User Behavior Modeling with Large-Scale Graph Analysis

Alex Beutel

May 2016
CMU-CS-16-105

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored under the National Science Foundation Graduate Research Fellowship Program (Grant no. DGE-1252522) and a Facebook Fellowship. This research was also supported by the National Science Foundation under grants numbered IIS-1017415, CNS-1314632 and IIS-1408924, as well as the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053. I also would like to thank the CMU Parallel Data Laboratory OpenCloud for providing infrastructure for experiments; this research was funded, in part, by the NSF under award CCF-1019104, and the Gordon and Betty Moore Foundation, in the eScience project. Also, I would like to thank the Open Cloud Consortium (OCC) and the Open Science Data Cloud (OSDC) for the use of resources on the OCC-Y Hadoop cluster, which was donated to the OCC by Yahoo!

Keywords: behavior modeling, graph mining, scalable machine learning, fraud detection, recommendation systems, clustering, co-clustering, factorization, machine learning systems, distributed systems

Dedicated to my family.
Jess, Mom, Dad, Stephen, Grandma, Papa, Nanny and Pa—I love you.

Abstract

Can we model how fraudsters work to distinguish them from normal users? Can we predict not just which movie a person will like, but also why? How can we find when a student will become confused or where patients in a hospital system are getting infected? How can we effectively model large attributed graphs of complex interactions?

In this dissertation we understand user behavior through modeling graphs. Online, users interact not just with each other in social networks, but also with the world around them—supporting politicians, watching movies, buying clothing, searching for restaurants and finding doctors. These interactions often include insightful contextual information as attributes, such as the time of the interaction and ratings or reviews about the interaction. The breadth of interactions and contextual information being stored presents a new frontier for graph modeling.

To improve our modeling of user behavior, we focus on three broad challenges: (1) modeling abnormal behavior, (2) modeling normal behavior and (3) scaling machine learning. To more effectively model and detect abnormal behavior, we model *how* fraudsters work, catching previously undetected fraud on Facebook, Twitter, and Tencent Weibo and improving classification accuracy by up to 68%. By designing flexible and interpretable models of normal behavior, we can predict *why* you will like a particular movie. Last, we scale modeling of large hypergraphs by designing machine learning systems that scale to hundreds of gigabytes of data, billions of parameters, and are 26 times faster than previous methods. This dissertation provides a foundation for making graph modeling useful for many other applications as well as offers new directions for designing more powerful and flexible models.

Acknowledgements

First and foremost, I want to thank my entire family for fostering my curiosity to get me to graduate school and then supporting me through it. Thank you Jess, for giving me such deep love and happiness for the past five years. Thank you Mom and Dad, for always encouraging me to do my best, not just in my work but as a person, for being my compass through graduate school, and for always being there, even in my difficult times. Thank you Stephen, for always being a much-needed grounding force in my life, consistently providing clarity when I was lost. Thank you Papa and Grandma—it is your unreasonable confidence in me and combination of intellectual idealism and real-world perceptiveness that has brought me here and helped me succeed. Nanny and Pa—I think about you often and deeply wish you were still here. Thank you to my many aunts, uncles, cousins, and Jess's entire family, who have all been there for me at key points in graduate school.

It cannot be overstated the role that my advisors had in my thesis, my time in graduate school, my career, and my life going forward. I want to thank Christos Faloutsos, for his patience and persistence with me as a young student, and his support throughout graduate school. He has taught me how to be an academic, and more broadly, how to simultaneously be a mentor, colleague and friend. Additionally, I want to thank Alex Smola, who has, since even before he came to CMU, pushed me and my research forward, making me learn new concepts and broaden my perspective. From technical insights to career opportunities, he repeatedly guided me down paths that I didn't know were possible. I am forever grateful to you both.

Additionally, I want to thank the rest of my committee: Geoff Gordon and Philip Yu. Thank you both for your comments, feedback and guidance that significantly shaped and focused this thesis.

Graduate school has been an exciting experience primarily because of my friends. They have helped me through courses, been my collaborators, outlet for relaxation, and general source of support. Everyone in Christos's database group has become like family. Vagelis Papalexakis—despite entering graduate school at the same time, you have been an academic big brother to me, both literally and figuratively. I want to thank Aditya Prakash, Danai Koutra, Leman Akoglu, Polo Chau, and U Kang for creating a welcoming environment, mentoring me as a young graduate student and continuing to be be there for me long after leaving CMU. I also want to thank Neil Shah, Jay-Yoon Lee, Rishi Chandy, Bryan Hooi, Hemank Lamba, Hyun Ah Song, Dhivya Eswaran and Kijung Shin for making the database group awesome, being fun to work with, hang out with, and everything in between.

Beyond those in the database group, I have had many amazing collaborators without whom this dissertation would not have been possible—Abhimanu Kumar, Meng Jiang, Amr Ahmed, Chao-Yuan Wu, Kenton Murray, Partha Talukdar, and Stephan Günnemann. All of the work in this dissertation came from truly collaborative endeavors, and I feel lucky to have had such great friends to work with in these efforts. Additionally, I would like to thank Venkat Guruswami, Peng Cui, Eric Xing and Qirong Ho who also offered their valuable insights at key points in my research.

Last, but certainly not least, I want to thank my friends from outside of my research life. Thank you Matteus Tanha, Anthony Brooks, David Naylor, Nico Feltman, David Witmer, Richard Wang, John Dickerson, Dougal Sutherland, Elie Krevat, Andrew O'Rourke and the many, many others that have made me love Pittsburgh and my time here.

I also want to thank Marilyn Walgora and Deborah Cavlovich. You have both been essential in making my time at Carnegie Mellon go smoothly and pleasantly; I greatly appreciate all of your help and extra effort, far beyond what is reasonable. Marilyn—enjoy retirement! You deserve the break one thousand times over.

One of the most exciting parts of my time in graduate school has been my summer internships. I have been extremely lucky to have had great mentors every summer. Thank you Wanhong Xu, Chris Palow, Lars Bakstrom, Markus Weimer, Vijay Narayanan, Ed Chi, and Zhiyuan Cheng for providing such wonderful environments for research, and for grounding my work back at Carnegie Mellon. In particular, I want to thank Wanhong Xu, who in my first summer as an intern at Facebook, got me started working on fraud detection, and for which his intuition on the problem provided the foundation for much of the rest of my fraud detection research in graduate school. Additionally, I want to thank Ed Chi, both for making my time at Google interesting, and also for his selfless support during my job search.

Additionally, I want to thank Tina Eliassi-Rad, who has been indispensable in her guidance and support throughout my job search, and also always makes conferences far more fun.

I also do not want to forget how I got to graduate school. Thank you to my many friends and colleagues at Duke University who helped me get into graduate school and continued to be sources of support throughout graduate school. In particular, I want thank Pankaj Agarwal and Thomas Mølhave for making me fall in love with research and continuing to be mentors long after I left Duke. Additionally, I want to thank all of my childhood, high school and college friends who have kept me tied to the real world and worked to keep me grounded despite my esoteric explorations. You all know who you are.

This is a bittersweet end to a consequential period of my life. Thank you all for making it so.

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction and Background

Chapter 1

Introduction

Can we model how fraudsters work to distinguish them from normal users?
Can we predict not just which movie a person will like, but also why?
How can we effectively ***model large attributed graphs*** *of complex interactions?*

As "Big Data" has become pervasive, organizations in every industry are storing as many actions and interactions as they can. For both academia and industry these massive databases have significant untapped potential, with increasing investment in trying to discover useful patterns. These databases can all be viewed as large hypergraphs—graphs among entities of different types with many attributes contextualizing their interactions. Through modeling these interactions, computer science can provide insights beyond what we can see and understand individually and propel many fields forward. With more information being stored than ever before, we now have the opportunity to move beyond only predicting whether two entities have interacted but also understanding the context of those interactions based on the many meaningful attributes available. This rapid expansion in the types of interactions and contextual information being stored presents a new frontier for graph modeling, enabling new applications and presenting new challenges in data mining and scalable machine learning.

To give some examples: online users interact not just with each other in social networks, but also with the world around them—supporting politicians, watching movies, buying clothing, searching for restaurants and even finding doctors. These interactions often include insightful contextual information as attributes, such as the time or location of the interaction and ratings or reviews about the interaction. There are similar hypergraphs in healthcare interactions among patients, doctors, diseases and treatments, as well as in educational interactions among students, teachers, subjects and educational resources.

In each of these fields, researchers have been able to extract useful knowledge just from the graph structure, e.g., predicting what movies a person will like and finding communities of people with similar interests. However, to provide more insightful understanding, we need to model not just the interactions but also the context of the interactions, finding the meaningful attributes within a graph and imbuing our models of those attributes with our intuition. Making use of all of these data effectively presents many exciting research challenges, ranging from designing models that capture the relevant patterns in large, attributed hypergraphs to systems for scaling learning of these complex models. Focusing on where classic techniques do not meet real-world challenges, we can develop holistic solutions that maximize real-world impact. To do this, we bridge insights from applications, modeling and scalable machine learning systems.

Online User Behavior			
Modeling Abnormal Behavior (Part II)		**Modeling Normal Behavior (Part III)**	
Ch. 3	Detect Fraud in Static Graphs	Ch. 6	Flexible Models for Normal and Abnormal Behavior
Ch. 4	Detect Fraud in Graphs with Time	Ch. 7	Interpretable Recommendations
Ch. 5	Detect Fraud in Graphs with Multiple Attributes	Ch. 8	Explaining Recommendations
Scalable Machine Learning (Part IV)			
Ch. 9	Distributed Modeling of Attributed Hypergraphs		
Ch. 10	Fast in the face of Stragglers		

Table 1.1: Full-stack approach to user behavior modeling.

1.1 Overview and Contributions

This dissertation focuses on modeling online user behavior, in particular developing new mathematical and computational techniques where classic techniques fail. An overview of this dissertation can be seen in Table 1.1, with the dependency hierarchy shown in Figure 1.1. We give a more detailed outline of the dissertation below.

1.1.1 Modeling Abnormal Behavior

How can we detect fraudulent user behavior?

Fraud on services like Amazon, Facebook and Twitter deceives honest users, disrupts our models of normal behavior and erodes the value of these services. Therefore, in Part II, we focus on the crucial problem of detecting and removing fraud.

From a graph perspective, fraud is a set of paid edges; for example, fake Page Likes are paid edges in the "who-Likes-what" graph of Facebook, and fake reviews are paid edges in the "who-rates-what" graph of Amazon. Fraud of this type often creates suspiciously dense subgraphs, leaving a variety of anomalous patterns in the data [108, 111, 112, 174]. In Chapter 3, we offer a novel technique, CatchSync [108] for detecting these anomalous patterns in static graphs.

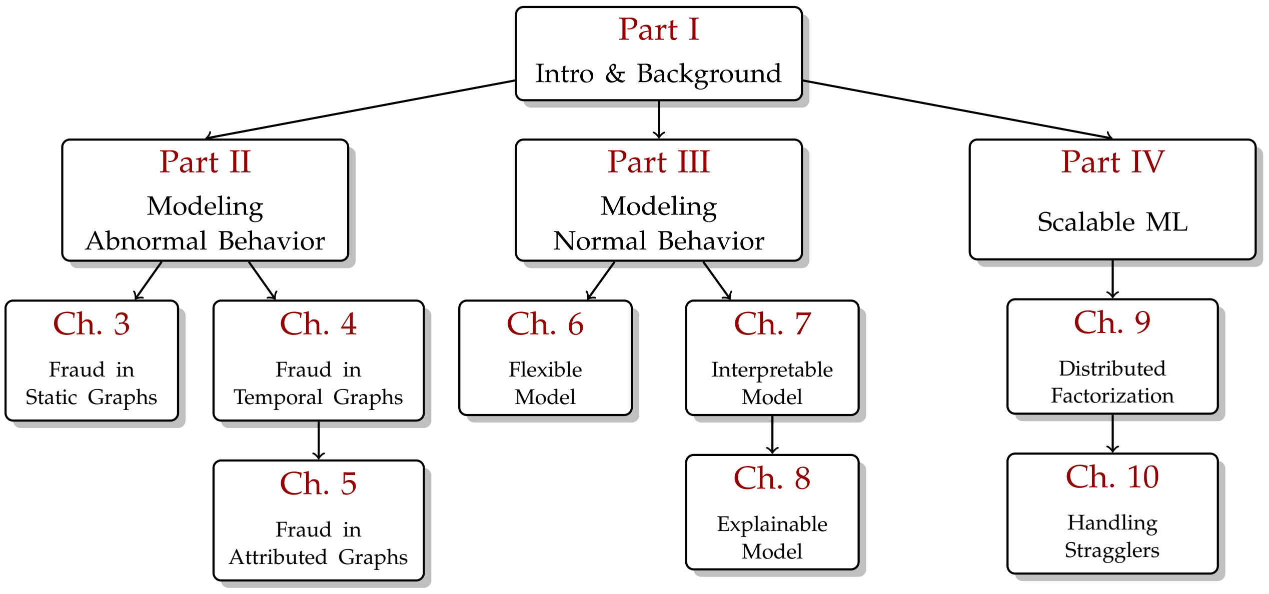
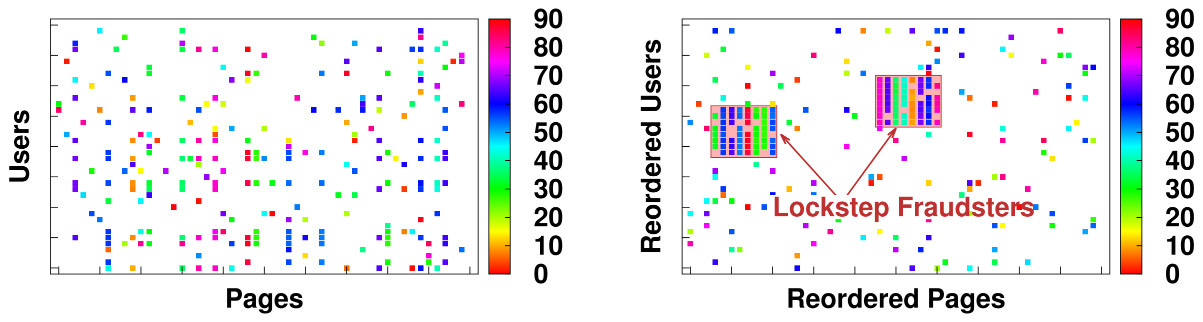


Figure 1.1: Dependency hierarchy for this dissertation.



(a) Before CopyCatch

(b) After CopyCatch

Figure 1.2: **Modeling abnormal behavior for fraud detection:** As shown in this example, CopyCatch detects suspicious lockstep behavior.

Because honest communities can also create dense subgraphs, they are difficult to distinguish from fraudulent behavior using only graph structure. Therefore, we turn our focus to using contextual information that leaves distinguishing patterns that are *costly* for fraudsters to avoid [33, 100, 107, 190].

In Chapter 4, we offer the CopyCatch algorithm [33] that detects temporally-coherent dense subgraphs or *lockstep behavior*—groups of users who perform the same action at approximately the same time. A visual demonstration of CopyCatch can be seen in Figure 1.2. Because fraudsters are obligated to deliver their product (fraud) in a short timespan, CopyCatch is difficult for adversaries to avoid. Furthermore, CopyCatch has high precision because normal users' actions are often uncorrelated with time. The CopyCatch implementation can distribute learning over a thousand machines, searches over billions of Page Likes and was relied on by Facebook for years to detect Page Like fraud.

Building on the success of contextual information in CopyCatch, we examine in Chapter 5 how we can easily incorporate multiple types of contextual data simultaneously. We offer a general suspiciousness metric and matching CrossSpot algorithm [107] to flexibly incorporate additional costly features, such as IP addresses, for finding spam and fraud. Our flexible search tool can take many different components of user behavior and find which components are most indicting for a particular group of users.

Contributions:

- **Detect Fraud in Static Graphs:** In Chapter 3, we offer a novel algorithm for detecting fraud in static graphs. Our algorithm, CatchSync, runs on graphs with billions of edges, improves detection accuracy by up to *36%*, and finds previously undetected fraud on Tencent Weibo and Twitter.
- **Detect Fraud in Graphs with Time:** In Chapter 4, we make use of the temporal graph patterns of fraud to detect fake Page Likes on Facebook. Our algorithm, CopyCatch, is implemented to run on Hadoop over Facebook's *10 billion edge graph* and was used at Facebook to detect fraud.
- **Detect Fraud in Graphs with Multiple Attributes:** In Chapter 5, we offer a metric for suspiciousness over that can combine many different signals. By using graph structure along with temporal and IP address patterns, our algorithm, CrossSpot, detects both hashtag hijacking and retweet boosting on Tencent Weibo and improves accuracy by up to *68%*.

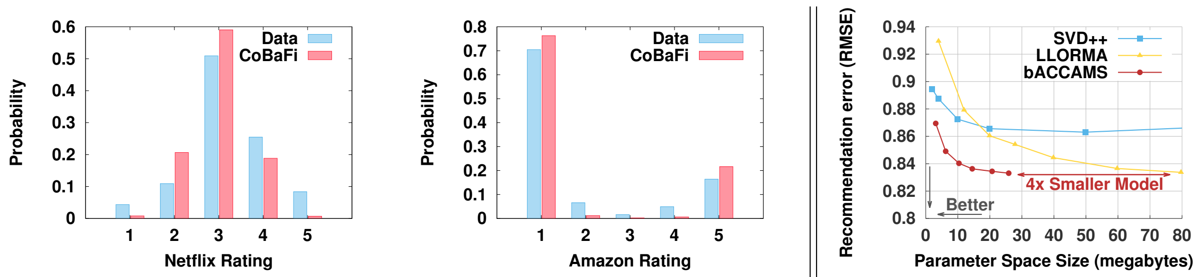
Impact:

- CatchSync, presented in Chapter 3, was a *Best Paper Award finalist* at *KDD* 2014.
- CopyCatch, presented in Chapter 4, was *used at Facebook* for years, and was extended for use on Instagram [42] and YouTube [143].
- We were granted a *patent* for CopyCatch [34], which was assigned to Facebook.
- CopyCatch received an Editor Highlight in the ACM Computing Review.
- CopyCatch has been included in Carnegie Mellon University's "Multimedia Databases and Data Mining" course (15-826) and the University of Florida's "Social Network Computing" course (CIS 6930).
- These insights were built upon for detecting fraudulent reviews on Flipkart [100].

1.1.2 Modeling Normal Behavior

How can we understand a user's preferences and predict their future actions?
How can we explain our predictions?

Online, we can interact with each other and the world around us at an unprecedented scale. Through modeling user behavior, we can understand these interactions and help sort through the deluge of options available online. In Part III, we examine new models that enable us to better to understand and predict user behavior.



(a) Netflix's Gaussian ratings (b) Amazon's bimodal ratings (c) ACCAMS

Figure 1.3: **Improved recommendations:** Examples of modeling normal behavior for capturing (a) Gaussian and (b) bimodal ratings, as well as (c) predicting ratings with a significantly smaller model .

Because recommender systems have been based on general matrix models, they make many implicit assumptions that do not match the reality of user behavior data. In particular, they typically assume all users and all items are equally similar and that there is some "correct" rating for each item, with some slightly personal perturbations. In Chapter 6, we challenge these assumptions, finding clusters of users and non-Gaussian rating patterns, as seen in Figure 1.3(a–b). We offer the CoBaFi model [31], which we find effectively clusters fraudulent users and finds polarizing items.

While recommendation accuracy is important, users are more likely to follow these recommendations if they are accompanied by an explanation. Thus, in Chapters 7 and 8 we offer simple, interpretable models that can offer explanations for recommendations. Previous factorization-based approaches create large models that are difficult to explain. In Chapter 7 we take a radically different approach to recommendation by learning a small set of discrete latent attributes about users and items, and the predicted ratings that accompany these attributes. That is, we learn an additive co-clustering model, ACCAMS [29], of the user and item ratings matrix. As seen in Figure 1.3(c), ACCAMS matches state-of-the-art prediction accuracy on Netflix with a model $^1/_4$ of the size. We build on this approach in Chapter 8 to directly offer explanations by predicting what words a given user would use when describing how he feels about a particular item. We offer an additive Poisson co-clustering model [229], enabling us to jointly model ratings and text reviews and to explain our recommendations with words.

Across all of this research, the models are designed to better understand attributes (ratings and reviews). As a result, these models are both more accurate and help explain *why* a user will like a particular item.

Contributions:

- **Flexible Models for Normal and Abnormal Behavior:** In Chapter 6, we design a more flexible model of user behavior. Our model, CoBaFi, models both Gaussian and bimodal ratings as well as clusters spammy users. We demonstrate that CoBaFi improves predictive accuracy by up to *17%* and find that services vary greatly in their rating patterns.

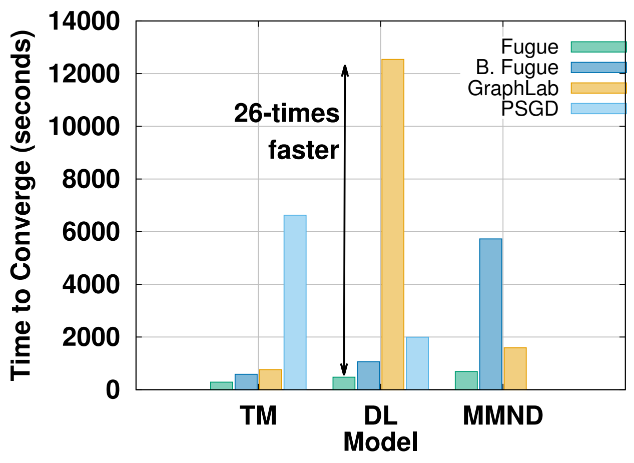


Figure 1.4: **Fugue is 26 times faster:** Improvements in distributed machine learning with Fugue for topic modeling, dictionary learning, and community detection.

- **Interpretable Recommendations:** In Chapter 7, a succinct, interpretable recommender system based on co-clustering. We find that our model, ACCAMS, matches state-of-the-art methods in accuracy while having a *4 times smaller model*.
- **Explaining Recommendations:** In Chapter 8, we build on the additive co-clustering model of Chapter 7 to *explain* recommendations to users. We offer PACO, a Poisson additive co-clustering model with an efficient sampling algorithm to predict personalized review words.

Impact:

- ACCAMS (Ch. 7) has been open-sourced at cs.cmu.edu/~abeutel/accams.
- Since it was published in *WWW* earlier this year, ACCAMS has been accessed over 150 times from 14 universities and over 20 countries.

1.1.3 Scalable Machine Learning

How can we efficiently learn user behavior models over many machines?
How can our learning algorithms adapt to the messy realities of "the cloud," such as stragglers?

To scale complex behavior models to the high volume and variety of data present online, it is important to efficiently distribute learning of our models. In Part IV, we exploit the structure of our models and the stochastic nature of learning [30, 32, 127]. While some methods above, such as CopyCatch (Ch. 4), include custom implementations that scale to large graphs, many other methods are based on more common latent factor structure. Therefore, we focus on effectively learning massive latent factor models: (1) for attributed hypergraphs, (2) at scale, (3) quickly, and (4) flexibly.

In Chapter 9 we examine how to learn latent factor models for complex datasets where entities have relations of multiple types, represented as tensors and joined tensors. In order to scale, we need to handle not just "big data" but also learn a model for many entities with complex interactions. We offer FlexiFaCT [30] to scale learning of huge models, with billions of parameters, from big datasets. FlexiFaCT scales effectively by understanding the intrinsic structure and independence assumptions of these models and thus partitioning both data and model across many machines. As a result, FlexiFaCT can scale learning to billions of parameters.

Distributed machine learning often relies on unreliable clusters, with concurrent programs causing slow machines or even high-demand machines being preempted. In Chapter 10 we offer a novel solution for fast machines to not waste time when waiting for slower machines, called *stragglers*. By exploiting the stochastic nature of machine learning, Fugue [127] is up to 26 times faster than competing methods, as seen in Figure 1.4.

Both of these solutions exploit the particular structure of relational models and the unique properties of machine learning algorithms to address often frustrating realities of learning these models in the real world. By focusing on the particular properties of learning this broad class of models we develop significantly improved solutions.

Contributions:

- **Distributed Modeling of Attributed Hypergraphs:** In Chapter 9, we offer FlexiFaCT, a distributed system for coupled tensor factorization. FlexiFaCT is flexible to a variety of objectives and scales to *billions of parameters*.
- **Fast in the face of Stragglers:** In Chapter 10, we offer a novel technique for efficient distributed learning, even if some machines are much slower than others, called stragglers. We demonstrate that our system, Fugue, is faster than competitors for topic modeling, dictionary learning, and community detection, offering up to *26 times faster* speeds.

Impact:

- FlexiFaCT [30] is the *most cited paper* of *SDM* 2014[1].
- FlexiFaCT (Ch. 9) is open-sourced on Github and has been forked seven times.
- FlexiFaCT (Ch. 9) and Fugue (Ch. 10) were taught in Carnegie Mellon's graduate course "Machine Learning with Large Datasets" (10-805) both in 2014 and 2015.

[1]Based on scholar.google.com/scholar?cites=7979229435915048938, as of January 1, 2016.

1.2 Overarching Thesis Statements

Across all parts of this dissertation, we follow three primary guiding principles:

Driving Thesis Statements

T1. **Making use of Interaction Context:** By modeling rich edge attributes in graphs, we can give greater insight into user behavior.

T2. **Normal and Abnormal Behavior—Two Sides of the Same Coin:** By unifying fraud detection and recommendation perspectives, we can use the same user behavior modeling techniques to improve both tasks.

T3. **Bridging Models and Scalable Systems:** By jointly understanding model structures, learning algorithms, and distributed systems, we can better scale learning of user behavior models.

At the end of this dissertation, in Section 12.1, we will review how the models and algorithms offered support these theses.

Chapter 2

Preliminaries and Background

We begin with an overview of the notation and concepts that will be used throughout this dissertation. In discussing background concepts, we will cite the general related work, but will go into more depth on these topics in the relevant parts of this thesis.

Unless noted otherwise, we will use the following notation. We will use capital script characters, e.g., $\mathcal{E}$, to denote sets, capital boldface script characters, e.g., $\boldsymbol{\mathcal{X}}$ to denote a tensor, capital boldface non-script characters, e.g., $\mathbf{Y}$ to denote a matrix, lowercase boldface character, e.g., $\mathbf{y}$, to denote a vector, and non-boldface non-script characters, e.g., i, to denote scalars. $\boldsymbol{\mathcal{X}}_{i,j,k}$ denotes the scalar in the (i, j, k) position of the tensor $\boldsymbol{\mathcal{X}}$, $\mathbf{Y}_{i,j}$ denotes the scalar in the (i, j) position of matrix $\mathbf{Y}$, and $\mathbf{y}_i$ denotes the scalar in the ith position of vector $\mathbf{y}$. We use $\mathbf{Y}_{i,*}$ to denote the vector of scalars $\mathbf{Y}_{i,j}$ for all j. We will often denote different instances of a matrix or tensor with an exponent, e.g., $\boldsymbol{\mathcal{X}}^{(\ell)}$.

2.1 Graph Structure

As given by the title, this dissertation takes a graph-based perspective to user behavior modeling. Therefore, we begin by formally defining the types of graphs we will consider. A graph is broadly defined by a set of nodes (also called vertices) $\mathcal{V}$ and a set of edges $\mathcal{E}$ that connect nodes. In many of our examples, nodes are users or items and edges are interactions between these nodes. By varying the precise definition of $\mathcal{V}$ and $\mathcal{E}$ we can describe different graph patterns:

Undirected graphs In undirected graphs an edge describes a symmetric relation between two nodes. That is, we say that an edge exists between $u \in \mathcal{V}$ and $v \in \mathcal{V}$ if $(u, v) \in \mathcal{E}$ *or* $(v, u) \in \mathcal{E}$. As such, $\mathcal{E} \subseteq \{(u, v) | u \in \mathcal{V}, v \in \mathcal{V}\}$. Facebook's and LinkedIn's friendship graphs are examples of undirected graphs of this type, where each node is a person and there is an edge between two users if they are friends.

Bipartite graphs In many of our settings, we observe users interacting with items in the world around them. In this case, we see that there are two classes of nodes and all edges represent interactions between nodes of opposite classes. For example, users Like pages on Facebook or users watch movies on Netflix. In these cases, we can consider our nodes to come from two distinct sets: $\mathcal{V} = \mathcal{U} \cup \mathcal{W}$ and $\mathcal{U} \cap \mathcal{W} = \emptyset$. As such, edges represent relations between nodes in sets $\mathcal{U}$ and $\mathcal{W}$; formally, $\mathcal{E} \subseteq \{(u, v) | u \in \mathcal{U}, v \in \mathcal{W}\}$. Bipartite graphs cover a wide variety of applications online, such as Facebook's "who-Likes-what" graph, Netflix's "who-watches-what" graph, Amazon's "who-buys-what" graph, Yelp's "who-reviews-what" graph, and many others.

Directed graphs In some applications, relationships are not symmetric. For example on Twitter, just because User-A follows User-B does not mean that User-B follows User-A. In this case, edges are given as ordered sets, where there is an edge from u to v if $(u, v) \in \mathcal{E}$. In this case, just because $(u, v) \in \mathcal{E}$ does not mean that $(v, u) \in \mathcal{E}$. Note, we can sometimes view directed graphs of this form similar to bipartite graphs, where we separate a node u into two distinct entities, one performing actions and one receiving actions. For example, we can consider a user on Twitter either by who he follows or who follows him.

Hypergraphs In some cases, edges can represent relations between more than two entities. This is called a hypergraph; most broadly edges can represent relations between any subset of nodes $\mathcal{E} \subseteq 2^{\mathcal{V}}$. In many of the applications considered here, we focus on a constrained type of hypergraph, where there are multiple types of nodes and edges represent relations among one node of each type. For example, on YouTube we may observe the relation User-A watches Video-1 from IP-1. In this case, we have three classes of nodes: $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \mathcal{V}_3$ and edges represent undirected relations among nodes of each class: $\mathcal{E} = \{(u, v, w) | u \in \mathcal{V}_1, v \in \mathcal{V}_2, w \in \mathcal{V}_3\}$.

Subgraphs In some cases, we only want to consider part of a graph. To do this, we focus on *subgraphs*. Given a subset of nodes $\mathcal{V}' \subseteq \mathcal{V}$, we can observe the induced subgraph $G' = (\mathcal{V}', \mathcal{E}') \subseteq G$ where $\mathcal{E}' = \{(u, v) \in \mathcal{E} | u, v \in \mathcal{V}'\}$. That is, a subgraph induced over a set of nodes is the graph consisting of that set of nodes and the edges among them from the original graph.

Joined graphs The graph formulations above can describe many types of relationships observed online, but in most applications there are more than one type of relation. As such, we will in some cases jointly consider multiple graphs. For example, we could jointly consider Facebook's "who-Likes-what" graph $G_1 = (\mathcal{U} \cup \mathcal{W}, \mathcal{E}_1)$ at the same time as considering Facebook's friendship graph $G_2 = (\mathcal{U}, \mathcal{E}_2)$. In this case we have two different sets of edges, but the user nodes $\mathcal{U}$ are the same in both graphs.

Attributed nodes While binary edge relations can be very informative, they are not the only type of data available. In many applications, we may observe one or more attributes about the nodes in our graph. For example, on many services we may observe the time that an account was created or the latitude and longitude of the account holder's billing address. In these cases, we can consider that we observe a function $a : \mathcal{V} \to \mathbb{R}^d$, where we observe d real valued attributes for each node. In some cases, our attribute data may be more complex than a real valued vector. A subtle point is that if the observed attributes are unordered discrete values, we can consider the node attributes to be a joined graph.

Attributed edges Another type of data of particular importance in this dissertation are edge attributes. In many applications, interactions between users and the world around them contain lots of context. For example, we may observe the time at which a user watches a movie on Netflix or reviews a restaurant on Yelp. We can formulate this contextual data as observing a function $c : \mathcal{E} \to \mathbb{R}^d$, where we observe d real-valued attributes for each edge. In some cases, our contextual data may be more complex than a real-valued vector.

Here too, we find that if our edge attributes are unordered discrete values then we can formulate the dataset as a hypergraph. In some cases we will convert real-valued edge attributes to unordered discrete attributes so as to use hypergraph methods (and described later tensor methods). This choice highly depends on the underlying assumptions of the model, and if viewing the attribute as unordered and discrete is reasonable.

2.2 Mathematical data structures

While thinking about real world interactions as large, attributed hypergraphs is a powerful formalization, it is often useful to refer to these graphs in terms of linear algebra primitives.

Graphs as matrices One of the most common formulations for graphs is as matrices. If we have a bipartite graph $G = (\mathcal{U} \cup \mathcal{W}, \mathcal{E})$ where $|\mathcal{U}| = n$ and $|\mathcal{W}| = m$, then we can observe matrix $\boldsymbol{X} \in \mathbb{R}^{n \times m}$, where

$$\boldsymbol{X}_{u,v} = \begin{cases} 1 & \text{if } (u,v) \in \mathcal{E} \\ 0 & \text{if } (u,v) \notin \mathcal{E} \end{cases} \tag{2.1}$$

Under this formulation, unipartite undirected graphs are given by symmetric matrices.

If we observe a single attribute, such as the rating in the user-rates-movie graph of Netflix, then we will often consider the graph to be represented by matrix $\boldsymbol{X} \in \mathbb{R}^{n \times m}$, where

$$\boldsymbol{X}_{u,v} = \begin{cases} c((u,v)) & \text{if } (u,v) \in \mathcal{E} \\ 0 & \text{if } (u,v) \notin \mathcal{E} \end{cases} \tag{2.2}$$

Depending on the semantics of our application, we will consider the cells in the matrix for which there is no edge to be either 0 or *missing*. For example, if the attribute is the rating a user gives a movie, then we do not want to consider a user to give a 0 rating to all movies he hasn't watched; rather, we will consider those values to be unobserved and thus missing. However, if the attribute is the number of times that a user visits a particular website or the amount of time the user has spent on that website, then all websites that the user has not visited should in fact have an observed value of 0. In applications where cells have missing values, we will slightly abuse the notation to iterate over observed cells in the matrix by being able to check if $(u, v) \in \boldsymbol{X}$.

Hypergraphs as tensors As described above, many applications contain relations among more than two types of nodes, thus creating a hypergraph $G = (\mathcal{V}_1 \cup \mathcal{V}_2 \cup \mathcal{V}_3, \mathcal{E})$, where $|\mathcal{V}_1| = n_1$, $|\mathcal{V}_2| = n_2$, and $|\mathcal{V}_3| = n_3$. In these cases, we can represent our hypergraph as a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$, where

$$\boldsymbol{\mathcal{X}}_{i,j,k} = \begin{cases} 1 & \text{if } (i,j,k) \in \mathcal{E} \\ 0 & \text{if } (i,j,k) \notin \mathcal{E} \end{cases} \tag{2.3}$$

Additionally, similar to the attributed graphs above, the tensor syntax can be used to encode an attribute on the edges of the hypergraph.

Joined graphs In cases where we have joined graphs with overlapping nodes, we can view each graph as a separate matrix or tensor and then define models that include multiple data sources.

2.3 Model Structure

To effectively model graphs of user behavior, we will build on a couple common model structures. We begin now by outlining these models.

2.3.1 Factorization

One of the most prevalent techniques to model graph data is based on decomposing or approximating a matrix or tensor by a product of component parts, called matrix or tensor factorization. Likely, the oldest technique in this class of models is the *singular value decomposition* (SVD), where a matrix $\boldsymbol{X} \in \mathbb{R}^{n \times m}$ can be decomposed into matrices $\boldsymbol{U} \in \mathbb{R}^{n \times k}$, $\boldsymbol{V} \in \mathbb{R}^{m \times k}$, and $\boldsymbol{\Sigma}$ is a diagonal $k \times k$ matrix, such that $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top = \boldsymbol{X}$ [203]. The size of the model is given by k, which is often referred to as the *rank* of the matrix, or if the model is truncated, as the rank of the model. The SVD has many interesting properties, but most are outside the scope of this dissertation.

The generalization of the SVD to tensors is called the PARAFAC or Canonical Polyadic (CP) decomposition [93]. In the PARAFAC decomposition, a tensor $\boldsymbol{\mathcal{X}} \in \mathbb{R}^{n_1 \times n_2 \times n_3}$ is decomposed into component matrices $\boldsymbol{U} \in \mathbb{R}^{n_1 \times k}$, $\boldsymbol{V} \in \mathbb{R}^{n_2 \times k}$, and $\boldsymbol{W} \in \mathbb{R}^{n_3 \times k}$, such that $\boldsymbol{\mathcal{X}} = \boldsymbol{U} \otimes \boldsymbol{V} \otimes \boldsymbol{W}$. Here, $\otimes$ represents the outer product, and as a result, a given entry $\boldsymbol{\mathcal{X}}_{u,v,w} = \sum_{r=1}^{k} \boldsymbol{U}_{u,r}\boldsymbol{V}_{v,r}\boldsymbol{W}_{w,r}$. Like the SVD, the PARAFAC decomposition has been well studied and has many interesting properties, but is not a primary focus of this dissertation.

Both the SVD and the PARAFAC decomposition are precise algorithms for learning a decomposition, requiring certain properties of the underlying data (e.g., that there are no missing values) and offering certain guarantees of the resulting decompositions. However, we can loosen these guarantees and examine the broader class of factorizations. (We will focus here on matrix factorizations, but the explanation can be easily generalized to tensor factorizations.) Broadly speaking, matrix factorization approximates a matrix $\boldsymbol{X}$ by $\boldsymbol{U}\boldsymbol{V}^{\top}$. We can view our model, $\boldsymbol{U}$ and $\boldsymbol{V}$, from different perspectives. One way to view the model is as a sum of rank-1 models:

$$\boldsymbol{X} \approx \sum_{k=1}^{K} \boldsymbol{U}_{*,k}\boldsymbol{V}_{*,k}^{\top} \tag{2.4}$$

In an SVD, each rank-1 model captures an eigenvector of your matrix (or graph). These eigenvectors have been used to find trusted communities [118]. More broadly, each rank-1 factorization can be thought of as capturing a block of common behavior or cluster in the matrix [134, 177]. These groups of common behavior have been found to match genres in movie-ratings matrices, topics in document-by-word matrices [36], functional regions in the brain [175], and many other interesting patterns.

An alternative perspective on factorizations is to view them as latent factor models finding latent representations or embeddings of nodes in the graph. That is, if our matrix contains the user-rates-movie graph, then $\boldsymbol{U}_{i,*}$ is the latent representation of user i and $\boldsymbol{V}_{j,*}$ is the latent representation of movie j; we will often use $\mathbf{u}_i$ and $\mathbf{v}_j$ for shorthand for these per-user and per-item embeddings (note, indices i and j here don't return the scalar in the vector but instead denote which vector we are using). Under this perspective, each value in the matrix $\boldsymbol{X}_{i,j}$ is approximated by $\mathbf{u}_i \cdot \mathbf{v}_j$. Because of this representation, matrix factorization is often referred to as learning a bilinear model. (Tensor factorization, as defined above, is a tri-linear model.)

Joining both perspectives, we can view the latent representations as offering the nodes "membership" or "preferences" across the different communities/topics/genres in the factorization.

A wide variety of factorization models have been proposed, adding different constraints or components to match different application concerns. We will discuss the specific relevant related work in each part of this dissertation.

2.3.2 Clustering

Aside from factorization models, we also build on clustering algorithms to understand and model graphs. Broadly speaking, clustering finds subsets of items in the data or nodes in the graph that are similar.

One view of clustering comes from the matrix perspective of our data. Given data matrix $\boldsymbol{X}$, we can consider each row to be a data point $\mathbf{x}_i$ and find groups of data points that are close together under a particular distance metric. Typically, we try to find a cluster center $\mathbf{v}$ such that many points are close to the center. While there are many different clustering variants, we will focus on a few different perspectives and design decisions in using clustering for user behavior modeling.

Global vs. local clustering Depending on our application, we may want all points to be assigned to clusters or we may want to just find certain types of well distinguished clusters. For example, if we want to cluster movies to discover genres, then we would want all movies to fall into a genre, and thus would perform a global clustering. Alternatively, if we want to find groups of fraudulent users, then we do not expect most users to fall into any fraudulent cluster and instead can focus on finding locally dense clusters.

Hard vs. soft clustering Another important distinction between different clustering techniques is if cluster membership is "hard" or "soft." Hard cluster membership means that a data point either belongs or does not belong to a cluster; for example, a movie is is either rated "PG-13" or its not. In contrast, soft cluster membership allows for data points to partially belong to a given cluster; for example, a movie can be considered a little scary. Mathematically, hard cluster membership is defined by binary assignment to clusters, and soft cluster membership is defined by real-valued, non-negative membership to clusters. K-means clustering is an example of hard clustering [22, 151]; non-negative matrix factorization [134] or topic modeling [36] can be interpreted as soft clustering.

Subspace clustering Many clustering techniques are based on a predefined distance metric between two points $d(\mathbf{x}, \mathbf{y})$. In some applications, we do not care that a cluster of points are similar in all dimensions, but rather that they are similar in a subset of dimensions. For example, a group of people can be considered similar if they like similar music, independent of their religious beliefs.

In subspace clustering, the algorithm must find both the set of data points that are similar as well as the set of features over which they are similar [126]. For example, if we define $\mathbf{m} \in \{0, 1\}^f$ and our metric to be Euclidean distance in a subspace

$$d_{\mathbf{m}}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{f} \mathbf{m}_i (\mathbf{x}_i - \mathbf{y}_i)^2,$$

then subspace learning must learn both cluster centers as well as the subspace $\mathbf{m}$. For a broad survey see [126].

Co-clustering Subspace clustering requires that sets of rows are similar over a set of columns. We can generalize these clustering ideas to *co-clustering* (also called *bi-clustering*), where we simultaneously cluster both the rows and columns of the data matrix [27]. In co-clustering, each row i is assigned to cluster $\mathbf{c}_i \in \{1 \ldots k\}$ and each column j is assigned to cluster $\mathbf{d}_j \in \{1 \ldots k\}$. We can then consider cluster assignments to be good if we can find a model $\boldsymbol{T} \in \mathbb{R}^{k \times k}$ such that $\boldsymbol{X}_{i,j}$ is approximated well by $\boldsymbol{T}_{\mathbf{c}_i, \mathbf{d}_j}$.

Co-clustering can be viewed as a matrix factorization in which $\boldsymbol{X} \approx \boldsymbol{U}\boldsymbol{T}\boldsymbol{V}^\top$, where $\boldsymbol{U}$ and $\boldsymbol{V}$ store the cluster assignments for the rows and columns respectively [198]. The technique was originally used for understanding the clustering of rows and columns of a matrix in biology [94]. However, many algorithms and applications have since been proposed for co-clustering, which we will discuss in more detail in the relevant sections of this dissertation.

Graph clustering The above clustering concepts can also be mapped to clustering over graphs. For example, clustering on the adjacency matrix can be viewed as clustering nodes based on who they connect to. Co-clustering can be viewed as jointly clustering both classes of nodes in a bipartite graph.

A significant amount of research in the graph mining community has also focused on graph clustering, with the goal of finding subgraphs that meet a pre-specified criteria. For example, one common challenging problem is to find the subgraph with the largest average degree [47]. A survey of graph clustering methods can be found [138].

2.4 Learning

So far we have focused on the types of data we will encounter as well as the types of models we will use to describe that data. However, how to learn the model parameters that best match the given data has not yet been described. For matrix factorization, we observe the data $\boldsymbol{X}$ but need to find the values of the model parameters $\boldsymbol{U}, \boldsymbol{V}$; for co-clustering, we need to find values of $\boldsymbol{T}$ as well as the cluster assignments $\mathbf{c}, \mathbf{d}$. We will now give a high-level overview of the way learning problems are framed.

Objective functions In order to learn the model parameters, we need to specify goals for a "good" setting of the model parameters. Generally speaking, as described above, we want our model to approximate our data well. To be more precise, we provide a loss function between the data and our model. For example, the SVD minimizes the Frobenius norm between our data and our model:

$$\arg\min_{\boldsymbol{U},\boldsymbol{\Sigma},\boldsymbol{V}} \|\boldsymbol{X} - \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top\|_F^2$$

Particularly common in this dissertation is the use of squared error for real-valued prediction on matrices with missing values:

$$\arg\min_{\boldsymbol{U},\boldsymbol{V}} \sum_{(i,j)\in\boldsymbol{X}} (\boldsymbol{X}_{i,j} - \mathbf{U}_{i,*} \cdot \mathbf{V}_{j,*})^2$$

Note, here we use the earlier specified notation that we can iterate over only the observed values in our matrix $\boldsymbol{X}$. In all cases, our goal is learn the values of our parameters so as to minimize the given objective.

Regularization and constraints In many cases, we would like to restrict the parameter search to meet certain application criteria. For example, it can be useful to have a non-negative matrix factorization, $\boldsymbol{U}, \boldsymbol{V} \geq 0$. In other cases, we may want part of the parameter space to be sparse, with only a few non-zero values. This can be encouraged through including the ℓ_1 norm over the parameters as regularization in the objective function:

$$\arg\min_{\boldsymbol{U},\boldsymbol{V}} \sum_{(i,j)\in\boldsymbol{X}} (\boldsymbol{X}_{i,j} - \mathbf{U}_{i,*} \cdot \mathbf{V}_{j,*})^2 + \|\boldsymbol{V}\|_1$$

A wide variety of different constraints and regularizations have been proposed through machine learning.

Optimization The field of optimization studies how to efficiently learn parameters that minimize the given objective. While there are many different optimization algorithms, we will primarily focus on those that match our data, model and application constraints. Of particular note are stochastic gradient descent [37] and alternating least squares [93, 104]. We will give a more in-depth explanation of the approaches in the relevant sections.

Bayesian models Apart from specifying our model by the approximation loss, we can also specify it as a generative Bayesian model. That is, we can describe a Bayesian model by which we believe our data was generated, and then learn the parameters of our Bayesian model that make the maximize the likelihood of our data being generated. We can use as an example Probabilistic Matrix Factorization [159]. Under this model, we believe that a particular rating $\boldsymbol{X}_{i,j}$ is generated by the Gaussian $\mathcal{N}(\mathbf{U}_{i,*} \cdot \mathbf{V}_{j,*}, \sigma^2)$. We want to learn the parameters that maximize the likelihood of the data, or alternatively framed, minimize the negative log-likelihood of the data:

$$\begin{aligned}
&\arg\min_{\boldsymbol{U},\boldsymbol{V}} \sum_{(i,j)\in\boldsymbol{X}} -\log \mathcal{N}(\boldsymbol{X}_{i,j} | \mathbf{U}_{i,*} \cdot \mathbf{V}_{j,*}, \sigma^2) \\
&\quad = \arg\min_{\boldsymbol{U},\boldsymbol{V}} \sum_{(i,j)\in\boldsymbol{X}} \frac{1}{2\sigma^2} (\boldsymbol{X}_{i,j} - \mathbf{U}_{i,*} \cdot \mathbf{V}_{j,*})^2
\end{aligned}$$

As we see above, this is equivalent to minimizing the squared error.

Under this Bayesian framework, the model [159] believes that each user's and each item's latent representation is generated from a spherical Gaussian distribution, e.g., $\mathbf{u}_i \sim \mathcal{N}(0, \sigma_u^2 \mathbf{I})$. These terms also contribute to the objective, as we try to minimize the entire negative log likelihood, but can be viewed as analogous to the regularization described previously.

By framing our model as a generative Bayesian model, we can make use of the broad set of literature on Bayesian learning, such as sampling and belief propagation. We will describe the relevant techniques at the corresponding parts of the dissertation.

Part II

Modeling Abnormal Behavior

Based on work previously published in
TKDD [111], *WWW* 2013 [33], and *ICDM* 2015 [107].

Introduction

How can we detect fraudulent user behavior?

In many domains, it is valuable to detect and understand anomalous behavior. Online, this challenge is even more important; as web services have increasingly relied on social data to provide information to their users, fraudulent behavior has increasingly become a threat. For just a few dollars, anyone can buy popularity, promote their own products or opinions, or in some cases slander a competitor. Fraudsters sell followers on Twitter, fake Page Likes on Facebook, upvotes on Reddit, ratings on mobile app stores, or reviews on Yelp and Amazon. This sort of fraudulent behavior deceives honest users and distorts our models of normal behavior, corroding the usefulness of these online services. The risks posed by fraud online are significant. In S-1 filings to the U.S. Securities and Exchange Commission, both Facebook and Twitter list deceptive behavior as risks that "may harm our reputation and negatively affect our business" and even state that 5% of their users are likely fake [70]. Because of the damaging effects of fraud, it is crucial that we detect and remove fraudulent behavior.

In all of the examples above, fraud is a group of purchased edges in a graph. Fake followers on Twitter are purchased edges in the "who-follows-whom" graph, illegitimate Page Likes are purchased edges in the "who-Likes-what" graph, and fake ratings on Netflix are edges in the "who-rates-what" graph. Because the attacks only require adding edges to the graph, previous content-based approaches for finding spam, such as analyzing users' tweets and profiles [11, 41, 180], will often miss this dubious behavior.

Instead, in the following chapters, we attack the fraudsters at their critical weakness. To be successful they need to add *many edges*. We therefore focus on modeling suspicious graph patterns and structures to detect and limit fraudsters. In pursuit of this goal, we offer the following contributions:

- **Detect Fraud in Static Graphs:** Central to all of the attacks above is that fraudsters are selling many edges. Therefore, in Chapter 3 we offer CatchSync, an algorithm to detect suspicious behavior based purely on graph structure. CatchSync has detected fraud on both Twitter and Tencent Weibo, improving over previous methods by up to 36%.
- **Detect Fraud in Graphs with Time:** In many applications, the temporal patterns of edges can be very insightful. We describe in Chapter 4 CopyCatch, a fraud detection system that makes use of temporal graph patterns to detect purchased Page Likes on Facebook. We offer a theoretical analysis of this graph based approach

and demonstrate the success of CopyCatch at Facebook.

- **Detect Fraud in Graphs with Multiple Attributes:** Interactions online often include rich contextual data, such as the IP address on which it took place as well as the time. To make use of the full variety of data, we design in Chapter 5 a novel metric of suspiciousness and fraud detection algorithm that makes use of all data available, while ignoring noisy data. Our algorithm, CrossSpot, meets all theoretical requirements and detects purchased retweets and hashtag boosting on Tencent Weibo.

II.1 Related Work

We begin with a brief survey of the related work in fraud and anomaly detection. In each of the following chapters we will discuss in more detail their relationship to the most relevant pieces of related work.

II.1.1 Spammer and Fraudster Detection

Detecting fraudsters and spammers online has been a central research focus in multiple communities and for many companies. In this effort, there have been a few different approaches taken to find the unique patterns left by fraudsters.

One branch of recent research to detect spammers has focused on content-based methods, primarily finding useful features about the users or the items [11, 28, 103, 114]. For example, Jindal *et al.* analyzed Amazon reviews, examining product, reviewer, rating, date, review title/body and feedbacks, to catch opinion spam [114]. Perez *et al.* proposed SPOT to catch suspicious Twitter profiles, by modeling text and malicious URLs in tweets and scoring their suspiciousness [180]. Our work in the following chapters is orthogonal to these approaches because we primarily focus on graph structure and group attacks.

A second branch of research has focused on directly exploiting the social and web linkage patterns to detect fraudsters. Many methods have been proposed based on belief propagation (BP) [15, 48, 171] or PageRank-like scores [41, 79, 90, 227]. For example, NetProbe [171] uses a small list of known fraudsters to blame all the fraudulent nodes on the graph.

II.1.2 Graph-based Anomaly Detection

Many algorithms have been developed for finding general anomalies in graphs [12, 45, 193], including discovering structural anomalies [66, 168] and propagating beliefs for surprising nodes [160, 204]. For example, AUTOPART [43] finds outlier edges across node groups, ODDBALL [17] finds near-cliques and stars, and [154, 179] focus on finding novel subgraphs in large networks.

SVD-based Methods: Decomposition methods have been widely used to understand graphs, including subspace clustering [105], community detection [49, 183], and pattern discovery [85, 119, 120, 206]. Implicitly, the SVD focuses on dense regions of a matrix. Prakash *et al.* proposed EigenSpokes which reads scatter plots of pairs of singular vectors to find patterns and communities [183]. These ideas were extended in [109, 110, 190] for finding other anomalous patterns based on SVD embeddings. Chen *et al.* extracted dense subgraphs using a spectral cluster framework [49]. For multimodal data, tensor decompositions have been applied in many applications [119, 206]. For example, high-order singular value can be thought to represent the importance of the cluster [105].

II.1.3 Subgraph Mining

One common approach to finding interesting patterns in graphs is to find interesting subgraphs. A number of subgraph mining algorithms have been investigated [55], such as mining frequent subgraph patterns [147, 231, 244], mining dense subgraphs [49, 80, 128, 222] and finding quasi-cliques [179, 213]. In addition, some variants of community detection algorithms have been proposed to discover dense clusters [74] and well-connected communities, such as the SVD-based methods described above, e.g., [183].

Of particular noteworthiness, significant work has focused on looking for dense subgraphs with high average degree [20, 23, 26, 138]. Charikar gave simple greedy approximation algorithms to find highly connected subgraphs of large average degree [47].

II.1.4 Co-clustering

As described in Section 2.3.2, we can sometimes view graph clustering as co-clustering over the adjacency matrix or similar matrices. As described previously, co-clustering is NP-hard, so many approaches use an approximation of the problem. There has been extensive research on co-clustering including [18, 27, 44, 63, 176], with applications to anomaly and intrusion detection [174]. Papadimitriou *et al.* [173] offer an iterative distributed algorithm for performing co-clustering with MapReduce.

Chapter 3

Detect Fraud in Static Graphs

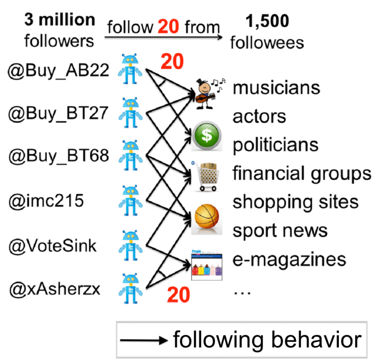
Given a directed graph of millions of nodes, how can we automatically spot anomalous, suspicious nodes, judging only from their connectivity patterns? Suspicious graph patterns show up in many applications, from Twitter users who buy fake followers, manipulating the social network, to botnet members performing distributed denial of service attacks, disturbing the network traffic graph. We propose a fast and effective method, CatchSync, which exploits two of the tell-tale signs left in graphs by fraudsters: (a) *synchronized* behavior: suspicious nodes have extremely similar behavior pattern, because they are often required to perform some task together (such as follow the same user); and (b) *rare* behavior: their connectivity patterns are very different from the majority. We introduce novel measures to quantify both concepts ("synchronicity" and "normality") and we propose a parameter-free algorithm that works on the resulting synchronicity-normality plots. Thanks to careful design, CatchSync has the following desirable properties: (a) it is *scalable* to large datasets, being linear on the graph size; (b) it is *parameter free*; and (c) it is *side-information-oblivious*: it can operate using only the topology, without needing labeled data or profile information. We applied CatchSync on two large, real datasets *1-billion-edge* Twitter social graph and *3-billion-edge* Tencent Weibo social graph, and several synthetic ones; CatchSync consistently outperforms existing competitors, both in detection accuracy by 36% on Twitter and 20% on Tencent Weibo, as well as in speed.

3.1 Introduction

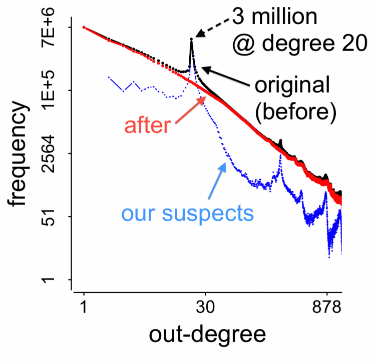
Given a directed graph within millions of nodes, can we tell which nodes are suspicious just based on the graph structure? For many applications, fraudsters try to manipulate networks for personal gain. For example, in social networks, like Twitter's "who-follows-whom" graph, fraudsters are paid to make certain accounts seem more legitimate or famous through giving them many additional followers. The spammers deliver these purchases through either generating fake accounts or controlling real accounts through malware and using them to follow their "customers" [1, 2].

In this case, the attack is strictly manipulating the Twitter graph to give certain accounts undue credibility. As described above and will be demonstrated in Section 3.4, content-based approaches analyzing users' tweets and profiles [11, 41, 180] often miss these purchased followers. Rather, we take a strictly graph mining approach, using exclusively the graph structure to find nodes that are suspicious because of their position in the graph.

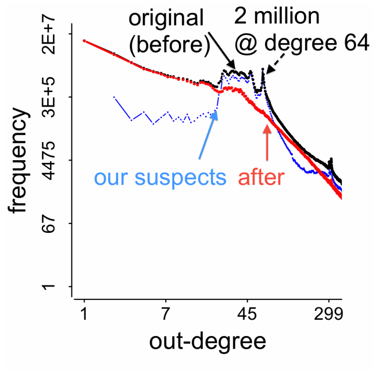
By abstracting the attack to a graph mining problem, we find that it covers a wide variety of suspicious behavior found in the real world. For example, botnets often control hundreds of thousands of machines and use them to perform distributed denial of service (DDoS) attacks on websites, creating a similar pattern in the "who-visits-whom" web traffic graph. Online, on sites like Amazon or Yelp, spammers will create accounts to skew ratings for certain products or places, manipulating edges in the "who-rates-what" graph.



(a) Synchronized behavior



(b) TWITTERSG



(c) WEIBOSG

Figure 3.1: **Suspicious followers and their footprints**: (a) We spot synchronized behavior that millions of Twitter accounts follow the same group of followees. (b) Synchronized behavior causes spikes at out-degree distribution and the distribution becomes smoother after the removal of our suspects. (c) We have the same result on Tencent Weibo.

In this chapter, we focus on the Twitter attack, looking for groups of accounts used to unfairly bolster the popularity of their customers. Figure 3.1(a) illustrates the scenario: it shows a set of suspicious followers and their followees. The followers, all 3 million of them, follow exactly 20 users from the same group of followees, creating a strange, rare connectivity structure. The side information, like the similarity of the login-names (@Buy_AB22, @Buy_BT27, @Buy_BT68), is an extra reason to suspect that they were created by a script.

Our main viewpoint: In more detail, suspicious nodes including suspicious followers and botnets exhibit behavior that is (a) *synchronized* (cause to occur at the same rate): they often connect to the very same 10, 100 or 500 targets and (b) *abnormal/rare*: their behavior pattern is very different from the majority of nodes. In this chapter, we offer a fast and effective method, CatchSync, to measure the two properties (*synchronicity* and the *normality*) of a group of nodes; we spot the suspicious nodes and efficiently catch

		Synchronized behaviors?	Parameter free?	No side information?
Proposed CatchSync		√	√	√
Graph-based Anomaly	AUTOPART [43]	×, edges across groups	√	√
	OUTRANK [160]	×, high scoring nodes	√	√
	ODDBALL [17]	×, cliques or stars	√	√
	NETPROBE [171]	×, fraudulent nodes	×, prop. matrix	×, committed fraud
Subgraph Mining	SPOKEN [183]	×, dense communities	×	√
	DSE [49]	×, d_G-dense subgraphs	×, density d_G	√
Spammer Detection	SPOT [180]	×, twitter spammers	√	×, tweet text & URLs
	SYBILRANK [41]	×, social sybils	√	×, early non-Sybils

Table 3.1: **Compare CatchSync with existing approaches**. It does not require any parameter or side information.

them in the synchronicity-normality plot. We study two real social graphs from Twitter and Tencent Weibo (denoted by TWITTERSG and WEIBOSG for abbreviation) and use them for evaluations. Note that both have millions of nodes and billions of edges.

Figure 3.1 gives an elaborate illustration on the effectiveness of CatchSync. As we mentioned earlier, the distributions of the social network data have been seriously distorted by the volume of suspicious followers. Here we plot the out-degree distribution of TWITTERSG and WEIBOSG in log-log scale, which should have smooth, power-law-like distributions. However, several spikes appear, which are presumably caused by suspicious followers [38]. For example, as shown in Figure 3.1(a), the 3 million followers on Twitter who connect to exactly 20 users create a spike at out-degree 20 on the distribution in Figure 3.1(b). After removing the nodes that CatchSync flags as suspicious (blue points), the distributions become much smoother and closer to a power law (red points).

Main contributions: In short, our method CatchSync has the following desirable properties:

- **Effectiveness:** It indeed spots groups of source-target groups, with suspicious behavior (see Section 3.4).
- **Scalability:** It is linear in the number of edges, and thus applicable to internet-scale graphs.
- **Parameter free:** The operator does not need to specify any parameters such as the density, the number of groups and the scale of each group.
- **Side information oblivious:** It needs no side information. It is solely based on topology, and it requires neither a training set, nor labeled nodes, nor node attributes.

Relationship to Related Work In Section II.1, we have given a general overview of the related work. As shown in Table 3.1, most of the previously proposed methods are orthogonal to our approach. Unlike most of the other graph-based anomaly detection methods, we focus here on labeling suspicious nodes based on finding suspiciously synchronized and strange behavior in static graphs. Methodologically, our algorithm follows a similar intuition to that of two-sample tests; [86] gives a good overview of such algorithms. However, as two-sample tests are more loosely defined than the synchronized and abnormal patterns detected by CatchSync, it is unclear if using a two-sample testing algorithm would give equivalent results or find non-fraudulent anomalies; this is an interesting direction for future exploration. We compare CatchSync to previous spam detection methods for finding fraud on Twitter, we do not make use of any content-based data, such as profile information or tweets [11, 103, 180]. In Section 3.4 we combine our results with the content-based method of [180], and demonstrate that these methods are complimentary, achieving better results by combining the methods.

3.2 Synchronized Behavior Detection

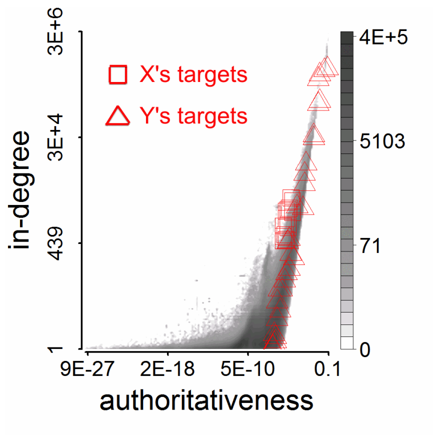
In this section, we first propose the problem of synchronized behavior detection and then present a fast and effective solution.

3.2.1 Problem Definition

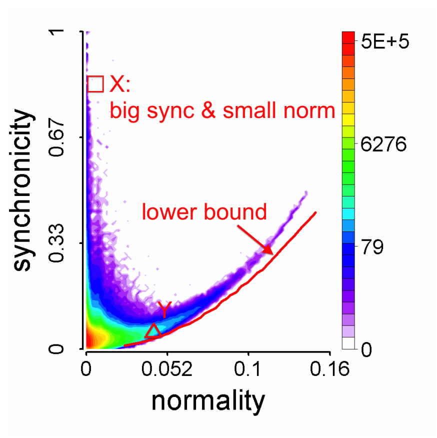
Our goal is to find suspicious nodes on a directed graph and thus the problem is defined as:
Given: a directed graph of N nodes in the node set $\mathcal{U}$
Find: a set of suspicious source nodes (fake followers, botnets, etc.) $\mathcal{U}_{sync}$, and a set of suspicious target nodes (followees, target hosts, etc.) $\mathcal{V}_{sync}$ that the source nodes have *synchronized* and *abnormal* behaviors connecting to the target nodes. The word "synchronized" means that the source nodes have very similar behavior pattern, and "abnormal" means that their behavior pattern is very different from the majority of nodes. Table 3.2 gives a list of the symbols we use throughout the chapter.

3.2.2 Proposed Approach

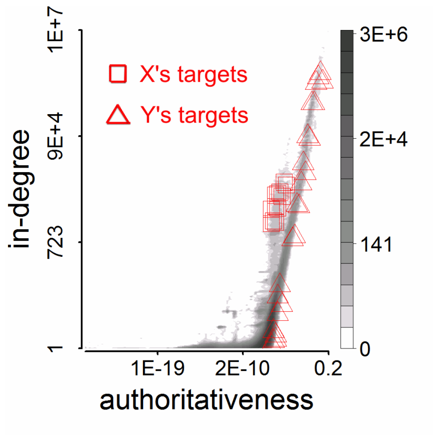
In this section, we introduce our approach towards the above problem. First, we give a feature space for target nodes. Second, we show the definitions of synchronicity and normality that measure the nodes' behavior patterns. Then we provide a general theorem of the normal shape of the synchronicity-normality plot. Next, we detect the outliers on the plot, which are suspicious nodes with synchronize behavior on the graph.



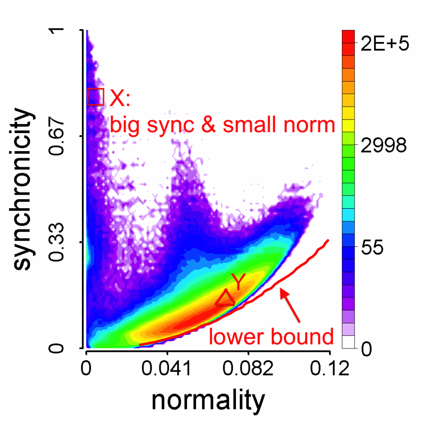
(a) InF-plot on TWITTERSG



(b) SN-plot on TWITTERSG



(c) InF-plot on WEIBOSG



(d) SN-plot on WEIBOSG

Figure 3.2: **Synchronicity-normality plot**: the source nodes X have synchronized and abnormal behaviors that their targets are coherent in the InF-plots (a) and (c), while Y's targets are not. X has big synchronicity and small normality in the SN-plots (b) and (d) while Y is near the parabolic, theoretical lower limit.

Symbol	Definition and Description
$\mathcal{U}$	The set of nodes
N=$\|\mathcal{U}\|$	The number of node
$\mathcal{I}(u)$	The set of node u's sources
$\mathcal{O}(u)$	The set of node u's targets
$d_i(u)$=$\|\mathcal{I}(u)\|$	In-degree of node u (number of sources)
$d_o(u)$=$\|\mathcal{O}(u)\|$	Out-degree of node u (number of targets)
$hub(u)$, $aut(u)$	"Hubness" and "authoritativeness" of u
$sync(u)$	"Synchronicity" of node u's targets
$norm(u)$	"Normality" of node u's targets
$\mathbf{p}(u)$	k-dimensional feature vector of node u
$c(u,v)$	Closeness of u and v in feature space

Table 3.2: Symbols and Definitions

Feature space

It has been established by past works that many data mining approaches on graphs benefit from exploiting the features from the nodes' behavior patterns, including (a) out-degree and in-degree, (b) HITS score (hubness and authoritativeness), (c) betweenness centrality, (e) node in-weight and out-weight, if the graph is weighted, (f) the score of the node in the i-th left or right singular vector, and many more. We denote k-dimensional feature vector of node u by $\mathbf{p}(u) \in \mathbb{R}^k$. We extract the feature vector from graph structure that somewhat reflects the node's behavior pattern. The features could be *any* from the above and the vector could have *any* dimensionality. In this chapter, we choose the degree values and HITS score. We denote a set of u's source nodes by $\mathcal{I}(u)$ and a set of u's target nodes by $\mathcal{O}(u)$. The in-degree $d_i(u)$ of node u is the number of its sources, i.e. the size of $\mathcal{I}(u)$. The out-degree $d_o(u)$ of node u is the number of its targets, i.e. the size of $\mathcal{O}(u)$. Also we denote by $hub(u)$ the hubness of node u and by $aut(u)$ the authoritativeness of u, according to Kleinberg's famous work [118]. We choose these features for two reasons: they are fast to compute, as well as easy to plot. As our experiments show, they work well in pin-pointing suspicious nodes. Note that if the side information is available, it could be regarded as additional features that would be easily incorporated, and hopefully, the performance could be better.

Here we present some plots of the feature spaces. For a source node u, we plot a heat map of the 2-D feature space of out-degree $d_o(u)$ vs hubness $hub(u)$ in log-log scale, called "OutF-plot". Similarly, for a target node u, the heat map of the feature space of in-degree $d_i(u)$ vs authoritativeness $aut(u)$ in log-log scale is called "InF-plot". Table 3.3 summarizes the description of all the plots.

Specifically, Figure 3.2(a) and 3.2(c) are InF-plots of TWITTERSG and WEIBOSG.

Plot	Description
OutF-plot	A heat map of source nodes in feature space: typically, out-degree vs hubness
InF-plot	A heat map of target nodes in feature space: typically, in-degree vs authoritativeness
SN-plot	A heat map of source nodes in synchronicity vs normality of their targets

Table 3.3: Plots and Descriptions

On TWITTERSG, we denote by X one of the suspicious followers we mentioned in Figure 3.1(a) and by Y an ordinary user whose out-degree is the same as X's. We tag their targets (followees) in the Figure 3.2(a) and find out that X's targets are coherent in the InF-plot, while Y's targets are not. In other words, X's target nodes have similar in-degree and authoritativeness, but Y's targets are diverse in the feature space, ranging from top popular to ordinary users just like Y. Similar thing happens on WEIBOSG. Figure 3.2(c) shows that X's targets are located at a micro-cluster which is away from the majority of followee nodes. They have large in-degree values from 1,000 to 100,000 but they are not as authoritative as the ones who are followed by Y and of the same in-degree.

Synchronicity and normality

We propose two novel concepts to investigate the behavior patterns of the source nodes: (a) "synchronicity" $sync(u)$ to qualify how synchronized the node u's targets are in the feature space (in-degree vs authoritativeness); and (b) "normality" $norm(u)$ to qualify how normal u's targets are relative to the rest of the data. These two measures consider the relative position of u's target nodes in the feature space. We denote by $c(v,v')$ the closeness (similarity) between two target nodes v and v' in the feature space (InF-plot). For fast computing the closeness of each pair of nodes, we divide the feature space into G grid cells and map each node to a specific grid cell. If two nodes are in the same grid cell, they have similar feature vectors, and they are close in the feature space. Thus, we have

$$c(v,v^{'}) = \begin{cases} 1 & \text{if nodes } v \text{ and } v^{'} \text{ are in the same grid cell} \\ 0 & \text{otherwise} \end{cases}$$

Then we have the definition of synchronicity and normality.

Definition 1 Synchronicity and Normality

We define synchronicity of node u as the average closeness between each pair of u's targets $(v,v^{'})$:

$$sync(u) \quad = \quad \frac{\sum_{(v,v^{'})\in\mathcal{O}(u)\times\mathcal{O}(u)} c(v,v^{'})}{d_o(u)\times d_o(u)} \tag{3.1}$$

We define normality of node u as the average closeness between each pair of u's targets and other nodes (v,v'):

$$norm(u) \quad = \quad \frac{\sum_{(v,v')\in\mathcal{O}(u)\times\mathcal{U}} c(v,v')}{d_o(u)\times N} \tag{3.2}$$

Both values of synchronicity and normality range from 0 to 1. We know that a suspicious source node u has uncommonly large $sync(u)$ and abnormally small $norm(u)$. For a source node u, we name u's target nodes in the InF-plot as *foreground* points and name all the nodes in the plot as *background* points. We provide a theorem of the normal shape of SN-plot, which could be the basis for catching suspicious nodes

Theorem 1 *For any foreground/background distribution, there is a parabolic lower limit in the synchronicity-normality plot.*

Proof 1 *The complete proof can be found in Appendix A. It is based on Lagrange multipliers. The parabolic low limit is*

$$s_{min} = (-Mn^2 + 2n - s_b)/(1 - Ms_b)$$

where M is the total number of grid cells, s_{min} is the minimum value of synchronicity of foreground points, s_b is the synchronicity of background points, and n is a given normality value.

Figure 3.2(b) and 3.2(d) show the SN-plots of source nodes in TWITTERSG and WEIBOSG. Note that the source node *X* has synchronized and abnormal behavior and *Y* does not, as shown in Figure 3.2(a) and 3.2(c). *X* has much bigger synchronicity and smaller normality than *Y*. The red parabola is the theoretical lower limit of synchronicity with a given normality, which has been given in the proof. *Y* is close to the parabola, while *X* is far away from the lower bound. The next step to find the suspicious nodes like *X* is to detect the outliers in the SN-plots.

Outliers in SN-plot

Here we introduce how to catch the outliers in the SN-plot based on Theorem 1. Informally, we want the nodes that are too far away from the lower limit. Formally, we denote by $r_{source}(u)$ the *residual score* of a source node u's synchronicity which indicates how suspicious it is. The set of suspicious source nodes $\mathcal{U}_{sync}$ includes the nodes whose suspiciousness is $\alpha = 3.0$ standard deviations away from the mean:

$$\mathcal{U}_{sync} \leftarrow \{u : r_{source}(u) > \mu[r_{source}] + \alpha \times \sigma[r_{source}]\} \tag{3.3}$$

Similarly, we denote by $r_{target}(v)$ the suspiciousness of a target node v, which is the proportion of v's sources that are reported in $\mathcal{U}_{sync}$. Then we could have the set of suspicious targets $\mathcal{V}_{sync}$:

$$\mathcal{V}_{sync} \leftarrow \{v : r_{target}(v) > \mu[r_{target}] + \alpha \times \sigma[r_{target}]\} \tag{3.4}$$

The default value of α is chosen according to Tax's classical outlier detection work in [208]. In the experimental section, we will validate that the performance of our method does not depend much on α.

3.3 CatchSync Algorithm

In this section, we present the implementation of CatchSync and analyze the complexity.

Implementation The approach is outlined below in Algorithm 1. We first derive a feature space for target nodes. We then compute synchronicity and normality of the source nodes' behaviors, according to the relative positions of their target nodes in the feature space. Finally, we use a distance-based outlier detection method to detect the outliers in the synchronicity-normality plot.

Algorithm 1: CatchSync: Catch suspicious nodes with synchronized behaviors in a large, directed graph.

```
Input: A directed graph of $N$ nodes in the set $\mathcal{U}$.
Output: A set of source nodes $\mathcal{U}_{sync}$ who have synchronized and abnormal
        behaviors and a set of targets in $\mathcal{V}_{sync}$ .
Step 1: plot a (2-D) feature space of target nodes.
foreach node $v$ as a target do
    Compute in-degree $d_i(v)$ and authoritativeness $aut(v)$.
Give InF-plot $d_i(v)$ vs $aut(v)$ (see Figure 3.2(a) and 3.2(c)).
Step 2: plot synchronicity-normality of source nodes.
Divide the InF-plot into grids.
foreach node $u$ as a source do
    Compute synchronicity $sync(u)$ and normality $norm(u)$ with Eq. (3.1) and (3.2).
Give SN-plot $sync(u)$ vs $norm(u)$ (see Figure 3.2(b) and 3.2(d)).
Step 3:
Adapt a distance-based method to report suspicious sources $\mathcal{U}_{sync}$ and targets $\mathcal{V}_{sync}$.
```

	Nodes	Edges	Description	Labeled suspicious nodes
TWITTERSG	41,652,230	1,468,365,182	Twitter graph in 7/2009 [130]	173 / 1,000
WEIBOSG	117,288,075	3,134,074,580	Weibo graph in 1/2011	237 / 1,000

Table 3.4: **The real world graphs we study** are complete social graphs with multimillion nodes and billion edges. We have used human labor to label a small piece of both of them for ground truth.

In detail, we choose 2-dimensional feature spaces and specifically out-degree vs hubness, for each source node, and in-degree vs authoritativeness, for each target node. The out-degree (in-degree) is the size of set of a source's targets (a target's sources). The hubness (authoritativeness) is the first left- (right-) singular vector of the graph's adjacency matrix. The algorithm to compute these values is omitted for saving space. When we divide the InF-plot into grids in Step 2, the length of each side of a grid is $log2$, that is, the grid lines at degree and HITS score are on powers of 2.

Complexity analysis We now examine CatchSync's complexity. We first compute degree and HITS score of each node. This process is linear in the number of edges E. Second, the process of computing synchronicity and normality is linear in the number of nodes N. We denote by G the number of grids that we divided the InF-plot into. Thus, the total time complexity is $\mathcal{O}(E+NG)$. CatchSync is a scalable algorithm and able to process huge, directed graphs.

3.4 Experiments

In this section we present an empirical evaluation of CatchSync, demonstrating its effectiveness in spotting suspicious behavior. Although much of the research on anomaly detection frames the problem as a labelling task, in real world anomaly detection is a combination of machine learning, manual verification, and discovering new types of attacks as they arise. Here, we provide evidence that CatchSync is effective at both the classic problem of labelling suspicious behavior, as well as surfacing new patterns of unusual group behavior:

- **Detection effectiveness:** We demonstrate CatchSync's ability to accurately label suspicious behavior and remove anomalies through three techniques.
    1. *Injected attacks*: We begin by testing the accuracy, precision, and recall on synthetic graphs with injected group attacks. We compare our algorithm against state-of-the-art methods and show that CatchSync performs the best.
    2. *Labelling task*: We also test our accuracy, precision, and recall on two real datasets, where we use the labeled data from random sampling of TWITTERSG and WEIBOSG as ground truth of suspicious and normal nodes.
    3. *Restore normal patterns*: For all of these cases we show that removing the suspicious nodes restores the power law properties of the graph's edge degree, which when distorted is a common sign of spam, and remove anomalous patterns in the feature spaces (OutF-plots and InF-plots).
- **CatchSync properties:** We test a number of properties of CatchSync, including the robustness with respect to α, the speed and the scalability.
- **Discovery:** We demonstrate the effectiveness of CatchSync as a discovery tool. We discuss a number of the unusual accounts caught and patterns detected in the TWITTERSG and WEIBOSG datasets.

3.4.1 Evaluation: Data and Ground Truth

We carry out experiments on synthetic and real datasets to evaluate the performance of CatchSync. The synthetic datasets are described in Table 3.5, while the real datasets are in Table 3.4.

	Number of Nodes	Number of Injected sources	Camouflage
SYNTH-1M	1,034,100	31,000	-
SYNTH-2M	2,034,100	=16K	-
SYNTH-3M	3,034,100	+8K+4K	-
SYNTH-3M-RAND	3,034,100	+2K+1K	+10% Random
SYNTH-3M-POP	3,034,100		+50% Popular

Table 3.5: **Synthetic data**: we inject 5 different sizes of group attacks on random power law graphs of 1-3M nodes.

Synthetic data

Description We generate random power-law graphs, following the Chung-Lu model [1] [53], and with a power-law exponent -1.5 since most real-world networks have been shown to have this value [72]. Next we inject groups of source and target nodes.

To demonstrate the effectiveness, we vary the following properties of the synthetic graphs:

- **Size of graph:** The random power-law graphs we generate contain approximately 1M, 2M, or 3M nodes, named as SYNTH-1M, SYNTH-2M and SYNTH-3M.
- **Size of injection:** We inject 5 source and target nodes groups of different sizes. The smallest group has 1,000 new sources, connecting to *20* of 100 new targets, because of the real case that the smallest number of fake followers a user can buy is often 1,000 [1]. The size of the injected group doubles one by one and thus the largest group has 16,000 sources and 1,600 targets. The total number of injected sources is 31,000.
- **Camouflage:** The injected source nodes may try to use "camouflage" to evade the detection, for example, the fake accounts can follow Barack Obama, Taylor Swift, or some random users, though they connect to tens or hundreds of customers. Inspired by this, we try two different techniques on SYNTH-3M: for each injected node, let it connect to (a) some "random", ordinary targets; (b) some from the top 100 "popular" targets. We also vary the weights of camouflage d_{camou}: (a) d_{camou}

[1]Following the model, we assign out-degrees $d_o(u)$ and in-degrees $d_i(v)$ to each node u and v respectively; we then create edge (u, v) with probability proportional to $d_o(v)d_i(u)$.

= 10%, 18 injected targets and 2 for camouflage; (b) d_{camou} = 50%, 10 injected ones and 10 for camouflage. Specifically, we denote by SYNTH-3M-RAND the injected graph with 10% random camouflage, and by SYNTH-3M-POP the graph with 50% popular camouflage.

With different settings of the above, we have the 5 synthetic datasets.

Evaluation If we denote the injected nodes by positive samples, and the others by negative samples, we can record the true positive (TP), true negative (TN), false positive (FP), and false negative (FN) rates, which we use the standard definition [73] to calculate the three popular metrics: accuracy, precision and recall. High accuracy, precision and recall will be a better method.

Real data

Description We also use our two real world datasets, TWITTERSG and WEIBOSG, both of which are complete graphs of popular online social networks with billions of edges. Thanks to the public download links [2], CatchSync is reproducible. As the webpage says, due to Twitter's new Terms of Services, academic researchers cannot access the side information like the tweet data. Fortunately, we can usually get the who-follows-whom data, or directed graphs from different applications. Then we can operate our side-information oblivious method CatchSync.

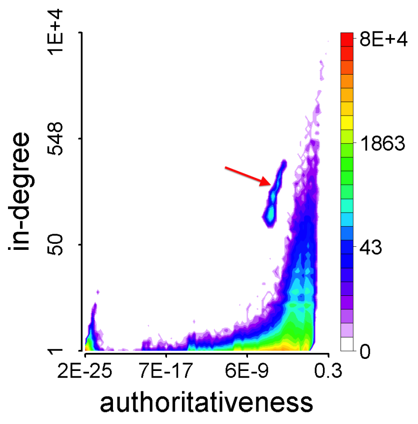
Synthetic graph	SYNTH-1M	SYNTH-2M	SYNTH-3M	SYNTH-3M-RAND		SYNTH-3M-POP	
Camouflage (d_{camou})	None (0)	None (0)	None (0)	10%	50%	10%	50%
CatchSync	**0.998**	**0.987**	**0.956**	**0.910**	**0.764**	**0.885**	**0.792**
ODDBALL	0.827	0.796	0.755	0.702	0.525	0.657	0.433
OUTRANK	0.805	0.777	0.725	0.678	0.516	0.694	0.392
SPOKEN	0.695	0.682	0.677	0.586	0.470	0.553	0.351

Table 3.6: **CatchSync consistently wins, despite of "camouflage"**: it reaches higher accuracy on detecting injected nodes.

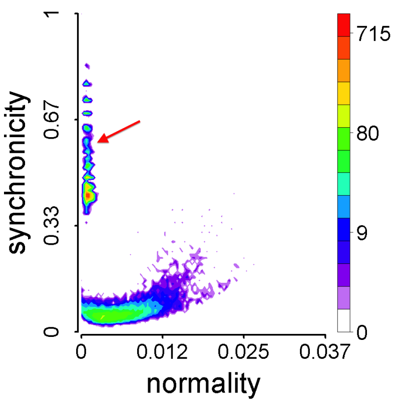
WEIBOSG was crawled in January 2011 from Tencent Weibo, one of the biggest microblogging services in China. For each dataset CatchSync only uses the graph structure, but we also have user id and name associated with the nodes, so that we can provide real links to check the users' profile information.

Evaluation For WEIBOSG and TWITTERSG, we sample 1,000 nodes and conduct user study to label them as suspicious or normal accounts. Half of the nodes are randomly selected from the set U_{sync} and half are not. Although the average suspiciousness of

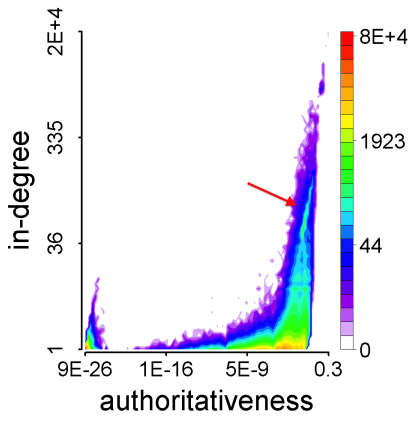
[2]http://an.kaist.ac.kr/traces/WWW2010.html



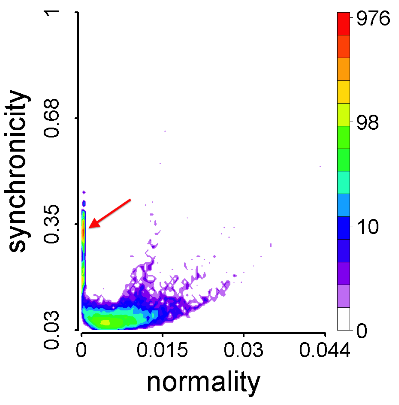
(a) SYNTH-3M-RAND



(b) SYNTH-3M-RAND



(c) SYNTH-3M-POP



(d) SYNTH-3M-POP

Figure 3.3: **Our CatchSync catches injections, despite of "camouflage"**: camouflage can hide the injected nodes in or put them close to dominating parts in (a) and (c), but SN-plots can catch them with big synchronicity and small normality in (b) and (d).

samples is higher than that of the entire dataset, it is fair for all the algorithms in our experiments. The 5 volunteers are all 20 to 25-year-old college students who have been social network users for at least 3 years. They are provided URL links directed to the 1000 users' Twitter or Tencent Weibo pages, and read their tweets and profile information. A user is labeled as a suspicious one if the volunteer finds he or she matches too many of the following clues:

- **Disabled account:** It has been disabled by the services. For example, Weibo user @marra_xiao_bai had 9 followers and 36 followees in 2011. Twitter user @wYWvk0310 had 666 followers and 926 followees in 2010. But both of them have been disabled now.
- **Suspicious user name:** They have strange self-declared information that follows a narrow pattern such as Twitter names in the form of "@Buy_XX##" (@Buy_AB22, @Buy_BT47), Weibo names in the form of "a#####" (@a58444, @a70054).

- **Many followees but few or zero tweets:** It has hundreds of followees but it never posts a single tweet. Twitter user @P8igBg801 had 923 followees in 2010 and @AjaurNYj2 had 869 followees, but both of them post nothing.
- **Malicious tweet content:** The account posts duplicated tweets or malicious links for monetary purposes. For example, Twitter user @Buy_BT66 posts only 3 messages but all of them are about "bed flat for sale". Weibo account @aa52011 posts hundreds of similar messages about online games.

Finally, we give a user a "suspicious" label if 3 (more than a half) of the volunteers think it is suspicious. Our task here is to detect the users with the "suspicious" labels. Similarly with the evaluation method on synthetic data, we also use accuracy, precision and recall to evaluate the effectiveness. A good detection algorithm will have high values of accuracy, precision and recall.

3.4.2 Competing Algorithms

We carefully implement the following state-of-the-art methods as competing algorithms: (a) ODDBALL [17], looking for near-cliques and stars that are suspected as strange nodes in the graph; (b) OUTRANK [160], using random walk model across the similarity measure to give the outlierness of each node; (c) SPOKEN [183], using pairs of eigenvectors to find well-connected communities. When operating on the labeled real data, we implement a content-based spammer detection method SPOT [180], which learns the words and the number of malicious links in the accounts' tweets.

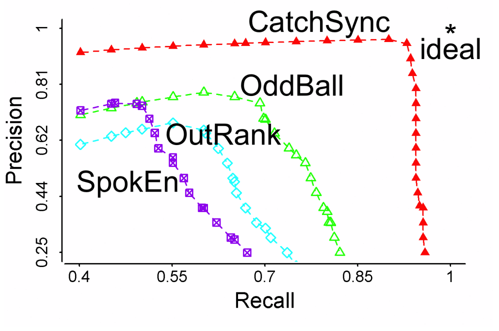
As mentioned before, our CatchSync is orthogonal to the text-based methods like SPOT. Thus, we develop a hybrid method, CATCHSYNC+SPOT, that suspects the nodes detected by *either* CatchSync *or* SPOT. It learns from both the graph structure and text-based features from tweets.

All the algorithms are implemented with JAVA, and all experiments are performed on a single machine with Intel Xeon CPU at 2.40GHz and 32GB RAM.

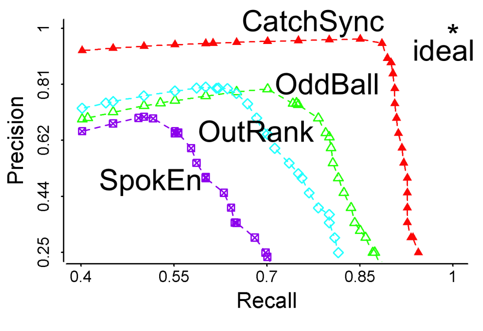
3.4.3 Detection Effectiveness on Synthetic Data

Injected group attacks shown in feature space and SN-plot We plot the feature space (InF-plots) and SN-plots of two synthetic graphs with camouflage SYNTH-3M-RAND and SYNTH-3M-POP in Figure 3.3. When the weight of camouflage is small (d_{camou}=10% in SYNTH-3M-RAND), the InF-plot in Figure 3.3(a) shows the injected node groups as outliers from the majority. With the SN-plot in Figure 3.3(c), CatchSync can easily catch them since they fall along the synchronicity axis. When d_{camou} is as big as 50%, Figure 3.3(c) shows that the camouflage can hide the injected nodes in the dominating part. Our SN-plot in Figure 3.3(c) can catch them for their big synchronicity and small normality values.

Accuracy, precision and recall on injected node detection Table 3.6 shows the accuracy on detecting the injected nodes from all the 5 synthetic datasets. When there is



(a) SYNTH-3M-RAND

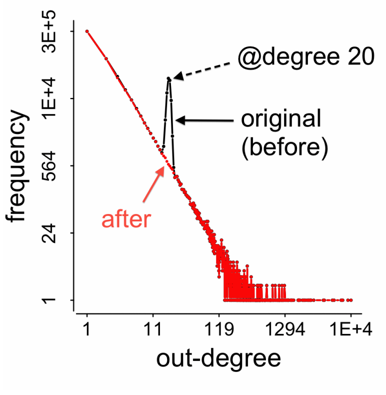


(b) SYNTH-3M-POP

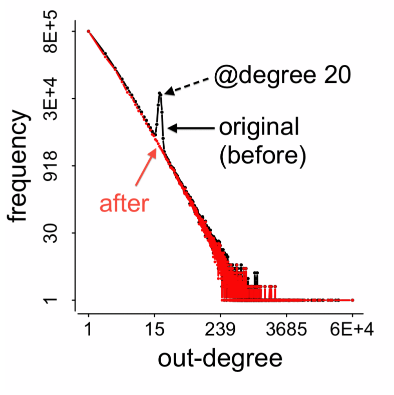
Figure 3.4: CatchSync achieves higher precision and recall.

no "camouflage", CatchSync reaches greater than 95% accuracy. When the nodes have camouflage, it can still outperforms the best of the other methods by 29.6% accuracy on SYNTH-3M-RAND and 27.5% accuracy on SYNTH-3M-POP. Figure 3.4 plots the precision-recall curves to test the performance of ranking the suspiciousness of nodes. Our method CatchSync (the red filled triangle) can achieve both higher precision and higher recall.

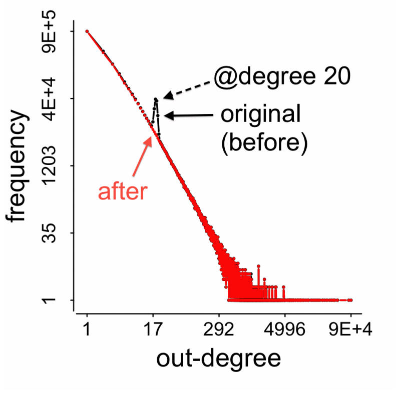
Restoring the power law The injected source nodes connect to 20 from the same set of targets. Due to this anomalous behavior pattern, the out-degree distribution has a spike at degree 20. Figure 3.5 plots the out-degree distributions before and after we operate CatchSync on the synthetic graphs of different sizes. The spike on the distribution shrinks with the size of the graph increasing. No matter how big the spike is, our method can detect the injected nodes and we see if we remove them and their out-going edges, then the power-law degree distribution is restored.



(a) SYNTH-1M



(b) SYNTH-2M



(c) SYNTH-3M

Figure 3.5: **CatchSync restores the power law**: the degree distribution is recovered after the removal of suspicious nodes.

3.4.4 Detection Effectiveness on Real Data

Accuracy, precision and recall on real data Table 3.7 shows the accuracy on detecting the labeled suspicious nodes from the two real social graphs. Also in Figure 3.6, we plot the precision-recall curves of CatchSync, OUTRANK, SPOT and the hybrid algorithm CATCHSYNC+SPOT. We examine the results and give the following observations and explanations.

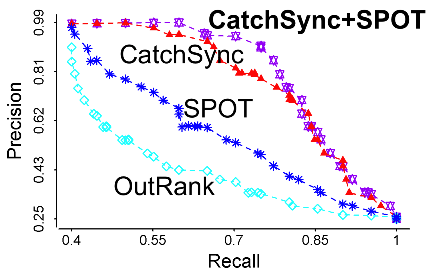
- **CatchSync outperforms OUTRANK.** CatchSync learns graph-based features of synchronized behavior that OUTRANK cannot capture using a random walk model with a data dedicated threshold.
- **CatchSync outperforms SPOT.** CatchSync learns graph-based features from the structural information, and SPOT learns text-based features from users' tweets. Since the main characteristics of the suspicious users are group attacks, CatchSync has high accuracy, precision and recall than SPOT.

Actually, CatchSync is *complementary* to SPOT: combining the flagged nodes, we get even better performance (purple line on Figure 3.6). The hybrid algorithm uses both of them to catch the different types of attackers. CATCHSYNC+SPOT consistently outperforms the competitors in detection accuracy by 36% on TWITTERSG and 20% on WEIBOSG. We suggest the social network applications to operate our CatchSync on their who-follows-whom graphs, while they have used methods like SPOT that learns text-based features from their tweets and profiles.

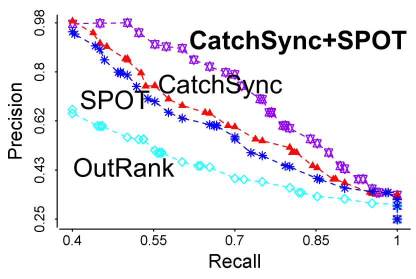
	TWITTERSG	WEIBOSG
CatchSync	0.751	0.694
OUTRANK	0.412	0.377
SPOT	0.597	0.653
CATCHSYNC+SPOT	**0.813**	**0.785**

Table 3.7: **CATCHSYNC+SPOT outperforms each part**: CatchSync is better than OUTRANK at learning the structure, while SPOT learns the text; the combination wins the last.

Restoring the power law While in the synthetic datasets the recovery of the power law followed directly from our high recall, this is not necessarily the case on real world data sets because we can only measure our accuracy on the subset of nodes we label. Looking at the out-degree distribution of TWITTERSG in Figure 3.1(b) and WEIBOSG in Figure 3.1(c), we see that removing the millions of caught suspicious nodes from the graph does leave only a smooth power law distribution on the remaining part of the graph. Because a power law distribution has been found to be typical of social networks and because the original distribution is not directly used in CatchSync, this is strong evidence that our recall on the full datasets is high and that CatchSync is effective.



(a) TWITTERSG



(b) WEIBOSG

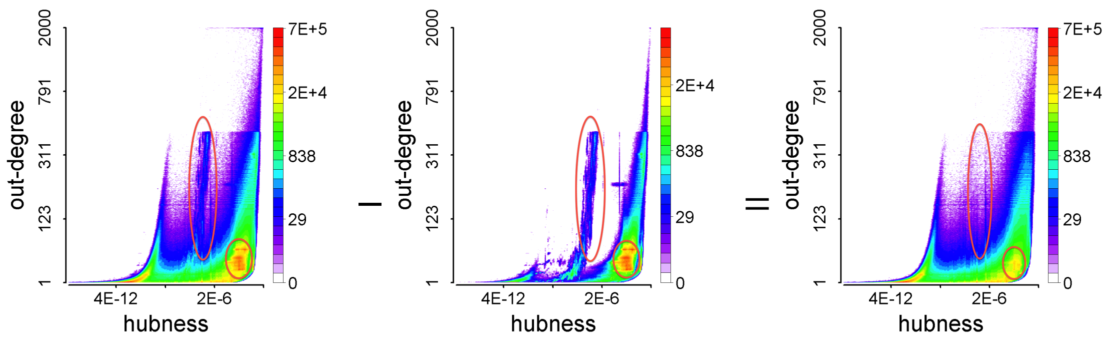
Figure 3.6: **CATCHSYNC+SPOT is the best at ranking the suspiciousness**: it reaches the highest precision and recall.

Observations in the feature space We provide interesting observations from the change of feature space before and after we operate CatchSync on WEIBOSG. Figure 3.7(a), 3.7(b) and 3.7(c) are OutF-plots arranged as an equation: all nodes *minus* suspicious nodes with synchronized behaviors *equals* normal nodes. Figure 3.7(b) shows the suspicious source nodes look synchronized and abnormal in the OutF-plot: they are coherent in red clusters or on blue stripes that deviate from the majority. The red clusters and blue stripes disappear in Figure 3.7(c) after we remove them from the graph. Figure 3.7(d), 3.7(e) and 3.7(f) show a similar equation of InF-plots. Figure 3.7(e) shows that the suspicious targets are in a purple cluster in Figure 3.7(f) the cluster disappears after we remove them. The above observations provide evidence of the suspiciousness of the nodes who have synchronized behaviors. Our method CatchSync can remove the strange patterns in the feature space.

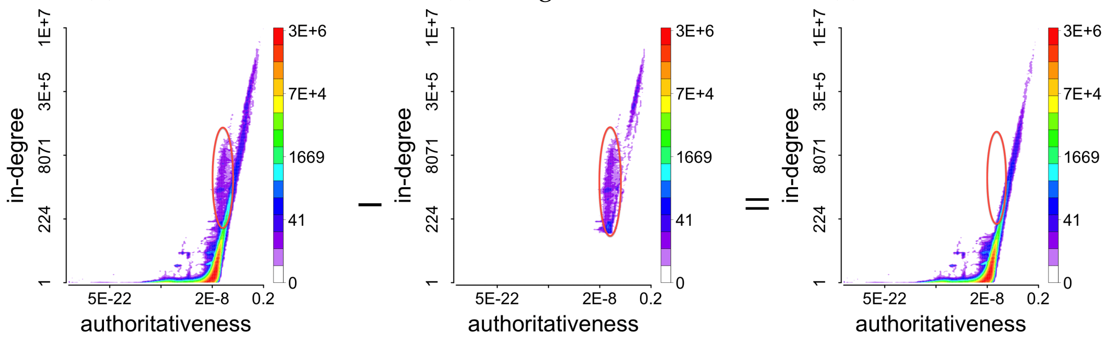
3.4.5 CatchSync Properties

Robustness with respect to α Within the synthetic data, we conduct experiments on the robustness to changes in α, the number of standard deviations from the mean for a node to be labelled as suspicious. In short, α=3.0 gives either the best result, or very close to it, and so do nearby values of α. In more detail, we test the sensitivity of precision and recall with respect to α, on the synthetic graphs of 3 different sizes. Figure 3.8 plots precision-recall curves: the ideal point is, of course, (1.0, 1.0); although α changes from 0.5 to 5.0, both precision and recall are still over 0.8. The performance of our algorithm is rather robust on α. We set $\alpha = 3.0$ as the default value for all of our other experiments. Note that approximately 99.7% of the observations fall within 3 standard deviations of the mean in the normal distribution. The suspicious nodes take the small percentage (0.3%) but still a big number since the graphs often contain millions of nodes, which makes this detection problem rather challenging.

Speed and scalability We measure the run time on synthetic graphs with 1-3 million nodes. Figure 3.9 plots processor time vs graph size, showing that CatchSync (the red



(a) All sources (b) Caught sources (c) Normal sources



(d) All targets (e) Caught targets (f) Normal targets

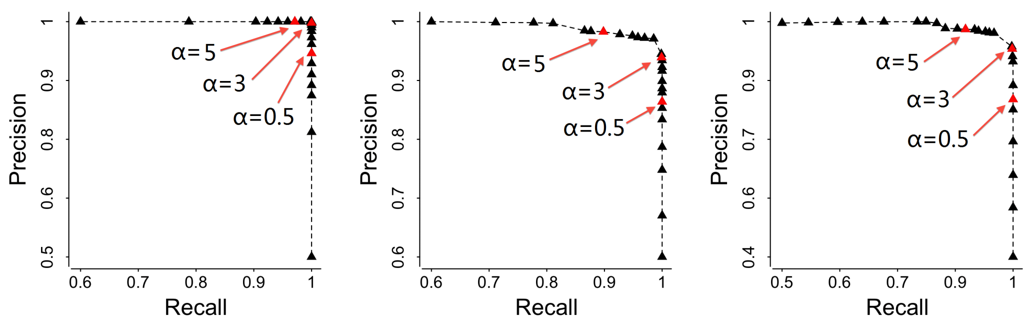
Figure 3.7: **Sources and targets caught by CatchSync are outliers**: (a,b,c) and (d,e,f) form two equations of OutF-/InF-plots. (a) minus (b) equals (c); (d) minus (e) equals (f), where (a,d) show all nodes, (b,e) show suspicious nodes and (c,f) show normal ones.

filled triangles) scales linearly with the graph size and runs faster than alternatives. The measures, synchronicity and normality, could be computed very fast, taking only 10% time of the features (degree and HITS score), while the features can be previously chosen and calculated. Therefore, CatchSync performs fast online for large graphs.

3.4.6 Discovery: A Case Study

As was mentioned earlier, detecting suspicious behavior is not merely a labeling problem. In the real world there are always new types of attacks that arise and distort the service being provided. While we have demonstrated that CatchSync is successful at detecting classic spammy behavior, it also discovers more subtle types of suspicious behavior that a simpler labeling analysis would miss.

Looking online, it is easy to see that fraud on Twitter is much more complex than individual users posting tweets for money. In general users can get paid to tweet and the amount is based on how many followers they have [1, 2]. As a result, this has created marketplaces for buying Twitter followers, which besides providing politicians the appearance of popularity, also raises the value of the "Tweeter." Additionally there are



(a) SYNTH-1M (b) SYNTH-2M (c) SYNTH-3M

Figure 3.8: **Robustness is perfect**: the performance of CatchSync is rather insensitive. We suggest $\alpha = 3.0$ as default.

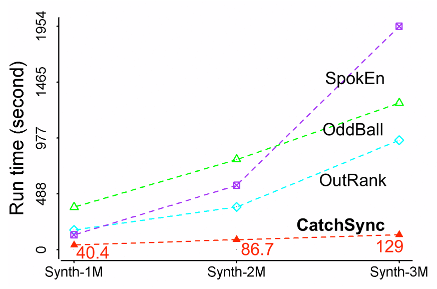
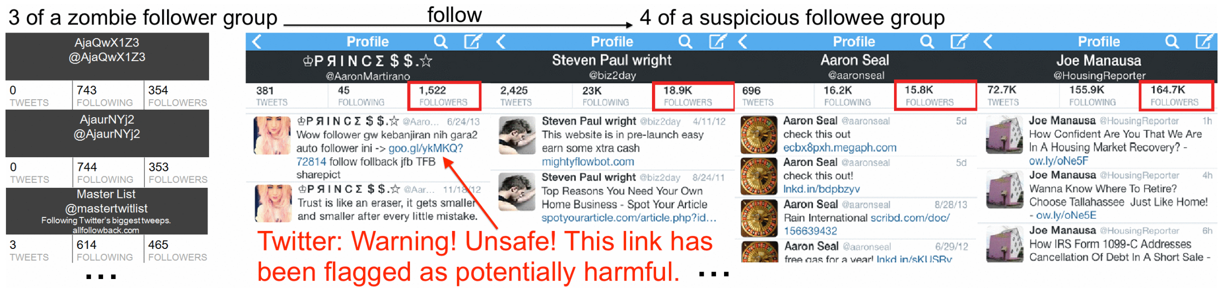


Figure 3.9: **CatchSync is fast and scalable**: run time to detect injected nodes as the graph grows.


3 of a zombie follower group
follow
4 of a suspicious followee group
Twitter: Warning! Unsafe! This link has been flagged as potentially harmful.

AjaQwX1Z3 @AjaQwX1Z3 — 0 TWEETS, 743 FOLLOWING, 354 FOLLOWERS

AjaurNYj2 @AjaurNYj2 — 0 TWEETS, 744 FOLLOWING, 353 FOLLOWERS

Master List @mastertwitlist Following Twitter's biggest tweeps. allfollowback.com — 3 TWEETS, 614 FOLLOWING, 465 FOLLOWERS

...

♔PЯINCΣ$$.☆ @AaronMartirano — 381 TWEETS, 45 FOLLOWING, 1,522 FOLLOWERS

♔PЯINCΣ$$.☆ @Aaro... 6/24/13 Wow follower gw kebanjiran nih gara2 auto follower ini -> goo.gl/ykMKQ?72814 follow follback jfb TFB sharepict

♔PЯINCΣ$$.☆ @Aar... 11/18/12 Trust is like an eraser, it gets smaller and smaller after every little mistake.

Steven Paul wright @biz2day — 2,425 TWEETS, 23K FOLLOWING, 18.9K FOLLOWERS

Steven Paul wright @biz2day 4/11/12 This website is in pre-launch easy earn some xtra cash mightyflowbot.com

Steven Paul wright @biz2day 8/24/11 Top Reasons You Need Your Own Home Business - Spot Your Article spotyourarticle.com/article.php?id...

Aaron Seal @aaronseal — 696 TWEETS, 16.2K FOLLOWING, 15.8K FOLLOWERS

Aaron Seal @aaronseal 5d check this out ecbx8pxh.megaph.com

Aaron Seal @aaronseal 5d check this out! lnkd.in/bdpbzyv

Aaron Seal @aaronseal 8/28/13 Rain International scribd.com/doc/156639432

Aaron Seal @aaronseal 6/29/12 free gas for a year! lnkd.in/sKUSRv

Joe Manausa @HousingReporter — 72.7K TWEETS, 155.9K FOLLOWING, 164.7K FOLLOWERS

Joe Manausa @HousingReporter 1h How Confident Are You That We Are In A Housing Market Recovery? - ow.ly/oNe5F

Joe Manausa @HousingReporter 4h Wanna Know Where To Retire? Choose Tallahassee Just Like Home! - ow.ly/oNe5E

Joe Manausa @HousingReporter 6h How IRS Form 1099-C Addresses Cancellation Of Debt In A Short Sale -

...

Figure 3.10: **CatchSync at work**: using *only* structure information, we illustrate the biggest group that CatchSync flagged (91,035 followers, 667 followees); we show 3 of the former and 4 of the latter. Side information *corroborates* our findings, raising several red flags: (a) the 3 shown followers have ~0 tweets, and near-identical counts of followers and followees, (b) the 4 shown followees mainly tweet URLs, one of which is flagged by Twitter as unsafe.

marketplaces, e.g., buytwitteraccounts.org [2] and socialsellouts.com [3] to buy and sell Twitter accounts, again with the number of followers being the primary value. Because the market is complex, labeling accounts can be difficult with only a subtle red flags raising eyebrows. Here we examine closer some of the Twitter accounts we caught and we use side information to explain the range of suspicious behavior detected by CatchSync.

Figure 3.10 shows a tiny subset (3 followers and 4 followees), from a large, suspicious group of 91K followers and about 700 followees), that was caught by CatchSync. We see 3 accounts on the left that follow the 4 accounts on the right (and many others). Overall, each account, on its own, raises a few small suspicions, but our point is that, collectively, these accounts raise many more suspicions. Below we break down the types of accounts we find:

Dedicated Followers Looking in Figure 3.10, we see on the left three followers: @AjaQwX1Z3, @AjaurNYj2 and @mastertwitlist. All three accounts have a slightly unusual name, few or no tweets, follow approximately 700 other accounts, and are surprisingly followed by approximately 400 accounts. Alone, each account may look slightly unusual but none of this evidence looks truly incriminating. As a group, however, the accounts are clearly suspicious because, along with them all having the same red flags, they all follow the same group of slightly unusual people.

Surprising Followees On the right side of Figure 3.10 we see four of the accounts being followed. Within the group of followees, we find a few common patterns of obviously spam accounts, "SEO experts" tweeting suspicious content, and small business owners with an unusual number of followers. In the first case, @AaronMartirano is slightly suspicious with a most recent gibberish tweet (with words "auto follower", "follback") linking to an empty Blogger that had been labeled "*Unsafe*" by Twitter. Slightly different to the right we see @aaronseal, whose profile offers the GPS coordinates of a Bell Credit Union in Kansas and tweets to free Wordpress themes or asks users to "Like our Facebook Page" for a restaurant. Similarly we observe @biz2day, a self described "webmaster in the advertising and SEO business," who does not tweet "Unsafe" content, but links to other suspicious content like get-rich-quick schemes. For both @aaronseal and @biz2day, the consistent odd linking raises a red flag, and paired with the synchronized followers suggests that these are possibly purchased tweets and followers were bought to inflate the price. Last we see @HousingReporter a real estate agent with 164,700 followers—more than Massachusetts Senator Elizabeth Warren. A small red flag, but of course it is possible for a small business owner to want to appear more popular and thus buy followers.

In all of these cases it is of course impossible to know for sure how their followers were obtained or why they tweet the way they do. However, given that CatchSync found these very different accounts based *only* on the graph structure, all of these other contextual red flags provide strong additional evidence that the followees caught are in fact very suspicious and that CatchSync is effective at catching even subtle or hidden suspicious behavior.

3.5 Summary

We propose a novel method called CatchSync that exploits two signs of artificial and non-organic behavior, synchronicity and normality, to automatically report and catch suspicious nodes on large directed graphs. CatchSync has desirable properties:

- **Effectiveness:** It spots synchronized behavior and indeed catches suspicious source-target groups.
- **Scalability:** its complexity is linear in the number of edges.
- **Parameter free:** The operator can easily implement the algorithm without specifying any parameters such as the density, the number and scale of groups.
- **Side information oblivious:** It is solely based on graph structure, and it requires neither labeled nodes nor profile information.

Experimental results using both real and synthetic datasets demonstrated that CatchSync can catch the suspicious behavior patterns that previous approaches cannot capture.

Chapter 4

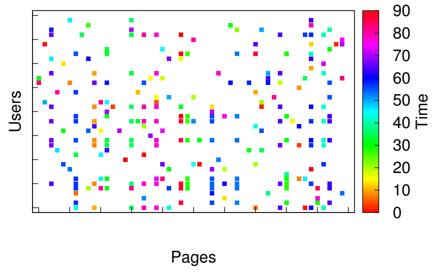
Detect Fraud in Graphs with Time

What temporal patterns do fraudsters leave? How can we use edge creation time to better detect and stop fraudsters? In this chapter we focus on the social network Facebook and the problem of discerning ill-gotten Page Likes, made by spammers hoping to turn a profit, from legitimate Page Likes. Our method, CopyCatch, detects lockstep Page Like patterns on Facebook by analyzing only the social graph between users and Pages and the times at which the edges in the graph (the Likes) were created. We offer the following contributions: (1) We give a novel problem formulation, with a simple concrete definition of suspicious behavior in terms of graph structure and edge constraints. (2) We offer two algorithms to find such suspicious lockstep behavior - one provably-convergent iterative algorithm and one approximate, scalable MapReduce implementation. (3) We show that our method severely limits "greedy attacks" and analyze the bounds from the application of the Zarankiewicz problem to our setting. Finally, we demonstrate and discuss the effectiveness of CopyCatch at Facebook and on synthetic data, as well as potential extensions to anomaly detection problems in other domains. CopyCatch is actively in use at Facebook, searching for attacks on Facebook's social graph of over a *billion* users, many *millions* of Pages, and *billions* of Page Likes.

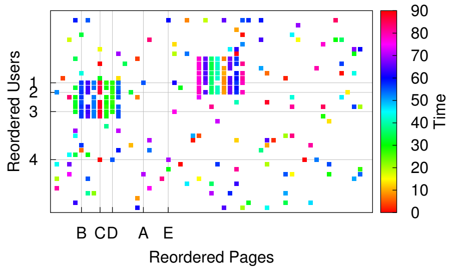
4.1 Introduction

As we found in the previous chapter, graph structure provides valuable signal for detecting fraudulent behavior, such as fake followers on Twitter and Tencent Weibo. While insightful, graph structure only provides part of the picture—how can we use other data collected online to stop fraudsters? How can we improve our accuracy and make it even more difficult for fraudsters to add fraud without being caught? In this chapter we analyze the fraud detection problem through the lens of detecting purchased Page Likes on Facebook.

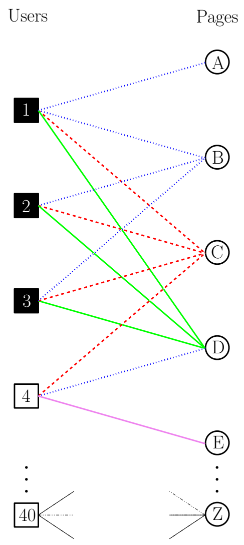
On Facebook, Pages are used by organizations to interact with their fans. Users can "Like" a Page to let their friends know about their interests and to receive content from that Page in their News Feed, the primary distribution channel on Facebook. Other users



(a) Without CopyCatch



(b) With CopyCatch



(c) Graphical view

Figure 4.1: **A toy example of Page Likes over time** with a subset of users and Pages organized to clearly show two detected attempts to inflate Page Like counts.

may interpret a high Like count as a Page being popular and also will see their friends' Page Likes in their News Feeds.

Because of its utility as a distribution channel, attackers frequently attempt to boost Page Like counts to get increased distribution for their content. At Facebook, we have found that attackers have attempted to inflate Like counts through a variety of deceitful methods, including malware, credential stealing, social engineering, and fake accounts. We define an ill-gotten Like, including those from the methods named previously, as "a Like that doesn't come from someone truly interested in connecting with a Page" [71]. As Facebook Security recently posted:

> Real identity, for both users and brands on Facebook, is important to not only Facebook's mission of helping the world share, but also the need for people and customers to authentically connect to the Pages they care about. When a Page and fan connect on Facebook, we want to ensure that connection involves a real person interested in hearing from a specific Page and engaging with that brand's content. [71]

From Page Likes to ratings and reviews throughout the web, the mission Facebook describes above holds—user generated content must be honest and legitimate if users are to trust and get value out of that web service.

To attack this problem, we build on our success with CatchSync and focus explicitly on the *many* fraudulent accounts needed to add *many* edges for attacks to be impactful and thus successful. However, Facebook already has many anti-phishing [69] and anti-malware [67, 68] mechanisms making it difficult for real accounts to be compromised, and many algorithms to detect fake accounts [201]. As a result, it is hard for an adversary to control many accounts, and instead they need to use the same few to Like many Pages.

Additionally, if fraudsters want to add *many* edges, they need to do so relatively quickly. Therefore, we look for *lockstep behavior* - groups of users acting together, generally Liking the same Pages at around the same time. We call our algorithm to detect such behavior CopyCatch and describe a process of *multi-user rate limiting* where, because of our constraint on Like times, we limit the rate at which a group of users can perform actions together (much stricter than you can limit individual users). Figure 4.1 demonstrates the challenge and the strength of CopyCatch in detecting such behavior.

To detect attackers attempting to deceive users, we again take a graph based approach to the problem. As we see in Figure 4.1(c), in the case of Facebook Page Likes, we have a bipartite graph between users and Pages, with the time at which each edge (Page Like) was created. Our algorithm searches for near-bipartite cores, where the same set of users Like the same set of Pages, and add constraints on the relationship between the edge properties (Like times) in this core. This can be extended to the bipartite graph of users to products where edges represent product reviews, or to general graphs such as the user to user connections of Instagram followers. We will discuss such extensions in Section 4.8.

In this chapter we offer a number of contributions, which build toward solving this problem:

1. **Problem Formulation:** We offer a novel problem formulation to a relevant, real-world challenge realized at Facebook and relevant in many online settings. We call our approach CopyCatch.
2. **Algorithm:** Pulling from work in one-class clustering and subspace clustering, we offer two algorithms to spot lockstep behavior: a provably-convergent serial iterative algorithm, and an approximate, scalable MapReduce implementation.
3. **Theoretical Analysis:** We show that catching anomalous behavior, as we have defined and as our algorithms detect, severely limits the damage an adversary can do when following a "greedy attack" strategy. We then apply research on the Zarankiewicz problem to our setting, showing that it is hard to find an optimal strategy against CopyCatch.

In Section 4.3 we give our problem formulation. In Section 4.4 we formulate the problem as an optimization problem, describe our serial algorithm, and prove that it converges. In Section 4.5 we describe our MapReduce algorithm and implementation. Finally in Section 4.6 we discuss the worst case damage an adversary could inflict, in Section 4.7 we offer experiments demonstrating the usefulness of our implementation at Facebook and on synthetic data, and in Section 4.8 we discuss the applicability of our approach to problems in other domains.

4.2 Relationship to Related Work

Our work in this chapter pulls from many different fields of research. Much of the related work on anomaly detection and fraud detection was previously outlined in Section II.1. Our problem formulation in this chapter differs from such prior work in our novel use of edge constraints, in this case based on time, to discern normal behavior from suspicious behavior. Since publication [33], further research has found temporal patterns to be an important signal in finding fraudulent behavior [234].

Algorithmically, we approach the problem similar to the co-clustering [27, 44, 63] and subspace clustering [126] literature. In order to scale effectively, we make use of local clustering and MapReduce, outlined below.

4.2.1 Local clustering

Clustering is one of the classic problems in both machine learning and data mining, with a wide range of methods still being developed. In this research, we build off Crammer *et al.*'s work on local one-class optimization and related work [57, 89], which focuses on finding dense clusters in noisy data through local search. Our algorithm also operates similarly to mean-shift clustering with a flat kernel [50].

4.2.2 MapReduce

To make our algorithm scale to large, web-scale data, we implement our algorithm in the MapReduce framework [61]. Hadoop [75] is an open source implementation of the MapReduce framework that is widely used. Facebook has a large Hadoop installation on which we built our implementation. In general, Hadoop and the Hadoop file system (HDFS) offer a distributed platform to store data and run parallel algorithms over a cluster of computers. We will give more details about the data flow and capabilities of Hadoop relevant for this Chapter in Section 4.5 and a more in-depth explanation of distributed machine learning on Hadoop in Part IV.

4.3 Problem Formulation

We now describe the mathematical details of our problem. Table 4.1 gives a list of the different symbols we will use throughout the chapter.

As we previously described, Facebook has the challenge of preventing adversaries from artificially inflating a Page's "Like count" in an effort to try to improve the Page's legitimacy and get distribution through the site. Since each user can only Like each Page once, a Page's Like count can only be increased through many users Liking the same Page.

Unfortunately, there is no ground truth to whether any individual Page Like is legitimate or not. Therefore, we take an unsupervised approach and only define suspicious behavior in terms of graph structure and edge creation times. Although our methods can be easily extended to many other settings, we will often describe the work in terms of Facebook users, Pages, and Likes for simplicity and clarity.

Before defining suspicious lockstep activity, we must give some notation surrounding our problem. We assume we have a set of users indexed from 1 to N, $\mathcal{U} = \{i\}_{i=1}^{N}$ and a set of M pages $\mathcal{P} = \{j\}_{j=1}^{M}$ similarly indexed. We define $\mathcal{E}$ as the set of edges in the

Symbol	Definition and Description
N and M	Number of nodes on either side of the bipartite graph (users and Pages)
$\mathbf{L}$	$N \times M$ data matrix of edge data (Like times)
$\mathbf{I}$	$N \times M$ adjacency matrix
$\mathcal{U}$ and $\mathcal{P}$	Set of indices of rows and columns (indexed users and Pages)
n and m	Bipartite core size threshold for suspiciousness
$\mathcal{P}'$	Subset of columns (Pages) that are suspicious
$\mathbf{c}$	Vector of times for each column around which there are suspicious rows (users)
$2\Delta t$	Width of time window
ρ	Percent of $\mathcal{P}'$ for which an suspicious user must be within the time window
ϕ	Thresholding function to compare two data points
s	Number of clusters being search for in parallel
$\mathscr{P}$	Set of $\mathcal{P}'$ for multiple clusters
$\mathscr{C}$	Set of $\mathbf{c}$ for multiple clusters

Table 4.1: Symbols and Definitions

graph, where $(i, j) \in \mathcal{E}$ is user i has Liked Page j. We also define an indicator matrix $\mathbf{I}$ such that $\mathbf{I}_{i,j} = 1$ if $(i, j) \in \mathcal{E}$, and $\mathbf{I}_{i,j} = 0$ otherwise. Last, we define our data matrix $\mathbf{L}$ such that $\mathbf{L}_{i,j} = t_{i,j}$ for all $(i, j) \in \mathcal{E}$, where $t_{i,j}$ is the time at which user i Liked Page j.

We can now broadly define our problem as:

Given: A graph of Likes between users and Pages $\mathbf{I}$ and the edge creation times $\mathbf{L}$

Find: Suspicious lockstep behavior - Bipartite cores of at least size (n, m) such that for each of the m Pages, all n users Liked that Page in a $2\Delta t$ time window. We call this an $[n, m, \Delta t]$-*temporally coherent bipartite core* (TBC).

We define suspicious lockstep behavior precisely below.

Definition 2 *We define an $[n, m, \Delta t]$-temporally coherent bipartite core (TBC) as a set of users $\mathcal{U}' \subseteq \mathcal{U}$ and a set of Pages $\mathcal{P}' \subseteq \mathcal{P}$ such that*

$$|\mathcal{U}'| \geq n \qquad \text{Size} \qquad (4.1)$$

$$|\mathcal{P}'| \geq m \qquad (4.2)$$

$$(i, j) \in \mathcal{E} \ \forall i \in \mathcal{U}', j \in \mathcal{P}' \qquad \text{Complete} \qquad (4.3)$$

$$\exists t_j \in \mathbb{R} \text{s.t.} |t_j - \mathbf{L}_{i,j}| \leq \Delta t \ \forall i \in \mathcal{U}', j \in \mathcal{P}' \qquad \text{Temporal} \qquad (4.4)$$

We consider users in an $[n, m, \Delta t]$-TBC to be in lockstep behavior and thus suspicious.

This can be interpreted a number of different ways, each of which we will use later in the chapter. From the graphical perspective, our indicator matrix $\mathbf{I}$ is an adjacency matrix for the bipartite graph between users and Pages, and our data matrix $\mathbf{L}$ contains the edge

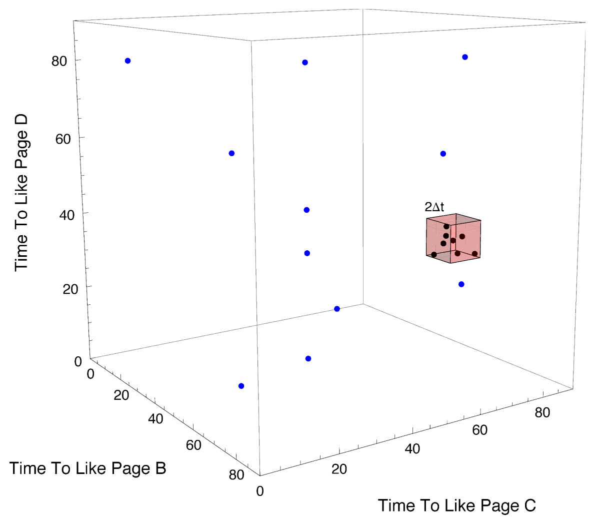


Figure 4.2: **Subspace clustering perspective**: An example of a subspace in which we find a clear clustering of Page Likes in time. Blue dots are normal users and black dots are suspicious users part of a $[8, 3, \Delta t]$-TBC.

creation time. A depiction of **L** pointing out clusters of users in near temporally coherent bipartite cores can be seen in Figure 4.1(a–b). In terms of the graph, we have defined suspicious behavior to be bipartite cores of size greater than (n, m) in the Facebook graph where all edges going into the same Page were created in a small time window. This graphical view of the data and anomalous behavior is shown on a subset of the nodes in Figure 4.1(c).

A second interpretation can be of **L** as a data matrix where each user represents a point in M dimensional space, $\mathbf{L}_{i,*} \in \mathbb{R}^M$. (Because users do not necessarily Like all Pages, they would often fall in a subspace of the M dimensional space, but thinking about each user as a point in the M dimensional space can provide good intuition.) We then consider a group of users to be part of an $[n, m, \Delta t]$-TBC and suspicious if there exists a hypercube of width $2\Delta t$ in at least m dimensions such that at least n users fall within that hypercube. Framing the problem this way is more similar to the standard clustering literature in machine learning and the subspace clustering problem. A depiction of a 3-dimensional subspace of the M dimensional space is shown in Figure 4.2, where a cluster of users are all found in the same small hypercube.

Later, we will show experimentally that, for appropriate values of n, m, and Δt, such behavior is extremely uncommon and thus in fact suspicious. We additionally will show the advantage of defining it this particular way as compared to other formulations, such as only looking for bipartite cores where all edges come from one time window. This particular formulation, with constraints on the inbound edges of each node, is novel in the literature, includes these other formulations as special cases, and provides a number of advantages for preventing fraudulent behavior.

4.4 Methodology

With Definition 2 of suspicious lockstep behavior, the challenge remains to detect when it occurs. As shown in [178], finding bipartite cores in a graph is NP-hard. To create an algorithm to find clusters of this type, we define the problem as an optimization problem. From here we offer an iterative algorithm, which monotonically improves our results, and in Section 4.5 we offer an approximate MapReduce implementation that searches for many bipartite cores in parallel.

4.4.1 Optimization Formulation

To formulate the problem succinctly we must add additional notation. We define $\mathbf{c} \in \mathbb{R}^M$ to be a vector for the center of our cluster, such that $\mathbf{c}_j = t_j$ where t_j comes from Definition 2.

We also relax our definition of suspicious lockstep behavior to include users that are part of *temporally-coherent* <u>*near*</u> *bipartite cores* (TNBC). We introduce the term $\rho \in [0, 1]$, which broadly describes how many of the Page Likes a user must match (in time) to be considered suspicious. More precisely, we say that a user is suspicious if he Likes at least $\rho|\mathcal{P}'|$ of the Pages in $\mathcal{P}'$ in the designated time window. This is clearly a relaxation since all of the users in any $[n, m, \Delta t]$-TBC would also be in a $[n, m, \Delta t, \rho]$-TNBC. We give the formal definition of $[n, m, \Delta t, \rho]$-TNBC below.

Definition 3 *A set of users $\mathcal{U}' \subseteq \mathcal{U}$ and a set of Pages $\mathcal{P}' \subseteq \mathcal{P}$ comprise an $[n, m, \Delta t, \rho]$-temporally coherent near bipartite core (TNBC) if there exists $\mathcal{P}'_i \subseteq \mathcal{P}'$ for all $i \in \mathcal{U}'$ such that:*

$$|\mathcal{U}'| \geq n \qquad \text{Size} \qquad (4.5)$$

$$|\mathcal{P}'| \geq m \qquad (4.6)$$

$$|\mathcal{P}'_i| \geq \rho|\mathcal{P}'| \; \forall i \in \mathcal{U}' \qquad \text{Near} \qquad (4.7)$$

$$(i, j) \in \mathcal{E} \; \forall i \in \mathcal{U}', j \in \mathcal{P}'_i \qquad \text{Complete} \qquad (4.8)$$

$$\exists t_j \in \mathbb{R} \text{s.t.} |t_j - \mathbf{L}_{i,j}| \leq \Delta t \; \forall i \in \mathcal{U}', j \in \mathcal{P}'_i \qquad \text{Temporal} \qquad (4.9)$$

Given these definitions our goal broadly is to maximize the number of suspicious users and the number of Page Likes of suspicious users that are suspicious (fall within the designated time window). Since we are ultimately trying to catch as many suspicious users as possible, we set $|\mathcal{P}'| = m$ and only try to grow $\mathcal{U}'$.

The optimization problem is given specifically below:

$$\max_{\mathbf{c}, \mathcal{P}' : |\mathcal{P}'| = m} \sum_i q(\mathbf{L}_{i,*} | \mathbf{c}, \mathcal{P}') \qquad (4.10)$$

where

$$q(\mathbf{u}|\mathbf{c},\mathcal{P}') = \begin{cases} \sigma & \text{if } \sigma = \sum_{j\in\mathcal{P}'} \mathbf{I}_{i,j}\phi(\mathbf{c}_j,\mathbf{u}_j) \geq \rho\, m \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

$$\phi(t_c, t_u) = \begin{cases} 1 & \text{if } |t_c - t_u| \leq \Delta t \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

This is a simple formulation of the problem described previously. The difference is under this formulation we are trying to find $\mathbf{c}$ and $\mathcal{P}'$ to maximize the number of users and their Likes inside the cluster centered at $\mathbf{c}$ in subspace $\mathcal{P}'$.

Additionally, because we expect most users to not be engaging in fraudulent Liking, we frame the problem similar to one-class clustering literature or to a flat kernel, where our optimization only focuses on data points *in* the cluster, and there is no penalty for data points outside the cluster.

4.4.2 A Serial Algorithm

To optimize this objective function, we must set both $\mathbf{c}$ and $\mathcal{P}'$. Placing $\mathbf{c}$ is similar to many density-seeking clustering problems in machine learning and data mining. Likewise, selecting $\mathcal{P}'$ from $\mathcal{P}$ is similar to subspace clustering. Therefore, we offer an iterative algorithm that in each step alternates between updating the center $\mathbf{c}$ and the subspace choice $\mathcal{P}'$, while holding the other variable constant. Note, given $\mathbf{c}$ and $\mathcal{P}'$, the users that fall within the cluster are fully determined. The algorithm can be seen in Algorithm 2.

In the step UPDATECENTER we keep $\mathcal{P}'$ constant and update $\mathbf{c}$. This update is performed iteratively over each dimension $j \in \mathcal{P}'$. For each such dimension, we find all users $\mathcal{U}'$ that fall within the cluster but loosen the width to $\beta\Delta t$ for the current dimension we are adjusting, where $\beta > 1$ thus including users who are just outside of the time window in dimension j. Given these users we find a new center in dimension j with subroutine FINDCENTER. We can sort the points in $\mathcal{U}'$ based on their position in dimension j, and then in one pass, weighting users by the number of Likes from $\mathcal{P}'$ they have, find the $2\Delta t$ span for which we capture the most users and the most Likes. We use this span to update $\mathbf{c}_j$. We note that UPDATECENTER runs in $O(m(mN + \log(n)))$ where we assume clusters are on the order of $O(n)$ in size.

In the step UPDATESUBSPACE we keep $\mathbf{c}$ constant and update $\mathcal{P}'$. Given the previous values of $\mathbf{c}$ and $\mathcal{P}'$ we can find the users currently in the cluster and attempt to improve our choice of $\mathcal{P}'$, such that more Likes are included for the same current set of users in the clusters. Here we take an incremental approach. For each $j \in \mathcal{P}'$ we search among all $j' \in \mathcal{P}$. We say a user i is *covered* by a column j if $\mathbf{I}_{i,j}\phi(\mathbf{c}_j, \mathbf{L}_{i,j}) = 1$. *We only consider those columns j' for which every user covered by column j is also covered by column j'.* We can then replace column j by column j' that has the most users covered. As such, any user that was covered previously will still be covered, but we can also be adding additional coverage (for more Likes) to other users. This is not necessarily the optimal choice, but it does improve our objective and runs in $O(nmM)$ time.

We repeatedly update **c** and $\mathcal{P}'$ until neither change and the algorithm has converged, or simply for some fixed number of rounds.

4.4.3 Proof of Convergence

We now prove that our algorithm converges. It should be clear that our objective function is bounded, as there are a limited number of users and Page Likes, and therefore there is a maximum or set of local maxima. Therefore, we must merely show that both UPDATECENTER and UPDATESUBSPACE monotonically improve our objective function.

Lemma 1 UPDATECENTER, *as defined in Algorithm 2, monotonically improves our objective function in* (4.10).

Proof 2 UPDATECENTER *works by updating each dimension's center one at a time, holding the others constant. For each update in each dimension, we take all the points within $\beta\Delta t$ of the previous center and find the center that will most improve our objective. Of course, all points previously covered will be included in this width of $\beta\Delta t$ since $\beta > 1$. Since we find the location for which we cover the most points weighted by the number of Likes for each point, we will only move the center if we find a location that covers more points with more Likes than before. Therefore, if our center moves the objective function must increase, and if it does not move then the objective function stays constant.*

Lemma 2 UPDATESUBSPACE, *as defined in Algorithm 2, monotonically improves our objective function in* (4.10).

Proof 3 *As was described previously,* UPDATESUBSPACE *only replaces a $j \in \mathcal{P}'$ with a j' if all Likes covered by Page j are also covered by Page j'. Therefore, we can only improve our objective function or stay constant. Therefore* UPDATESUBSPACE *monotonically improves our objective.*

Because UPDATECENTER and UPDATESUBSPACE monotonically improve our objective, the algorithm converges.

4.5 A MapReduce Implementation

Although Algorithm 2 works well theoretically, it has inefficiencies in both speed and convergence. To address these issues we offer here a new algorithm similar to the serial algorithm, which operates in the MapReduce framework. This implementation operates under the trade-off of making the algorithm scalable to massive data sets and trivially parallelizable, such that we can search for many clusters simultaneously, with the cost of the algorithm not provably converging. However, the heuristics used here have performed well in practice on real world data and converge quickly, as will be demonstrated in Section 4.7.

Unlike many optimization problems, where the goal is to find the global maximum, here we want to find all local maxima that meet our criteria (as there could be many attacks happening simultaneously with different users on different Pages). Therefore,

Algorithm 2: Serial CopyCatch

```
function S-CopyCatch($\mathbf{x}, j$):
Require: Preset parameters $\Delta t$, $n$, $m$, and $\rho$
Initialize $\mathbf{c} = \mathbf{x}$, $\mathcal{P}' = \{j\}$
repeat
    $\mathcal{P}'_\ell = \mathcal{P}'$
    $\mathbf{c}_\ell = \mathbf{c}$
    $\mathbf{c} = \text{UpdateCenter}(\mathbf{c}, \mathcal{P}')$
    $\mathcal{P}' = \text{UpdateSubspace}(\mathbf{c}, \mathcal{P}')$
until $\mathbf{c} = \mathbf{c}_\ell$ and $\mathcal{P}' = \mathcal{P}'_\ell$ // Run to convergence
return $[\mathbf{c}, \mathcal{P}]$
```

```
function UpdateCenter($\mathbf{c}, \mathcal{P}'$):
$\mathcal{U}' = \text{FindUsers}(\mathcal{U}, \mathbf{c}, \mathcal{P}')$ // Get current users
Set $\mathbf{c}'$ to the average of $\mathbf{L}_{i,*}$ for all $i \in \mathcal{U}'$
// Update center for each Page
for $j \in \mathcal{P}'$ do
    $[\mathcal{U}', \mathbf{w}] = \text{FindUsers}(\mathcal{U}, \mathbf{c}, \mathcal{P}', j, \beta\Delta t)$
    $[\mathcal{U}'', t_j] = \text{FindCenter}(\mathcal{U}', \mathbf{w}, j)$
    $\mathbf{c}'_j = t_j$
end for
return $\mathbf{c}'$
```

```
function UpdateSubspace($\mathbf{c}, \mathcal{P}'_\ell$):
$\mathcal{P}' = \mathcal{P}'_\ell$
$\mathcal{U}' = \text{FindUsers}(\mathcal{U}, \mathbf{c}, \mathcal{P}'_\ell)$ // Get current users
for $j' \in \mathcal{P}'_\ell$ do
    $j'' = j'$
    $\mathcal{U}'_{j''} = \text{FindUsers}(\mathcal{U}', \mathbf{c}_{j''}, \{j''\})$
    // See if another Page is better
    for $j \in \mathcal{P} \setminus \mathcal{P}'$ do
        $\mathcal{U}'_j = \text{FindUsers}(\mathcal{U}', \mathbf{c}_j, \{j\})$
        if $\mathcal{U}'_{j''} \subset \mathcal{U}'_j$ then
            $j'' = j$, $\mathcal{U}'_{j''} = \mathcal{U}'_j$
        end if
    end for
    $\mathcal{P}' = (\mathcal{P}' \setminus \{j'\}) \cup \{j''\}$
end for
return $\mathcal{P}'$
```

```
// Find weighted center of $\mathcal{U}$ in dimension $j_c$
function FindCenter($\mathcal{U}, \mathbf{w}, j_c$):
Sort $\mathcal{U}$ by $\mathbf{L}_{i,j_c}$ for $i \in \mathcal{U}$
Scan sorted $\mathcal{U}$ for $2\Delta t$-width subset $\mathcal{U}'$
    s.t. $\sum_{i \in \mathcal{U}'} \mathbf{w}_i$ is maximized
Set $c_j$ to the center of this subset $\mathcal{U}'$
return $[\mathcal{U}', c_j]$
```

```
// Find users from $\mathcal{U}$ based on $\mathbf{c}$ and $\mathcal{P}'$
function FindUsers($\mathcal{U}, \mathbf{c}, \mathcal{P}', j_c, \Delta t'$):
$\mathcal{U}' = \{\}$, $\mathbf{w} = \mathbf{0}$
for $i \in \mathcal{U}$ do
    for $j \in \mathcal{P}'$ do
        if $\mathbf{I}_{i,j} = 1 \wedge (|\mathbf{c}_j, \mathbf{L}_{i,j}| < \Delta t \vee$
            $(j = j_c \wedge |\mathbf{c}_j, \mathbf{L}_{i,j}| < \Delta t'))$ then
            $\mathbf{w}_i = \mathbf{w}_i + 1$
        end if
    end for
    if $\mathbf{w}_i \geq \rho m$ then
        $\mathcal{U}' = \mathcal{U}' \cup \{i\}$
    end if
end for
return $[\mathcal{U}', \mathbf{w}]$
```

the ability to run the algorithm from many starting points in parallel is both useful and more efficient than running from different starting points serially.

4.5.1 Algorithm

Because we now would like to run the algorithm for multiple clusters in parallel, we must introduce some additional notation. We define s to be the number of clusters being run simultaneously. Each cluster has a center $\mathbf{c}^{(k)} \in \mathbb{R}^M$ and a set of currently selected columns $\mathcal{P}'_k \subseteq \mathcal{P}$ (each defined as before). We define $\mathcal{C}$ to be the set of all $\mathbf{c}^{(k)}$ and $\mathcal{P}$ to be the set of all $\mathcal{P}'_k$, both for $k = 1 \dots s$.

Like the serial algorithm, the MapReduce CopyCatch algorithm operates by updating $\mathbf{c}$ and $\mathcal{P}'$ iteratively. The core of the algorithm can be seen in Algorithm 3, where we note that we run one MapReduce job per iteration, each time updating $\mathcal{C}$ and $\mathcal{P}$. As in the serial algorithm, we can keep iteratively updating $\mathcal{C}$ and $\mathcal{P}$ until no changes are made. In practice, we will merely run a fixed number of iterations.

MapReduce It is worth taking a moment to note the data flow in a MapReduce job before describing the details of our algorithm. In the Map step, our input is split among many mappers. Each mapper gets a pair of data of the form $\langle \text{KEY}_{\text{map}}, \text{VALUE} \rangle$ and can output zero or more results of the form $\langle \text{KEY}_{\text{reduce}}, \text{VALUE} \rangle$. In the reducer step, for each unique $\text{KEY}_{\text{reduce}}$ a reducer is formed which takes as an input $\langle \text{KEY}_{\text{reduce}}, \text{VALUES} \rangle$, where VALUES is a set of the VALUE outputs from the mapper step which correspond to that reducer's particular $\text{KEY}_{\text{reduce}}$. The reducer can then output data to disk. Aside from this data flow, we make use of Hadoop's Distributed Cache, which lets us store data as global read-only data. For more information on MapReduce and Hadoop see [61, 75].

In our implementation, the mapper for the MapReduce job USERMAPPER is shown in Procedure 4, the reducer ADJUSTCLUSTER-REDUCER is shown in Procedure 5, and $\mathcal{C}_\ell$ and $\mathcal{P}_\ell$ are stored in the Distributed Cache.

USERMAPPER finds which users are currently in which clusters based on $\mathcal{C}$ and $\mathcal{P}$ and maps those users to a reducer based on which cluster it is within. More specifically, the Map step takes as input $\mathbf{L}$ and $\mathbf{I}$, where each $(\mathbf{L}_{i,*}, \mathbf{I}_{i,*})$ for all $i \in \mathcal{U}$ is input to a mapper. Each mapper checks the $\mathbf{L}_{i,*}$ across all s clusters to see if it falls within that cluster following the definition given in our optimization objective (4.10). If it does, it emits an output where the key is the cluster ID k, and the value is the row (user) information $\mathbf{L}_{i,*}$ and $\mathbf{I}_{i,*}$. Each mapper runs in $O(sm)$ time, and since this is being run over all data the entire step takes $O(smN)$ not taking into account parallelization.

AdjustCluster-Reducer takes in all of the users currently in a given cluster k and updates $\mathbf{c}^{(k)}$ and $\mathcal{P}'_k$. Each reducer takes as an input $\langle k, \mathcal{U}' \rangle$, where $\mathcal{U}'$ here contains pairs $(\mathbf{L}_{i,*}, \mathbf{I}_{i,*})$ for all users in the cluster (as was output by the mappers). As shown in Procedure 5, we must be careful to only use values from each user in the dimensions for which it falls within the cluster. However, beyond this, the update generally works fairly simply. The center $\mathbf{c}$ is updated by merely taking an average of the points in the cluster, similar to a mean-shift algorithm with a flat kernel [50]. The selected columns are chosen based on which columns cover the most users from the previous cluster parameters, and then by which columns have the lowest variance among these users. By using the previous centers for this update, we can do the calculation online, passing over each user only once. Because we assume each cluster is $O(n)$ in size, each reducer takes $O(nM)$ time, and the reduce step as a whole takes $O(snM)$ when not considering parallelization. The reducers output the updated $\mathcal{C}$ and $\mathcal{P}$ to be placed in the Distributed Cache in subsequent iterations.

Algorithm 3: MapReduce CopyCatch

```
1: Require: Preset parameters $\Delta t$, $m$, and $\rho$
2: $\mathcal{C}, \mathcal{P} = \textsc{Initialize}()$
3: repeat
4:   $\mathcal{C}_\ell = \mathcal{C}, \mathcal{P}_\ell = \mathcal{P}$
5:   $\mathcal{C}, \mathcal{P} = \textsc{MapReduceJob}(\mathcal{C}_\ell, \mathcal{P}_\ell)$
6: until $\mathcal{C}_\ell = \mathcal{C} \wedge \mathcal{P}_\ell = \mathcal{P}$
7: return $[\mathcal{C}, \mathcal{P}]$
```

Algorithm 4: $\textsc{UserMapper}(\langle \textsc{Null}, (\mathbf{L}_{i,*}, \mathbf{I}_{i,*})\rangle)$

```
1: Globals: $\mathcal{C}, \mathcal{P}$
2: for $k = 1 \ldots s$ do
3:   $\sigma = \sum_{j \in \mathcal{P}'_k} \mathbf{I}_{i,j} \cdot \phi(\mathbf{c}_j^{(k)}, \mathbf{L}_{i,j})$
4:   if $\sigma \geq \rho|\mathcal{P}'_k|$ then
5:     emit $\langle k, (\mathbf{L}_{i,*}, \mathbf{I}_{i,*})\rangle$
6:   end if
7: end for
```

4.5.2 Implementation Optimizations

While the description above gives the general overview of the algorithm, there are a number of implementation details that make the algorithm run efficiently on huge data sets.

Data Format Because $\mathbf{L}$ is expected to be very large and sparse, we do not want to, nor need to, store the full data matrix. Instead, we store the matrix as an adjacency list. For each user $i \in \mathcal{U}$, we store on one line the user ID i, a list of page IDs j where $\mathbf{I}_{i,j} = 1$, and the value $\mathbf{L}_{i,j}$. As a result, NULL values were $\mathbf{I}_{i,j} = 0$ do not take up space or time.

$\mathcal{C}$ and $\mathcal{P}$ are stored similarly where each line of a data file is indexed by the cluster ID k, and then contains the Page IDs j, the times $\mathbf{c}_j^{(k)}$ and a 1 if $j \in \mathcal{P}'_k$.

Algorithm 5: ADJUSTCLUSTER-REDUCER(k,$\mathcal{U}'$)

```
1:  Globals: $\mathcal{C}, \mathcal{P}$
2:  Initialize $\mathbf{c} = \mathbf{0}$, $\mathbf{p} = \mathbf{0}$, $\mathbf{v} = \mathbf{0}$
3:  for all map values $(\mathbf{L}_{i,*}, \mathbf{I}_{i,*}) \in \mathcal{U}'$ do
4:    for $j = 1 \dots M$ do
5:      if $\mathbf{I}_{i,j} = 1 \wedge \phi(\mathbf{c}_j^{(k)}, \mathbf{L}_{i,j}) = 1$ then
6:        $\mathbf{c}_j = \mathbf{c}_j + \mathbf{L}_{i,j}$
7:        $\mathbf{p}_j = \mathbf{p}_j + 1$
8:        $\mathbf{v}_j = \mathbf{v}_j + (\mathbf{c}_j^{(k)} - \mathbf{L}_{i,j})^2$
9:      end if
10:   end for
11: end for
12: $\mathbf{c}^{(k)} = \mathbf{c}/\mathbf{p}$
13: $\mathbf{v} = \mathbf{v}/\mathbf{p}$
14: Sort $\{j\}_1^M$ by $\mathbf{p}$ (decreasing), then $\mathbf{v}$ (increasing)
15: Set $\mathcal{P}'_k$ to top $m$ columns from previous sort
16: return Updated $\mathbf{c}^{(k)}$ and $\mathcal{P}'_k$
```

Seeds and Initial Iterations Figuring out where to start the clusters can be very difficult. To avoid any bias, we sample seeds randomly from the list of all edges in the graph, using the edge's Page and Like time to initialize both $\mathcal{P}'$ and $\mathbf{c}$. (While we could use suspicious users from one of Facebook's many other security mechanisms, this would introduce prior assumptions about attackers that are unnecessary and could make it easier for an adversary to hide.)

As a result, in the initial iterations users only need to have Liked a single Page at around the same time, which is not uncommon. There are often so many users found in these initial iterations that we sample a small percentage of them at random to keep the algorithm efficient. Even with the sampling, this method lets us quickly find lots of users that Liked one Page around the same time, and then see what else they have in common. This sampling is performed in the first two iterations until $|\mathcal{P}'| = m$.

Page sampling Because $\mathcal{C}$ and $\mathcal{P}$ are stored in the Distributed Cache, they are passed to every MapReduce node. This communication time slows the algorithm down if the data becomes too large. To avoid this we limit the length of $\mathbf{c}^{(k)}$ by including only Pages from $\mathcal{P}'_k$, Pages that were close to being in $\mathcal{P}'_k$ in the last iteration, and a random sampling of the other Pages. With this method we can still find users that are similar, but without the risk of becoming too slow.

4.6 An Adversarial Challenge

Given our definition of suspicious lockstep behavior and our approach to detecting spam, how much damage could an adversary do without appearing suspicious? We will again discuss this in the context of Facebook, although it can be extended to other applications.

To be more concrete, we can frame the question as follows: If an adversary controls N' accounts and wants to Like M' Pages, for $N \gg N' \gg n$ and $M \gg M' \gg m$, how long would it take the adversary to Like all M' Pages with all N' accounts without creating an $[n, m, \Delta t, \rho]$-TNBC? (For this analysis we assume that we catch all lockstep behavior meeting Definition 3.) It turns out this is an extension of an old open problem in extremal graph theory. We analyze below a couple different approaches to this problem.

4.6.1 "Greedy Attacks"

We first analyze the consequences of an adversary performing a naïve greedy attack, particularly because it matches a common business model for adversaries, demonstrates the difficulties for an adversary, and shows the strength of our security approach.

In this initial example, we assume that the adversary has from the start the set of M' Pages he wants to Like, and we analyze the effect if he iteratively Likes each Page with as many accounts as he can without getting caught. That is, for the first Page the adversary will Like the Page using as many accounts as possible without getting caught. He will then move onto the second Page and do the same thing, etc. In this case, we are unconcerned with the time component of our bipartite core definition because we assume the adversary will add all of the Likes instantaneously. Therefore, an adversary will be caught if he creates an (n, m) complete bipartite core in the graph. If we look at the $N' \times M'$ adjacency matrix of Likes added by the adversary, this is equivalent to creating an n, m submatrix filled with ones. We will call this $N' \times M'$ adjacency matrix $\mathbf{I}'$.

Lemma 3 *We assume there is an adversary with N' accounts performing a greedy attack on M' Pages. Catching and preventing all behavior meeting Definition 2 limits the adversary to obtaining $(m-1)N' + (n-1)M' - (n-1)(m-1)$ Likes at a rate greater than $\frac{n-1}{2\Delta t}$ per Page per unit time.*

Proof 4 *A simple pattern created by following the greedy approach can be seen in Figure 4.3. We see here for the first $m-1$ pages, the adversary can Like the Page with all N' accounts. This is because we will only detect the accounts if at least m pages have been Liked, so there is no harm in Liking the first $m-1$ Pages with as many accounts as possible. However, for the mth Page, if any n users Like that page in a $2\Delta t$ time window, then he will have created an $n \times m$ full submatrix. This is true for all Pages $\geq m$. As a result, following this greedy approach we can only Like each Page an average of $\frac{(m-1)N'+(n-1)M'-(n-1)(m-1)}{M'}$ times.*

While not a complex example, the results are quite interesting. First, we see that as an adversary adds more accounts, the average number of Likes added by each account asymptotically approaches $m-1$. Similarly, as the accounts Like more Pages, the average number of Likes per Page asymptotically approaches $n-1$.

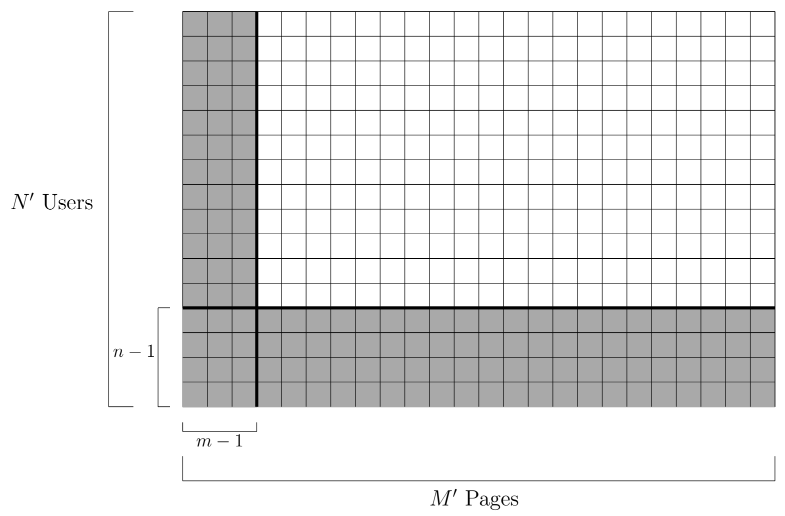


Figure 4.3: **Illustration of a greedy attack**: $N' \times M'$ adjacency matrix $\mathbf{I}'$ of all Likes, where grey cells denote $\mathbf{I}'_{i,j} = 1$, and white cells denote $\mathbf{I}'_{i,j} = 0$.

We have so far focused on how many Likes can an adversary add instantaneously, but it is worth looking at the effect of our time constraint. To illustrate the impact of our temporal coherence restriction, we look at the case where an adversary sells Likes on eBay, promising to add the Page Likes within some time period (as some adversaries are known to do). As new orders come in, the adversary must add those Page Likes within the requested amount of time. Because we have a distinct time window for each Page j (centered around $\mathbf{c}_j$), waiting to add Likes to a new Page does not help an adversary avoid being caught. That is, if an adversary gets $m-1$ orders in January and adds all N' Likes as the greedy attack suggests, then even if he does not get any more orders until July he still cannot add more than $n-1$ Likes in a $2\Delta t$ time window without being caught.

This is in contrast to previous research, which could set a time window for the entire bipartite core (through limiting the input), and thus merely waiting out that time window gave accounts a clean slate. As a result of our construction, once the greedy attack has been run, the adversary can only continue to add Page Likes to each Page at a rate slower than $\frac{n}{2\Delta t}$. We call this effect *multi-user rate limiting*.

To make the effect of these restrictions clear and concrete, let's look at the example limits $n = 20$ accounts, $m = 5$ Pages, and $2\Delta t =$ one week. (Note, for security reasons these are *not* the actual limits used at Facebook.) Any adversary performing a greedy attack with $N' = 1000$ accounts can on average only like up to 5 Pages per account for 980 of his accounts. Similarly, if the adversary wants to boost the Like counts for more than 5 Pages, he can on average only Like the Pages 20 times. To add additional likes to any given Page can only be done at a rate slower than $\frac{20 \text{ accounts}}{1 \text{ week}} \approx 3$ Likes per Page per day. This is clearly much harsher rate-limiting than we could enforce on a single user, but when looking at the group acting together makes sense.

4.6.2 Optimal Strategy: An Open Problem

It should be clear from the previous analysis that greedy attacks do not work well for spammers. Ideally, we would like to find an upper bound for the amount of damage an adversary could inflict. However, this turns out to be an old, open problem in extremal graph theory. In 1951 Kazimierz Zarankiewicz posed the following problem [239]: how many edges can there be in a bipartite graph G of size $N' \times M'$ without creating a complete bipartite core of size $n \times m$? This is known as the Zarankiewicz problem, and the maximum number of edges is denoted as $z(N', M', n, m)$.

Although the problem is old, little progress has been made in solving it for the general case. Füredi [76] is currently known to have the best general upper bound

$$z(N', M', n, m) \leq (n - m + 1)^{\frac{1}{m}} N' M'^{1-\frac{1}{m}} + M'm + mN'^{2-\frac{2}{m}}$$

for $n > m$. However, this is only known to be asymptotically optimal for $m = 2$ and $n = m = 3$. If we use larger values of n and m and reasonable values of N' and M', then $z > N'M'$ and thus offers us no information.

Additionally, the proof offered by Füredi [76], as well as most work surrounding the Zarankiewicz problem, uses non-constructive methods. That is, although the proof finds an upper bound for $z(N', M', n, m)$, it does not give any information about how to actually add edges to reach that bound without being caught. It would therefore be a challenge for an adversary to optimally add edges (and would be interesting for the field of extremal graph theory if solved).

It is worth noting that our problem deviates from the classic Zarankiewicz problem in two regards: (1) we look for near-bipartite cores, and (2) we require temporal coherence. With respect to the first point, by looking for near bipartite cores and not just complete bipartite cores, we would also catch cases where certain 1's were missing from the $n \times m$ submatrix of $\mathbf{I}'$. We call this the *approximate Zarankiewicz problem*. While any upper bound for the Zarankiewicz problem is also an upper bound to this approximate Zarankiewicz problem, the maximum number of edges added without creating a $[n, m, \Delta t, \rho]$-TNBC would be lower, and thus this is an even harder problem for an adversary.

Second, we require temporal coherence in our bipartite cores for the behavior to be considered suspicious. This helps us more accurately and flexibly discern normal behavior from illegitimate behavior and effectively rate limits an adversary, forcing them to add Likes very slowly if they do not want to be caught. As we saw in our analysis of a greedy attack, this multi-user rate limiting is significantly stricter than a rate limit we could enforce on individual users. When generalizing this concept to the Zarankiewicz problem, the constraint adds complexity to an already challenging problem. While we cannot find a precise optimal rate at which adversaries can add edges without knowing an optimal strategy for the Zarankiewicz problem, it should be clear that the multi-user rate limiting principle holds. Setting $2\Delta t = \infty$ restricts an adversary to solving the Zarankiewicz problem; decreasing Δt allows an adversary to exceed the maximum solution to the Zarankiewicz problem but at a slow rate.

4.7 Experimental Analysis

4.7.1 Experimental Setup

CopyCatch was written in Java 1.6.0_14 with Hadoop 0.20.1, generally matching the algorithm outlined in Section 4.5. The experiments were run on one of Facebook's Hadoop clusters, running Hadoop, Hive, and HDFS on over 1000 machines. More information about Facebook's infrastructure can be found in [211]. Our MapReduce jobs ran with 3000 mappers and 500 reducers. We ran the algorithm on a few different datasets from Facebook as well as synthetic data to demonstrate a number of different properties.

The Facebook datasets used come from real Likes between users and Pages on the site. We pull data from periods of time ranging from weeks to multiple months, where the data has not already had Likes removed by this particular method (although of course many other security measures already keep malicious users and fake Likes off the site). For our scalability tests, data ranged from approximately 760 million Page Likes (25 gigabytes on HDFS) to 10.4 billion Page Likes (294 gigabytes on HDFS). When not testing scalability over data size, we used an intermediate dataset of 3.3 billion Likes (100 gigabytes on HDFS). Our parameter choices for n, m, and Δt are those used currently on Facebook systems, but can not be given for security reasons.

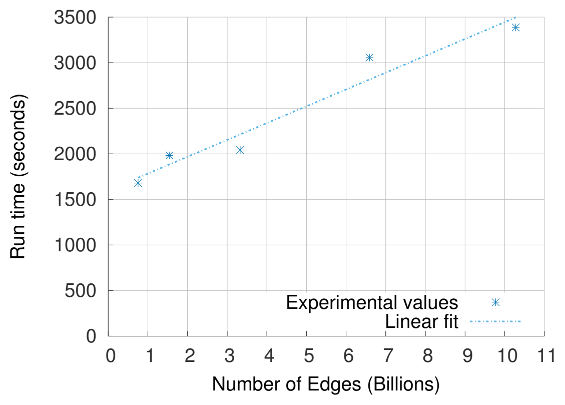
We also ran our discoverability experiments on synthetic data so that results could be replicated by other researchers. Our synthetic data was generated following the RTM method [16], where the time evolving graph is generated with a repeated Kronecker product. The code used to generate the time evolving graph, including our initial generator/tensor, can be found online at cs.cmu.edu/~abeutel/www2013. The generated graph is a bipartite graph between 38 million and 10 million nodes with 410 million edges. After our data formatting, the graph is 10 gigabytes on HDFS.

4.7.2 Scalability

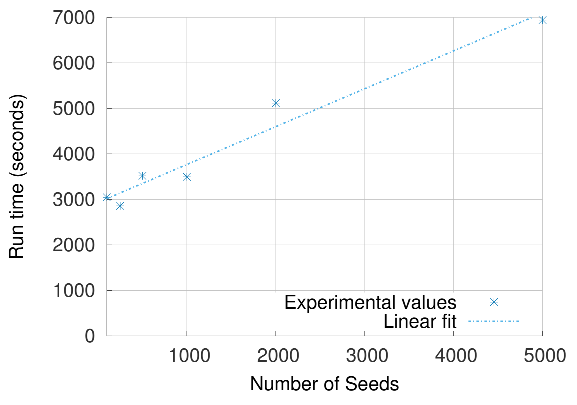
Facebook now has over a billion users and is continuing to grow. Therefore, it is important that our algorithm scales well to large datasets. We test this a few different ways.

A necessary preprocessing step to keep our algorithm efficient is to format the data similar to an adjacency list as described in Section 4.5.2. This takes approximately 45 minutes for our 100GB Facebook dataset. Because this is merely formatting and not required each time we run the algorithm, we do not consider this time in future tests.

For each of our timing experiments, we run our algorithm on the on the 100GB Facebook dataset and time the first 3 iterations, which includes the initial iterations of starting with individual Likes as seeds. We chose to run these tests on the Facebook data because run time is heavily influenced by the size of the data and the size of clusters. Since we do not know a-priori where large clusters are, we sample our seeds randomly so that we get the same distribution of small and large clusters that we would get in our real runs.



(a) Scaling in graph size



(b) Scaling in number of seeds

Figure 4.4: **Scalability experiments**: (a) shows the linear increase in computation time as the graph grows into the billions of Page Likes, (b) shows the increase in computation time as the number of seeds increases. Running time for is based on the total time for the first three iterations of CopyCatch.

Even with large Hadoop clusters, it is important that the algorithm can scale as the data scales. We test this by running our implementation over Page Likes from increasingly large periods of time. As we explained previously, this data ranges from 25GB with 760 million Page Likes to 294GB with 10.4 billion Page Likes. We ran the algorithm with 100 seeds, 3000 mappers, and 500 reducers. The run time for the first 3 iterations can be seen in Figure 4.4(a). As seen in the plot, the run time increases approximately linearly with the number of edges (Page Likes). However, we note that only after a greater than 10 times increases in data size does our running time double. Therefore as our data grows, the running time of CopyCatch grows at a much slower rate.

As the data scales, we also face the challenge of having to use more seeds to sufficiently sample the space. Therefore, we also tested how the run time increases as the number of clusters being run in parallel increases. We time the first 3 iterations for running the algorithm on the 100GB Facebook dataset with 3000 mappers and 500 reducers. We vary the number of seeds used from 100 to 5000. As can be seen in Figure 4.4(b), we find a similar linear relationship between number of seeds and run time. We note that 50 times increase from 100 clusters to 5000 clusters takes only a little over twice as much time. This is again reassuring that our implementation exploits the parallelism of the problem and can continue to scale as the data scales.

4.7.3 Convergence

Because our MapReduce implementation is not provably convergent, we also test our algorithm's convergence. We ran the algorithm on the 100GB Facebook dataset, starting with 1000 seeds and tracked the sum of the values of the objective functions per iteration,

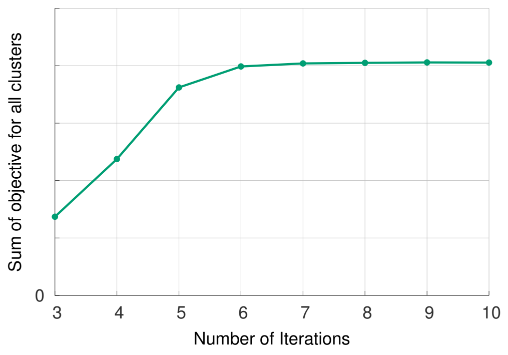
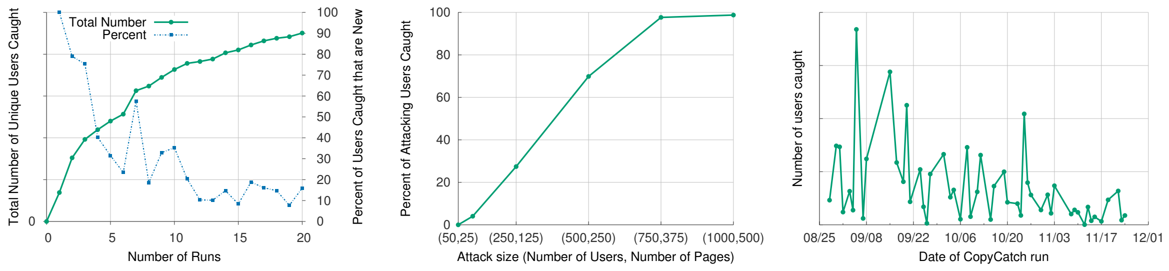


Figure 4.5: **The convergence of the CopyCatch MapReduce implementation** over 10 iterations on Facebook data.



(a) Facebook (b) Synthetic (c) Facebook live runs

Figure 4.6: **Plots demonstrating the effectiveness of CopyCatch in discovering attacks**. (a) shows the total number of users caught after multiple runs of CopyCatch. (b) shows the success in finding planted attacks in synthetic data after one run. (c) shows the decrease in attacks on Facebook over the last 3 months.

starting at the third iteration when $|\mathcal{P}'| = m$. As can be seen in Figure 4.5, the algorithm quickly converges in only a few iterations as desired.

4.7.4 Discovery

The effectiveness of the algorithm is measured by its ability to detect fraudulent behavior. This is difficult to gauge in the real world where there isn't labelled data, and it can be impossible to know if a user intentionally, honestly Liked a given Page. Here we take two different approaches to evaluate our method.

First, we analyze the success of the algorithm in finding suspicious groups of users on Facebook. Our goal is for the algorithm to find as many of the $[n, m, \Delta t, \rho]$-TNBC's in the Facebook data as possible. To test CopyCatch's success, we run the algorithm repeatedly and check that we eventually are mostly finding users we had caught in previous runs. In our experiment we run the algorithm 20 times for 5 iterations, each run starting from 1000 seeds on the 100GB data set. In Figure 4.6(a) we see that over the course of 20 runs we quickly decrease to finding mostly the same users repeatedly, and only catching a small percentage of new users with each successive run. More precisely, after the eleventh run, the average percent of caught users that are new is only 13% percent. Because we use random independently drawn seeds for each run, this test suggests that the algorithm has found most of the attacks in the dataset.

For a more precise analysis we use our synthetic dataset to test the algorithm's ability to find attacks in the graph. We set $n = 50$, $m = 25$, $\Delta t = 50$, and $\rho = 0.9$. For our tests we add 20 randomly-placed attacks to the graph, and test the ability of the algorithm to find the attacks after one run starting with 5000 randomly chosen seeds. We vary our attack size as a multiple of our defined suspicious lockstep behavior, ranging from 50 users and 25 Pages to 1000 users and 500 Pages. In each case the time of the attack for each Page is chosen at random, and each user's Likes falls within the $2\Delta t$ time window for 95% of the Pages being attacked. The percent of attackers caught after 1 run for each attack size are plotted in Figure 4.6(b). Also we note that 0% of our caught users were false positives. As we see in the plot, small attacks exactly at our threshold size were hard to spot, but as attacks grow in size in just one run we catch nearly all of the attackers. It is worth noting that because we choose seeds randomly, running the algorithm again should catch an independent set of the attackers, and thus merely running the algorithm a few times should catch a high percentage of the attackers even in cases where 1 run does not. Overall, this experiment shows the algorithm is generally successful in detecting a few relatively small attacks in large graphs.

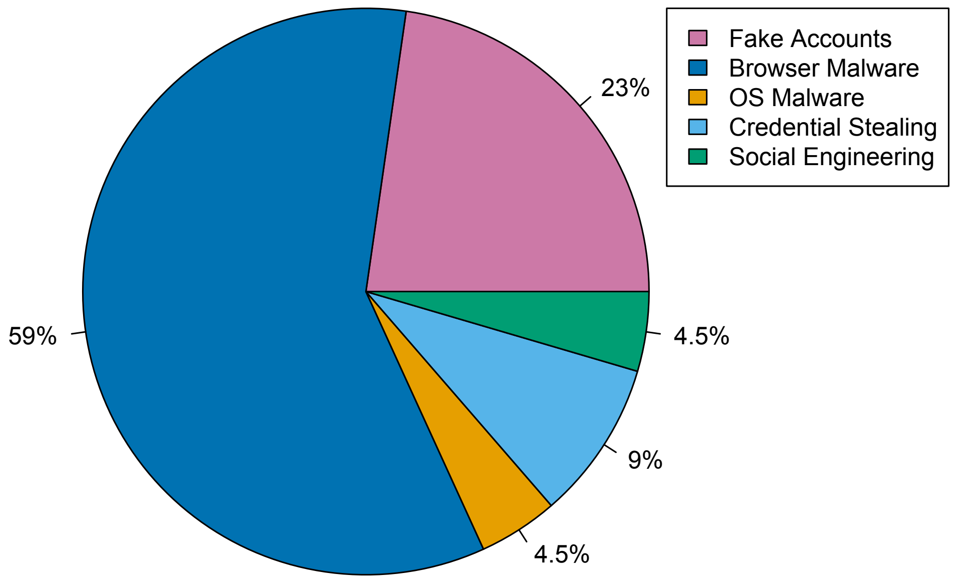


Figure 4.7: **Breakdown of attack vectors found on Facebook**. Note, most attacks are from real but compromised users.

4.7.5 Deployment at Facebook

CopyCatch is run regularly at Facebook, searching for new attacks. Parameters have been chosen to significantly distinguish natural user behavior from ill-gotten Page Likes. In practice at Facebook, false positives are very rare due to the sparsity of the Page Like matrix. In particular, we labelled 22 randomly selected clusters caught in February 2013. After intensive manual investigation, we found that 100% of the clusters were caught due to Likes generated through deceitful means. As can be seen in Figure 4.7, of the attacks, 5 were from fake accounts, 13 from malicious browser extensions, 1 from OS malware, 2 from credential stealing, and 1 from social engineering. Note, this means that most attacks were from real users who were compromised, and not Sybil accounts. When caught, users that have contributed to ill-gotten Page Likes have a portion of their past month's Likes removed and are prevented from Liking pages for the next month. This removal of Page Likes are reflected in the Like counts of the Pages that benefited. As shown in Figure 4.6(c), we have seen a general decrease in ill-gotten Page Likes since the start of this method. Overall, this method, in combination with the numerous other measures at Facebook mentioned in Section 4.1, has proved effective in decreasing ill-gotten Page Likes.

4.8 Discussion: Applications

As mentioned previously, the algorithm can be used in a number of settings including Twitter followers and Amazon product reviews. The use case at Twitter demonstrates the ability of the algorithm to be applied to non-bipartite graphs. However, in this case, the general structure of the algorithm is the same—we look for some set of users that follow another set of users at around the same time.

In this case of product reviews, we again have a bipartite graph between users and products where edges represent a review that was given to the product. Although we have described the edge constraints in terms of time, the algorithm only requires some center $\mathbf{c}$ and a comparison function ϕ. Therefore, we could hypothetically extend the formulation as applied to Page Likes to use multiple linguistic and behavioral cues when attempting to find near bipartite cores in settings with more metadata.

Since the CopyCatch method was published in [33], the ideas have been extended to detect fraud on Instagram [42] as well as on YouTube [143].

4.9 Summary

In this chapter we have attacked the problem of detecting fraudulent user feedback through the context of catching lockstep behavior in Facebook Page Likes. Our main contributions are:

1. **Problem Formulation:** We offer a novel definition of suspicious behavior based only on graph structure and edge constraints.

2. **Algorithm:** We describe two new algorithms to find lockstep behavior ($[n, m, \Delta t, \rho]$-TNBC's) - one provably convergent serial algorithm and one scalable, efficient MapReduce implementation.
3. **Theoretical analysis:** We show that catching lockstep behavior limits the damage an adversary can do. By analyzing the effect of a "greedy attack" and by applying the 60 year old Zarankiewicz problem to our setting, we show that finding an optimal adversarial attack is very hard.

Finally, we experimentally demonstrate on Facebook and synthetic data that CopyCatch is scalable, generally converges, and effective in catching lockstep behavior.

Chapter 5

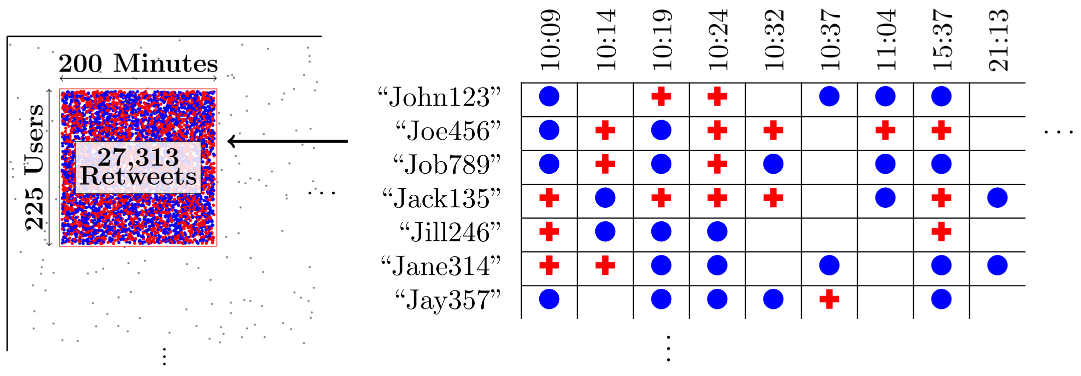
Detect Fraud in Graphs with Multiple Attributes

Which seems more suspicious: 5,000 tweets from 200 users on 5 IP addresses, or 10,000 tweets from 500 users on 500 IP addresses but all with the same hashtag and all in 10 minutes? Building on the effectiveness of incorporating time in fraud detection in Chapter 4, we focus now on scoring suspiciousness in multimodal data. The literature has many methods that try to find dense blocks in matrices, and, recently, tensors, but no method gives a principled way to score the suspiciousness of dense blocks with different numbers of modes and rank them to draw human attention accordingly. Our main contribution is that we show how to unify these methods and how to give a principled answer to questions like the above. Specifically, (a) we give a list of axioms that any metric of suspiciousness should satisfy; (b) we propose an intuitive, principled metric that satisfies the axioms, and is fast to compute; (c) we propose CrossSpot, an algorithm to spot dense regions, and sort them in importance ("suspiciousness") order. Finally, we apply CrossSpot to real data, where it improves the F1 score over previous techniques by *68%* and finds retweet-boosting and hashtag-hijacking in social datasets spanning *0.3 billion* posts.

5.1 Introduction

Imagine your job at Twitter is to detect when fraudsters are trying to manipulate the most popular tweets for a given trending topic. Given time pressure, which is more worthy of your investigation: 2,000 Twitter users, all retweeting the same 20 tweets, 4 to 6 times each; or 225 Twitter users, retweeting the same 1 tweet, 10 to 15 times each? Now, what if the latter batch of activity happened within 3 hours, while the former spanned 10 hours? What if all 225 users of the latter group used the same 2 IP addresses?

Figure 5.1 shows an example of these patterns from Tencent Weibo, one of the largest microblogging platforms in China; our method CrossSpot detected a block of 225 users, using 2 IP addresses ("blue circle" and "red cross"), retweeting the same tweet 27,313



(a) Dense block (b) Magnification

Figure 5.1: **Dense blocks in multiple modes are suspicious**. Left: A dense block of 225 users on Tencent Weibo (Chinese Twitter) retweeting one tweet 27,313 times from 2 IP addresses over 200 minutes. Right: magnification of a subset of this block. ● and ✚ indicate the two IP addresses used. Notice how synchronized the behavior is, across several modes (IP-address, user-id, timestamp).

times, within 200 minutes. Further, manual inspection shows that several of these users get activated every 5 minutes. This type of lockstep behavior is suspicious (say, due to automated scripts), and it leads to dense blocks, as in Figure 5.1. These blocks may span several modes (user-id, timestamp, hashtag, etc.). Although our main motivation is fraud detection in a Twitter-like setting, our proposed approach is suitable for numerous other settings, like distributed-denial-of-service (DDoS) attacks, link fraud, click fraud, even health-insurance fraud, as we discuss next.

Thus, the core question we ask in this chapter is: what is the right way to compare the severity, suspiciousness, or surprise of two dense blocks, that span 2 or more modes? Informally, the problem is:

Informal Problem 1 (Suspiciousness score) *Given a K-mode dataset (tensor) $\mathcal{X}$, with counts of events (that are non-negative integer values), and two subtensors $\mathcal{Y}_1$ and $\mathcal{Y}_2$, which is more suspicious and worthy of further investigation?*

Why multimodal data (tensor): Graphs and social networks have attracted huge interest, and they are often modeled as K=2 mode datasets, that is, matrices. With K=2 modes we can model Twitter's "who-follows-whom" network (Ch. 3), Facebook's "who-friends-whom" and "who-Likes-what" graphs (Ch. 4), eBay's "who-buys-from-whom" graph [171], financial activities of "who-trades-what-stocks", and scientific relations of "who-cites-whom". Several high-impact datasets make use of higher mode relations. With K=3 modes, we can consider how all of the above graphs change over time, as in Chapter 4, or what words are used in product reviews on eBay or Amazon. With K=4 modes, we can analyze network traces for intrusion detection and distributed denial of service (DDoS) attacks by looking for patterns in the source IP, destination IP, destination port, and timestamp [153, 154]. Health-insurance fraud detection is another example, with (patient-id, doctor-id, prescription-id, timestamp) where corrupt doctors prescribe fake, expensive medicine to senile or corrupt patients [101, 184].

Why are dense regions worth inspecting: Dense regions are surprising in all of the examples above.[1] Past work has repeatedly found that dense regions in these tensors correspond to suspicious, lockstep behavior: Purchased Page Likes on Facebook result in a few users "Liking" the same "Pages" always at the same time (when the order for the Page Likes is placed) (Ch. 4). Spammers paid to write deceptively high (or low) reviews for restaurants or hotels will reuse the same accounts and often even the same text [15, 161]. Zombie followers, botnets who are set up to build social links, will inflate the number of followers to make their customers seem more popular than they actually are (Ch. 3). This high-density outcome has a reason: Spammers have constrained resources (users, IP addresses, time, etc.) and they want to add as many edges to the graph/tensor as possible, to maximize their profit while minimizing their costs. Intuitively, the more synchronized the data is, in higher number of modes, the more worthy it is of further inspection.

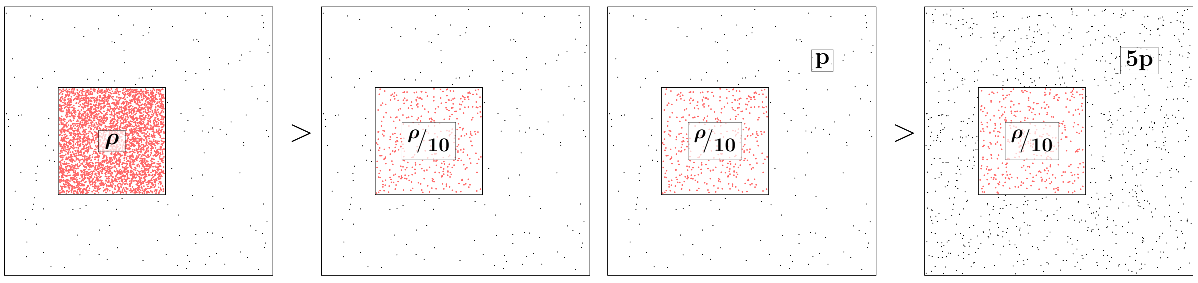
Our new perspective: There are numerous papers on finding dense subgraphs [26, 49, 179, 213], blocks and communities [20, 65, 138], including matrix algebra methods like singular value decompositions (SVD) [40, 59], tensor decompositions like multi-way decomposition methods (CANDECOMP/PARAFAC) and high-order SVD (HOSVD) [105, 119], and PageRank/TrustRank [169, 204]; several more papers apply such methods for anomaly and fraud detection [90, 154, 183]. These methods do effectively find suspicious behavior, nearly always related to dense subgraphs. However, none of them answers the problem of interest (Problem 1). The features that set this work apart are the following (also presented in Table 5.1):

- **Block score:** How would you label an individual Like on Facebook or follower on Twitter? These actions are impossible to evaluate in isolation but can be understood in the aggregate. Therefore, we focus on finding and measuring the suspiciousness of blocks of data. Other methods either return no score (like SVD/EigenSpokes, and PARAFAC/Tucker tensor decomposition) or they return a score for each node (like PageRank, TrustRank, and belief propagation), but not for the whole group. These prior methods are harder to interpret and are more easily deceived through adversarial noise.
- **Cross modes:** We look for suspicious density in all K modes, as well as any subset of the modes. In contrast, SVD and dense subgraph mining methods work only for K=2 modes; (sparse) PARAFAC, HOSVD and related tensor analysis return blocks in all modes, which we will show in Sections 5.2.3 and 5.5 has limitations.

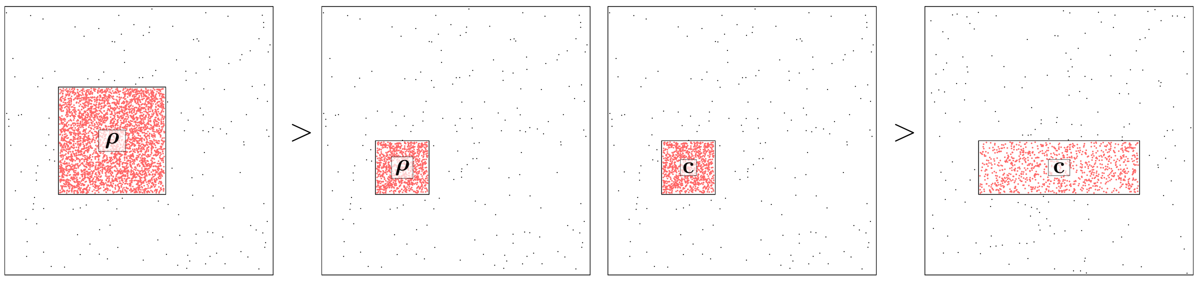
It is worthwhile to highlight our contributions in this chapter as follows:

1. **Metric criteria:** We propose a set of basic axioms that a good metric must meet to detect dense subregions in sparse multimodal data (e.g., if two blocks are the same size, the denser one is more surprising). We demonstrate that while simple, meeting all of the criteria is non-trivial.

[1] Extremely sparse regions are also surprising and worthy of inspection, but fraud cannot be the result of inaction and as such sparse regions are not the focus here.



(a) Density Axiom (b) Contrast Axiom



(c) Size Axiom (d) Concentration Axiom

Figure 5.2: **A visual representation of the axioms**: Density, Contrast, Size and Concentration. The blocks on the left are more suspicious (of higher "suspiciousness") than those on the right. ($\rho = 0.1$, $c = 1000$, $p = 0.0008$)

2. **Novel metric:** We introduce a novel suspiciousness metric to evaluate how suspicious a subvector, a submatrix or a subtensor is in multimodal data. Our metric is derived from basic probability and meets the specified criteria.
3. **The CrossSpot algorithm:** We design a scalable search algorithm to find suspicious regions of a tensor. The time complexity is quasi-linear in the number of nodes and linear in the number of non-zero entries.
4. **Validation:** Extensive experiments have demonstrated the effectiveness in catching trending-hashtag manipulation and detecting tweet promotion through retweets. We find that directly optimizing our metric significantly improves the results over just applying computationally-convenient methods like the SVD.

The remainder of this chapter is organized as follows. In Section 5.2 we propose a set of basic axioms as metric criteria and Section 5.3 presents our novel suspiciousness metric which meets all of the criteria. In Section 5.4 we develop a scalable algorithm to detect multimodal blocks with the suspiciousness metric. In Section 5.5 we report empirical results on synthetic and real-world datasets.

	Method	Scores Blocks	Axioms				
			Density 1	Size 2	Concentration 3	Contrast 4	Multimodal 5
Metrics	**SUSPICIOUSNESS**	✔	✔	✔	✔	✔	✔
	Mass	✓	✓	✗	✗	✗	✓
	Density	✓	✓	✗	✓	✗	✗
	Average Degree [47]	✓	✓	✗	✗	✗	N/A
	Singular Value [49]	✓	✓	✓	✓	✗	✗
Methods	**CrossSpot**	✔	✔	✔	✔	✔	✔
	Subgraph [49, 179, 213]	✓	✓	✓	✓	✗	N/A
	CopyCatch (Ch. 4)	✓	✓	✓	✓	✗	N/A
	CatchSync (Ch. 3)	✗			N/A		
	EigenSpokes [183]	✗			N/A		
	fBox [190]	✗			N/A		
	TrustRank [90][228]	✗			N/A		
	SybilRank [41]	✗			N/A		
	CollusionRank [79]	✗			N/A		
	BP [15, 171]	✗			N/A		

Table 5.1: **Comparison of state-of-the-art metrics and methods**. While the methods have been successful in particular applications, they do not meet the general goals set out here.

5.2 Proposed Metric Criteria

Having given the high level intuition behind our perspective and its relation to prior work, we now give a precise definition of the problem. We focus on tensors where each cell contains a non-negative integer, typically representing counts of events. We consider the mass of a subtensor to be the sum of entries in that subtensor, and the density to be the mass divided by the volume of the subtensor. A full list of our notation can be found in Table 5.2.

Symbol	Definition
K	Number of modes in our dataset
$\mathcal{X}$	K-mode tensor dataset
$\mathcal{Y}$	Subtensor within $\mathcal{X}$
$\mathbf{N}$	K-length vector for the size of each mode of $\mathcal{X}$
C	The mass of $\mathcal{X}$ (summing the entries of $\mathcal{X}$)
$\mathbf{n}$	K-length vector for the size of each mode of $\mathcal{Y}$
c	The mass of $\mathcal{Y}$
p	The density, $C/\prod_k N_k$ of $\mathcal{X}$
ρ	The density, $c/\prod_k n_k$, of $\mathcal{Y}$
f	Suspiciousness metric, parameterized by the masses
$\hat{f}$	Suspiciousness metric, parameterized by the densities
$D_{KL}(\rho\|p)$	Directed KL-divergence of Poisson(p) & Poisson(ρ) $p - \rho + \rho \log \frac{\rho}{p}$

Table 5.2: The notation used throughout this chapter.

5.2.1 Problem Formulation

Now we can formally give the definition of the problem of evaluating the suspiciousness of suspicious behaviors - mathematically, giving a suspiciousness score of dense blocks in multimodal data.

Formal Problem 1 (Suspiciousness score) ***Given*** *a K-mode tensor $\mathcal{X}$ with non-negative entries, of size $\mathbf{N} = [N_k]_{k=1}^K$ and with mass C (describing C events by summing entries of the tensor),* ***define*** *a score function $f(\mathbf{n}, c, \mathbf{N}, C)$ for how suspicious a subtensor $\mathcal{Y}$ of size $\mathbf{n} = [n_k]_{k=1}^K$ with mass c.*

We consider an alternative parameterization using density. Here ρ is the density of $\mathcal{Y}$ and p is the density of $\mathcal{X}$:

$$\hat{f}(\mathbf{n}, \rho, \mathbf{N}, p) = f\left(\mathbf{n}, \rho \prod_{k=1}^{K} n_k, \mathbf{N}, p \prod_{k=1}^{K} N_k\right)$$

In the rare case that the number of modes being considered is unclear, we will refer to the functions by f_K and $\hat{f}_K$.

Note that we restrict f to only focus on blocks for which $\rho > p$, that is the density inside the block is greater than the density in the general tensor. While extremely sparse regions are also unusual, they are not the focus of this work.

5.2.2 Axioms

We now list five basic axioms that any suspiciousness metric f must meet. A pictorial representation can be found in Figure 5.2.

Axiom 1 ***Density*** *If there are two blocks of the same size in the same number of modes, the block of bigger mass is more suspicious than the block of less mass. Formally,*

$$c_1 > c_2 \iff f(\mathbf{n}, c_1, \mathbf{N}, C) > f(\mathbf{n}, c_2, \mathbf{N}, C)$$

Axiom 2 ***Size*** *If there are two blocks of the same density in the same number of modes, the bigger block is more suspicious than smaller block. Formally,*

$$n_j > n'_j \wedge n_k \geq n'_k \;\forall k \implies \hat{f}\,(\mathbf{n}, \rho, \mathbf{N}, p) > \hat{f}\,(\mathbf{n}', \rho, \mathbf{N}, p)$$

Axiom 3 ***Concentration*** *If there are two blocks of the same mass in the same number of modes, the smaller block is more suspicious than bigger block. Formally,*

$$n_j < n'_j \wedge n_k \leq n'_k \;\forall k \implies f(\mathbf{n}, c, \mathbf{N}, C) > f(\mathbf{n}', c, \mathbf{N}, C)$$

Axiom 4 ***Contrast*** *If two identical blocks lie in two tensors each of the same size but one is sparser, then the block in the sparser tensor is more suspicious. Formally,*

$$p_1 < p_2 \iff \hat{f}(\mathbf{n}, \rho, \mathbf{N}, p_1) > \hat{f}(\mathbf{n}, \rho, \mathbf{N}, p_2)$$

Axiom 5 ***Multimodal*** *A block which contains all possible values within a mode is just as suspicious as if that mode was ignored (was collapsed*[2] *into the remaining modes). Formally,*

$$f_{K-1}\big([n_k]_{k=1}^{K-1}, c, [N_k]_{k=1}^{K-1}, C\big) = f_K\left(([n_k]_{k=1}^{K-1}, N_K), c, [N_k]_{k=1}^{K}, C\right)$$

Lemma 4 ***Cross-mode comparisons*** *Learning of a new mode about our data can only make blocks in that data more suspicious. Formally,*

$$f_{K-1}\big([n_k]_{k=1}^{K-1}, c, [N_k]_{k=1}^{K-1}, C\big) \leq f_K\big([n_k]_{k=1}^{K}, c, [N_k]_{k=1}^{K}, C\big)$$

Proof 5

$$\begin{aligned} & f_{K-1}\left([n_k]_{k=1}^{K-1}, c, [N_k]_{k=1}^{K-1}, C\right) \\ = \; & f_K\left(([n_k]_{k=1}^{k-1}, N_K), c, [N_k]_{k=1}^{K}, C\right) \\ \leq \; & f_K\left(([n_k]_{k=1}^{k-1}, n_K), c, [N_k]_{k=1}^{K}, C\right) \end{aligned}$$

Above we find that the first equality is given by Axiom 5 and the second by Axiom 3.

[2]*Collapsing* a tensor $\mathcal{X}$ on mode K sums the values of $\mathcal{X}$ across all indices in mode K [106], e.g., collapsing a tensor to a matrix: $\mathbf{X}_{i,j} = \sum_k \mathcal{X}_{i,j,k}$.

5.2.3 Shortcomings of Competitors

While these axioms are simple and intuitive, they are non-trivial to meet. As shown in Table 5.1, simple metrics fail a number of the axioms.

Mass: One possible metric is the mass $f(\mathbf{n}, c, \mathbf{N}, C) = c$. This does not change if the same mass is concentrated in a smaller region, and hence fails Axiom 3 (Concentration); it does not consider the background density p, and so fails Axiom 4 (Contrast) as well.

Density: Another possible metric is the density of the block $\hat{f}(\mathbf{n}, \rho, \mathbf{N}, p) = \rho$. However, this does not consider the size of the dense block, and hence fails Axiom 2 (Size). It also does not consider the background density, and fails Axiom 4 (Contrast). Since density in general decreases with more modes, Axiom 5 (Multimodal) is also broken.

Average degree: Much of the research on finding dense subgraphs focuses on the average degree of the subgraph [23][20], $f(\mathbf{n}, c, \mathbf{N}, C) = c/n_1$. This metric breaks both Axioms 2 and 3 by not considering n_2 and breaks Axiom 4 by not considering C and $\mathbf{N}$. Additionally it is unclear how we would define the average degree for $K > 2$, making it unsuitable for multimodal data.

Singular value (in SVD): The SVD of a matrix $\mathbf{A}$ is a factorization of the form $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. The singular values of $\mathbf{A}$ correspond to $\mathbf{\Sigma}_{r,r}$, and $\mathbf{U}, \mathbf{V}$ are the singular vectors. The top singular values and vectors indicate big, dense blocks/clusters in the multimodal data and have been used to find suspicious behavior [105][183]. As shown in [190], an independent block of size $n_1 \times n_2$ with mass c has a singular value σ corresponding to that block of $\sigma = \frac{c}{\sqrt{n_1 n_2}} = \sqrt{\rho c}$. Given the SVD prioritizes the parts of the data with higher singular values, we can view this as a competing metric of suspiciousness. While this metric now meets Axioms 1 through 3, it has a challenge generalizing. First, it is clear that this metric ignores the density of the background data. As a result, Axiom 4 is broken. Second, how can we extend this metric to more modes? HOSVD does not have the same provable guarantees as SVD and thus does not necessarily find the largest, densest blocks. Even if we consider density in higher modes, what we find is that with each additional mode added, the volume of a block becomes greater and thus the density lower. This breaks Axiom 5 and would make an algorithm collapse all data down to one mode rather than consider the correlation across all K modes. Later, we will observe in proposed metric definition (Section 5.3) and experiments (Section 5.5) the drawbacks of the singular value in higher modes.

From the above, we see that methods building on average degree and SVD meet the requirements for many cases, but break down on certain corner cases, limiting their path toward a general approach to finding surprising/suspicious behavior. We now offer our approach and demonstrate its effectiveness across all of these challenges.

5.3 Proposed Suspiciousness Metric

Our metric is based on a model of the data in which the C events are randomly distributed across the tensor data $\mathcal{X}$. For binary data this corresponds to a multimodal Erdös-Rényi model [166], where the value in each cell follows a binomial distribution. Because each cell in the tensor can contain more than one occurrence, we instead use a Poisson distribution, resulting in the Erdös-Rényi-Poisson model:

Definition 4 ***Erdös-Rényi-Poisson (ERP) model*** *A tensor $\mathcal{X}$ generated by the ERP model, has each value in the tensor sampled from a Poisson distribution parameterized by p.*

$$\mathcal{X}_{\mathbf{i}} \sim \text{Poisson}(p)$$

In general, we set p to be the density of the overall tensor. Using this model we define our metric:

Definition 5 ***The suspiciousness metric*** *The suspiciousness score of a multimodal block is the negative log likelihood of block's mass under an Erdös-Rényi-Poisson model. Mathematically, given an $n_1 \times \cdots \times n_K$ block of mass c in $N_1 \times \cdots \times N_K$ data of total mass C, the suspiciousness score is*

$$\boxed{f(\mathbf{n}, c, \mathbf{N}, C) = -\log\left[Pr(Y_n = c)\right]} \tag{5.1}$$

where Y_n is the sum of entries in the block.

5.3.1 Dense Subvector and Submatrix: 1-Mode and 2-Mode Suspiciousness

Consider an N-length vector $\mathbf{X}$, which we believe to be generated by the ERP model defined above. We can think of this vector as the number of tweets per IP address. If there are C tweets total, then the density is $p=\frac{C}{N}$ and each X_i has a Poisson distribution:

$$\Pr(X_i|p) = \frac{p^{X_i}}{X_i!}e^{-p}$$

We are searching for an n-length subvector $X_{i_1}, \ldots, X_{i_n}$ that is unlikely and hence has a high suspiciousness score.

Lemma 5 *The suspiciousness of an n-length subvector $[X_{i_1}, \ldots, X_{i_n}]$ in the N-length vector data $[X_1, \ldots, X_N]$ is*

$$f(n, c, N, C) = c\big(\log \tfrac{c}{C} - 1\big) + C\tfrac{n}{N} - c\log \tfrac{n}{N}$$
$$\hat{f}(n, \rho, N, p) = n\Big(p - \rho + \rho \log \tfrac{\rho}{p}\Big) = nD_{KL}(\rho||p)$$

Here $c = \sum_{j=1}^{n} X_{i_j}$ and $D_{KL}(\rho||p)$ is the Kullback-Leibler (KL) divergence of $\text{Poisson}(p)$ from $\text{Poisson}(\rho)$.

Proof 6 *We denote the sum of n variables by $Y_n = \sum_{j=1}^{n} X_{i_j}$. From the Poisson property, we know $Y_n \sim \text{Poisson}(pn)$. The probability that Y_n equals a given number of retweets c is*

$$Pr(Y_n = c) = \frac{(pn)^c e^{-pn}}{c!} = \frac{C^c}{c!}\left(\frac{n}{N}\right)^c e^{-\frac{Cn}{N}}$$

Since the approximation for factorials (Stirling's formula) is

$$\log(c!) \quad = \quad c \log c - c + \mathcal{O}(\log c),$$

we obtain the suspiciousness score:

$$\begin{aligned} f(n, c, N, C) = -\log\left[Pr(Y_n = c)\right] = -\log\left[\frac{C^c}{c!}\left(\frac{n}{N}\right)^c e^{-\frac{Cn}{N}}\right] \\ \approx c\Big(\log\frac{c}{C} - 1\Big) + C\frac{n}{N} - c\log\frac{n}{N}. \end{aligned}$$

We now extend suspiciousness to a 2-mode matrix.

Lemma 6 *The suspiciousness of an $n_1 \times n_2$ block of mass c in $N_1 \times N_2$ data of total mass C is:*

$$\begin{aligned} f([n_1, n_2], c, [N_1, N_2], C) &= c\Big(\log\tfrac{c}{C} - 1\Big) + C\tfrac{n_1 n_2}{N_1 N_2} - c\log\tfrac{n_1 n_2}{N_1 N_2} \\ \hat{f}([n_1, n_2], \rho, [N_1, N_2], p) &= n_1 n_2 D_{KL}(\rho || p) \end{aligned}$$

5.3.2 Dense Subtensor: K-Mode Suspiciousness

We now extend suspiciousness to a K-mode tensors.

Lemma 7 *Given an $n_1 \times \cdots \times n_K$ block of mass c in $N_1 \times \cdots \times N_K$ data of total mass C, the suspiciousness function is*

$$\boxed{f(\mathbf{n}, c, \mathbf{N}, C) = c(\log\frac{c}{C} - 1) + C\prod_{i=1}^{K}\frac{n_i}{N_i} - c\sum_{i=1}^{K}\log\frac{n_i}{N_i}} \tag{5.2}$$

Using ρ as the block's density and p is the data's density, we have the simpler formulation

$$\boxed{\hat{f}(\mathbf{n}, \rho, \mathbf{N}, p) = \left(\prod_{i=1}^{K} n_i\right) D_{KL}(\rho || p)} \tag{5.3}$$

From the non-negativity of KL divergence, we have $f = \hat{f} \geq 0$.

5.3.3 Proofs: Satisfying the Axioms

Now that we have defined our suspiciousness metric, we prove that it meets all of the desired axioms proposed in Section 5.2.

Axiom 1: **Density**

Proof 7 *Using Eq. (5.2), the derivative of the suspiciousness function with respect to the block's mass c is*[3]

$$\frac{\mathrm{d}\hat{f}}{\mathrm{d}c} = \log\frac{c}{C} + \log\left(\prod_{i=1}^{K}\frac{N_i}{n_i}\right) = \log\frac{\rho}{p}$$

since $p = \frac{C}{\prod_{i=1}^{K} N_i}$ and $\rho = \frac{c}{\prod_{i=1}^{K} n_i}$. We are only considering blocks with higher density than the overall data, i.e. $\rho > p$, so $\frac{\mathrm{d}\hat{f}}{\mathrm{d}c} > 0$, i.e. suspiciousness increases with density.

Axiom 2: **Size**

Proof 8 *Using Eq. (5.3), fixing n_k for $k \neq j$, the derivative of the suspiciousness function with respect to n_j is:*

$$\frac{\mathrm{d}\hat{f}}{\mathrm{d}n_j} = \left(\prod_{k\neq j} n_k\right) D_{KL}(\rho \| p) > 0$$

Thus, for fixed density ρ, as we increase any one dimension of the block with the remaining dimensions kept fixed, suspiciousness increases.

Axiom 3: **Concentration**

Proof 9 *Using Eq. (5.2), fixing n_k for $k \neq j$, the derivative of the suspiciousness function with respect to n_j is:*

$$\frac{\mathrm{d}\hat{f}}{\mathrm{d}n_j} = \frac{C}{n_j}\prod_{i=1}^{K}\frac{n_i}{N_i} - \frac{c}{n_j} = \frac{c}{n_j}\left(\frac{p}{\rho} - 1\right)$$

The last expression is negative since $\rho > p$. Thus, for a fixed mass, larger blocks are less suspicious.

Axiom 4: **Contrast**

Proof 10 *Using Eq. (5.3), the derivative of suspiciousness with respect to the data density p is*

$$\frac{\mathrm{d}\hat{f}}{\mathrm{d}p} = \left(\prod_{k=1}^{K} n_i\right)\left(1 - \frac{\rho}{p}\right)$$

Since $\rho > p$, we have $\frac{\mathrm{d}\hat{f}}{\mathrm{d}p} < 0$, so as the overall matrix gets denser, the block gets less suspicious.

Axiom 5: **Multimodal**

Proof 11 *Using Eq. (5.2):*

$$\begin{aligned}
& f_K\left(([n_k]_{k=1}^{K-1}, N_K), c, [N_k]_{k=1}^{K}, C\right) \\
&= c\left(\log\tfrac{c}{C} - 1\right) + C\tfrac{N_K}{N_K}\prod_{i=1}^{K-1}\tfrac{n_i}{N_i} - c\left(\log\tfrac{N_K}{N_K} + \sum_{i=1}^{K-1}\log\tfrac{n_i}{N_i}\right) \\
&= c\left(\log\tfrac{c}{C} - 1\right) + C\prod_{i=1}^{K-1}\tfrac{n_i}{N_i} - c\sum_{i=1}^{K-1}\log\tfrac{n_i}{N_i} \\
&= f_{K-1}\left([n_k]_{k=1}^{K-1}, c, [N_k]_{k=1}^{K-1}, C\right)
\end{aligned}$$

[3]Formally, to take derivatives with respect to a discrete variable such as c, we extend Eq. (5.2) to take in c as real numbered input, then differentiate it with respect to c to show the desired monotonicity property. Then, the original, integer version of Eq. (5.2) agrees with the real numbered version whenever c is an integer, proving the desired monotonicity property for Eq. (5.2).

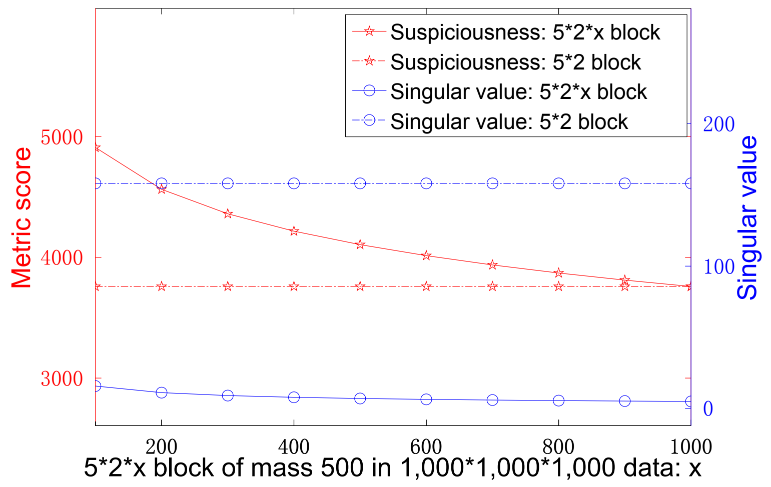


Figure 5.3: **Cross-mode comparisons**: Our suspiciousness metric obeys all the axioms, but the singular value breaks Axiom 5 (Multimodal).

After the extensive proofs, we give an example to highlight the advantages of our "suspiciousness" over the singular value.

Example 1 *Given a 3-mode tensor data of mass 10,000 and size 1,000×1,000×1,000, suppose that we spot a 3-mode dense block of mass 500 and size 5×2×x; collapsing the 3rd mode of the 3-mode tensor, or directly given a 2-mode data of mass 10,000 and size 1,000×1,000, suppose that we spot a 2-mode dense block of mass 500 and size 5×2. Figure 5.3 compares the suspiciousness scores and singular values of these dense blocks. Two observations are as follows.*

- *With the value of x decreasing, both metric scores of the 3-mode blocks increase. So, both the "suspiciousness" and the singular value obey Axiom 3 (Concentration).*
- *When x<1,000, the suspiciousness of the higher-mode block is bigger than the lower-mode one, however, the singular value of the lower-mode block is bigger. So, the "suspiciousness" obeys Axiom 5 (Multimodal) and the singular value breaks it.*

5.4 Suspicious Block Detection

Having defined a metric for measuring the suspiciousness of a block, in this section we formally define the problem of detecting suspicious blocks across modes, and give a scalable algorithm based on our proposed metric to identify the blocks.

5.4.1 Problem Definition

Now we can formally give the definition of the problem of detecting suspicious behaviors - mathematically, detecting dense blocks in multimodal data.

Problem 1 (Suspicious block detection) ***Given*** *dataset $\mathcal{X}$ which is a $N_1 \times \cdots \times N_K$ tensor of mass C,* ***find*** *a list of blocks in $\mathcal{X}$, in any subset of modes, with high suspiciousness scores, in descending order, based on Eq.* (5.2) *and* (5.4).

As before, we have a K-mode tensor $\mathcal{X}$ and a k-mode subtensor $\mathcal{Y}$ to represent the suspicious block. Mode j of the tensor has N_j possible values: $\mathcal{P}_j = \{p_1^{(j)}, \ldots, p_{N_j}^{(j)}\}$. Subtensor $\mathcal{Y}$ covers a subset of values in each mode: $\tilde{\mathcal{P}}_j \subseteq \mathcal{P}_j, \forall j$. Define $\tilde{\mathcal{P}} = \{\tilde{\mathcal{P}}_j\}_{j=1}^K$. Let $c(\tilde{\mathcal{P}})$ be the number of events in the subtensor defined by $\tilde{\mathcal{P}}$.

The dimensions of our block $\mathbf{n}$ are $n_j = |\tilde{\mathcal{P}}_j|$. If a mode j is not included, we consider $\tilde{\mathcal{P}}_j = \mathcal{P}_j$, based on Axiom 5 and the properties of collapse operation. For the sake of notational simplicity we define one last alternative parameterization for our suspiciousness function

$$\tilde{f}(\tilde{\mathcal{P}}, \mathcal{D}) = f([|\tilde{\mathcal{P}}_j|]_{j=1}^K, c(\tilde{\mathcal{P}}), [|\mathcal{P}_j|]_{j=1}^K, |\mathcal{X}|) \tag{5.4}$$

5.4.2 Proposed Algorithm CrossSpot

We define here a local search algorithm to search for suspicious blocks in the dataset. We start with a seed suspicious block, then perform an iterative alternating optimization, where we find the optimal set of values in mode j while holding constant the included values in all other modes. We run this sequence of updates until convergence. The complete algorithm is shown in Algorithm 6.

Algorithm 6: Local Search

```
Require: Data $\mathcal{X}$, seed region $\mathcal{Y}$ with $\tilde{\mathcal{P}} = \{\tilde{\mathcal{P}}_j\}_{j=1}^K$
1: while not converged do
2:   for $j = 1 \ldots K$ do
3:     $\tilde{\mathcal{P}}_j \leftarrow$ ADJUSTMODE($j$)
4:   end for
5: end while
6: return $\tilde{\mathcal{P}}$
```

Adjusting a Mode: During each iteration of ADJUSTMODE, we optimally choose a subset of values from $\mathcal{P}_j$ holding constant the values in other modes, i.e. fixing $\tilde{\mathcal{P}}_{j'}$ for $j' \neq j$. Denote $\Delta c_{p_i^{(j)}}$ as the number of events in the intersection of row i (in mode j) and the currently fixed values in the other modes, i.e. $\tilde{\mathcal{P}}_{j'}$ for $j' \neq j$. We refer to $\Delta c_{p_i^{(j)}}$ as the "benefit" of $p_i^{(j)}$. In Algorithm 7 we use these benefit scores to order the values in $\mathcal{P}_j$, from greatest to least benefit. We will refer to this ordered list as $\mathbf{P}_j$.

Lemma 8 *Holding constant $\tilde{\mathcal{P}}_{j'}$ for all $j' \neq j$, the optimal choice of values $\tilde{\mathcal{P}}_j \subseteq \mathcal{P}_j$ is the first n_j values in $\mathbf{P}_j$ for some $n_j \leq N_j$.*

Algorithm 7: ADJUSTMODE(j)

```
1: $\tilde{\mathcal{P}}'_j \leftarrow \{\}$;
2: $\mathbf{P}_j \leftarrow \{p_i^{(j)}\}_{i=1}^{N_j}$ sorted in descending order by $\Delta c_{p_i^{(j)}}$
3: for $p_i^{(j)} \in \mathbf{P}_j$ do
4:     $\tilde{\mathcal{P}}'_j \leftarrow \tilde{\mathcal{P}}'_j \cup p_i^{(j)}$
5:     $\tilde{\mathcal{P}}' \leftarrow \{\tilde{\mathcal{P}}_{j'}\}_{j' \neq j} \cup \tilde{\mathcal{P}}'_j$
6:     if $\tilde{f}(\tilde{\mathcal{P}}, \mathcal{D}) \leq \tilde{f}(\tilde{\mathcal{P}}', \mathcal{D})$ then
7:         $\tilde{\mathcal{P}}_j \leftarrow \tilde{\mathcal{P}}'_j$
8:     end if
9: end for
10: return $\tilde{\mathcal{P}}_j$
```

Dataset	Mode #1	Mode #2	Mode #3	Mode # 4	Mass
Retweeting	User id 29,468,040	Tweet id 19,755,875	IP address 27,817,611	Time (min.) 56,943	Retweet 221,719,535
Tweeting hashtag	User id 81,186,369	Hashtag 1,580,042	IP address 47,717,882	Time (min.) 56,943	Tweet 276,944,456
Network traffic	Source IP 2,345	Destination IP 2,355	Port number 6,055	Time (sec.) 3,610	Packet 230,836

Table 5.3: **Data statistics**: multimodal datasets from social networks and network traffic.

Proof 12 *We prove this by contradiction. Assume there is a subset $\tilde{\mathcal{P}}_j \subseteq \mathcal{P}_j$ that we believe to be the optimal choice of values but that $\tilde{\mathcal{P}}_j$ is not the first $|\tilde{\mathcal{P}}_j|$ values of $\mathbf{P}_j$. Therefore, there must exist a pair of values $p_i^{(j)}, p_{i'}^{(j)}$ where $p_i^{(j)} \in \tilde{\mathcal{P}}_j$ and $p_{i'}^{(j)} \notin \tilde{\mathcal{P}}_j$ but $\Delta c_{p_{i'}^{(j)}} > \Delta c_{p_i^{(j)}}$. By Axiom 1, it is clear that removing $p_i^{(j)}$ and adding $p_{i'}^{(j)}$ to $\tilde{\mathcal{P}}_j$ results in a block with a higher suspiciousness score than the original, supposedly optimal block. From this contradiction, the optimal set of values for $\tilde{\mathcal{P}}_j$ must come from the top of $\mathbf{P}_j$.*

Theorem 1 *Holding constant $\tilde{\mathcal{P}}_{j'}$ for all $j' \neq j$, the $\tilde{\mathcal{P}}_j$ that maximizes $f(\mathbf{n}, c, \mathbf{N}, C)$ is found by* ADJUSTMODE(j) *in Algorithm 6.*

Proof 13 *Because* ADJUSTMODE *sorts $\mathcal{P}_j$ and checks all possible values of n_j for mode j, Lemma 8 implies that* ADJUSTMODE *makes the optimal choice of values in each step.*

Seeds: In Algorithm 6, we start from a seed subtensor $\mathcal{Y}$. In the simplest case, we start from a randomly chosen seed, containing an individual cell of the tensor or a larger randomly chosen block. As we will show in Section 5.5, even using randomly chosen seeds does well.

This starting point offers significant flexibility for CrossSpot to benefit from the findings of previous data mining work and side information. For example, we can use as a seed the dense regions found in each rank of a singular value decomposition. Searching with multiple seeds is trivially parallelizable, so with more computational resources we can always choose additional random seeds or use additional prior methods as starting points for CrossSpot.

Complexity: The time complexity of Algorithm 6 is $\mathcal{O}(T \times K \times (E + N \log N))$, where T is the number of iterations, K is the number of modes, E is the number of non-zero entries in the data, and $N = \max_j N_j$ is the maximum size of any mode. Because T and K are often set to constant values, the complexity is quasi-linear in N and linear in the number of non-zero entries. Thus, Algorithm 6 is scalable for real applications to catch suspicious behavior.

Convergence Guarantees: Our algorithm converges to a local optimum: by Theorem 1, we find that each time AdjustMode is run, the value of $\tilde{f}(\tilde{\mathcal{P}}, \mathcal{X})$ improves or stays constant. As such, since there are a finite number of possible subtensors and hence a finite number of possible (non-infinite) values of the objective, the algorithm must converge.

5.5 Experiments

In this section, we conduct experiments to answer the following questions: (1) How effective is the proposed method CrossSpot in finding suspicious blocks? (2) Can CrossSpot discover suspicious behavioral patterns in real datasets? (3) How efficient is CrossSpot? The experimental results show that CrossSpot detects suspicious blocks more accurately and is more computationally efficient than competing baselines. We also use CrossSpot to identify large, dense blocks in a retweeting dataset, a hashtag promoting dataset and a network traffic dataset, and use side information to show that suspicious behavior is indeed identified.

5.5.1 Datasets

In our experiments we used extensive datasets including synthetically generated datasets, two large, new social networking datasets and a public network traffic dataset. A summary of the datasets can be found in Table 5.3.

Synthetic data: We adapt the Erdös-Rényi-Poisson model to generate multimodal data. The synthetic data is generated as a K-mode tensor of size $N_1 \times \cdots \times N_K$ with mass C. Within the tensor we inject b dense blocks. Each block is assigned a size $n_1 \times \cdots \times n_K$ and mass c. When an injected block falls in only a subset of modes $\mathcal{I}$, we set $n_i = N_i$.

Retweeting data: We use retweeting data from Tencent Weibo, one of the largest social networking platforms in China. These retweets consist of user id, tweet id, IP address, timestamp (from November 9 to December 20 in 2011) and retweeting comment. On Weibo, retweet boosting is common, where retweets can be purchased to make a particular tweet seem more popular than it actually is. This results in a distorted user experience.

Tweeting hashtag data: As well as retweeting data, we use original tweets from Tencent Weibo that include hashtags in their content. The dataset consists of tuples of user id, hashtag, IP address, timestamp and tweet content. This dataset is interesting for hashtag hijacking and hashtag promotion, where purchased tweets will use popular hashtags to promote their own content or will tweet many times using a hashtag in an attempt to make it trend. By searching for dense, multimodal behavior, we hope to spot suspicious patterns evident of hashtag hijacking.

Network traffic data: The network traffic log is public through a research effort to study internet traffic of enterprises [172]. The data of thousands of packets was collected on servers within the Lawrence Berkeley National Lab (LBNL). Each packet trace includes source IP, destination IP, port number and a timestamp in seconds. We look for dense structures.

5.5.2 Experimental Setup

In this subsection, we introduce how we set up our experiments: baseline methods, parameter settings and evaluation methods.

Baselines: We compare our proposed method CrossSpot with the following baseline methods. All the methods utilize structured behavioral information in different ways.

- SVD and HOSVD (Higher-Order SVD) [59][183] compute the orthonormal spaces associated with the different modes of a tensor. The threshold value for partitioning the decomposition vector is adaptively determined [183].
- MAF (MultiAspectForensics) [154] looks for spikes indicating the high-order subtensor (representing dense bipartite-core pattern) with eigenscore histogram vector and threshold parameters.
- AvgDeg (Dense Graph Components) [47] defines average degree as a metric of dense subgraph and develops a greedy algorithm to find the dense components.

Parameter settings: We look for the best performance of every method. When running CrossSpot, we generate 1,000 random seeds to find their final blocks. We randomly decide the modes of a seed block and the set of values on each mode. We implement CrossSpot in Python. For the sake of efficiency in Algorithm 7 we prune out sparse values in each mode by stopping early if line 6 returns false. For SVD and HOSVD, we compare with different decomposition ranks such as 5, 10 and 20. We vary the threshold from 0 to 1 for every singular vector, considering rows (or columns) to be included in

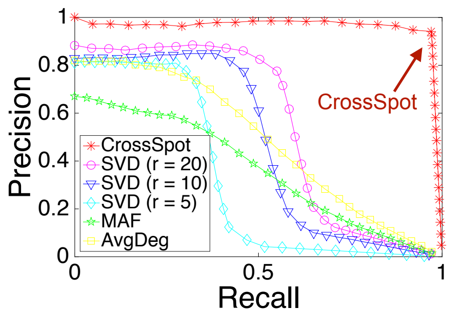


Figure 5.4: **Finding dense subgraphs**: CrossSpot has nearly perfect performance on catching injected 2-mode blocks.

the block if their value in the singular vector is greater than the threshold. For other baselines, we use their standard implementations. We perform the experiments on a 2.40GHz$\times$8 Intel Xeon CPU with 64GB RAM, running Windows Server 2008-64 bit.

Evaluation methods: To assess the effectiveness of our detection strategy in classifying suspicious and normal behaviors we use the standard information retrieval metrics of recall, precision and F1 score [233]. The recall is the ratio of the number of behaviors correctly classified to the number of suspicious behaviors. The precision is the ratio of the number of behaviors classified correctly to the total predicted suspicious behaviors. The F1 score is the harmonic mean of precision and recall.

5.5.3 Synthetic Experiments

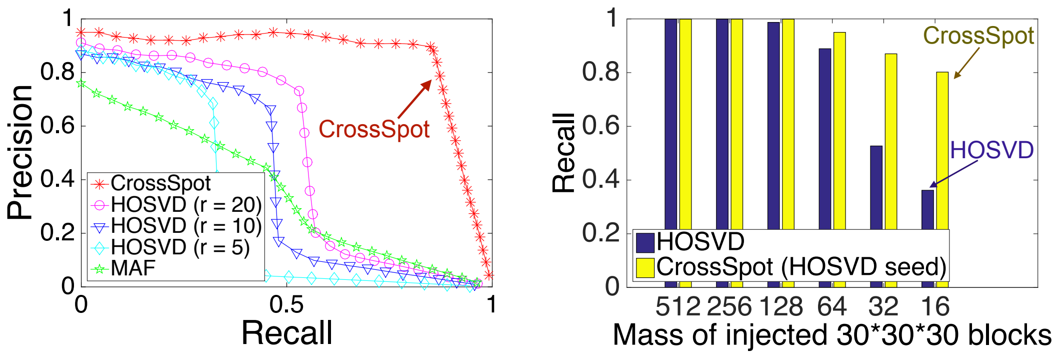
We first evaluate CrossSpot on synthetic datasets. Overall, CrossSpot is effective: it detects dense subgraphs in 2-mode data, dense k-mode blocks in k-mode tensor data, and even dense k'-mode blocks in k-mode tensor data ($k' < k$) with very high precision and recall. It is also efficient: it has faster execution time than complex traditional methods.

Effectiveness evaluations: We test the effectiveness of our proposed method CrossSpot in three tasks of finding suspicious behaviors in synthetic datasets represented by big, dense blocks in multimodal generated data, as well as the robustness of random seed number.

Finding dense subgraphs (2-mode blocks): We generate a random matrix under the ERP model with parameters as (1) the number of modes k=2, (2) the size of data $N_1 = 1000$ and $N_2 = 1000$, and (3) the mass of data C=10,000. We inject $b = 6$ blocks of $k' = 2$ modes into the random data, so, $\mathcal{I} = \{1, 2\}$. The size of every block is 30$\times$30 and the block's mass

	Recall				Overall Evaluation		
	Block #1	Block #2	Block #3	Block #4	Precision	Recall	F1 score
HOSVD (r=20)	93.7%	29.5%	23.7%	21.3%	**0.983**	0.407	0.576
HOSVD (r=10)	91.3%	24.4%	18.5%	19.2%	0.972	0.317	0.478
HOSVD (r=5)	85.7%	10.0%	9.5%	11.4%	0.952	0.195	0.324
CrossSpot	**100%**	**99.9%**	**94.9%**	**95.4%**	0.978	**0.967**	**0.972**

Table 5.4: **CrossSpot catches more lower-mode blocks**: CrossSpot has high accuracy in finding the injected 4 blocks: (1) 30×30×30, (2) 30×30×1,000, (3) 30×1,000×30, and (4) 1,000×30×30, each of which has mass 1,000.



(a) Performance of dense block detection (b) Recall with HOSVD seed

Figure 5.5: **Finding dense blocks**: CrossSpot outperforms baselines in finding 3-mode blocks, and directly method improves the recall on top of HOSVD.

$c \in \{16, 32, 64, 128, 256, 512\}$. The task is to classify all the data entries into suspicious (injected) and normal classes. Figure 5.4 reports the classification performances of our proposed CrossSpot and the baselines of finding dense subgraphs. We observe that

- CrossSpot has nearly perfect precision: it only includes the entries that increase the suspiciousness because they belong to the dense blocks. It also has perfect recall: the local search does not miss any values in the block's modes. The highest F1 of CrossSpot is 0.967, while the highest F1 scores of SVD, MAF, AVGDEG are 0.634, 0.439, and 0.511. MAF catches a big number of similar objects on some mode and thus it can catch very large blocks. AVGDEG catches very dense blocks but it will miss larger, less dense blocks.
- SVD has small recall but high precision. However, the SVD can hardy catch small, sparse injected blocks such as 30×30 submatrices of mass 16 and 32, even though they are denser than the background. Higher decomposition rank brings higher classification accuracy.

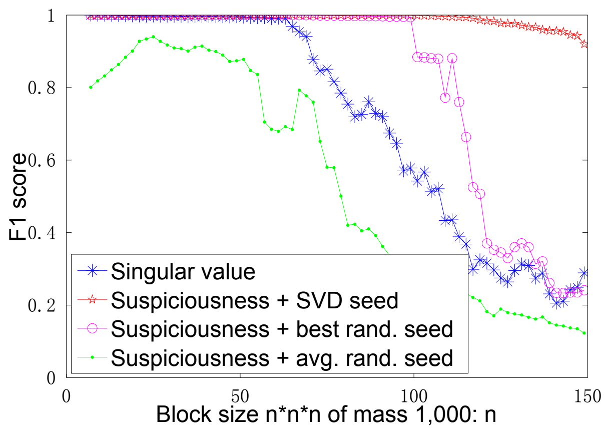


Figure 5.6: Performance comparisons between different versions of CrossSpot and SVD. CrossSpot performs the best with SVD seeds.

Finding dense high-order blocks in multimodal data: We generate random tensor data with parameters as (1) the number of modes k=3, (2) the size of data N_1=1,000, N_2=1,000 and N_3=1,000 and (3) the mass of data C=10,000. We inject b=6 blocks of k'=3 modes into the random data, so, $\mathcal{I} = \{1, 2, 3\}$. Each block has size 30×30×30 and mass $c \in \{16, 32, 64, 128, 256, 512\}$. The task is again to classify the tensor entries into suspicious and normal classes. Figure 5.5(a) reports the performances of CrossSpot and baselines. We observe that in order to find all the six 3-mode injected blocks, our proposed CrossSpot has better performance in precision and recall than baselines. The best F1 score CrossSpot gives is 0.891, which is 46.0% higher than the F1 score given by the best of HOSVD (0.610). If we use the results of HOSVD as seeds to CrossSpot, the best F1 score of CrossSpot reaches 0.979. Figure 5.5(b) gives the recall value of every injected block. We observe that CrossSpot improves the recall over HOSVD, especially on slightly sparser blocks.

Figure 5.6 shows the performances of dense block detection: The different versions of CrossSpot include the performance with SVD seeds, the best performance with random seeds, and the average performance with random seeds. We inject a 3-mode dense block of mass 1,000 and size $n \times n \times n$ into the random tensor data of mass 10,000 and size 1,000×1,000×1,000. We observe that

- CrossSpot performs better than high-order SVD when it uses SVD seeds or the best of random seeds.
- When the blocks become bigger, the F1 scores decrease, because the density of the dense blocks are smaller.
- CrossSpot with SVD seeds can perform almost perfectly: F1 scores are consistently more than 0.90.

Finding dense low-order blocks in multimodal data: We generate random tensor data with parameters as (1) the number of modes k=3, (2) the size of data N_1=1,000,

	Rank #	User×Tweet×IP×Minute	Mass c	Suspiciousness score
CrossSpot	1	14×1×2×1,114	41,396	1,239,865
	2	225×1×2×200	27,313	777,781
	3	8×2×4×1,872	17,701	491,323
HOSVD	1	24×6×11×439	3,582	131,113
	2	18×4×5×223	1,942	74,087
	3	14×2×1×265	9,061	381,211

Table 5.5: Big dense blocks with top metric values discovered in the retweeting dataset.

N_2=1,000 and N_3=1,000 and (3) the mass of data C=10,000. We inject b=4 blocks into the random data:

- Block #1: The number of modes is k'_1=3 and $\mathcal{I}_1$={1,2,3}. The size is 30×30×30 and the block's mass is c_1=512.
- Block #2: The number of modes is k'_2=2 and $\mathcal{I}_2$={1,2}. The size is 30×30×1,000 and the block's mass is c_2=512.
- Block #3: The number of modes is k'_3=2 and $\mathcal{I}_3$={1,3}. The size is 30×1,000×30 and the block's mass is c_3=512.
- Block #4: The number of modes is k'_4=2 and $\mathcal{I}_4$={2,3}. The size is 1,000×30×30 and the block's mass is c_4=512.

Note, blocks 2–4 are dense in only 2 modes and random in the third mode. Table 5.4 reports the classification performances of CrossSpot and baselines. We show the overall evaluations (precision, recall and F1 score) and recall value of every block. We observe that CrossSpot has 100% recall in catching the 3-mode block #1, while the baselines have 85%–95% recall. More impressively, CrossSpot successfully catches the 2-mode blocks, where HOSVD has difficulty and low recall. The F1 score of overall evaluation is as large as 0.972 with 68.8% improvement.

Testing robustness of the random seed number: We test how the performance of our CrossSpot improves when we use more seed blocks in the low-order block detection experiments. Figure 5.7(a) shows the best F1 score for different numbers of random seeds. We find that when we use 41 random seeds, the best F1 score is close to the results when we use as many as 1,000 random seeds. Thus, once we exceed a moderate number of random seeds, the performance is fairly robust to the number of random seeds.

Efficiency analysis: CrossSpot can be parallelized into multiple machines to search dense blocks with different sets of random seeds. The time cost of every iteration is linear in the number of non-zero entries in the multimodal data as we have discussed in Section 5.4. Figure 5.7(b) reports the counts of iterations in the procedure of 1000 random seeds.

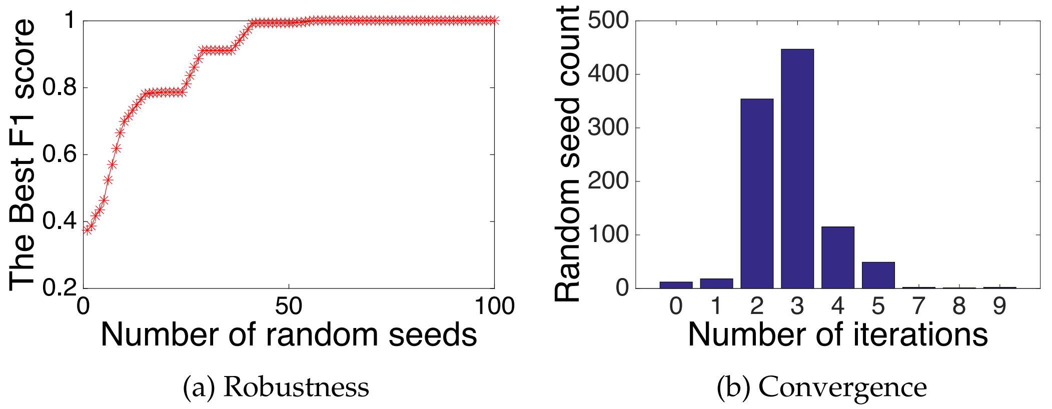


Figure 5.7: **CrossSpot is robust to the number of random seeds**. In detecting the 4 low-order blocks, when we use 41 seeds, the best F1 score has reported the final result of as many as 1,000 seeds. CrossSpot converges very fast: the average number of iterations is 2.87.

We observe that usually CrossSpot takes 2 or 3 iterations to finish the local search. Each iteration takes only 5.6 seconds. Tensor decompositions such as HOSVD and PARAFAC used in MAF often take more time. On the same machine, HOSVD methods of rank r=5, 10 and 20 take 280, 1750, 34,510 seconds respectively. From Table 5.4 and Figure 5.7(a), even without parallelization, we know that CrossSpot takes only 230 seconds to have the best F1 score 0.972, while HOSVD needs more time (280 seconds if r=5) to have a much smaller F1 score 0.324.

5.5.4 Retweeting Boosting

Table 5.5 shows big, dense block patterns of Tencent Weibo retweeting dataset. CrossSpot reports blocks of high mass and high density. For example, we spot that 14 users retweet the same content for 41,396 times on 2 IP addresses in 19 hours. Their coordinated, suspicious behaviors result in a few tweets that seem extremely popular. We observe that CrossSpot catches more suspicious (bigger and denser) blocks than HOSVD does: HOSVD evaluates the number of retweets per user, item, IP, or minute, but does not consider the block's density, mass nor the background.

Table 5.6 shows an example of retweeting boosting from the big, dense 225×1×2×200 block reported by our proposed CrossSpot. A group of users (e.g., A, B, C) retweet the same message "*Galaxy note dream project: Happy happy life travelling the world*" in lockstep every 5 minutes on the same two IP addresses in the same city. We spot that their retweet comments are generated from some literature or art books. The periodicity of the retweets and the nonsensical comments are strong independent evidence that the suspicious behavior found by CrossSpot is actually fraudulent.

User ID	Time	IP address	Retweet comment (Google translator)
USER-A	11-26 10:08:54	IP-1	Qi Xiao Qi: "unspoken rules count ass ah…
USER-B	11-26 10:08:54	IP-1	You gave me a promise, I will give you a result...
USER-C	11-26 10:09:07	IP-2	Clouds have dispersed, the horse is already…
USER-A	11-26 10:13:55	IP-1	People always disgust smelly socks, it remains…
USER-B	11-26 10:13:57	IP-2	Next life do koalas sleep 20 hours a day…
USER-C	11-26 10:14:03	IP-1	all we really need to survive is one person…
USER-A	11-26 10:18:57	IP-1	Coins and flowers after the same amount…
USER-C	11-26 10:19:18	IP-2	My computer is blue screen
USER-B	11-26 10:19:31	IP-1	Finally believe that in real life there is no…
USER-A	11-26 10:23:50	IP-1	Do not be obsessed brother, only a prop.
USER-B	11-26 10:24:04	IP-2	Life is like stationery, every day we loaded pen
USER-C	11-26 10:24:19	IP-1	"The sentence: the annual party 1.25 Hidetoshi…

Table 5.6: **Retweeting boosting**: We spot a group of users retweet "Galaxy note dream project: Happy happy life travelling the world" in lockstep (every 5 minutes) on the same group of IP addresses from Liaocheng Shandong. (Block $225\times1\times2\times200$ in Table 5.5)

5.5.5 Hashtag Hijacking

Big, dense block patterns of tweeting hashtag data are illustrated in Table 5.7. CrossSpot reports blocks of high mass and high density. We spot (1) continuous attacks: 582 users post as many as 5,941,821 tweets of the same 3 hashtags on 294 IP addresses for almost *every* minute in 43 days; (2) extensive attacks: 75 users post 689,179 tweets of the same hashtag on only 2 IPs in 35 hours. The top 2 big dense blocks discovered by our CrossSpot take almost all the values of the time mode. HOSVD does not consider cross-mode scenarios. We observe that it cannot catch continuous attacks. The blocks that CrossSpot reports are more suspicious than those HOSVD does.

Table 5.8 shows an example of hashtag hijacking from the big, dense $582\times3\times294\times56{,}940$ block. A group of users (e.g., D, E, F) post tweets of advertising hashtags (e.g., #Snow#, #Li Ning - a weapon with a hero# and #Toshiba Bright Daren#) on multiple IP addresses of two cities in the same Province. This demonstrates that CrossSpot catches the use of advertising hashtags to inflate popularity.

5.5.6 Network Traffic

We illustrate big, dense block patterns of LBNL network traffic dataset in Table 5.9, comparing our proposed CrossSpot and HOSVD. CrossSpot reports blocks of high mass and high density. We spot (1) very big and dense blocks: 411 source IP addresses send a total of 47,449 packets ($\geq$100 from each) to 9 destination IPs on 6 ports, or 533 source IPs send 30,476 packets to 6 destination IPs on the same port; (2) small but very dense blocks:

	Rank #	User×Hashtag×IP×Minute	Mass c	Suspiciousness score
CrossSpot	1	582×3×294×**56,940**	5,941,821	111,799,948
	2	188×1×313×**56,943**	2,344,614	47,013,868
	3	75×1×2×2,061	689,179	19,378,403
HOSVD	1	2,001×1×4×135	77,084	2,931,982
	2	327×1×2×401	212,519	8,599,843
	3	851×2×4×337	103,873	3,903,703

Table 5.7: Dense blocks discovered in hashtag data.

User ID	Time	IP address	Tweet text with hashtag
USER-D	11-18 12:12:51	IP-1	**#Snow#** the Samsung GALAXY SII QQ Service ...
USER-E	11-18 12:12:53	IP-1	**#Snow#** the Samsung GALAXY SII QQ Service ...
USER-F	11-18 12:12:54	IP-2	**#Snow#** the Samsung GALAXY SII QQ Service ...
USER-E	11-18 12:17:55	IP-1	**#Li Ning - a weapon with a hero#** good support activities!
USER-F	11-18 12:17:56	IP-2	**#Li Ning - a weapon with a hero#** good support activities!
USER-D	11-18 12:18:40	IP-1	**#Toshiba Bright Daren#** color personality test to find out ...
USER-E	11-18 17:00:31	IP-2	**#Snow#** the Samsung GALAXY SII QQ Service ...
USER-D	11-18 17:00:49	IP-2	**#Toshiba Bright Daren#** color personality test to find out ...
USER-F	11-18 17:00:56	IP-2	**#Li Ning - a weapon with a hero#** good support activities!

Table 5.8: **Hashtag hijacking**: We spot a group of users post tweets of multiple hashtags continuously on the same two IP addresses from Zaozhuang and Deyang, Shandong. (Block 582×3×294×56,940 in Table 5.7)

5 source IPs send 18,881 packets ($\geq$3,600 from each, $\geq$1 for every second) to 5 destinations on 2 ports, or 11 source IPs send 20,382 packets to 7 destination IPs on 7 different port numbers. These subsets of events are extremely suspicious: the probability of their occurrence is smaller than 10^{-10^6}. We observe that the top four blocks that CrossSpot catches are all due to continuous attacks. HOSVD cannot catch these 3-mode blocks (collapsing the time mode). Therefore, it only catches extensive attacks that are less suspicious than the continuous attacks.

These subsets of events take all the values in time mode, forming 3-mode dense blocks in 4-mode data. The cross-mode results indicate that a group of source IP addresses continuously send packets to multiple destination servers with the same group of ports in every second of one hour.

	Rank #	Source IP×Dest. IP×Port×Second	Mass c	Suspiciousness score
	1	411×9×6×**3,610**	47,449	552,465
CrossSpot	2	533×6×1×**3,610**	30,476	400,391
	3	5×5×2×**3,610**	18,881	317,529
	4	11×7×7×**3,610**	20,382	295,869
	1	15×1×1×1,336	4,579	80,585
HOSVD	2	1×2×2×1,035	1,035	18,308
	3	1×1×1×1,825	1,825	34,812
	4	1×13×6×181	1,722	29,224

Table 5.9: **Big dense blocks in LBNL network data**. The final suspicious blocks take all the time values indicating that suspicious traffic continuously happens in the hour.

5.6 Summary

In this chapter, we propose a general metric of surprise/suspiciousness for a dense block, in arbitrary number of modes. The main motivation was fraud detection, and more generally, attention routing—among two or more dense blocks, which ones are worth most of the human attention. The specific contributions are the following:

- **Metric criteria:** We propose a set of axioms that any metric of suspicious dense behavior should meet.
- **Novel metric:** We propose a suspiciousness metric, that is based on a principled, probabilistic model; and we prove that it obeys our axioms.
- **CrossSpot algorithm:** We propose a scalable algorithm to find dense, suspicious blocks in multimodal data.
- **Empirical results:** We demonstrate the effectiveness of our approach on synthetic as well as on real world data, spanning 0.3 billion entries. CrossSpot consistently improves the F1 score, up to 68.8%.

Part III

Modeling Normal Behavior

Based on work previously published in
WWW 2014 [31], *WWW* 2015 [29], and *WWW* 2016 [229].

Introduction

How can we understand a user's preferences and predict their future actions?
How can we explain our predictions?

Given users' ratings of movies or products, how can we understand a user's preferences for different types of items and recommend other items that the user will like? This problem was popularized by the Netflix Prize competition, in which Netflix released a dataset of 100 million ratings and offered 1 million dollars to anyone who could design a recommender system that gave a 10% improvement in prediction accuracy. The competition generated a flurry of research in collaborative filtering, with a variety of proposed matrix factorization models and inference methods [123]. Today, recommender systems are pervasive on the web—Facebook predicts what stories you will like for your News Feed, Amazon predicts what products you may want to buy, and Yelp predicts what restaurants you will like. Offering these recommendations is a core component of all of these services.

Much of the research in this area focuses on designing models that take in a database of user ratings of items and predicting the ratings for unobserved user and item combinations. The gold standard, set by the Netflix competition, was to have low average squared error.

However, while much of the research focus has been on having on-average high prediction accuracy, accuracy is only a small part of what makes recommender systems practically useful. In the following chapters, we move beyond average accuracy and focus on designing models that let us better *understand* users and their preferences. We make the following contributions:

- **Flexible Models for Normal and Abnormal Behavior:** In Chapter 6, we give a flexible model of user behavior that can capture both Gaussian and bimodal rating distributions, allowing us to detect polarizing content, as well as cluster fraudsters for detection. Our model, CoBaFi, improves accuracy by up to 17%.
- **Interpretable Recommendations:** In Chapter 7 we offer ACCAMS, a succinct, interpretable recommendation model that is 4 times smaller than previous methods while maintaining high accuracy.
- **Explaining Recommendations:** In Chapter 8 we offer PACO to directly give text descriptions of our model as well as personalized predictions for how a given user would describe *why* she likes a particular item. PACO does this by jointly modeling

both ratings and reviews.

III.1 Related Work

We will begin with a survey of research in modeling user behavior and recommender systems in particular. Given the breadth of this field, we will discuss the uniquely related work to each contribution in the relevant chapters that follow.

III.1.1 Recommender Systems

Collaborative filtering algorithms have been some of the best performing methods for recommender systems. Koren *et al.* [123] presents an overview of various methods for collaborative filtering. One of the simplest and most effective methods for collaborative filtering is low-rank matrix factorization. The intuition behind factor-based approaches is that preferences for users are determined by small numbers of unobserved factors. Linear factor models make use of two factor vectors—a user factor vector and an item factor vector —with the inner product representing a user's rating for an item. These vectors can be appended to form User Factor and Item Factor Matrices [200]. A variety of factorization models have bee proposed to capture different patterns or application goals, such as SVD++ for neighborhood effects [121], temporal effects in [122], implicit feedback [104], ranking [225] and others [207, 240]. Several works consider alternative distributions over attribute vectors, e.g., sparse non-negative factors [133], jointly sparse factorizations [99], cluster aggregation [10], or discrete factorizations [181].

Bayesian recommendation A variety of papers have adapted frequentist intuition on factorization to Bayesian models [159, 188, 202]. Of particular note are Probabilistic Matrix Factorization (PMF) [159] and Bayesian Probabilistic Matrix Factorization (BPMF) [188]. Other research in behavior modeling has built on Bayesian non-parametric approaches. For instance, [84, 87] use the Indian Buffet Process (IBP) for recommendation. In doing so they assume that each user (and movie) has certain preferential binary attributes. [181] proposed a factorization model based on a Dirichlet process over users and columns. All these models are closely related to the mixed-membership stochastic blockmodels of [14].

Side Information A separate line of research in recommender systems has focused on using side information to improve prediction quality. This is particularly important when parts of the data are extremely sparse, i.e. the cold start problem. Content-based filtering is a popular approach to alleviate this problem. Regression based latent factor models (RLFM) [8] address cold-start problem by utilizing user and item features. Cold-start users and items are able to share statistical strength with other users and items through similarity in features space. fLDA [9] uses text associated with items and user features to regularize item and user factors, but does not make use of review text.

Ensembles Recent models have achieved improved accuracy through ensembling multiple recommendation models. For example, DFC [150] and LLORMA [135, 136] have focused on using ensembles of factorizations that exploit local structure.

III.1.2 Co-Clustering

While co-clustering was originally used for understanding row and column clusters in biology [94], it has evolved to cover a wider variety of objectives [27]. [177] defined a soft co-clustering objective akin to a factorization model. Recent work has defined a Bayesian model for co-clustering focused on matrix modeling [191]. [221] focuses on exploiting co-clustering ensembles to find a single consensus co-clustering.

Prior to our research, co-clustering had been applied to user behavior modeling to a limited extent [78, 191, 192]. More recently, since the publication of our results, other papers have followed this direction of making use of co-clustering during recommendation [230].

III.1.3 Matrix Approximation

There exists a large body of work on matrix approximation in the theoretical computer science community (TCS). They focus mainly on efficient low-rank approximations, e.g., by projection or by interpolation [81, 92]. Essentially one aims to find a general low-rank approximation of the matrix, similar to recommender models.

A more parsimonious strategy is to seek *interpolative* decompositions, approximating columns of a matrix by a linear combination of a subset of other columns [142]. Nonetheless this requires us to store at least one, possibly more scaling coefficients per column. Also note the focus on column interpolations—this can easily be extended to row and column interpolations.

Chapter 6

Flexible Models for Normal and Abnormal Behavior

Given a large dataset of users' ratings of movies, how can we accurately model users' ratings? And how can we prevent spammers from tricking our algorithms into suggesting a bad movie? Is it possible to infer structure between movies simultaneously?

In this chapter we describe a unified Bayesian approach to Collaborative Filtering that accomplishes all of these goals. It models the discrete structure of ratings and is flexible to the often non-Gaussian shape of the distribution. Additionally, our method finds a co-clustering of the users and items, which improves the model's accuracy and lets the model detect fraud. We offer three main contributions: (1) We provide a novel model and Gibbs sampling algorithm that accurately models the quirks of real world ratings, such as convex ratings distributions. (2) We provide proof of our model's ability to handle spam and anomalous behavior. (3) We use several real world datasets to demonstrate the model's effectiveness in accurately predicting user's ratings, avoiding prediction skew in the face of injected spam, and finding interesting patterns in real world ratings data.

6.1 Introduction

As described above, collaborative filtering is one of the key tools for providing relevant content and for driving successful electronic commerce. Of late, Bayesian approaches have demonstrated good performance, when applied to collaborative filtering [186, 188]. Here we make the following contributions to the recommender problem:

Rating Model: Many collaborative filtering models simply minimize the squared error between prediction and observed rating. Alternatively, others make the assumption of a Gaussian noise model, assuming that the observed rating is given by the latent rating plus a small perturbation. Unfortunately this assumption is not realistic since the observed ratings are *discrete*, typically ranging from 1 to 5. Moreover, they need not be unimodal—some products are quite polarizing, attracting a significant number of 1 and 5 ratings with very little in between. We address this by modeling such dependencies explicitly.

Profile Distribution: A common approach to modeling user and item profiles is to draw them independently and identically (i.i.d.) from a distribution that is essentially normal around a common mean [188, 198]. However, often the objects we would like to recommend form clear clusters (e.g., romantic comedies, summer action movies). Hence, sharing of statistical strength even within the same object category is likely to improve accuracy. This extends work of [10, 14, 181] on dyadic factorization models.

Our motivation is that we have rather different amounts of information for different movies (and users). That is, some movies are very popular and using only a small number of latent factors amounts to considerable underfitting; as a result, using a larger number of parameters would be greatly beneficial. On the other hand, many movies have few ratings and it would be good if we could aggregate similar ones into a larger cluster for estimation purposes.

Inference: We propose an efficient sampling algorithm that can be parallelized with relative ease. It combines collapsed inference for discrete variables, uncollapsed inference for cluster parameters, and a Metropolis-Hastings iteration for non-conjugate terms arising from the recommender distribution.[1]

Spam Detection: A useful side-effect of the clustering approach is that similar users and items are grouped into the same cluster. This means that users who exhibit consistently abnormal behavior are likely to aggregate into the same clusters, thus limiting their impact on other recommendations and making it easier to detect and remove them.

Outline: We begin discussing our model for addressing the discrete nature of the rating distribution in Section 6.3. This is followed in Section 6.4 by a description of a nonparametric mixture of Gaussians akin to [165]. We then give an overview of the statistical model used for collaborative filtering in Section 6.5, and we subsequently give an in-depth description of statistical inference techniques in Section 6.6. We then in Section 6.7 analyze the model's ability to limit the impact of fraud. Last, we do an in depth analysis of our model on a variety of datasets in Section 6.8.

6.2 Relationship to Related Work

In addition to the related work described in Section III.1, we also build on and improve on previous research for alternative loss functions and robust recommendation. We discuss the relevant research in these areas below.

[1] It is over two orders of magnitude faster than the sampling algorithms carried out in Stan mc-stan.org using Hamiltonian Monte-Carlo [98].

6.2.1 Recommendation Metrics

The least mean squares loss is a mainstay in collaborative filtering, not the least due to the Netflix contest. Nonetheless, there are many other and better objectives that one could strive to optimize for. For instance, [225, 232] study preference ranking instead of estimating scores directly. In fact, research on ranking teems with alternative formulations of preferences [39, 218]. There are variants using distance similarity [226]. However, we do not know of any previous work that directly considered the skewed nature of ratings during modeling and recommendation.

6.2.2 Rating Spam and Robust Recommendation

As we observed in Part II, spam detection for online reviews is a significant problem, due to the financial incentives for malicious behavior. There are of course specialized approaches for detecting online spam, including the techniques in Part II as well as text mining approaches for detecting review fraud [114, 144]. However, given that our recommender systems consider all user behavior, it makes sense for them to capture the difference between normal and abnormal behavior. A small amount of research has approached this goal [157]; for example, [217] propose an algorithm with very good theoretical guarantees of robustness, albeit without the explicit ability to detect spammers. More related to our work is [196] who use Bayesian mixture models for detecting outliers. Most similar to our work in this chapter is the previously discussed fraud and anomaly detection research that finds fraud by analyzing the latent user and item embeddings from the SVD [110, 183, 190].

In this chapter we use a factorization model for recommendation where the factors are derived from a Gaussian mixture model. Moreover, we model the score distribution over ratings directly to deal with the often-observed bimodality of the reviews. This is all jointly addressed by a Metropolis-Hastings sampler for a unified Bayesian treatment. The advantage of our approach is that we only need to build *one* model to perform recommendation, spam detection, to model the distribution and confidence of a rating distribution explicitly, and to group users and movies.

6.3 Rating Distribution

One of the key components of our approach is to provide a model that captures the bimodality in many of the reviews found on rating sites. For instance, consider the example of ratings for a WD15EZRX hard disk (newegg.com, April 27, 2013) as shown in Figure 6.1. It clearly shows a very polarized view of the product, with significant numbers of users rating it very poorly whereas others rate it as excellent.

This type of rating distribution has been commonly ignored in collaborative filtering. In fact, when aiming for a least mean squares fit the optimal rating would be 2.5319 stars, the mean of the reviews. Such a score would be very misleading since only a small minority of reviews, namely only 7% of them actually assume such a value. Moreover,

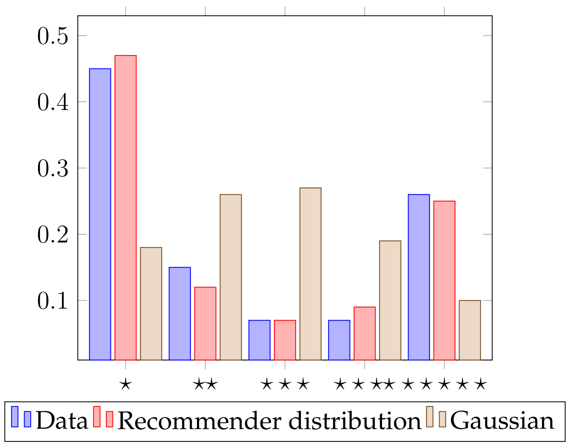


Figure 6.1: **Modeling ratings for WD15EZRX hard disk** (newegg.com, April 27, 2013).

while an RMSE of 2.85 would indicate that the data is a poor fit, it would tell us precious little about the actual *distribution*. The distribution itself in many cases is just as important as the predicted rating: suggesting an item with a bimodal review distribution, be it an unreliable hard drive or a polarizing film, may be undesirable for a risk-avoiding user.

Note that this is not an isolated case. We have observed many such examples in both product reviews and movies. However, not all items are equally polarizing and, in fact, the majority of reviews follow a more conventional unimodal distribution. Hence it is desirable to design a likelihood model that takes such nonstandard settings into account.

For this purpose we employ an exponential families model of a discrete distribution over the set $\mathcal{X} = \{1, \ldots R\}$, where typically we have $R = 5$. In the following we will refer to this distribution as the *recommender distribution*. Define

$$\phi(x) = \left(\left(x - \tfrac{R}{2}\right), \left(x - \tfrac{R}{2}\right)^2\right) \tag{6.1}$$

to be the sufficient statistic of the data on $\mathcal{X}$. In this case we have with $\theta \in \mathbb{R}^2$ the model

$$p(x|\theta) = e^{\langle \phi(x), \theta \rangle - g(\theta)} \text{ where } g(\theta) = \log \sum_{j=1}^{R} e^{\langle \phi(j), \theta \rangle}. \tag{6.2}$$

The above generalizes normal distributions to a discrete finite domain. Like the Gaussian, we can adjust mean and variance by modifying the first and second coordinate of θ accordingly. However, quite unlike in a Gaussian, we can also choose coefficients $\theta_2 \geq 0$. Such a choice would amount to no variance or negative variance respectively, which is clearly impossible (with correspondingly diverging integrals). On a finite countable domain, however, normalization is easy. As common in exponential families, we have

$$-\partial_\theta \log p(x|\theta) = \sum_{x'} \phi(x') p(x'|\theta) - \phi(x) \tag{6.3}$$

$$-\partial_\theta^2 \log p(x|\theta) = \operatorname*{Cov}_{x' \sim p(x'|\theta)} [\phi(x')] \tag{6.4}$$

Note that the covariance is *independent* of x since it depends only on the log-partition function $g(\theta)$. We will exploit this property in the context of a Metropolis-Hastings sampler to infer user and movie dependent parameters.

To illustrate the model we fitted the recommender distribution to the rating data for the WD15EZRX hard disk. This yielded $\theta = (-0.157, 0.391)$ as shown in Figure 6.1. For comparison we also display the fit obtained by a Gaussian distribution when discretized over the domain $\{1, \ldots 5\}$ and using the mean and variances inherent in the data. It is clear that the Gaussian thoroughly misrepresents the data whereas the recommender distribution is an excellent fit.

The attraction of (6.2) is that it allows us infer the parameter θ efficiently, e.g., by means of simple convex optimization problem: as all exponential family models, the negative log-likelihood $-\log p(x|\theta)$ is convex in θ. The major difference is that we now need to fit two parameters: one for what effectively amounts to the mean, and one that amounts to the skew of the distribution. In summary the recommender distribution has the following properties:

- Negative values of θ_2 it will yield a unimodal distribution, as the log-likelihood mimics a discretized Gaussian. The magnitude of θ_2 determines how peaked the distribution is. Large values of θ_1 in this context imply that the popularity is fairly uncontested. This is nontrivial to achieve in a Gaussian fit of the data.
- Positive values of θ_2 can lead to a bimodal distribution. The extent of bimodality is determined by the value of θ_1. If it is very large (or very small), it leads to what is essentially a single mode at either one of the extremes.
- Large (small) values of θ_1 correspond to an object that is rather more (less) popular. This can be seen in our example where we inferred $\theta_1 = -0.157$, i.e. the disk was rather unpopular.

In our model we will assume that the amount of polarization, i.e. the value of θ_2, is determined in an additive fashion between users and items. That is, the distribution is bimodal if the item encourages such ratings (e.g., an unreliable hard disk) and if the user is prone to praise and condemnation. For the linear term we use an inner product model.

To assess this new rating model we need to evaluate its predictive log-likelihood, since a least mean squares fit clearly obscures the properties of the data. Hence, in our experiments we will compute $\sum_x -\log p(x|\theta)$. By basic facts from information theory [56], this is bounded from above by the entropy of the uniform distribution, i.e. $\log 5$ for 5 different outcomes. The lower bound is naturally the conditional entropy $H(R|U, M)$, where we conditioned on the user and item pairs. By definition, it is impossible to compute this lower bound explicitly, short of solving the recommendation problem optimally.

6.4 Co-Clustering

Collaborative filtering is nontrivial whenever the number of ratings is nonuniform over items and users. Like much other natural data, the number of ratings often follows a power-law distribution on a per row and per column basis. For instance, blockbuster bigbudget Hollywood movies will attract many more ratings than independent productions, regardless of their quality. Likewise, a small number of enthusiasts can generate an enormous amount of ratings when compared to more casual users.

Conventional collaborative filtering models assume that all users (and items respectively) are parameterized by a fixed dimensional model. In fact, many experiments show that the accuracy of the models plateaus (or even decreases) once a fairly modest number of parameters has been reached. Part of this effect is due to the fact that we have comparatively little data per row (or column) of the recommendation matrix. Furthermore, data scarcity is different, when viewed on a per-column or per-row basis. For instance, we might have millions of users but only thousands of movies.

This suggests that it would be very much desirable to have a mechanism for assigning capacity dynamically. That is, users who generate significant amounts of ratings are sufficiently well-specified to afford a 'personal' parameter vector for them. On the other hand, users who generate very little data are probably best modeled as unspecific members of a much larger pool of similar participants. Such a model can be obtained by clustering users, e.g., as mixture of Gaussians.

To accomplish this, we use co-clustering of the rows and columns of the recommendation matrix to allow for dynamic allocation of statistical capacity between sets of users. This allows us to increase dimensionality beyond what is optimal when treating all users/items equally. Hence we need to introduce a statistical model for partitioning. We resort to the Dirichlet Process for this purpose. This yields a nonparametric mixture of Gaussians. To keep the presentation self-contained we give a brief overview of the model below.

One of the prototypical tools for nonparametric modeling is the Dirichlet Process $DP(H, \gamma)$. It allows for a discrete distribution of observations drawn from an arbitrary base measure H over the domain $\mathcal{X}$ in such a way as that the marginals match draws from H, while simultaneously obtaining a countable set of distinct items. Denote by $\mathcal{X}_i$ a partition of $\mathcal{X}$ and let $c_i := \mu_H(\mathcal{X}_i)$ be the measure of $\mathcal{X}_i$ under H. Then the discretization of draws from G_0 with $G_0 \sim DP(H, \gamma)$ onto the sets $\mathcal{X}_i$ behaves as if it had been drawn from a multinomial distribution θ with $\theta \sim \mathrm{Dir}(\gamma c)$.

A useful view of the Dirichlet Process is the Chinese Restaurant metaphor. In it, each data point is considered as a customer in a restaurant with an infinite number of tables.Initially all tables are empty. As the tables fill up, customers pick existing tables in proportion to their popularity, resulting in a "rich get richer" scenario. The probability for customer i to pick table j is given by

$$\Pr\{z_i = j\} = \begin{cases} \frac{N_j}{\sum_k N_k + \gamma} & \text{for an existing table} \\ \frac{\gamma}{\sum_k N_k + \gamma} & \text{for a new table.} \end{cases} \tag{6.5}$$

Here z_i encodes the choice of customer i. N_j denotes the *current* number of customers sitting at table j and $\sum_k N_k$ is the total number of customers so far. This makes the Chinese Restaurant Process a single parameter distribution over partitions of the integers. Figure 6.2 provides a pictographical representation of the process. One may show that

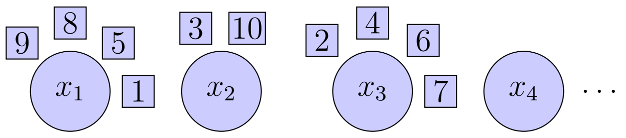


Figure 6.2: **An example of the Chinese Restaurant Process metaphor**. Here we observe 10 customers who have so far occupied tables 1 (customers 1, 5, 8, 9), 2 (customers 3, 10) and 3 (customers 2, 4, 6, 7). A new customer will pick the first three tables with probabilities $\frac{4}{10+\gamma}$, $\frac{2}{10+\gamma}$ and $\frac{4}{10+\gamma}$ respectively, or a new table with probability $\frac{\gamma}{10+\gamma}$.

the distribution over partitions is invariant in the order in which customers arrive, i.e. it is exchangeable.

A benefit of (6.5) is that it affords for collapsed Gibbs sampling inference by simply drawing from

$$z_i \sim \Pr(z_i|\text{rest}) \text{ where } \Pr(z_i|\text{rest}) \propto p(z_i|z\backslash z_i)p(\text{ratings}|z)$$

Inference proceeds by traversing users (or items) and resampling their cluster assignments in sequence.

6.5 Generative Model

We describe a model for obtaining collaborative filtering estimates in a pure rating setting rather than a recommendation setting [232]. That is, we assume that for a subset of (user,item) pairs (i, j) we are given ratings $r_{ij} \in \{1, \ldots R\}$. It is our goal to infer user and item specific parameters u_i, v_j such that we are able to estimate the distribution of $r_{ij}|i, j$ efficiently. For this purpose we employ a directed graphical model, as depicted in Figure 6.3.

This model extends existing recommendation algorithms such as [188, 198] by assuming a more complex structure for the latent user and item attributes. We assume that the ratings r_{ij} are drawn from the recommender distribution. The parameters $\langle u_i, v_j\rangle + u_i^{(1)} + v_j^{(1)}, u_i^{(2)} + v_j^{(2)}$, as associated with parts of the user and item specific vectors, account for preference and skew explicitly:

$$r_{ij} \sim \text{Recommender}\left(\langle u_i, v_j\rangle + u_i^{(1)} + v_j^{(1)}, u_i^{(2)} + v_j^{(2)}\right)$$

From this we have for each user a $d-2$ length vector u_i and scalars $u_i^{(1)}, u_i^{(2)}$ giving us d latent parameters per user (and item). For simplicity, we will often refer to this set of parameters as a single vector u_i.

We assume that the user and item parameter distributions are described by a mixture of Gaussians, where

$$u_i \sim \mathcal{N}(\mu_{a_i}, \Sigma_{a_i}) \text{ and } v_j \sim \mathcal{N}(\mu_{b_j}, \Sigma_{b_j}).$$

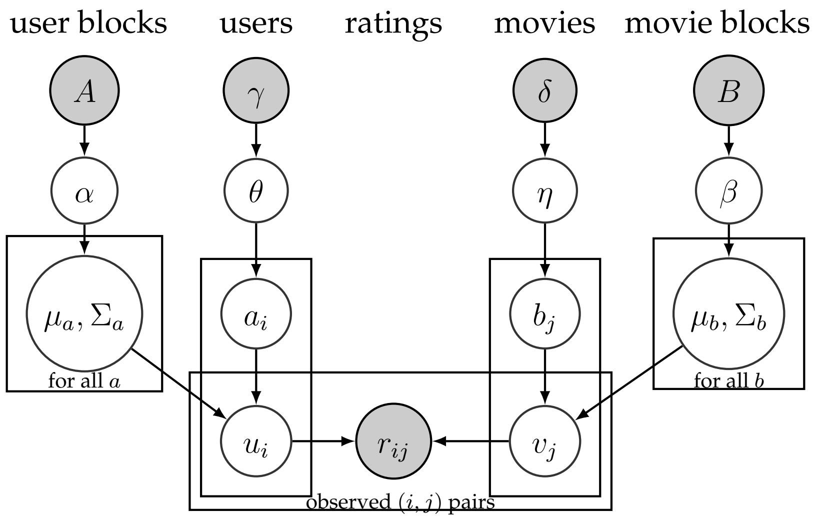


Figure 6.3: **Factorized rating model**. Note the symmetry between users and items. In both cases we use a mixture of Gaussians to represent the latent parameter distribution.

Here a_i and b_j are the clusters that user i and item j belong to respectively. The cluster means and variances μ_a, Σ_a are drawn from a Gauss-Wishart distribution with ν_0 degrees of freedom and parameters $\{\mu_\alpha, W_\alpha, \lambda_\alpha\}$ and $\{\mu_\beta, W_\beta, \lambda_\beta\}$, respectively:

$$(\mu_a, \Sigma_a) \sim \mathrm{GW}(\lambda_\alpha, W_\alpha, \mu_\alpha) \text{ and } (\mu_b, \Sigma_b) \sim \mathrm{GW}(\lambda_\beta, W_\beta, \mu_\beta)$$

This is convenient since in this case the priors are conjugate and we are able to draw from $\mu_a, \Sigma_a | \alpha, u, a$ exactly without the need for approximation.

The common parameters W_α, μ_α (and W_β, μ_β) allow us to share information about variance and means between clusters via the conjugate Wishart distribution. Finally, we impose an inverse Gauss-Wishart hyperprior on $W_\alpha, \mu_\alpha, W_\beta, \mu_\beta$ respectively. For computational convenience we will maximize the likelihood with regard to those hyperparameters.

Finally, the cluster indicators, a_i and b_j, allow us to assign users (and items respectively) to particular groups. To sample cluster assignments, we draw

$$a_i \sim \mathrm{Mult}(\theta) \text{ and } b_j \sim \mathrm{Mult}(\eta) \tag{6.6}$$

In turn, we assume that the distribution over clusters follows a Dirichlet distribution conjugate to the multinomials governing cluster membership. Likewise in the nonparametric case (where the number of clusters itself is unbounded), we assume that it is given by a draw from a Dirichlet process.

$$\theta \sim \mathrm{Dir}(\gamma) \text{ and } \eta \sim \mathrm{Dir}(\delta) \tag{6.7}$$

Therefore, in a nutshell, we assume that the interaction between users u_i and items v_j can model ratings r_{ij} similarly to many other collaborative filtering methods. Unlike [188] who assume one common mean and variance for all users (and items respectively), clustering provides the ability to better model what is a similar user.

6.6 Parameter Inference

Inference in the statistical model introduced previously requires sampling and (or) optimization. Matters are simple whenever the distributions are conjugate since the associated parameters can either be integrated out (collapsed) or, alternatively, it is easy to draw from them. The non-conjugate parts do not admit a closed-form treatment. In the following we discuss in detail how to sample the various parameters and variables for the sampler's Markov chain, subject to these constraints. Since user and item-related parameters are entirely identical, we only discuss users.

6.6.1 Sampling the user and item parameters u_i

We omit details for the case where r_{ij} is Gaussian, as it is almost identical to [188]. Since the recommender distribution does not have conjugacy, sampling the user and item parameters u_i and v_j is challenging and we resort to a Metropolis-Hastings procedure [21]. In its simplest form the idea is that rather than sampling from some difficult distribution $p(x)$ we draw $\bar{x}$ from an approximation $q(x)$ and accept the move $x \to \bar{x}$ with probability $\min(1, p(\bar{x})q(x)/p(x)q(\bar{x}))$.

In our case the distribution p is given by the log-concave distribution $p(u_i|\text{rest})$ and we obtain q by performing a Laplace approximation of p. In other words, we perform a second order Taylor approximation of $\log p(u_i|\text{rest})$ at its mode, which yields a Gaussian. This is possible since log-concave distributions have a global mode and the second derivative of the negative log-posterior is positive semidefinite.

$$\begin{aligned}
&-\log p(u_i|\text{rest}) - \text{const.} \\
=&\frac{1}{2}(u_i - \mu_a)^\top \Sigma_a^{-1}(u_i - \mu_a) + \\
&\sum_{j\in\text{ratings}(i)} g(u_i^{(1)} + v_j^{(1)} + \langle u_i, v_i\rangle, u_i^{(2)} + v_j^{(2)}) - \\
&\sum_{j\in\text{ratings}(i)} \left\langle \phi(r_{ij}), (u_i^{(1)} + v_j^{(1)} + \langle u_i, v_i\rangle, u_i^{(2)} + v_j^{(2)}) \right\rangle
\end{aligned}$$

As before $u_i, u_i^{(1)}, u_i^{(2)}$ denote the latent parameterization, the linear biases and the quadratic biases for the user, with corresponding terms for item parameters v. The log-partition function g is as defined in (6.2) with appropriate derivatives given by (6.3) and (6.4).

Convex minimization of $-\log p(u_i|\text{rest})$ yields the mode $\bar{\mu}$. By first order conditions and since the objective is analytic, we know that at $\bar{\mu}$ the gradient vanishes and we may compute the 'variance' of $q(u_i)$ via

$$\bar{\Sigma}^{-1} = \sigma_a^{-1} + \sum_{j\in\text{ratings}(i)} \partial_{u_i}^2 g(u_i^{(1)} + v_j^{(1)} + \langle u_i, v_i\rangle, u_i^{(2)} + v_j^{(2)}).$$

Using $\bar{u} \sim \mathcal{N}(\bar{\mu}, \bar{\Sigma})$ we accept the MH proposal with

$$\min\left(1, \frac{p(\bar{u}|\text{rest})\mathcal{N}(u_i(t)|\bar{\mu}, \bar{\Sigma})}{p(u_i(t)|\text{rest})\mathcal{N}(\bar{u}|\bar{\mu}, \bar{\Sigma})}\right) \tag{6.8}$$

Otherwise we set $u_i(t+1) \leftarrow u_i(t)$. The advantage of this strategy is that we need to compute $\bar{\mu}$ and $\bar{\Sigma}$ only once per resample of u_i. Multiple MH steps ensure proper mixing.

The computational cost of this step is $O(d^3 + d^2 n_{i,\text{ratings}})$ per user, i.e. for the purpose of the Taylor approximation we need to aggregate all ratings ($n_{i,\text{ratings}}$) of a given user and subsequently draw from it. Hence the total cost is $O(d^3|V| + d^2|E|)$, when viewing the set of (user,item) pairs as a graph with V vertices and $|E|$ edges.

6.6.2 Sampling the cluster membership a_i

Our generative model represents cluster assignments of users and items as draws of indicator variables a_i and b_j from multinomial distributions, parameterized by θ and η. These multinomials are in turn drawn from their conjugate, the Dirichlet distribution. We take advantage of a collapsed Gibbs sampler to deal with the Dirichlet process in a (and b respectively) [88, 165]. More specifically, we may integrate out θ, η such that we are left with an *exchangeable* distribution over $\{a_i\}|\gamma$ (and corresponding terms in b, δ). This leaves us with (6.5), i.e.

$$p(a_i = \bar{a}|a\backslash\{a_i\}) = \begin{cases} \frac{n_{\bar{a}}^{-i}}{n+\gamma-1} & \text{for existing cluster} \\ \frac{\gamma}{n+\gamma-1} & \text{for new cluster} \end{cases} \tag{6.9}$$

Analogous values can be obtained if we want to use the Pitman-Yor process [113]. Secondly, we need to evaluate the likelihood of $u_i|a_i$, as given by $\mathcal{N}(\mu_{a_i}, \Sigma_{a_i})$. This governs how well u_i fits to an *existing* cluster. Whenever we instantiate a new cluster, matters are slightly more involved—the acceptance probability of drawing a new random set of parameters μ, Σ from the associated Gaussian-Wishart distribution would be very low. Instead, we integrate out μ, Σ and evaluate directly $p(u_i|\mu_\alpha, W_\alpha, \lambda_\alpha)$. However, to do this we must integrate and collapse out the intermediate cluster parameters, see e.g., [91]. Thus the collapsed probability of u_i is

$$\begin{aligned} &p(u_i|\mu_\alpha, W_\alpha, \lambda_\alpha) \\ &=\mathcal{T}\left(\nu_0 - d + 2, \frac{\lambda_\alpha\mu_\alpha + u_i}{\lambda_\alpha + 1}, \frac{\lambda_\alpha + 2}{(\lambda_\alpha + 1)(\nu_0 - d + 2)}\Sigma_m\right) \end{aligned}$$

where $\Sigma_m = W_\alpha^{-1} + \dfrac{\lambda_\alpha}{\lambda_\alpha + 1}(u_i - \mu_\alpha)(u_i - \mu_\alpha)^\top$. Here $\mathcal{T}$ is the student-t distribution, ν_0 is a user specified parameter and d is the rank of the decomposition. Ultimately, this gives us an estimate of the likelihood of a new cluster for u_i:

$$p(a_i = \bar{a}|\text{rest}) \propto \begin{cases} \frac{n_{\bar{a}}^{-i}}{n+\gamma-1}\mathcal{N}(u_i|\mu_{\bar{a}}, \Sigma_{\bar{a}}) & \text{if cluster exists} \\ \frac{\gamma}{n+\gamma-1}p(u_i|\mu_\alpha, W_\alpha, \lambda_\alpha) & \text{for new cluster} \end{cases}$$

Lastly, if we draw a new cluster, we need to instantiate the values of μ_a, Σ_a. Since this is no different for clusters of size 1 or larger, we describe the general case below.

Resampling the cluster membership for a user requires us to evaluate the cluster likelihood for each existing (and one new) cluster. Each of these steps cost $O(d^2)$ computation. Given k clusters this amounts to $O(kd^2|V|)$ CPU cost for one sampling pass.

6.6.3 Sampling cluster parameters μ_a, Σ_a

Since the Gauss-Wishart distribution is conjugate to the Gaussian, it follows that the conditional posterior $p(\mu_a, \Sigma_a|\mathrm{rest})$ is also a Gauss-Wishart. Moreover, the update equations for a cluster are particularly simple: draw μ_a, Σ_a from a Gauss-Wishart distribution with parameters

$$\bar{\lambda}_a = \lambda_\alpha + N_a \tag{6.10a}$$

$$\bar{\lambda}_a \bar{\mu}_a = \lambda_\alpha \mu_\alpha + \sum_{i:a_i=a} u_i \tag{6.10b}$$

$$\bar{\lambda}_a \bar{W}_a^{-1} = \lambda_\alpha W_\alpha^{-1} + \lambda_\alpha \mu_\alpha \mu_\alpha^\top + \sum_{i:a_i=a} u_i u_i^\top \tag{6.10c}$$

$$\bar{\nu} = \nu_0 + N_a \tag{6.10d}$$

Here $N_a = \sum_{i:a_i=a} 1$ is the size of the cluster a and $\bar{\mu}_a$ is the mean of the u_i vectors for the user in cluster a. See e.g., [163] for extended details and how to sample from it via a multivariate Student-t distribution.

The cost for sampling the cluster means and variances is $O(N_a d^2 + d^3)$ per cluster, hence the total cost is $O(|V|d^2 + kd^3)$ to sample all clusters in the algorithm.

6.6.4 Inference for hyperparameters $\mu_\alpha, W_\alpha, \lambda_\alpha$

The last remaining step for inference is to estimate a suitable value for the parameters of the Gauss-Wishart distribution. Choosing these terms appropriately is important since they provide the default (prior) for the cluster specific factors. We resort to maximization rather than sampling, since the number of clusters is large relative to the number of parameters involved in determining $\mu_\alpha, W_\alpha, \lambda_\alpha$, hence we expect the posterior to be rather peaked. For the sake of completeness, we give the full derivation below. In a nutshell we compute the parameters using a conjugate to the Gauss-Wishart distribution. While this is not available in closed form, it is easily constructed implicitly by adding an additional spurious normal distribution. We choose one with unit variance and zero mean. For notational convenience, we will include such a 'prior' in the set of $(\mu_a, \Sigma_a, \lambda_a)$ terms. The Gauss-Wishart distribution is given by

$$p(\mu, \Sigma|\mu_0, \lambda, W, \nu) = \mathcal{N}(\mu|\mu_0, \lambda^{-1}\Sigma)\mathcal{W}(\Sigma^{-1}|W, \nu) \tag{6.11}$$

$$\mathcal{W}(\Sigma^{-1}|W, \nu) = 2^{-\frac{\nu d}{2}} |\Sigma|^{\frac{\nu}{2}} \Gamma_d^{-1}\left(\frac{\nu}{2}\right) |W|^{\frac{\nu-d-1}{2}} e^{-\frac{1}{2}\operatorname{tr}\Sigma W}$$

Here $\mathcal{W}$ denotes the Wishart distribution with ν degrees of freedom and covariance scale W. Moreover, Γ_d denotes the multivariate Gamma function. The negative log-likelihood given k normal distributions is given (up to constants independent of μ_0, W, λ) by

$$\begin{aligned}
&- \log p(\{\mu_a, \Sigma_a\} | \mu_0, \lambda, W, \nu) \\
=&\frac{kd}{2} \log \lambda + \frac{1}{2\lambda} \sum_{a=1}^{k} (\mu_a - \mu_0)^\top \Sigma_a^{-1} (\mu_a - \mu_0) + \frac{\nu}{2} \sum_{a=1}^{k} \log |\Sigma_a| \\
&+ k \log \Gamma_d \left(\tfrac{\nu}{2}\right) + \tfrac{k(\nu-d-1)}{2} \log |W| + \frac{1}{2} \operatorname{tr} W \sum_{a=1}^{k} \Sigma_a + \text{const.}
\end{aligned}$$

Taking derivatives with respect to μ_0, W, and λ respectively yields that the log-likelihood is maximized for

$$\mu_0 = \left[\sum_{a=1}^{k} \Sigma_a^{-1}\right]^{-1} \sum_{a=1}^{k} \Sigma_a^{-1} \mu_a \tag{6.12}$$

$$W = k(\nu - d - 1) \left[\sum_{a=1}^{k} \Sigma_a\right]^{-1} \tag{6.13}$$

$$\lambda = \frac{1}{kd} \sum_{a=1}^{k} (\mu_a - \mu_0)^\top \Sigma_a^{-1} (\mu_a - \mu_0) \tag{6.14}$$

That is, the mean μ_0 is given by what amounts to Gaussian posterior averaging; the prior on covariances W is given by the average covariance, suitably normalized by dimensions and degrees of freedom; the variance scale λ is given by an average over clusters and dimensions. The degrees of freedom $\nu > d - 1$ control the effective dimensionality of the space. Large values of ν encourage a larger volume of the associated covariance matrices, whereas a values emphasize low-dimensional per-cluster distributions. We set $\nu = \frac{3}{2}d$.

Computing W, μ_0 and λ costs $O(kd^3)$ work since we need to aggregate over all cluster centers. In summary, we have the following runtime properties:

Remark 1 *Per iteration the algorithm requires $O(kd^3 + |V|d^3 + |V|d^2k + |E|d^2)$ computation. That is, it is cubic in the number of latent parameters, linear in the number of clusters, and linear in the amount of data given.*

Furthermore, if needed, these steps could be parallelized efficiently, since most work is specific per user (or per item) with communication required only once for each round of sampling and optimization.

6.7 Spam Detection during Recommendation

Fraud detection is a persistent challenge and requires a multi-tiered approach. By default, most recommender systems are not very resilient, allowing any spam that is not caught ahead of time to skew recommendations significantly. To illustrate this, consider Bayesian matrix factorization [188, 198]. In this case the influence of a malicious user cannot be bounded: the sufficient statistic governing the normal distribution over users is a linear combination of the statistics of all constituents. Hence it follows that each individual contribution can assume arbitrary amounts, provided that the associated statistic is also unbounded. Moreover, even for bounded sufficient statistics, the aggregate influence can be arbitrarily high, provided that a sufficiently large number of malicious users contributes. In other words, if a sufficiently high number of fraudulent likes for an objectionable movie is received, this will lead to the movie being rated very highly. The example below quantifies this effect.

Example 2 (Spam and sufficient statistics) *Denote by $\phi(x)$ the sufficient statistic of $x \in \mathcal{X}$. Moreover, let $X := \{x_1, \ldots, x_m\}$ and $X' := \{x'_1, \ldots x'_{m'}\}$ be regular and malicious observations respectively. Let m_0, μ_0 be sufficient statistics arising from a conjugate prior. Then the posterior mean yields*

$$\hat{\mu} := \frac{1}{m_0 + m + m'}\Big[m_0\mu_0 + \underbrace{\sum_{i=1}^{m}\phi(x_i)}_{=:m\cdot\mu[X]} + \underbrace{\sum_{i=1}^{m'}\phi(x'_{i'})}_{=:m'\cdot\mu[X']}\Big] \tag{6.15}$$

Hence parameter estimates will be influenced $O(\frac{m'}{m_0+m+m'})$ by outliers whenever we have bounded sufficient statistics and by an unbounded amount whenever the statistics are unbounded, even for $m' = 1$.

Several researchers, such as [217], have attempted to address this problem. However, [217] deals primarily with robustness in the nearest neighbor case, and the extension to other estimators is highly nontrivial. Our approach borrows from [196] which investigates mixture models for outlier-robust density estimation. The key difference is that we perform co-clustering of pairs of conditioning attributes. Moreover, we use clustering as latent information in an estimation problem.

Our analysis relies on the idea that when generating multiple clusters, malicious users can typically only affect a real users very little. *Whenever spammers are very different from regular users they will be aggregated into clusters of their own. Alternatively, whenever they are very similar to regular users, they cannot do much damage.* It exploits the fact that whenever a user is rather different, (6.9) suggests that it should be assigned to a new cluster.

Denote by X and X' two sets of m and m' observations respectively. In a Dirichlet process, the probability of two vs. one cluster is given by $\frac{mm'}{(n+\alpha)^2}$ vs. $\frac{m+m'}{n+\alpha}$. Hence, if the data likelihood for two separate clusters exceeds that of a single joint cluster via

$$p_{\text{cluster}}(X)p_{\text{cluster}}(X')\frac{mm'}{(n+\alpha)^2} > p_{\text{cluster}}(X \cup X')\frac{m+m'}{(n+\alpha)}$$

then there is benefit in splitting observations. From this we see that a split is likely whenever m' is large, i.e. whenever we have a large number of spammers, or alternatively, whenever their behavior deviates substantially from regular users.

On the other hand, if the spammers try being subtle (to avoid detection), then their presence will improve overall parameter estimates. As a result, egregious spammers are clustered together, limiting their impact on others' recommendations and making them easier to directly spot, through analyzing the clusters or testing users within the clusters.

6.8 Experiments

The model described above is particularly interesting because of the insight it gives into the many common datasets used throughout data mining. Here we demonstrate its effectiveness accurately modeling real world data, both matching some expectations of reviews and breaking others.

	Users	Items	Ratings
Netflix-48k	48,090	17,770	10,015,137
Netflix-24k	24,047	17,770	5,038,411
BeerAdvocate	12,652	5000	112,981
Amazon Electronics	29,294	2000	30,378
Amazon Clothing	16,017	5000	51,438

Table 6.1: Review datasets used in our experiments

6.8.1 Datasets and Set up

Various ratings systems and datasets have very different characteristics. We look at a few real-world datasets with varying distributional properties. As some domains may be more polarizing than others, the simple assumption that ratings will look the same across domains does not hold. We looked at four different sets of review data comprising movies, beer, clothing, and electronics. A summary of the datasets can be found in Table 6.1.

In analyzing movies data, we used the Netflix Prize dataset. We create two subsets of the data by randomly sampling users but keeping all movies. This creates one dataset of 48,090 users and 17,770 movies, which we will refer to as Netflix-48k, and one dataset of 24,047 users and 17,770 movies, which we will refer to as Netflix-24k. The Netflix-48k dataset has a total of 10,015,137 ratings and the Netflix-24k dataset has 5,038,411 ratings.

Second, we use user reviews of beers from BeerAdvocate.com, as was scraped in [156]. Here we again take a subset of the data, comprising 5000 beers and their associated reviews from 12,652 users. This makes for a total of 112,981 reviews.

	Uniform	BPMF	CoBaFi
Netflix-24k	1.6904	1.2525	**1.1827**
BeerAdvocate	2.1972	1.9855	**1.6741**

Table 6.2: **Comparison of predictive log-likelihood** on Netflix-24k and BeerAdvocate data.

We also use two datasets of Amazon reviews [155]. We first use reviews of a subset of products from Amazon's Electronics category. This includes 2000 products and all of their associated from 29,294 users. The total number of reviews in the dataset is 30,378. We note that this is just over 1 review per user on average, making this dataset much sparser than the rest. Last, we use the reviews from a subset of the products in Amazon's Clothing & Accessories category. This is 51,348 reviews on 5000 products from 16,017 users.

Implementation: We implement our model and Gibbs sampling procedure in Matlab along with its parallel toolbox for speed. For simplicity and efficiency purposes we set a maximum number of clusters for users and items in each run, which we will specify for the experiments below. In all cases we use the PMF solution from [188] as our starting point. For all experiments below we set the rank (the length of u_i and v_j) $d = 30$ and $\nu_0 = \frac{3}{2}d$.

Comparing on predictive probability: To measure our model's fit, we use the predictive probability (PP):

$$PP = \frac{1}{N} \sum_{r_{ij}} - \log p(r_{ij}|u_i, v_j)$$

In order to make this a fair comparison with non-discrete distributions such as the typical Gaussian distribution, we discretize it first before any comparison. Because our recommender model can also act like a Gaussian, we can convert a Gaussian model $\mathcal{N}(\langle u_i, v_j \rangle, \sigma^2)$ to a recommender model Recommender$(\langle \hat{u}_i, \hat{v}_j \rangle, \hat{u}_i^{(2)} + \hat{v}_j^{(2)})$ as follows:

$$\begin{array}{cc} \hat{u}_i = \sigma \cdot u_i & \hat{v}_j = \sigma \cdot v_j \\ \hat{u}_i^{(1)} = \hat{v}_j^{(1)} = 0 & \hat{u}_i^{(2)} = \hat{v}_j^{(2)} = -\frac{1}{4\sigma^2} \end{array}$$

As a result, no weight of the distribution is wasted on regions for which there couldn't be a rating. We tested many different values of σ^2 that gave the best predictive probability for a Gaussian and ultimately chose to use $\sigma^2 = 1.0$ because it generally produced the best results. Additionally, to compare against BPMF we run our code with a maximum of 1 cluster and fitting a normal as was done in their model.

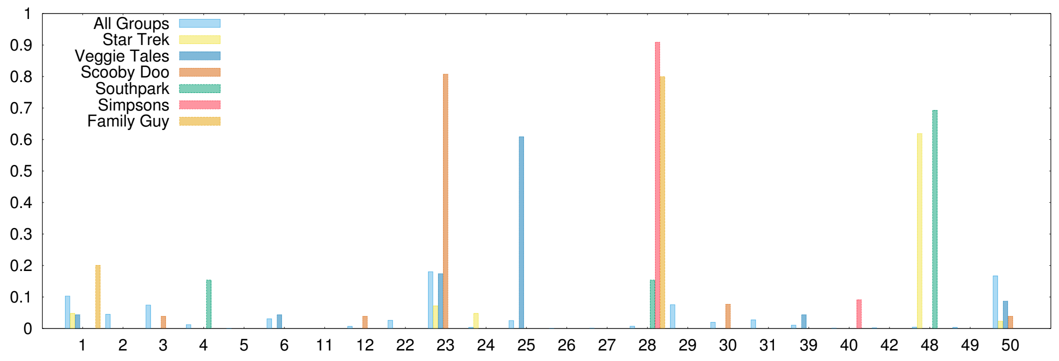


Figure 6.4: **Distribution of movies from TV series over clusters**. Movies from the same series are highly concentrated. Viewing series as ground truth, we see that the accuracy is fairly high. Note that movies within the same series are not necessarily uniform (different seasons, directors, script writers).

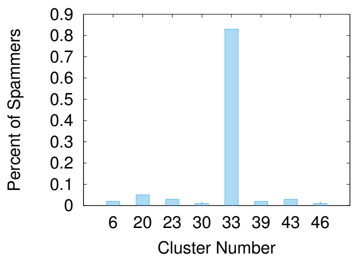
6.8.2 Model fit

We run our model on the Netflix-24k and BeerAdvocate datasets to verify that it better models the data. We compare the results of running BPMF [188] to that from our model with a maximum of 50 clusters. As explained before, we want to measure the *fit* of our model so we use average predictive probability. As seen in Table 6.2 (where smaller predictive probability is better), CoBaFi fits the data much better than BPMF for both Netflix-24k and BeerAdvocate.

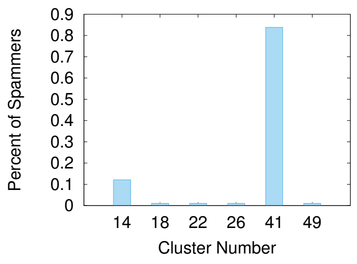
Sampling from the recommender distribution: As was mentioned earlier, one of the challenges of fitting such a model correctly is approximating our recommender distribution to perform the Metropolis-Hastings sampling. To verify that our approximation was working well, we recorded the average acceptance rate for a couple of our datasets. In the Netflix-24k dataset we see an average acceptance rate of 77.77%. In the robustness tests below on the Amazon Electronics dataset, we see a 99.12% acceptance rate.

6.8.3 Robustness to spam

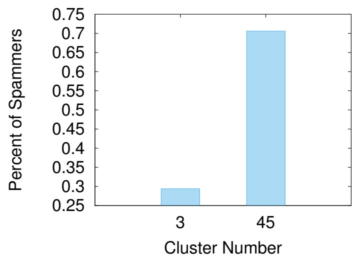
One of the benefits of incorporating clusters into a model is that similar users will get clustered together, whether it be users who have similar tastes, or anomalous users who may try to manipulate ratings for their own gain. Spammers are an issue in any large, community driven rating system as there is frequently a financial incentive for positive ratings and an incentive for negative ratings for competitors. By modeling the latent clusters of users, we are able to group similar users together. This means that anomalous users, such as spammers, may get grouped together, and thus have a smaller impact overall on ratings outside of their collective group or groups. To do this we ran two experiments. In each we take one of the datasets above and inject spammers who are trying to manipulate the ratings for a subset of items. We then measure how much these spammers effect the fit of our model on these items.



(a) Dumb spammers

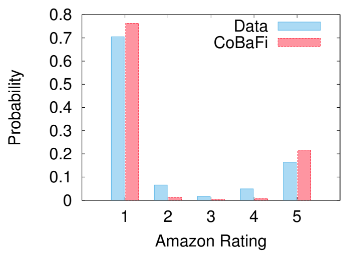


(b) Hijacked users

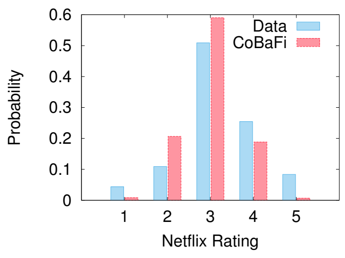


(c) Spammed movies

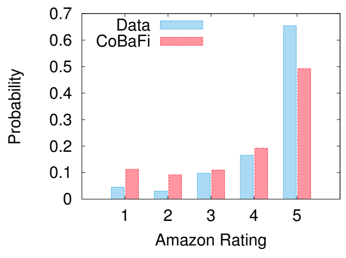
Figure 6.5: **Distribution of spammers over clusters**; Left: *Dumb spammers* on the electronics dataset; Middle: *Hijacked users* on the electronics dataset; Right: *Spammed movies* on the Netflix-24k spam dataset. In all three cases the spammers congregate tightly in a very small number of clusters.



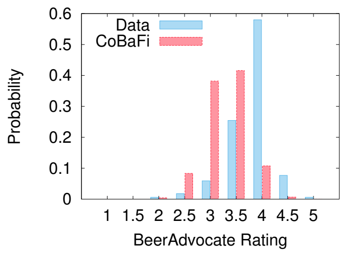
(a) Amazon Electronics



(b) Netflix



(c) Amazon Clothing



(d) BeerAdvocate

Figure 6.6: **Rating distributions from the data and as estimated by CoBaFi**. From left to right: *Sony 50CDQ80RS 80 Minute 700 MB 48x CD-R 50 Pack Spindle* on Amazon Electronics (the most skewed distribution); *The Rookie* on Netflix (the most Gaussian distribution); *Vanity Fair Women's My Favorite Illumination String Bikini Panty* on Amazon Clothing; *Ovila Abbey Saison, Sierra Nevada Brewing Co.* on BeerAdvocate.

Dumb spammers: For the first experiment we model spammers who are quite flagrant in how they try to manipulate certain reviews and do not camouflage their behavior by also providing typical ratings. To do this we use the Amazon Electronics dataset and see if we can take generally well-liked products and bring their ratings down (as a competitor would want). We select 31 products with a high average rating and at least

	PP Before	PP After
BPMF	1.7047	1.8146
CoBaFi	1.0549	1.7042

Table 6.3: **Predictive probability on "attacked items"** in the Amazon Electronics dataset before and after adding spammers.

	PP Before	PP After
BPMF	1.2375	1.3057
CoBaFi	0.9670	1.2935

Table 6.4: **Predictive probability on "attacked movies"** in the Netflix-24k dataset before and after add spam from hijacked accounts.

100 reviews. These products include Logitech speakers, an HDMI cable, a TiVo USB Network Adapter, and an HP LaserJet Printer, among many others. For these products we randomly select 100 ratings for each project to be included in the training dataset and the rest to be held out for the test dataset. We subsequently create 100 new accounts, and for all of these products our spam accounts give a rating of 1.0. As such, each product being manipulated now has half real reviews and half fake. As can be seen in Table 6.3, the predictive probability of our model is better than that for BPMF before and after we add the spam. Additionally, while the model is clearly effected by the spammers as it shifts to cover both the 1 star and 5 star reviews, it does an even better job of clustering spammers and the products they attack. As can be seen in Figure 6.5(a), most of the spammers (83%) are placed in the same clustered. Additionally, not graphed due to its simplicity, the attacked products are *all* placed into the same cluster. In an industry setting, this sort of interpretability and ability to understand group behavior would be valuable for investigating suspicious behavior.

Spammers with hijacked accounts: Second we test the model's robustness to more clever spammers—those who hijack real accounts. In this case we take real users and assume their account has been hijacked and begin providing misleading dubious reviews. This is a common issue in online systems and is the worst-case form of spammers who add realistic-looking reviews to try to camouflage their fraudulent behavior. To test our model in this setting we use the Netflix-24k dataset. Here we choose 85 movies with an average rating over 4.3 and with at least 200 reviews. Additionally we select 99 users at random (their average number of ratings is approximately 209). As before, for each of the movies selected to be attacked we include 100 of their ratings at random in the training set and use the rest for the test set. For the hijacked accounts we merely add 1 star reviews to all of the movies attacked. Again, we see in Table 6.4 that our model has a better predictive probability than BPMF both before and after we add the spam.

More impressively, CoBaFi is still successful in clustering together the hijacked users and the attacked movies, as seen in Figures 6.5(b–c). Note, this is particularly surprising for the hijacked users since they may have hundreds or thousands of legitimate reviews along with the only 85 fake reviews that contribute to their profile. From this we see that CoBaFi naturally handles spammers in a way that minimizes their impact on collaborative filtering and groups them together.

6.8.4 Natural clusters in real world

As mentioned earlier, being able to understand your data and interpret your latent parameters is useful in many applications and industry settings. Besides clustering anomalous behavior, our model provides interesting insight into the natural clusters of items based on their latent preferences in v_j.

We analyze the clusters produced from the Netflix-48k dataset. In fitting our model, we set the maximum number of clusters to 50, which given the number of movies/beers being rated leaves a still fairly coarse clustering. In order to isolate the effects of the clustering mechanism and not the flexible distribution, we use a Normal as our recommender distribution so that clusters are based only on v_j. In the Netflix-48k dataset we analyzed the clustering in two different ways. In Figure 6.4, we look at the distribution of different series across different clusters. We see that for most of the series nearly all of the seasons or films in that series are clustered together. Additionally, we see in Table 6.5 that clusters often contain movies within the same genre, with Cluster 28 containing generally comedies, Cluster 30 containing material for children, and Cluster 48 including science fiction. This sort of interpretable model is of course useful in practice to understand your data and user preferences.

Cluster 28	Cluster 30	Cluster 48
Simpsons	Scooby Doo	Star Trek
Family Guy	Spy Kids	Back to the Future
Monty Python	Stuart Little	Southpark
Curb your Enthusiasm	Dr. Dolittle	Lord of the Rings
The Twilight Zone	Lion King	Harry Potter
Arrested Development	Agent Cody Banks	The X-Files
Chappelle Show		
Monty Python		
Seinfeld		

Table 6.5: **Clusters of movies and shows from Netflix**. Series are only included if there are at least 2 items from that show or series in the cluster.

Amazon Clothing	Bra Disc Nipple covers
	Vanity Fair Women's String Bikini Panty
	Lee Men's Relaxed Fit Tapered Jean
	Carhartt Men's Dungaree Jean
	Wrangler Men's Cowboy Cut Slim Fit Jean
Amazon Electronics	Sony CD-R 50 Pack Spindle
	Olympus Stylus Epic Zoom Camera
	Sony AC Adapter Laptop Charger
	Apricorn Hard Drive Upgrade Kit
	Corsair 1GB Desktop Memory
BeerAdvocate	Weizenbock (Sierra Nevada)
	Ovila Abbey Saison (Sierra Nevada)
	Stoudt's Abbey Double Ale
	Stoudt's Fat Dog Stout
	Juniper Black Ale

Table 6.6: **Extreme distributions in data**: Items for Amazon Clothing and Amazon Electronics are those with the largest amount of skew, i.e. large positive θ_2 in the recommender distribution. This occurs for items that are universally liked, universally despised, or items that polarize. Alternatively, the beers from BeerAdvocate are those that are least skewed and have the largest negative θ_2.

6.8.5 Shape of real world data

The most interesting interpretable parameter of our model is understanding the wide range of distributions. In analyzing our results, we quickly see the world is not so normal. For each of our datasets we calculate the convexity of each item, $\hat{\theta}_j^{(2)}$:

$$\hat{\theta}_j^{(2)} = v_j^{(2)} + \frac{1}{|\text{ratings}(j)|} \sum_{i \in \text{ratings}(j)} u_i^{(2)}$$

With this term, we can generally see how convex ("U"-shaped) or concave (Gaussian) our item ratings actually are.

Beer Ratings: The simplest of the results, though surprising given the others, was with the BeerAdvocate ratings. For *all* beers we found $\hat{\theta}_j^{(2)} < 0$, meaning that the distribution was always Gaussian. Additionally, the distributions were not overly skewed as in other datasets; the items with the smallest values of $\hat{\theta}_j^{(2)}$ and thus the least variance were typically centered at 4.0 or less as seen in Figure 6.6(d). Looking at the distribution of variances $\hat{\theta}_j^{(2)}$ we see a fairly wide range of variances across the beers. This explains why our model has such an improved predictive probability than BPMF

Most skewed	Most Gaussian
The O.C. Season 2	The Rookie
Samurai X: Trust and Betrayal	The Fan
Aqua Teen Hunger Force, Vol. 2	Cadet Kelly
Sealab 2001: Season 1	Money Train
Aqua Teen Hunger Force: Vol. 1	Alice Doesn't Live Here
Gilmore Girls: Season 3	Sea of Love
Felicity: Season 4	Boiling Point
The O.C.: Season 1	True Believer
The Shield: Season 3	Stakeout
Queer as Folk: Season 4	The Package

Table 6.7: **A list of the most skewed and most Gaussian rating distributions on Netflix**. Note that the skewed movies are all exclusively TV shows whereas the tightly peaked 'Gaussian' ratings all correspond to blockbuster movies.

as shown above and demonstrates the importance of fitting the variance as well as the mean.

Netflix Ratings: Within the Netflix dataset, the results were not nearly as simple. Here we found that, possibly to the surprise of many data-miners, for most items on Netflix, $\hat{\theta}_2^{(2)} < 0$ and thus the distribution is Gaussian. Taking a closer look at the skews, we find some interesting patterns. In particular, as seen in Table 6.7, television shows were the most convex. For nearly all of these cases, we see that the shows produce highly polarized results that are usually heavily skewed in one direction. Alternatively, the items that were fitted to be the most Gaussian were mediocre movies such as "The Rookie." A comparison of the distributions can be seen in Figure 6.6(b). Intuitively, this makes sense. Watching a full television series is a lot more time consuming than viewing an individual movie, and users are probably more likely to self select, especially before rating a later season.

Amazon Ratings: For both the Amazon Electronics and Amazon Clothing datasets, we found that they exhibited very skewed distributions that were not Gaussian. Items were frequently viewed as very good, or very bad, with variances that were not well approximated by a normal distribution. Deviations were less likely to occur naturally. Furthermore, this makes sense as users often do not go online and rate an item they bought previously unless they have a strong reason for wanting to do so, e.g., the product is defective. In Figures 6.6(a,c) we see that our model handles the skew and bimodal distribution well. Additionally in Table 6.6 we list the most skewed distributions within both the Amazon Clothing and Amazon Electronics datasets.

Possible Explanations: We can hypothesize that such a difference in behavior can be attributed to a number of biases. For example, it is likely that only fans of a TV series will rate a later season of the show. Additionally, in rating the show is the rating a statement of (a) “This was a good/bad season of this show,” (b) “This was a good/bad season of TV,” or (c) “This was a good/bad piece of video entertainment?” Similarly, the selection bias arising from the additional effort to watch a full season of a show may also arise in the additional effort to go online and rate a product on Amazon. Additionally, we may only be able to compare quality against highly similar items as has been found in pricing studies in behavioral economics. From this we see polarized views from items that are difficult to evaluate or we have fewer instances to compare against, e.g., printers, seasons of a TV show, jeans, and more Gaussian ratings from items that we can compare against a wide variety of past experiences, e.g., comedy movies, action movies, beers.

While there is no definitive explanation for why these patterns emerge, it is clear that ratings have complex meaning. While our model is effective at fitting the wide range of distributions generated from such varied motives, we believe these discovered discrepancies demonstrate the need to further understand online ratings and better understand the latent factors that contribute to rating decision.

6.9 Summary

In this chapter we demonstrated that a single model suffices to provide both good recommendation performance, the ability to capture nontrivial bimodal and skewed distributions, the ability to analyze and group users and movies, and a mechanism for detecting spammers. Often these four problems are addressed by separate tools, often even by separate teams in commercial contexts. Our research shows that this is not needed. Instead, it is likely better to jointly model all these effects in a moderately detailed statistical generative process. While this may be costlier than each individual specialized solution, the overall engineering cost of a joint model is considerably less than maintaining these specialized methods. Furthermore, additional data can help improve *all* aspects simultaneously. We anticipate that large integrated data modeling will become a pervasive trend for recommendation, ranking and content analysis.

Chapter 7

Interpretable Recommendations

Matrix completion and approximation are popular tools to capture a user's preferences for recommendation and to approximate missing data. Instead of using low-rank factorization we take a drastically different approach, based on the simple insight that an additive model of co-clusterings allows one to approximate matrices efficiently. This allows us to build a concise model that, per bit of model learned, significantly beats all factorization approaches in matrix completion. Even more surprisingly, we find that summing over small co-clusterings is more effective in modeling matrices than classic co-clustering, which uses just one large partitioning of the matrix.

We provide our model ACCAMS, which stands for Additive Co-Clustering to Approximate Matrices Succinctly, along with an iterative minimization algorithm. We also offer a generative Bayesian form of our model, bACCAMS, for which we provide a collapsed Gibbs sampler. For both models we provide excellent empirical evidence for the efficacy of our approach. We match state-of-the-art results for matrix completion on Netflix at a fraction of the model complexity. Following Occam's razor principle, the fact that our model is more concise and yet just as accurate as more complex models suggests that it better captures the latent preferences and decision making processes present in the real world.

7.1 Introduction

As described previously, a wide variety of collaborative filtering approaches have been proposed to improve recommendation quality. Top recommender systems have used thousands of factors per item and per user, as was the case in the winning submissions in the Netflix prize [123]. Recent state-of-the-art methods have relied on learning even larger, more complex factorization models, often taking nontrivial combinations of multiple submodels [136, 150]. Such complex models are increasingly difficult to interpret, use large amounts of memory, and are often difficult integrate into larger systems.

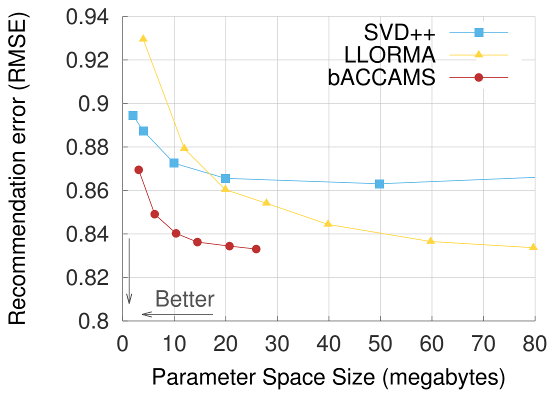


Figure 7.1: **Accuracy of bACCAMS on Netflix**, compared to [123] and [136]. Note that our model matches state of the art accuracy at a fraction of the model size.

7.1.1 Linear combinations of attributes

Our approach is drastically different from previous collaborative filtering research. Rather than start with the assumptions of a matrix factorization model, we make *co-clustering* effective for high quality matrix completion and approximation. Co-clustering finds a clustering of the rows and columns of a matrix $\mathbf{R}$ so as to partition $\mathbf{R}$ into blocks that are highly similar. It has been well studied [27, 191] but was not previously competitive in large behavior modeling and matrix completion problems. To achieve state of the art results, we use an *additive model of simple co-clusterings* that we call stencils, rather than building a large single co-clustering. The result is a model that is conceptually simple, has a small parameter space, has interpretable structure, and still matches the accuracy of other state-of-the-art methods for matrix completion on Netflix, as seen in Figure 7.1.

Using a linear combination of co-clusterings corresponds to a rather different interpretation of user preferences and movie properties. Matrix factorization assumes that a movie preference is based on a weighted sum of preferences for different genres, with the movie properties being represented in vector form. Therefore, even if a movie is a comedy and the user likes comedies, the model still also offsets by how scary the movie is and does the user like scary movies.

Co-clustering on the other hand assumes there exists some "correct" partitioning of movies (and users). For instance, a user might be part of a group that likes all comedies but does not like romantic movies. Correspondingly, all romantic comedies might be split out into a separate cluster. If a group of users likes new movies but not old ones then each genre may be further partitioned by decade. This quickly leads to a combinatorial explosion.

By taking a linear combination of co-clusterings we benefit from both perspectives: by modeling the discrete nature of attributes we can avoid the cost of high-dimensional factorization models, and by adding the preferences for different attributes we can avoid the large models necessary to cover all combinations of attributes. Rather, through backfitting, we create a more powerful, hierarchical representation. For instance, a movie may be {funny, scary, sad}, it was made in some era, it has a certain age rating, it may contain a certain group of actors or be shot in a certain visual style. However, if a user likes comedies but doesn't like scary movies, it is generally unlikely that this preference will suddenly flip depending on the decade the movie was produced. Therefore, by taking linear combinations of co-clusterings we can efficiently take all of these attributes into account.

7.1.2 Succinct Stencils

The mathematical challenge that motivated this work is, how can we make a *succinct* model of user behavior that still provides high quality predictions? Designing a succinct model is difficult because it requires making assumptions and restrictions while not decreasing the model's accuracy. Factorization models are very flexible. In order to encode a rank-k matrix by a factorization, we need k numbers per row (and column) respectively. Rather, consider a stencil - a small $k \times k$ template of a matrix and its mapping to the row and column vectors respectively. We only need $\log_2 k$ bits per row (and column) plus $O(k^2)$ floating point numbers regardless of the size of the matrix.

Taking a linear combination of stencils we can model quite complex matrices. This is best understood by the example below: assume that we have two simple stencils containing 3×2 and 2×3 co-clusterings. Their linear combination yields a rather nontrivial 9×8 matrix of rank 5. Alternatively, classic co-clustering would require a $(3 \cdot 2) \times (2 \cdot 3)$ partitioning to match this structure. When we have S stencils of size $k \times k$, this would require a single co-clustering of size $k^S \times k^S$.

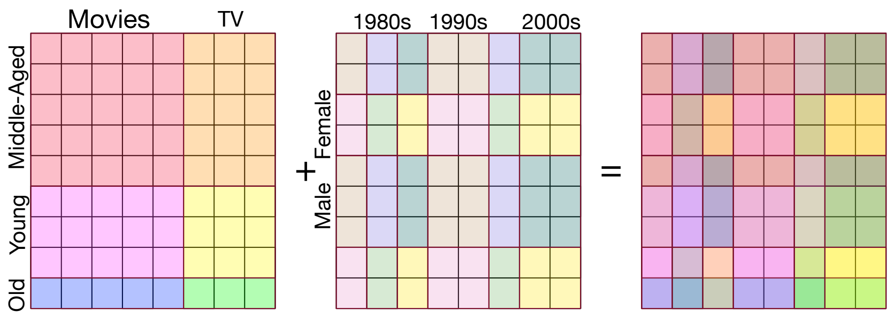


Figure 7.2: Additive co-clustering example

By design our model has a parameter space that is an order of magnitude smaller than competing methods, requiring only $S \log_2 k$ bits per user and per movie and Sk^2 floating point numbers, where k is generally quite small. While ACCAMS is more restrictive than classic factorization, we demonstrate that our assumptions do not increase the generalization error, achieving even better prediction accuracy than more complex models.

Finding succinct models for binary matrices, e.g., by minimizing the minimum description (MDL), has been the focus of significant research and valuable results in the data mining community [124, 216]. That said, these models are quite different. To the best of our knowledge, ours is the first work aimed at finding a parsimonious model for general (real-valued) matrix completion and approximation.

7.1.3 Contributions

We make a number of contributions to the problem of finding succinct representations of matrices.

- **Optimization algorithm:** In Section 7.3, we present ACCAMS, an iterative k-means style algorithm that minimizes the approximation error by backfitting the residuals of previous approximations.
- **Bayesian model:** In Section 7.4, we present a generative Bayesian non-parametric model and devise a collapsed Gibbs sampling algorithm, bACCAMS, for efficient inference.
- **State-of-the-art results:** Experiments confirm the efficacy of our approach, offering high accuracy matrix completion on Netflix, an interpretable hierarchy of content, and succinct matrix approximations for ratings, image, and network data.

We believe that these contributions offer a promising new direction for behavior modeling and matrix approximation.

Outline. We begin by discussing related work from recommender systems, non-parametric Bayesian models, co-clustering, and minimum description length. We subsequently introduce the simple k-means style additive co-clustering in Section 7.3. Subsequently, in Section 7.4.1 we define our Bayesian co-clustering model and collapsed Gibbs sampler for a single stencil. In Section 7.4.4 we extend our Bayesian model to multiple stencils. Section 7.5 reports our experimental results and we conclude with a discussion of future directions for the work.

7.2 Related Work

In this chapter, we build on the intuition from many different strands of the recommender systems literature, outlined in Section III.1. Most significantly, we build on the co-clustering intuition of CoBaFi in the previous chapter. However, while the co-clustering improved modeling accuracy, it did not reduce the model complexity of the underlying factorization.

As was previously mentioned, ensemble approaches to recommender systems have recently been found to work well [136, 150]. Our work in this chapter could be viewed as an ensemble of multiple co-clusterings. [221] also exploits co-clustering ensembles, but does so by finding a single consensus co-clustering. As far as we know, ours is the first work to use an additive combination of co-clusterings.

Methodologically, our model also follows the Bayesian non-parametric literature. [84, 87], which use an IBP for recommendation, can be seen as an extreme case of ACCAMS where the cluster size $k = 2$, while using a somewhat different strategy to handle cluster assignment and overall similarity within a cluster. Following a similar intuition as ACCAMS but different perspective and focus, [170] extended the IBP to handle $k > 2$ for link prediction tasks on binary graphs. Our work differs in its focus on general, real-valued matrices, its application of co-clustering, and its significantly simpler parameterization.

Succinct modeling The data mining community has focused on finding succinct models of data, often directly optimizing the model size described by the minimum description length (MDL) [187]. This approach has led to valuable results in pattern and item-set mining [216, 219] as well as graph summarization [124]. While we follow the intuition of this literature, these previous approaches typically focus on modeling databases of discrete items rather than real-valued datasets with missing values.

Additive Clustering Our model, built on additive co-clustering, takes a different approach from classic collaborative filtering literature. Our approach is conceptually similar to older literature from psychology on additive clustering [137, 164, 194, 210]. ADCLUS [194] argues that similarity between objects should be based on similarity on a subset of discrete properties. This perspective reinforces our belief that additive co-clustering is a good fit for user behavior modeling.

7.3 Matrix Approximation

We begin by defining our notation and the problem. A full list of the symbols used can be found in Table 7.1.

We consider now the problem of matrix approximation:

Problem Definition 1 ***(Matrix Approximation)***
Given: *a sparse matrix $\mathbf{R} \in \mathbb{R}^{N \times M}$ with indicator matrix $\mathbf{I}$*
Find: *a model $\mathcal{M}$ with parameters θ, such that the size of θ is small, $|\theta| \ll \mathbf{R}$, and $\mathcal{M}(\theta)$ approximates $\mathbf{R}$ well:*

$$\underset{\theta}{\text{minimize}} \sum_{u=1}^{N} \sum_{m=1}^{M} \mathbf{I}_{u,m}(\mathbf{R}_{u,m} - \mathcal{M}(\theta)_{u,m})^2 \tag{7.1}$$

As stated above, we assume $\mathbf{R}$ is a sparse matrix where there is no data for many values in the matrix. The indicator matrix $\mathbf{I}$ denotes this information, where $\mathbf{I}_{i,j} = 1$ when there is an observed value for $\mathbf{R}_{i,j}$ and $\mathbf{I}_{i,j} = 0$ otherwise. Without loss of generality we assume that our data matrix, $\mathbf{R}$, has more rows than columns, i.e. $\mathbf{R} \in \mathbb{R}^{N \times M}$ with $N \geq M$.

The concept of a small model in the context of behavior modeling has typically been captured by the rank of a factorization. We generalize this concept and define a small model by the number of bits required to store it, more commonly known as the minimum description length (MDL).

Symbol	Definition
N, M	Number of rows (users) and columns (movies)
$\mathbf{R}$	Data matrix $\in \mathbb{R}^{N\times M}$ (with missing values)
$\mathbf{I}$	Indicator matrix $\in \{0,1\}^{N\times M}$ for $\mathbf{R}$
S	Number of stencils
$k_n^{(\ell)}, k_m^{(\ell)}$	Number of user and movie clusters in stencil ℓ
$\mathbf{T}^{(\ell)}$	Matrix $\in \mathbb{R}^{k_n^{(\ell)}\times k_m^{(\ell)}}$ for stencil ℓ
$\mathbf{c}^{(\ell)}$	Vector of user assignments $\in \{1,\ldots,k_n^{(\ell)}\}^N$
$\mathbf{d}^{(\ell)}$	Vector of movie assignments $\in \{1,\ldots,k_m^{(\ell)}\}^M$
$\mathcal{S}(\mathbf{T},\mathbf{c},\mathbf{d})$	$\in \mathbb{R}^{N\times M}$ defined by $\mathcal{S}(\mathbf{T},\mathbf{c},\mathbf{d})_{u,m} = \mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}$
$\mathbf{n}_c$	Number of users in cluster c
$\mathbf{n}_c^{(-u)}$	Number of users in cluster c, ignoring user u
$\mathbf{N}_{c,d}$	Number of observations in block (c,d)
$\mathbf{r}_{c,d}$	Vector of observed ratings in block (c,d)

Table 7.1: Symbols used throughout this chapter.

7.3.1 Proposed Model

A stencil assigns each user u to a user cluster c, each movie to a movie cluster d, and for each (user cluster c, movie cluster d) combination there is a *block* of ratings which are predicted to have value $\mathbf{T}_{c,d}$. Formally:

Definition 6 (Stencil) *A stencil $\mathcal{S}(\mathbf{T},\mathbf{c},\mathbf{d})$ is a matrix $\mathcal{S} \in \mathbb{R}^{N\times M}$ with the property that $\mathcal{S}(\mathbf{T},\mathbf{c},\mathbf{d})_{u,m} = \mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}$ for a template $\mathbf{T} \in \mathbb{R}^{k_n\times k_m}$ and discrete index vectors $\mathbf{c} \in \{1,\ldots,k_n\}^N$ and $\mathbf{d} \in \{1,\ldots,k_m\}^M$ respectively.*

Therefore, our goal is to find a stencil $\mathcal{S}(\mathbf{T},\mathbf{c},\mathbf{d})$ with a small approximation error $\mathbf{R} - \mathcal{S}(\mathbf{T},\mathbf{c},\mathbf{d})$ and small cost for storing $\theta = \{\mathbf{T},\mathbf{c},\mathbf{d}\}$.

Lemma 9 (Compression) *Stencil $\mathcal{S}(\mathbf{T},\mathbf{c},\mathbf{d})$ can be stored with $N\log_2 k_n$ bits for row cluster assignments $\mathbf{c}$, $M\log_2 k_m$ bits for column cluster assignments $\mathbf{d}$ and $32k_nk_m$ bits to store a 32-bit floating point number for each block in $\mathbf{T}$:*

$$\text{Bits}(\{\mathbf{T},\mathbf{c},\mathbf{d}\}) = N\log_2 k_n + M\log_2 k_m + 32k_nk_m \tag{7.2}$$

Note, it is trivial to get zero approximation error by setting $k_n = N$ and $k_m = M$, but this creates a very large model (the size of the original data) that is not useful.

As mentioned above, we can efficiently improve the approximation accuracy by using

Algorithm 8: Matrix Approximation with ACCAMS

```
Require: matrix $\mathbf{R}$, indicator matrix $\mathbf{I}$, clusters $k_n, k_m$, max stencils $S$
 1: $\hat{\mathbf{R}} \leftarrow \mathbf{R}$
 2: for $\ell = 1$ to $S$ do
 3:     $(\mathbf{c}^{(\ell)}, \mathbf{V}^{(\text{row})}, \mathbf{L}) \leftarrow \textsc{Cluster}(\hat{\mathbf{R}}, k_n, \mathbf{I})$ {Rows}
 4:     $(\mathbf{d}^{(\ell)}, \cdot, \cdot) \leftarrow \textsc{Cluster}([\mathbf{V}^{(\text{row})}]^\top, k_m, \mathbf{L}^\top)$ {Columns}
 5:     for all $a, b \in \{1, \dots k_n\} \times \{1, \dots k_m\}$ do
 6:         $\mathbf{T}^{(\ell)}_{a,b} \leftarrow \text{mean}\left\{\hat{\mathbf{R}}_{u,m} | \mathbf{c}^{(\ell)}_u = a \text{ and } \mathbf{d}^{(\ell)}_m = b\right\}$
 7:     end for
 8:     $\hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}} - \mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ {Backfit on residuals}
 9: end for
10: return $\{\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}\}_{\ell=1}^S$
```

multiple stencils:

$$\underset{\{\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}\}}{\text{minimize}} \sum_{u=1}^{N} \sum_{m=1}^{M} \mathbf{I}_{u,m} \left(\mathbf{R}_{u,m} - \sum_{\ell=1}^{S} \mathcal{S}\left(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}\right)_{u,m} \right)^2$$

That is, we would like to find an additive model of S stencils that minimizes the approximation error to $\mathbf{R}$.

Given $\mathbf{R}$, it is our goal to *find* such stencils $\mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ with good approximation properties. Unfortunately, finding linear combinations of co-clusterings is NP-hard. It is easy to see this by reducing co-clustering, which is NP-hard, to our problem by setting $S = 1$. We describe below two algorithms to learn our stencils that offer good approximation guarantees and work well in practice.

7.3.2 Algorithm

We consider a simple iterative procedure to learn stencils to approximate our data $\mathbf{R}$. Algorithm 8 gives the high level algorithm of learning each stencil one at a time. In learning each stencil we use the CLUSTER algorithm, similar to k-means, as given in Algorithm 9.

Row clustering We first perform k-means clustering of the rows. That is, we aim to find an approximation of $\mathbf{R}$ that replaces all rows by a small subset thereof. Algorithm 9 is essentially a generalization of k-means clustering. By calling the algorithm with $\mathbf{M} = \mathbf{R}$, $k = k_n$, and $\mathbf{W} = \mathbf{1}^{N \times M}$, we find that the algorithm simplifies significantly to classic k-means. The cluster assignment in Line 5 is simply

$$\mathbf{c}_u \leftarrow \underset{c}{\text{argmin}} \, \|\mathbf{R}_{u,:} - \mathbf{v}_c\|_2^2 \tag{7.3}$$

and $\mathbf{L}$ stores the number of rows in each row cluster.

Algorithm 9: CLUSTER$(\mathbf{M}, k, \mathbf{W})$

```
Require: matrix $\mathbf{M} \in \mathbb{R}^{N_1 \times N_2}$, weights $\mathbf{W} \in \mathbb{R}^{N_1 \times N_2}$, number of clusters $k$
 1: Draw $k$ rows from $\mathbf{M}$ at random without replacement and copy them to
    $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \in \mathbb{R}^{k \times N_2}$.
 2: while not converged do
 3:   $\mathbf{L} \leftarrow \mathbf{0} \in \mathbb{R}^{k \times N_2}$ and $\mathbf{Y} \leftarrow \mathbf{0} \in \mathbb{R}^{k \times N_2}$
 4:   for $i = 1$ to $N_1$ do
 5:     $\mathbf{c}_i \leftarrow \mathrm{argmin}_c \sum_j \mathbf{W}_{i,j}(\mathbf{M}_{i,j} - \mathbf{V}_{c,j})^2$
 6:     $\mathbf{Y}_{\mathbf{c}_i,:} \leftarrow \mathbf{Y}_{\mathbf{c}_i,:} + \mathbf{M}_{i,:}$ {Increment statistics}
 7:     $\mathbf{L}_{\mathbf{c}_i,:} \leftarrow \mathbf{L}_{\mathbf{c}_i,:} + \mathbf{I}_{i,:}$ {Increment counts}
 8:   end for
 9:   for $c = 1$ to $k$ do
10:     $\mathbf{V}_{c,:} \leftarrow \mathbf{Y}_{c,:}/\mathbf{L}_{c,:}$ {New cluster center}
11:   end for
12: end while
13: return cluster assignments $\mathbf{c}$, clusters $\mathbf{V}$, counts $\mathbf{L}$
```

Column clustering Once we have run the clustering algorithm on the rows, we now cluster the columns of previously learned row clusters, $\mathbf{V}^{(\text{row})}$. In this case, we need to weight each row of $\mathbf{V}^{(\text{row})}$ (corresponding to a row cluster) by the number of rows it represents (the number of rows in that cluster). As a result, rather than choose a column cluster assignment by the Euclidean distance, we use the Mahalanobis distance. Still, Algorithm 9 is a generalization of this concept. We use the assignment (for Line 5):

$$\mathbf{d}_m \leftarrow \underset{d}{\mathrm{argmin}} \left(\mathbf{V}^{(\text{row})}_{:,m} - \mathbf{V}_{:,d}\right)^\top \mathbf{D} \left(\mathbf{V}^{(\text{row})}_{:,m} - \mathbf{V}_{:,d}\right) \tag{7.4}$$

where $\mathbf{V} \in \mathbb{R}^{k_n \times k_m}$ is the matrix obtained by stacking $\mathbf{V}_{:,d} = \mathbf{v}_d^\top$ and $\mathbf{D}$ would be the diagonal matrix of counts, i.e. $\mathbf{D}_{cc}$ is the number of rows of $\mathbf{R}$ in cluster c.

Missing entries In many cases, however, $\mathbf{R}$ itself is incomplete. This is addressed quite easily by using the assignment shown in Line 5 of Algorithm 9, where $\mathbf{W}$ can be set to the adjacency matrix to ignore missing entries. Therefore, in finding a good cluster for the row $\mathbf{R}_{u,:}$ we restrict ourselves to the coordinates in $\mathbf{V}_{c,:}$ where $\mathbf{R}_{u,m}$ exists (and also where $\mathbf{V}_{c,m}$ has been initialized).

For the purpose of obtaining the column clusters, we now need to weight each coordinate in $\mathbf{V}^{(\text{row})}$ by how many elements in $\mathbf{R}$ contributed to it. Correspondingly denote by $\mathbf{L}_{c,m} := \sum_{(u,m) \in \mathbf{R}: \mathbf{c}_u = c} 1$ the number of entries mapped into coordinate $\mathbf{V}^{(\text{row})}_{c,m}$. Then the assignment for column clusters is obtained via

$$\mathbf{d}_m \leftarrow \underset{d}{\mathrm{argmin}} \sum_c \mathbf{L}_{c,m} \left(\mathbf{V}^{(\text{row})}_{c,m} - \mathbf{V}_{c,d}\right)^2 \tag{7.5}$$

We observe that Algorithm 9 handles both row and column clustering correctly, by calling it appropriately as shown in Algorithm 8.

Backfitting The outcome of running the row and column clustering described above on $\mathbf{R}$ is a single stencil $\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$ consisting of the clusters obtained by first row and then column clustering. It may be desirable to alternate between row and column clustering for further refinement.

However, as noted above, additional stencils only reduce the objective function further—convergence to a local minimum is assured, with the same caveat on solution quality as in k-means clustering. Therefore, we take the residual $\hat{\mathbf{R}} = \mathbf{R} - \mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$ and use it as the starting point to learn a new stencil. By repeatedly learning new stencils on the residuals of the previous stencils, as shown in Algorithm 8, we obtain an additive model of co-clusterings that minimizes the approximation error to $\mathbf{R}$.

7.3.3 Approximation Guarantees

A key question is how well any given matrix $\mathbf{R}$ can be approximated by an appropriate stencil. For the sake of simplicity we limit ourselves to the case where all entries of the matrix are observed. Using covering numbers and the spectral properties of $\mathbf{R}$, we can obtain approximation guarantees for co-clustering. Previous results have offered guarantees for how well k-means co-clustering, similar to Algorithm 9, can approximate the optimal co-clustering [18]. We offer below a theorem bounding the approximation error between a matrix and a stencil approximating it, based on the singular values of the particular data matrix.

Denote by $\mathbf{\Sigma} = \mathrm{diag}(\sigma_1, \ldots \sigma_n)$ the singular values of $\mathbf{R}$.

Theorem 2 (Approximation Guarantees) *Using k clusters for both rows and columns, the matrix $\mathbf{R}$ can be approximated with error at most*

$$\|\mathbf{R} - \mathbf{R}'\|_\infty \leq 2 \|\mathbf{R}\|^{\frac{1}{2}} \epsilon_k \left(\mathbf{\Sigma}^{\frac{1}{2}}\right)$$
$$\|\mathbf{R} - \mathbf{R}'\|_2 \leq (\sqrt{N} + \sqrt{M}) \|\mathbf{R}\|^{\frac{1}{2}} \epsilon_k \left(\mathbf{\Sigma}^{\frac{1}{2}}\right)$$

where

$$\epsilon_k(\mathbf{\Sigma}) \leq 6 \sup_{j \in \mathbb{N}} \left(\frac{1}{k} \prod_{i=1}^{j} \sigma_i \right)^{\frac{1}{j}} \leq 6\epsilon_k(\mathbf{\Sigma})$$

The proof for Theorem 2 can be found in Appendix B.

Note that the above is a statement of *existence* rather than a constructive prescription. However, because ACCAMS is more restrictive than matrix factorization, the above theorem, connecting co-clustering to singular values, bounds the error induced by co-clustering. In practice, the results can be considerably better, as we show in Section 7.5.

As this theorem does not offer any insight into the structure of the residuals left from co-clustering, an interesting future direction for theoretical analysis is to extend the proof to approximation guarantees for *additive* co-clustering.

7.4 Generative Model

As many frequentist algorithms have a Bayesian counterpart, we now devise a Bayesian counterpart to ACCAMS, which we will refer to as bACCAMS. We begin with the single stencil case, describing the model in Section 7.4.1 and a collapsed Gibbs sampler in Section 7.4.2. We then extend the model and sampler to many stencils in Section 7.4.4.

7.4.1 Co-Clustering with a Single Stencil

We begin with a simple Bayesian model of co-clustering. This is the basic template for single-stencil inference, and our model of additive co-clusters will use the same idea. Our model can be broken into two parts: (1) generating block values and (2) generating cluster assignments. We will go through each part of the model and then the model as a whole, as shown in Figure 7.3.

Block values We begin by considering how we generate the prediction $\mathbf{T}_{c,d}$ for a particular block, corresponding to ratings from users in cluster c to movies in cluster d. Previously, as shown in Line 6 of Algorithm 8, each block merely took the average of the values that fell in that block. Subsequently, we would use that block mean directly as the prediction for all values in the block.

In our Bayesian model, we consider each $\mathbf{T}_{c,d}$ to come from a Gaussian distribution, centered at 0 and with variance τ^2. On top of this, each value in the matrix $\mathbf{R}_{u,m}$ is generated by a Gaussian with mean $\mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}$ and variance σ^2 (alternatively stated, with additive noise $\epsilon_{u,m} \sim \mathcal{N}(0,\sigma^2)$). As such we have

$$\mathbf{T}_{c,d} \sim \mathcal{N}(0,\tau^2) \qquad\qquad \mathbf{R}_{u,m} \sim \mathcal{N}\left(\mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m},\sigma^2\right) \tag{7.6}$$

Because τ^2 and σ^2 are not yet defined and are data dependent, they too are sampled from the conjugate prior distribution, specifically the Inverse Gamma distribution:

$$\tau^2 \sim \mathrm{IG}(\gamma) \qquad\qquad \sigma^2 \sim \mathrm{IG}(\eta) \tag{7.7}$$

Cluster assignments In Algorithm 9, a user u (or movie m) is assigned to a particular cluster $\mathbf{c}_u$ ($\mathbf{d}_m$ for movies) based purely on the distance to the cluster center. As with most frequentist algorithms, there is no prior on the cluster assignments.

In our Bayesian model, we put a Dirichlet prior on the cluster assignments. More simply understood, we believe cluster assignments are generated by a Chinese Restaurant Process (CRP). One big advantage of using a CRP is that it allows for an unrestricted number of clusters, where new clusters can always be created with small probability. To

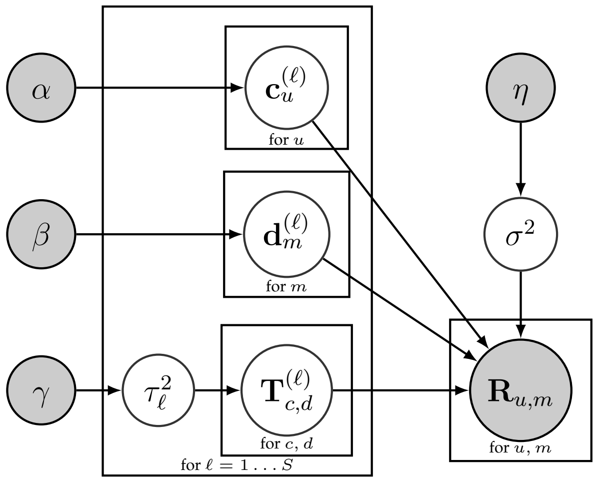


Figure 7.3: **Generative model for recommendation and matrix approximation (bAC-CAMS)**. For each stencil, as indexed by ℓ, row and cluster memberships $\mathbf{c}^{(\ell)}$ and $\mathbf{d}^{(\ell)}$ are drawn from a Chinese Restaurant Process. The values for the template $\mathbf{T}^{(\ell)}$ are drawn from a Normal Distribution. The observed ratings $\mathbf{R}_{u,m}$ are sums over the stencils $\mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$.

understand the Chinese Restaurant Process, consider the following metaphor. A user u is at a Chinese restaurant and trying to pick a table (cluster) at which to sit. Each table is chosen with probability proportional to the number of users already at that table (in that cluster), and a new table is started with some small probability, proportional to α. To be concrete, we assume there are N users, and cluster c has $\mathbf{n}_c$ users in it; $\mathbf{n}_c^{(-u)}$ is the number of users in cluster c not including user u. The probability that user u will go to a particular cluster c is given by:

$$p(\mathbf{c}_u = c|\mathbf{c}^{(-u)}, \alpha) = \begin{cases} \frac{\mathbf{n}_c^{(-u)}}{\alpha+N-1} & \text{if } \mathbf{n}_c^{(-u)} > 0 \\ \frac{\alpha}{\alpha+N-1} & \text{new cluster } c \end{cases} \tag{7.8}$$

An analogous expression is available for movies: $p(\mathbf{d}_m = d|\mathbf{d}^{(-m)}, \beta)$. Under this model, large values of α and β encourage the formation of larger numbers of clusters.

Complete model Putting these two elements together, we can now describe our complete Bayesian model for co-clustering:

$$\mathbf{c}_u \sim \mathrm{CRP}(\alpha) \qquad \mathbf{d}_m \sim \mathrm{CRP}(\beta) \tag{7.9a}$$
$$\mathbf{T}_{c,d} \sim \mathcal{N}(0, \tau^2) \qquad \mathbf{R}_{u,m} \sim \mathcal{N}\left(\mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}, \sigma^2\right) \tag{7.9b}$$
$$\tau^2 \sim \mathrm{IG}(\gamma) \qquad \sigma^2 \sim \mathrm{IG}(\eta) \tag{7.9c}$$

Consequently the joint probability distribution over all ratings, given the variances, is given by

$$p\left(\mathbf{R}, \mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})|\alpha, \beta, \sigma^2, \tau^2\right) = \mathrm{CRP}(\mathbf{c}|\alpha)\mathrm{CRP}(\mathbf{d}|\beta) \times \prod_{c,d} \tfrac{1}{\sqrt{2\pi\tau^2}} \exp\left(\tfrac{-\mathbf{T}_{c,d}^2}{2\tau^2}\right) \prod_{(u,m)} \tfrac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\tfrac{\left(\mathbf{R}_{u,m}-\mathbf{T}_{\mathbf{c}_u,\mathbf{d}_m}\right)^2}{2\sigma^2}\right) \tag{7.10}$$

This is an extremely simple model similar to [191], akin to a decision stump. The rationale for picking such a primitive model is that we will be taking linear combinations thereof to obtain a very flexible tool. We will now show that, by design, the model can be efficiently sampled.

7.4.2 Collapsed Gibbs Sampler

We now dive into the details of efficiently learning our model through a collapsed Gibbs sampler. There are three main parts of the model that need to be sampled: (1) the cluster assignments $\mathbf{c}$ and $\mathbf{d}$, (2) the block values $\mathbf{T}_{c,d}$, and (3) the variances τ^2 and σ^2. By design, in particular the use of conjugate priors, our model is efficient to sample. For sampling cluster assignments, we find that we can collapse out $\mathbf{T}_{c,d}$ so that sequential samples of cluster assignments are not based on stale values of $\mathbf{T}_{c,d}$ and we achieve a significantly faster sampler.

Inferring Cluster Assignments

We begin with the challenge of sampling the user cluster assignments $\mathbf{c}$ given only the data $\mathbf{R}$, the movie cluster assignments $\mathbf{d}$, and the priors α, σ^2, τ^2. To do this, we must collapse out the block values of $\mathbf{T}$.

In particular, at each step, we check how the likelihood of the data $\mathbf{R}$ changes by assigning a user (or movie) to a new cluster:

$$p(\mathbf{c}_u = c|\mathbf{R}, \mathbf{d}, \alpha, \sigma^2, \tau^2) \propto \mathrm{CRP}(c|\alpha) \frac{p(\mathbf{R}|\mathbf{c}^{(-u)}, \mathbf{c}_u = c, \mathbf{d}, \sigma^2, \tau^2)}{p(\mathbf{R}|\mathbf{c}^{(-u)}, \mathbf{d}, \sigma^2, \tau^2)}$$

We denote the observations for block (c, d), the ratings from users in cluster c to movies in cluster d, by the vector $\mathbf{r}_{c,d}$; the updated ratings in the block after the assignment is given by $\mathbf{r}'_{c,d}$. With this, the calculation above simplifies to:

$$p(\mathbf{c}_u = c|\mathbf{R}, \mathbf{d}, \alpha, \sigma^2, \tau^2) \propto \mathrm{CRP}(c|\alpha) \prod_d \frac{p(\mathbf{r}'_{c,d}|\sigma^2, \tau^2)}{p(\mathbf{r}_{c,d}|\sigma^2, \tau^2)}$$

As we shall see, this is easily calculated by keeping simple linear statistics of the ratings. Moreover, by integrating out $\mathbf{T}$ we avoid the problem of having to instantiate a new value whenever a new cluster is added.

For a given block (c, d) with associated $\mathbf{r}_{c,d}$, the distribution of ratings is Gaussian with mean 0 and with covariance matrix $\mathbf{\Sigma} = \sigma^2\mathbf{1} + \tau^2 11^\top$ (due to the independence of the variances and the additive nature of the normal distribution). Here we use $\mathbf{1}$ to denote the identity matrix and 1 to denote the vector of all 1. Denote by $\mathbf{N}_{c,d}$ the number of rating pairs (u, m) for which $\mathbf{c}_u = c$ and $\mathbf{d}_m = d$. Hence, the likelihood of the block (c, d), as observed in $\mathbf{r}_{c,d}$, is

$$p(\mathbf{r}_{c,d}|\sigma^2, \tau^2) = \frac{\exp\left[-\frac{1}{2}\mathbf{r}_{c,d}^\top \mathbf{\Sigma}^{-1}\mathbf{r}_{c,d}\right]}{(2\pi)^{\frac{\mathbf{N}_{c,d}}{2}} |\mathbf{\Sigma}|^{\frac{1}{2}}}$$

In computing the above expression we need to compute the determinant of $\mathbf{\Sigma}$, a diagonal matrix with rank-1 update, and the inverse of said matrix. For the former, we use the matrix-determinant lemma, and for the latter, the Sherman-Morrison-Woodbury formula:

$$\mathbf{r}_{c,d}^\top \mathbf{\Sigma}^{-1}\mathbf{r}_{c,d} = \frac{1}{\sigma^2}\left\|\mathbf{r}_{c,d}\right\|^2 - \frac{\tau^2}{\sigma^2}\cdot\frac{\left(1^\top \mathbf{r}_{c,d}\right)^2}{\sigma^2 + \mathbf{N}_{c,d}\tau^2}$$
$$\log|\mathbf{\Sigma}| = (\mathbf{N}_{c,d} - 1)\log\sigma^2 + \log\left[\sigma^2 + \mathbf{N}_{c,d}\tau^2\right]$$

This allows us to assess whether it is beneficial to assign a user u or a movie m to a different or a new cluster efficiently, since the only statistics involved in the operation are sums of ratings and of their squares, $1^\top \mathbf{r}_{c,d}$ and $\left\|\mathbf{r}_{c,d}\right\|^2$ respectively.

We denote by $\mathbf{N}'_{c,d}$ the new cluster count and by $\mathbf{r}'_{c,d}$ the new set of ratings, after having assigned user u to cluster c. Let

$$\Delta := \tfrac{\mathbf{N}'_{c,d} - \mathbf{N}_{c,d}}{2}\left[\log(2\pi) + \log\sigma^2\right] + \frac{1}{2\sigma^2}\left[\left\|\mathbf{r}'_{c,d}\right\|^2 - \left\|\mathbf{r}_{c,d}\right\|^2\right]$$

be a constant offset, in log-space, that only depends on the additional ratings that are added to a cluster. That is, Δ is *independent* of the cluster that the additional scores are assigned to and can be safely ignored. The result is:

$$p(\mathbf{c}_u = c|\mathbf{R}, \mathbf{d}, \alpha, \sigma^2, \tau^2) \propto \tfrac{\mathbf{n}_c^{(-u)}}{\alpha + N - 1}\prod_d \left[\frac{\sigma^2 + \mathbf{N}_{c,d}\tau^2}{\sigma^2 + \mathbf{N}'_{c,d}\tau^2}\right]^{\frac{1}{2}} \times \exp\left[\frac{\tau^2}{2\sigma^2}\sum_d \left[\frac{\left(1^\top \mathbf{r}'_{c,d}\right)^2}{\sigma^2 + \mathbf{N}'_{c,d}\tau^2} - \frac{\left(1^\top \mathbf{r}_{c,d}\right)^2}{\sigma^2 + \mathbf{N}_{c,d}\tau^2}\right]\right] \tag{7.11}$$

For a new cluster this can be simplified since there is no data, hence $\mathbf{N}_{c,d} = 0$ and $\mathbf{r}_{c,d} = []$.

$$p(\mathbf{c}_u = c_{\text{new}}|\mathbf{R}, \mathbf{d}, \alpha, \sigma^2, \tau^2) \propto \tfrac{\alpha}{\alpha + N - 1}\prod_d \left[\tfrac{\sigma^2}{\sigma^2 + \mathbf{N}'_{c,d}\tau^2}\right]^{\frac{1}{2}} \times \exp\left[\tfrac{\tau^2}{2\sigma^2}\sum_d \tfrac{\left(1^\top \mathbf{r}'_{c,d}\right)^2}{\sigma^2 + \mathbf{N}'_{c,d}\tau^2}\right] \tag{7.12}$$

The above expression is fairly straightforward to compute: we only need to track $\mathbf{N}_{c,d}$, i.e. the number of ratings assigned to a particular (user cluster, movie cluster) combination and $1^\top \mathbf{r}_{c,d}$, i.e. the sum of the ratings for this block.

Inferring Block Values

For the purpose of recommendation and for a subsequent combination of several matrices, we need to instantiate the block values $\mathbf{T}_{c,d}$. By checking (7.10) we see that $\mathbf{T}_{c,d}|\mathrm{rest}$ is given by

$$\mathbf{T}_{c,d}|\mathrm{rest} \sim \mathcal{N}\left(\frac{1^\top \mathbf{r}_{c,d}}{\rho \mathbf{N}_{c,d}}, \frac{\sigma^2}{\rho \mathbf{N}_{c,d}}\right) \text{ where } \rho = \left[1 + \frac{1}{\mathbf{N}_{c,d}} \frac{\sigma^2}{\tau^2}\right] \tag{7.13}$$

Here too sampling $\mathbf{T}_{c,d}$ only requires having the number of ratings $\mathbf{N}_{c,d}$ and sum of ratings in the block $1^\top \mathbf{r}_{c,d}$.

Inferring Variances

Last, we consider how to sample the variances for the priors, σ^2 and τ^2. Both σ^2 and τ^2 are generated by the Inverse Gamma distribution:

$$p(x|a,b) = b^a \Gamma^{-1}(a) x^{-a-1} e^{-\frac{b}{x}} \tag{7.14}$$

Denote by E the total number of observed values in $\mathbf{R}$. In this case, σ^2 is drawn from an Inverse Gamma prior with parameters (η_a', η_b'):

$$\eta_a' \leftarrow \eta_a + \frac{E}{2} \text{ and } \eta_b' \leftarrow \eta_b + \sum_{(u,m)} \mathbf{I}_{u,m}(\mathbf{R}_{u,m} - \mathbf{T}_{\mathbf{c}_u, \mathbf{d}_m})^2 \tag{7.15}$$

Analogously, we draw τ^2 from an Inverse Gamma with parameters

$$\gamma_a' \leftarrow \gamma_a + \frac{k_n k_m}{2} \text{ and } \gamma_b' \leftarrow \gamma_b + \sum_{c,d} \mathbf{T}_{c,d}^2 \tag{7.16}$$

k_n and k_m denote the number of user and movie clusters.

7.4.3 Efficient Implementation

With these inference equations we can implement an efficient sampler, as seen in Algorithm 10. The key to efficient sampling is to cache the per-cluster sums of ratings $1^\top \mathbf{r}_{c,d}$. Then reassigning a user (or movie) to a different (or new) cluster is just a matter of checking the amount of change that this would effect. Hence each sampling pass costs $O(k_n \cdot k_m \cdot (N + M) + E)$ operations. It is linear in the number of ratings and of partitions.

Note that once $\mathbf{y}^{(u)}$ and $\mathbf{l}^{(u)}$ are available for all users (or all movies), it is cheap to perform additional sampling sweeps at comparably low cost. It is therefore beneficial to iterate over all users (or all movies) more than once, in particular in the initial stages of the algorithm. Also note that the algorithm can be used on datasets that are being streamed from disk, provided that an index and an inverted index of M can be stored: we need to be able to traverse the data when ordered by users and when ordered by movies. It is thus compatible with solid state disks.

Algorithm 10: StencilSampler$(\mathbf{M}, \mathbf{T}, \mathbf{c}, \mathbf{d})$

```
1:  Initialize row-index and column-index of data in $\mathbf{M}$
2:  Initialize statistics for each partition
        $\mathbf{N}_{c,d} := |\{(u,m) : \mathbf{c}_u = c, \mathbf{d}_m = d\}|$ and $\mathbf{Y}_{c,d} := \sum_{(u,m):\mathbf{c}_u=c,\mathbf{d}_m=d} \mathbf{M}_{u,m}$
3:  while sampler not converged do
4:      for all users $u$ do
5:          For all movie clusters $\mathbf{d}$ compute the incremental changes
                $\mathbf{l}_d^{(u)} := |\{(u,m) : \mathbf{d}_m = d\}|$ and $\mathbf{y}_d^{(u)} := \sum_{(u,m):\mathbf{d}_m=d} \mathbf{M}_{u,m}$
6:          Remove $u$ from their cluster
                $\mathbf{N}_{\mathbf{c}_u,:} \leftarrow \mathbf{N}_{\mathbf{c}_u,:} - \mathbf{l}^{(u)}$ and $\mathbf{Y}_{\mathbf{c}_u,:} \leftarrow \mathbf{Y}_{\mathbf{c}_u,:} - \mathbf{y}^{(u)}$
7:          Sample new user cluster $\mathbf{c}_u$ using (7.11) and (7.12).
8:          Update statistics
                $\mathbf{N}_{\mathbf{c}_u,:} \leftarrow \mathbf{N}_{\mathbf{c}_u,:} + \mathbf{l}^{(u)}$ and $\mathbf{Y}_{\mathbf{c}_u,:} \leftarrow \mathbf{Y}_{\mathbf{c}_u,:} + \mathbf{y}^{(u)}$
9:      end for
10:     for all movies $m$ do
11:         Sample movie cluster assignments analogously.
12:     end for
13:     for all $(c,d)$ cluster partitions do
14:         Resample $\mathbf{T}_{c,d}$ using (7.13) and $\mathbf{N}_{c,d}, \mathbf{Y}_{c,d}$.
15:     end for
16:     Resample $\sigma^2$ and $\tau^2$ using (7.15) and (7.16).
17: end while
18: return $\mathbf{T}, \mathbf{c}, \mathbf{d}$
```

7.4.4 Additive Combinations of Stencils

As before, we find that using a linear combination of stencils is far more powerful than just a single stencil. Therefore, we enumerate the stencils by $\mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$, where stencil index ℓ ranges from 1 to S. Correspondingly we now need to sample from a set of $\mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ and τ_ℓ^2 *per* stencil. However, we keep the additive noise term $\mathcal{N}(0, \sigma^2)$ unchanged. This is the model of Figure 7.3. The additivity of Gaussians makes inference easy:

$$\mathbf{R}_{u,m} \sim \mathcal{N}\left(\sum_\ell \mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})_{u,m}, \sigma^2\right). \tag{7.17}$$

Note, though, that estimating $\mathcal{S}$ jointly *for all* indices ℓ is not tractable since various clusterings $(\mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ overlap and intersect with each other, hence the joint normal distribution over all variables would be expensive to factorize.

Algorithm 11: bACCAMS

```
1: initialize residuals $\hat{\mathbf{R}} \leftarrow \mathbf{R}$ and $\mathbf{T}^{(\ell)} = 0\ \forall \ell$
2: while sampler not converged do
3:   for all stencils $\ell = 1 \ldots S$ do
4:     $\hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}} + \mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ {Without stencil $\ell$}
5:     $(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)}) \leftarrow \mathrm{StencilSampler}(\hat{\mathbf{R}}, \mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$
6:     $\hat{\mathbf{R}} \leftarrow \hat{\mathbf{R}} - \mathcal{S}(\mathbf{T}^{(\ell)}, \mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)})$ {With stencil $\ell$}
7:   end for
8: end while
```

Instead, we sample over one stencil at a time, as shown in Algorithm 11. This algorithm only requires repeated passes through the dataset. Moreover, it can be modified into a backfitting procedure by fitting one matrix at a time and then fixing the outcome. Capacity control can be enforced by modifying α and β such that the probability of a new cluster decreases for larger ℓ, i.e. by decreasing α and β. As a result following the analysis in the single stencil case, each sampling pass costs $O(S \cdot (k_n \cdot k_m \cdot (N + M) + E))$ operations. It is linear in the number of ratings, in the number of partitions and in the number of stencils.

7.5 Experiments

We evaluate our method based on its ability to perform matrix completion, matrix approximation and to give interpretable results. Here we describe our experimental setup and results on real world data, such as the Netflix ratings.

7.5.1 Implementation

We implemented both ACCAMS, the k-means-based algorithm, as well as bACCAMS, the Bayesian model. Unless specified otherwise, we run Algorithm 9 for up to $T = 50$ iterations. Our system can also iterate over the stencils multiple times, such that earlier stencils can be re-learned after we have learned later ones. In practice, we observe this only yields small gains in accuracy, hence we generally do not use it.

We implemented bACCAMS using Gibbs sampling (Section 7.4.4) and used the k-means algorithm ACCAMS for the initialization of each stencil. Following standard practice, we bound the range of σ by $\sigma_{\max}$ from above. This rejection sampler avoids pathological cases. For the sake of simplicity, we set $k = k_n = k_m$ to be the maximum number of clusters that can be generated in each stencil. When inferring the cluster assignments for a given stencil, we run three iterations of the sampler before proceeding to the next stencil. As common in MCMC algorithms, we use a burn-in period of at least 30 iterations (each with three sub-iterations of sampling cluster assignments) and then average the predictions over many draws. Code for both ACCAMS and bACCAMS is available at cs.cmu.edu/~abeutel/accams.

7.5.2 Experimental Setup

Netflix We run our algorithms on data from a variety of domains. Our primary testing dataset is the ratings dataset from the Netflix contest. The dataset contains 100 million ratings from 480,189 users and 17,770 movies. Following standard practice for testing recommendation accuracy, we average over three different random 90:10 splits for training and testing.

CMU Face Images To test how well we can approximate arbitrary matrices, we use image data from the CMU Face Images dataset[1]. It contains black and white images of 20 different people, each in 32 different positions, for a total of 640 images. Each image has 128×120 pixel resolution; we flatten this into a matrix of 640×15360, i.e. an image by pixel matrix.

AS Peering Graph To assess our model's ability to deal with graph data we consider the AS graph[2]. It contains information on the peering information of 13,580 nodes. It thus creates a binary matrix of size $13,580 \times 13,580$ with 37k edges. Since our algorithm is not designed to learn binary matrices, we treat the entries $\{0, 1\}$ as real valued numbers.

Parameters For all experiments, we set the hyperparameters in bACCAMS to $\alpha = \beta = 10$, $\eta_\alpha = 2$, $\eta_\beta = 0.3$, $\gamma_\alpha = 5$, and $\gamma_\alpha = 0.3$. Depending on the task, we compare ACCAMS against SVD++ using the GraphChi [131] implementation, SVD from Matlab for full matrices, and previously reported state-of-the-art results.

Model complexity Since our model is structurally quite different from factorization models, we compare them based on the number of bits in the model and prediction accuracy. For factorization models, we consider each factor to be a 32 bit `float`. Hence the complexity of a rank r SVD++ model of N users and M movies is $32 \cdot r(N + M)$ bits.

For ACCAMS with S stencils and $k \times k$ co-clusters in each stencil, the cluster assignment for a given row or column is $\log_2 k$ bits and each value in the stencil is a `float`. As such, the complexity of a model is $S((N + M)\log_2 k + 32 \cdot k^2)$ bits.

In calculating the parameter space size for LLORMA, we make the very conservative estimate that each row and column is on average part of two factorizations, even though the model contains more than 30 factorizations that each row and column could be part of.

[1]http://www.cs.cmu.edu/~tom/faces.html
[2]http://topology.eecs.umich.edu/data.html

Method	Parameters	Size	Test RMSE
SVD++ [131]	$R = 25$	49.8MB	0.8631
DFC-NYS [150]	Not reported		0.8486
DFC-PROJ [150]	Not reported		0.8411
LLORMA [136]	$R = 1$	3.98MB	0.9295
LLORMA [136]	$R = 5$, $a > 30$	19.9MB	0.8604
LLORMA [136]	$R = 10$, $a > 30$	39.8MB	0.8444
LLORMA [136]	$R = 20$, $a > 30$	79.7MB	0.8337
ACCAMS	$k = 10$, $s = 13$	2.69MB	0.8780
ACCAMS	$k = 100$, $s = 5$	2.27MB	0.8759
bACCAMS	$k = 10$, $s = 50$	10.4MB	0.8403
bACCAMS	$k = 10$, $s = 70$	14.5MB	0.8363
bACCAMS	$k = 10$, $s = 125$	**25.9MB**	**0.8331**

Table 7.2: **Model accuracy and size**: bACCAMS achieves an accuracy for matrix completion on Netflix better than or on-par with other state-of-the-arts methods, while having a parameter space a fraction of the size of other methods. a denotes the number of anchor points for LLORMA and sizes listed are the parameter space size.

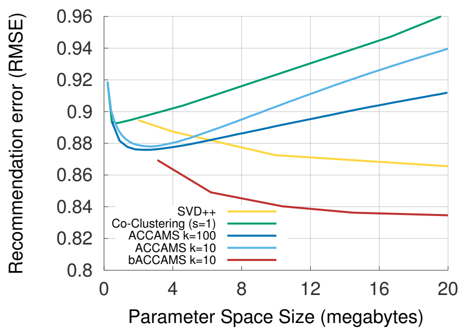
7.5.3 Matrix Completion

Since the primary motivation of our model is collaborative filtering we begin by discussing results on the classic Netflix problem; accuracy is measured in RMSE. To avoid divergence we set $\sigma_{\max} = 1$. We then vary both the number of clusters k and the number of stencils S.

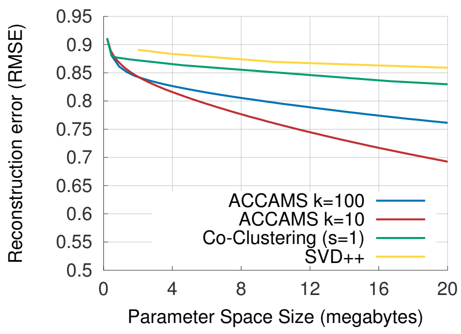
A summary of recent results as well as results using our method can be found in Table 7.2. Using GraphChi we run SVD++ on our data. We use the reported values from LLORMA [136] and DFC [150], which were obtained using the same protocol as reported here.

As can be seen in Table 7.2, bACCAMS matches the accuracy of other state-of-the-art methods. We achieve this while using a very different model that is significantly simpler both conceptually and in terms of parameter space size. We also did not use any of the temporal and contextual variants that many other models use to incorporate prior knowledge.

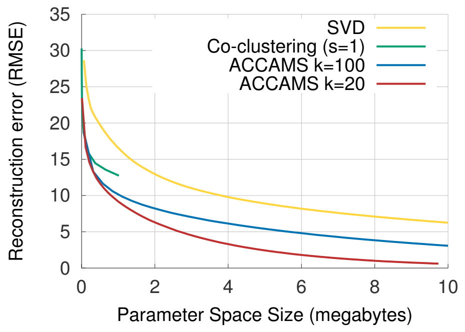
As shown in Figure 7.1, we observe that per bit our model achieves much better accuracy at a fraction of the model size. In Figure 7.4(a) we compare different configurations of our algorithm. As can be seen, classic co-clustering quickly overfits the training data and provides a less fine-grained ability to improve prediction accuracy than ACCAMS. Since ACCAMS has no regularization, it too overfits the training data.



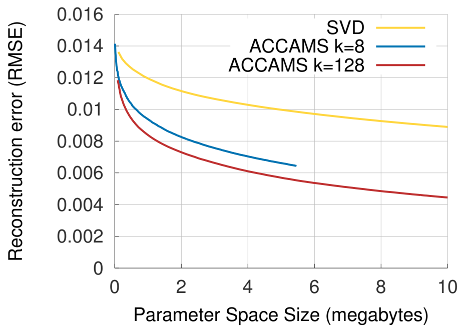
(a) Netflix Prediction Error



(b) Netflix Training Error



(c) Face Approximation



(d) AS Graph Approximation

Figure 7.4: On images, ratings, and binary graphs, ACCAMS approximates the matrix more efficiently than SVD, SVD++, or classic co-clustering.

By using a Bayesian model with bACCAMS, we do not overfit the training data and thus can use more stencils for prediction, greatly improving the prediction accuracy.

7.5.4 Matrix Approximation

In addition to matrix completion, it is valuable to be able to approximate matrices well, especially for dimensionality reduction tasks. To test the ability of ACCAMS to model matrix data we analyze both how well our model fits the training data from the Netflix tests above as well as on image data from the CMU Faces dataset and a binary matrix from the AS peering graph. (Note, for Netflix we now use the training data from one split of the dataset.) For each of these of datasets we compare to the SVD (or SVD++ to handle missing values). We also use our algorithm to perform classic co-clustering by setting $S = 1$ and varying k.

As can be seen in Figure 7.4(b–d), ACCAMS models the matrices from all three domains much more compactly than SVD (or SVD++ in the case of the Netflix matrix, which contains missing values). In particular, we observe on the CMU Faces matrix that ACCAMS uses in some cases under $\frac{1}{4}$ of the bits as SVD for the same quality matrix approximation. Additionally, we observe that using a linear combination of stencils is more efficient to approximate the matrices than performing classic co-clustering

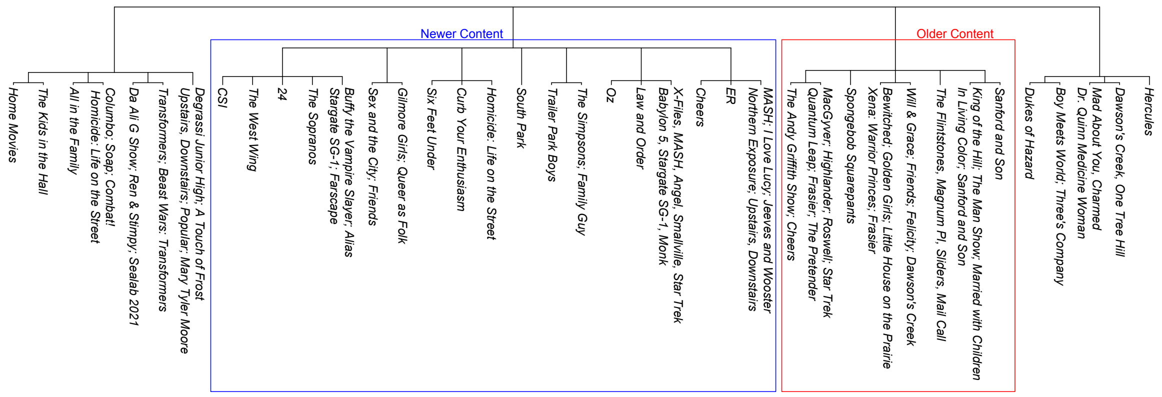

Newer Content
Older Content
Home Movies
The Kids in the Hall
All in the Family
Columbo; Soap; Combat!
Homicide: Life on the Street
Da Ali G Show; Ren & Stimpy; Sealab 2021
Transformers: Beast Wars; Transformers
Degrassi Junior High; A Touch of Frost
Upstairs, Downstairs; Popular; Mary Tyler Moore
CSI
The West Wing
24
The Sopranos
Buffy the Vampire Slayer; Alias
Stargate SG-1; Farscape
Sex and the City; Friends
Gilmore Girls; Queer as Folk
Six Feet Under
Curb Your Enthusiasm
Homicide: Life on the Street
South Park
Trailer Park Boys
The Simpsons; Family Guy
Oz
Law and Order
X-Files, MASH, Angel, Smallville, Star Trek
Babylon 5, Stargate SG-1, Monk
Cheers
ER
MASH; I Love Lucy; Jeeves and Wooster
Northern Exposure; Upstairs, Downstairs
The Andy Griffith Show; Cheers
MacGyver; Highlander; Roswell; Star Trek
Quantum Leap; Frasier; The Pretender
Spongebob Squarepants
Bewitched; Golden Girls; Little House on the Prairie
Xena: Warrior Princess; Frasier
Will & Grace; Friends; Felicity; Dawson's Creek
The Flintstones, Magnum PI, Sliders, Mail Call
King of the Hill; The Man Show; Married with Children
In Living Color; Sanford and Son
Sanford and Son
Dukes of Hazard
Boy Meets World; Three's Company
Mad About You, Charmed
Dr. Quinn Medicine Woman
Dawson's Creek, One Tree Hill
Hercules

Figure 7.5: **Hierarchy of TV Shows** on Netflix based on the first three stencils generated by ACCAMS.

where we have just one stencil. Ultimately, although the method was not designed specifically for image or network data, we observe that our method is effective for succinctly modeling the data.

7.5.5 Interpretability

In any model the structure of factors makes assumptions about the form of user preferences and decision making. The fact that our model uses a smaller parameter space while achieving an *improvement* in the generalization error suggests that our modeling assumptions better match the underlying data generation (how people make decisions). One advantage of our model being compact and conceptually simple is that we can understand our learned parameters.

To test the model's interpretability we use ACCAMS to model the Netflix data with $S = 20$ stencils and $k^2 = 100$ clusters (a model of similar size to a rank-3 matrix factorization). Here we look at two ways to interpret the results.

First we view the cluster assignments in stencils as inducing a hierarchy on the movies. That is, movies are split in the first level based on their cluster assignments in the first stencil. At the second level, we split movies based on their cluster assignments in the second stencil, etc. In Figure 7.5 we observe the hierarchy of TV shows induced by the first three stencils learned by ACCAMS (we only include shows where there is more than one season of that show in the leaf and we pruned small partitions due to space restrictions).

As can be seen in the hierarchy, there are branches which clearly cluster together shows more focused on male audiences, female audiences, or children. However, beyond a first brush at the leaf nodes, we can notice some larger structural differences. For example, looking at the two large branches coming from the root, we observe that the left branch generally contains more recent TV shows from the late 1990s to the present, while the right branch generally contains older shows ranging from the 1960s to the mid 1990s. This can be most starkly noticed by "Friends," which shows up in both branches;

2001: A Space Odyssey	**Sex and the City: Season 1**	**Seinfeld: Seasons 1 & 2**	**Mean Girls**
Taxi Driver	Sex and the City: Season 2	Seinfeld: Season 3	Clueless
Chinatown	Sex and the City: Season 3	Seinfeld: Season 4	13 Going on 30
Citizen Kane	Sex and the City: Season 4	Curb Your Enthusiasm: Season 1	Best in Show
Dr. Strangelove	Sex and the City: Season 5	Curb Your Enthusiasm: Season 2	Particles of Truth
A Clockwork Orange	Sex and the City: Season 6.1	Curb Your Enthusiasm: Season 3	Charlie's Angels: Full Throttle
THX 1138: Special Edition	Sex and the City: Season 6.2	Arrested Development: Season 1	Amelie
Apocalypse Now Redux	Hercules: Season 3	Newsradio: Seasons 1 and 2	Me Myself I
The Graduate	Will & Grace: Season 1	The Kids in the Hall: Season 1	Bring it On
Blade Runner	Beverly Hills 90210: Pilot	The Simpsons: Treehouse of Horror	Chaos
The Deer Hunter	The O.C.: Season 1	Spin City: Michael J. Fox	Kissing Jessica Stein
Deliverance	Divine Madness	Curb Your Enthusiasm: Season 4	Nine Innings from Ground Zero

Star Wars: Episode V	**The Silence of the Lambs**	**Scooby-Doo Where Are You?**	**Law & Order: Season 1**
Star Wars: Episode IV	The Sixth Sense	The Flintstones: Season 2	Law & Order: Season 3
Star Wars: Episode VI	Alien: Collector's Edition	Classic Cartoon Favorites: Goofy	Law & Order: SVU (2)
Battlestar Galactica: Season 1	The Exorcist	Transformers: Season 1 (1984)	Law & Order: Criminal Intent (3)
Raiders of the Lost Ark	Schindler's List	Tom and Jerry: Whiskers Away!	Law & Order: Season 2
Star Wars: Clone Wars: Vol. 1	The Godfather	Boy Meets World: Season 1	MASH: Season 8
Gladiator: Extended Edition	Seven	The Flintstones: Season 3	ER: Season 1
Star Wars Trilogy: Bonus Material	Colors	Scooby-Doo's Greatest Mysteries	MASH: Season 7
LOTR: The Fellowship of the Ring	The Godfather, Part II	Care Bears: Kingdom of Caring	Rikki-Tikki-Tavi
LOTR: The Two Towers	GoodFellas: Special Edition	Aloha Scooby-Doo!	The X-Files: Season 6
Indiana Jones: Bonus Material	Platoon	Scooby-Doo: Legend of the Vampire	The X-Files: Season 7
LOTR: The Return of the King	Full Metal Jacket	Rugrats: Decade in Diapers	ER: Season 3

Table 7.3: **Finding related content:** For a given movie or TV show on Netflix, we can use the cluster assignments to find related content.

Seasons 1 to 4 of "Friends" from 1994–1997 fall in the older branch, while Seasons 5 to 9 from 1998–2002 fall in the newer branch. Of course the algorithm does not know the dates the shows were released, but our model learns these general concepts just based on the ratings. From this it is clear the stencils can be useful for breaking down content in a meaningful structured way, something that is not possible under classic factorization approaches.

While the hierarchy demonstrates that our stencils are learning meaningful latent factors, it may be difficult to always understand individual clusters. Rather, to use knowledge from *all* of the stencils, we can look to the use case of "Users who watched X also liked Y," and ask given a movie or TV show to search, can we find other similar items? We do this by comparing the set of cluster assignments from the given movie to the set of cluster assignments of other items. Most simply, we can measure similarity between two movies using the Hamming distance between cluster assignments.

As can be seen in Table 7.3, we find the combination of clusters for different movies and TV shows can be used to easily find similar content. While we see some obvious cases where the method succeeds, e.g., "Sex and the City" returns six more seasons of "Sex and the City," we also notice the method takes into account more subtle similarities of movies beyond genre. For example, while the first season of "Seinfeld" returns the subsequent seasons of "Seinfeld," it is followed by three seasons of "Curb Your Enthusiasm," another comedy show by the same writer Larry David. Similarly, searching for Stanley Kubrick's "2001: A Space Odyssey" returns other Stanley Kubrick movies, as well as other critically acclaimed films from that era, particularly thematically similar science fiction movies.

Searching for “Scooby-Doo” returns topically similar children’s shows, specifically from the mid to late 1900’s. From this we get a sense that ACCAMS does not just find similarity in genre but also more subtle similarities.

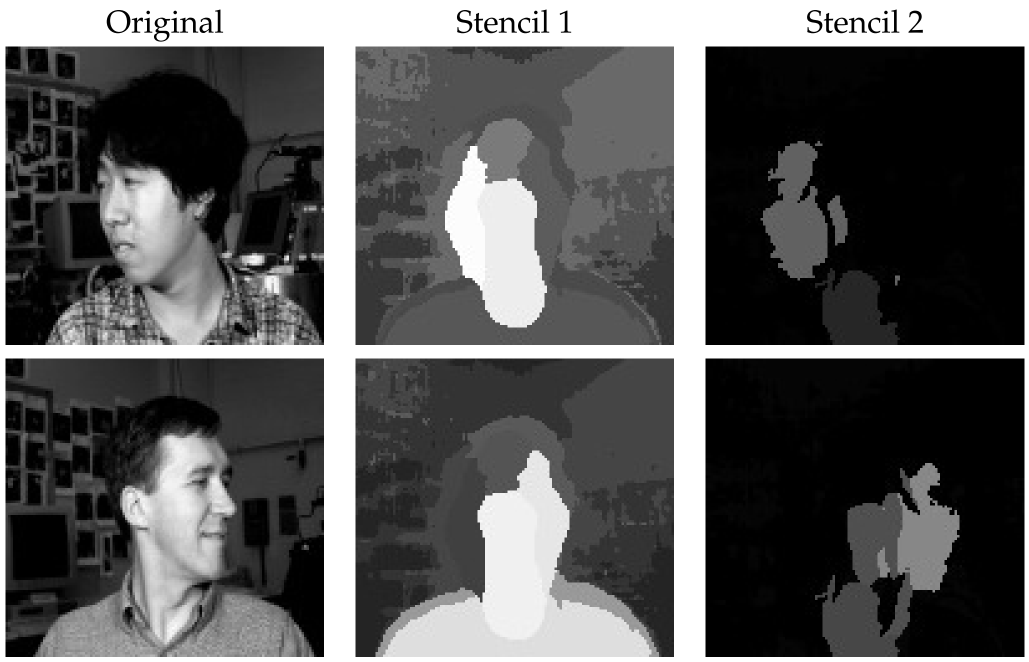
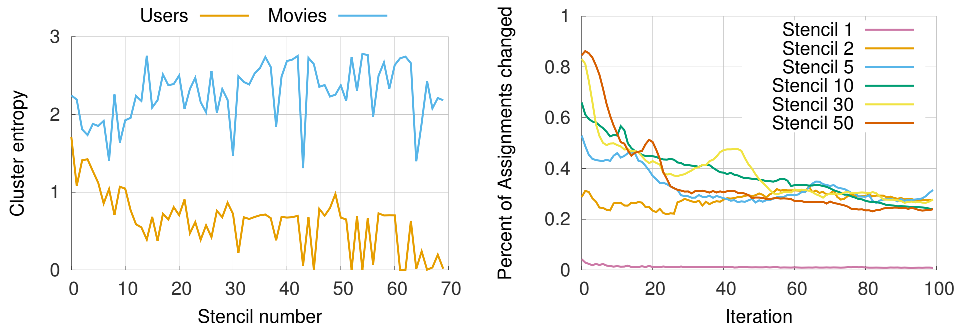


Figure 7.6: **Examples of original images and the first two stencils**. The decomposition is very similar to that of eigenfaces [214], albeit much more concise in its nature.

7.5.6 Properties of ACCAMS

Aside from ACCAMS’s success across matrix completion and approximation, it is valuable to understand how our method is working, particularly because of how different it is from previous models. First, because ACCAMS uses backfitting, we expect that the first stencil captures the largest features, the second captures secondary ones, etc. This idea is backed up by the theoretical results in Section 7.3.3, and we observe that this is working experimentally by the drop off in RMSE for our matrix approximation results in Figure 7.4. We can visually observe this in the image approximation of the CMU Faces. As can be seen in Figure 7.6, the first stencil captures general structures of the room and heads, and the second starts to fill in more fine grained details of the face.

The Bayesian model, bACCAMS, backfits in the first iteration of the sampler but ultimately resamples each stencil many times thus loosening these properties. In Figure 7.7, we observe how the distribution of users and movies across clusters changes over iterations and number of stencils, based on our run of bACCAMS with $S = 70$ stencils and a maximum of $k = 10$ clusters per stencil. As we see in the plot of entropy, movies, across all 70 stencils, are well distributed across the 10 possible clusters. Users, however,



(a) Cluster assignment entropy (b) Cluster stability

Figure 7.7: **bACCAMS properties:** Left: Entropy in cluster assignments for users and movies. Right: Stability of the assignments in the sampler.

are well distributed in the early stencils but then are only spread across a few clusters in later stencils. In addition, we notice that while the earlier clusters are stable, later stencils are much less stable with a high percentage of cluster assignments changing. Both of these properties follow from the fact that most users rate very few movies. For most users only a few clusters are necessary to capture their observed preferences. Movies, however, typically have more ratings and more latent information to infer. Thus through all 70 stencils we learn useful clusterings, and our prediction accuracy improves through $S = 125$ stencils.

7.6 Discussion

Here we formulated a model of additive co-clustering. We presented both a k-means style algorithm, ACCAMS, as well as a generative Bayesian non-parametric model with a collapsed Gibbs sampler, bACCAMS; we showed that our method is concise and accurate on a diverse range of datasets, including matching state-of-the-art accuracy for matrix completion on Netflix.

Given the novelty and initial success of the method, we believe that domain-specific variants of ACCAMS, such as for community detection and topic modeling, can and will lead to new models and improved results. In addition, given the modularity of our framework, it is easy to incorporate side information, such as explicit genre and actor data, in modeling rating data that should lead to improved accuracy and interpretability. We will show in the next chapter how to incorporate review text for precisely these reasons.

Chapter 8

Explaining Recommendations

Understanding a user's *motivations* provides valuable information beyond the ability to recommend items. Quite often this can be accomplished by perusing both ratings and review texts, since it is the latter where the reasoning for specific preferences is explicitly expressed.

Unfortunately matrix factorization approaches to recommendation result in large, complex models that are difficult to interpret and give recommendations that are hard to clearly explain to users. In contrast, in this chapter, we build on the successes in Chapter 7 and attack this problem through succinct additive co-clustering. We devise a novel Bayesian technique for summing co-clusterings of *Poisson distributions*. With this novel technique we propose a new Bayesian model for joint collaborative filtering of ratings and text reviews through a sum of simple co-clusterings. The simple structure of our model yields easily interpretable recommendations. Even with a simple, succinct structure, our model outperforms competitors in terms of predicting ratings with reviews.

8.1 Introduction

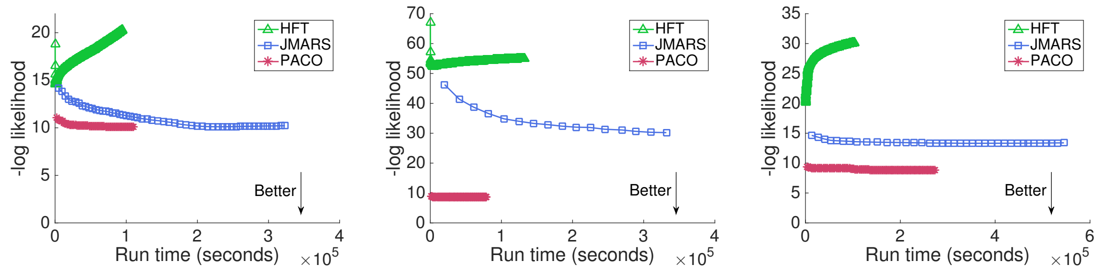
Recommender systems often serve a dual purpose—they are expected to generate suggestions that users might like, while simultaneously being able to *explain* why a certain recommendation was made. This increases a user's confidence in a recommender system and it offers valuable insight for debugging a malfunctioning model.

Matrix factorization [123] accomplishes this goal only to a limited extent, since it maps all users and movies into a rather low-dimensional space, where objects are compared by the extent of overlap they have in terms of their inner product. This limits attempts to understand the model to principal component analysis and nearest neighbor queries for specific instances.

On the other hand, in many cases users are actually happy to provide explicit justification for their preferences in the form of written reviews. They offer immediate insights into the reasoning, provided that we are able to capture this reasoning in the form of a model for the text inherent in the reviews. JMARS [64] exploited this insight by designing a topic model to capture reviews and ratings jointly, thus offering one of the first works to infer both topics and sentiments without requiring explicit aspect ratings.

A challenge in these approaches is that the model must fit a language model to a rather messy, high dimensional embedding of users and items. We addressed this problem in the context of recommender systems in the previous chapter with a novel additive co-clustering model for matrix completion. This approach yielded excellent predictive accuracy while providing a well-structured, parsimonious model. Because the model heavily relied on backfitting and an additive representation of a regression model, it is not possible to combine it with multinomial language models, i.e. a simple bag-of-words representation, since probabilities are not additive: They need to be normalized to 1.

We address this problem by introducing a novel *additive* language description in the form of a sum of *Poisson* distributions rather than a Binomial distribution. This strategy allows us to use backfitting for documents rather than just in a regression setting, and enables a wide variety of new applications. This is possible because the Poisson distribution is closed under addition. This means that sums of Poisson random variables remain Poisson. This property also applies to mixtures of Poisson random variables, i.e. the occurrence of multiple words.



(a) Amazon fine foods (b) RateBeer (c) Yelp

Figure 8.1: **Negative log likelihood**. PACO better jointly predicts held-out ratings and reviews than state-of-the-art JMARS [64] and HFT [155] on Amazon fine foods, Yelp and RateBeer datasets. The joint predictive power is captured by the normalized negative log likelihood as described in (8.18). (Lower negative log-likelihood and time are both better.)

With this approach we make a number of contributions:

- **Joint Model:** We design a Poisson additive co-clustering model for backfitting word counts in documents. We combine this with ACCAMS (Ch. 7) to learn a joint Bayesian model of reviews and ratings, with the ability to now interpret our model.
- **Efficient Sampler:** We describe a new, efficient algorithm for sampling from a sum of Poisson random variables to facilitate efficient inference. It relies on treating discrete counts as "residuals," similar to an additive regression model.
- **Empiral evidence:** We give a thorough experimental evaluation across multiple datasets that PACO has better prediction accuracy for ratings than competing methods, such as HFT [155] and JMARS [64]. Additionally, our method predicts

text reviews better than HFT, and achieves nearly as high quality review prediction as JMARS, while being far faster and simpler. As seen in Figure 8.1, PACO outperforms both competing models in jointly predicting ratings and reviews.

In summary, we propose a simple and novel model and sampler, capable of characterizing user and item attributes very concisely, while providing excellent accuracy and perplexity.

8.2 Related Work

In this chapter, we build on the ACCAMS model from the previous chapter but focus on making use of reviews to explain recommendations. In the process, we also build on the insights from a variety of related fields in pursuing this new direction.

Poisson Collaborative Filtering Recently, there has been a growing line of research on using Poisson distributions in matrix factorization models [46, 82, 189]. This work shows the exciting potential uses of Poisson distributions for understanding matrix data. However, all of these models are left with limited interpretability since they too rely on bilinear models. Additionally, all of these models rely on variational inference to learn the models. Our work provides the building blocks for using Poisson distributions in a wide array of additive clustering applications and is the first work to learn a model of this sort through Gibbs sampling rather than variational inference.

Review Mining and Modeling Modeling online reviews has long been a focus of the data mining, machine learning and natural language processing communities [102]. Significant research has focused on understanding and finding patterns in online reviews [58]. More closely related to our work, a variety of papers model aspects and sentiments of reviews [115, 117, 132, 145]. For example, [117] considers hierarchical structures in aspects and sentiments. However, in these works ratings are not considered jointly.

Multimodal Models Recently, there is increasing attention in jointly modeling review text and ratings. Collaborative topic regression (CTR) [220] combines topic modeling and collaborative filtering to recommend scientific articles. HFT [155] jointly models both ratings and reviews by designing link function to connect each topic dimension to a latent factor, demonstrating improvements in rating prediction. RMR [146] relaxes the hard link between topic and latent factor dimension for interpretable topics. [230] considers review texts with hidden user communities and item groups relationship. JMARS [64] jointly models aspects, sentiments, items, reviews, and ratings based on insights in review structure.

A related line of work models multi-aspect ratings [156, 212]. However, these works often rely on availability of aspect-specific ratings, which are often not available. In contrast, our models learns sentiments in different aspects without requiring multi-aspect ratings.

Symbol	Definition
N, M	Number of rows (users) and columns (items)
$\mathbf{R}$	Data matrix $\in \mathbb{R}^{N \times M}$ (with missing values)
$\mathcal{I}$	Indicator matrix $\in \{0,1\}^{N \times M}$ for $\mathbf{R}$
S	Number of stencils
$k_n^{(\ell)}, k_m^{(\ell)}$	Number of user and item clusters in stencil ℓ
$\mathbf{T}^{(\ell)}$	Matrix $\in \mathbb{R}^{k_n^{(\ell)} \times k_m^{(\ell)}}$ for stencil ℓ
$\mathbf{c}^{(\ell)}$	Vector of user assignments $\in \{1, \ldots, k_n^{(\ell)}\}^N$
$\mathbf{d}^{(\ell)}$	Vector of item assignments $\in \{1, \ldots, k_m^{(\ell)}\}^M$
$\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})$	$\in \mathbb{R}^{N \times M}$ defined by $\mathcal{S}(\mathbf{T}, \mathbf{c}, \mathbf{d})_{u,m} = \mathbf{T}_{\mathbf{c}_u, \mathbf{d}_m}$
$\mathcal{W}$	Set of all words used in the reviews
$n_{u,m,x}$	Count for word x in review (u, m)
$\mu_x^{(i)}$	Rate of Poisson in language model i for word x
$\mathcal{L}_{u,m}$	Set of language models used for review (u, m)

Table 8.1: Symbols used throughout this chapter.

Additive Co-Clustering As stated previously, we build primarily on the success of ACCAMS in the previous chapter. From a computational perspective, ACCAMS (Ch. 7) focuses on an additive co-clustering model of Gaussian distributions and describes a collapsed sampler for efficient learning. [170] uses a sum of clusters within a logistic function for modeling binary data but has difficulty scaling. Here we demonstrate that we can successfully scale learning a sum of clusters within a Poisson distribution.

8.3 Poisson Additive Co-Clustering

While ACCAMS (Ch. 7) explains well the rating matrix, it does not utilize the reviews associated with those ratings. In this section we introduce PACO, a Poisson Additive co-clustering model that jointly model text and reviews. PACO builds on the idea of stencils introduced in ACCAMS. Each stencil $\mathbf{T}$ assigns each user u to a cluster say a and each item m to a cluster say b. Given the block (i.e. co-cluster) denoted by $\mathbf{T}_{a,b}$, we design a model to jointly generates both the rating user u gives to item m as well as the review she wrote for this item. We endow each block with a Gaussian distribution that model the mean rating associated with cluster a and b. The question now is: how can we parameterize the text model of block (a, b)?

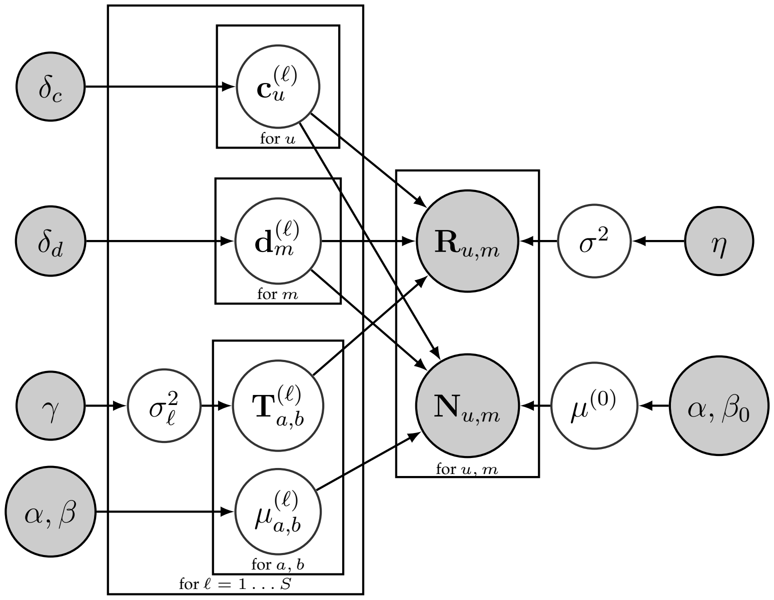


Figure 8.2: **The generative model for PACO to predict both ratings R and review text N**. (Note, for the sake of space we simplify the model slightly by not explicitly separating the different language models associated with each stencil.)

8.3.1 Modeling Reviews using an Additive Poisson Model

A standard approach in the text mining literature is to model reviews using a multinomial distribution, however, in PACO, as in ACCAMS, we want to combine multiple stencils to enhance the model. While it is easy to define an additive model over review scores, it is nontrivial to accomplish this using multinomial distributions for reviews. Quite obviously, if p and q are multinomial probability distributions, then $p + q$ is no longer in the probability simplex. Instead, the transform is given by $(p + q)/\|p + q\|_1$, thus making updates highly nontrivial, since additivity of the model is lost, which means that we would need to update the entire language model whenever even just a single stencil changes. This would make a backfitting algorithm very expensive.

Rather, we introduce a novel approach to modeling using the Poisson distribution. In a nutshell, we exploit the fact that the Poisson distribution is closed under addition, i.e. for

$$a \sim \text{Poi}(\lambda) \text{ and } b \sim \text{Poi}(\gamma) \text{ we have } a + b \sim \text{Poi}(\lambda + \gamma)$$

where λ and γ denote the rates of each of the random variables, i.e. $\mathbf{E}[a] = \lambda$ and $\mathbf{E}[b] = \gamma$.

For each user and item pair (u, m) pair we let $n_{u,m,x}$ denote the count for word x in the review. We now design an additive model for each review. The idea is that the distribution over each word is given by a Poisson random variable with rate $\sum_\ell \mu^{(\ell)}_{c_u^{(\ell)}, d_m^{(\ell)}, x}$, or equivalently a sum over Poisson random variables with rates $\mu^{(\ell)}_{c_u^{(\ell)}, d_m^{(\ell)}, x}$ for each word x across all stencils. The benefit of this approach is that we no longer need to ensure normalization across stencils. We will detail the generative model in the following subsection.

8.3.2 The Joint Generative Model

Now we are ready to present the full model. To design a joint model we face an important challenge: we need to assess whether to perform good recommendation or whether we strive to optimize for good perplexity. In the former case, it is undesirable if the reviews carry the majority of the statistical weight. Hence it is worthwhile to normalize the reviews by their length. This technique is common in NLP literature [223, 224]. This yields the following joint objective:

$$\begin{aligned}\underset{\{\mathbf{T}^{(\ell)},\mathbf{c}^{(\ell)},\mathbf{d}^{(\ell)},\lambda\}}{\text{minimize}} \sum_{(u,m)\in\mathcal{I}} &\left(\mathbf{R}_{u,m} - \sum_{\ell=1}^{S} \mathcal{S}\left(\mathbf{T}^{(\ell)},\mathbf{c}^{(\ell)},\mathbf{d}^{(\ell)}\right)_{u,m}\right)^2 \\ &+ \sum_{(u,m)\in\mathcal{I}} \frac{1}{|n_{u,m}|_1} \sum_{x\in\mathcal{W}} \log \mathrm{Poi}(\lambda_{u,m,x})\end{aligned}$$

where

$$\lambda_{u,m} = \mu^{(0)} + \mu^{(m)} + \left[\sum_{\ell=1}^{S} \mu^{(\ell)}_{c_u^{(\ell)},d_m^{(\ell)}} + \mu^{(m,\ell)}_{d_m^{(\ell)}} + \mu^{(u,\ell)}_{c_u^{(\ell)}}\right] \tag{8.1}$$

Our goal is to learn a set of stencils whose summation minimizes the prediction error on ratings and maximizes the likelihood of generating the text. To model review text, we allow each stencil to have three language models: a stencil-specific user language model $\mu^{(\ell)}_{c_u^{(\ell)}}$, a stencil-specific item language model $\mu^{(m,\ell)}_{d_m^{(\ell)}}$, and block language model, $\mu^{(\ell)}_{c_u^{(\ell)},d_m^{(\ell)}}$. The block language model captures the stencil-specific interaction between the item and the user. In addition, we add a global item language model, $\mu^{(m)}$, and a global background language model, $\mu^{(0)}$. The text of the review is modeled as a combination of these Poisson language models.

Minimizing the aforementioned objective function, is equivalent to maximizing the log-likelihood of the graphical model in Figure 8.2. The generative process proceeds as follows:

$$\mathbf{R}_{u,m} \sim \mathcal{N}\left(\sum_{\ell=1}^{S} \mathcal{S}\left(\mathbf{T}^{(\ell)},\mathbf{c}^{(\ell)},\mathbf{d}^{(\ell)}\right)_{u,m}, \sigma^2\right) \tag{8.2}$$

$$\mathbf{c}^{(\ell)}, \mathbf{d}^{(\ell)} \sim \mathrm{CRP}(\delta) \text{ for all } (\ell) \tag{8.3}$$

$$\mathbf{T}^{(\ell)}_{c,d} \sim \mathcal{N}(0, \sigma^2_{(\ell)}) \tag{8.4}$$

$$n_{u,m,x} \sim \mathrm{Poi}\left(\lambda_{u,m,x}\right) \tag{8.5}$$

$$\mu^{(*)}_x \sim \mathrm{Gamma}(\alpha, \beta) \tag{8.6}$$

where $\mathbf{c}_u^{(\ell)}, \mathbf{d}_m^{(\ell)}$ is the cluster (u, m) assigned to in stencil ℓ. In essence, we model user and item clusters inside each stencil using a Chinese restaurant process (CRP). Ratings are modeled similar to ACCAMS while (8.1), (8.5) and (8.6) contain the additional text modeling aspects of our model. To ensure no overfitting, we use conjugate Gamma prior for all the vectors μ:

$$P(\mu_x) = \frac{\beta^\alpha}{\Gamma(\alpha)} \mu_x^{\alpha-1} e^{-\beta\mu_x}$$

Since the mode of the Gamma distribution is $\frac{\alpha-1}{\beta}$, a very large β should ensure that only a small amount of data is blamed. Given this model, we now consider the challenge of learning the parameters in practice.

8.4 The Sampling Algorithm

The goal of inference is to learn a posterior distribution over stencils' parameters which are: user and item cluster assignments, stencil ratings of each block, and the multiple language models. To do this, we use Gibbs sampling.

Jointly sampling text and rating adds significant complications over just sampling ratings data and requires a novel sampling technique. In particular, our sampler offers (1) a new novel technique to learn the sum of Poisson rates μ and (2) an efficient method for sampling cluster assignments based on both the text model and ratings model. We describe each of these challenges and solutions below in Section 8.4.1, followed by the complete learner shown in Algorithm 12, which combines the new, novel Poisson sampler and ACCAMS's sampler for Gaussian rating data.

8.4.1 Sampling a Sum of Poisson Distributions

The Gamma distribution is conjugate to the Poisson distribution, typically allowing for an easy sampling of the Poisson's rate λ from the Gamma distribution. However, in this case we have a sum of Poisson distributions and we would like to sample the rate of each of these distributions.

To make this tractable, we create a multinomial the from rates of the involved Poisson distributions and sample form this multinomial the fraction of counts coming from each Poisson distributions. To be precise, for a particular $n_{u,m,x}$ we have $\lambda_{u,m,x} = \sum_{i \in \mathcal{L}_{u,m}} \mu_x^{(i)}$, where $\mathcal{L}_{u,m}$ is the set of Poisson distributions from which words in (u, m) are sampled. We define:

$$\left\{\mu_{u,m,x}^{(*)}\right\} := \left\{\mu_x^{(i)}\right\}_{\forall i \in \mathcal{L}_{u,m}} \tag{8.7}$$

$$\left\{\hat{n}_{u,m,x}\right\} := \left\{\hat{n}_{u,m,x}^{(i)}\right\}_{\forall i \in \mathcal{L}_{u,m}} \tag{8.8}$$

We can therefore sample $\{\hat{n}_{u,m,x}\}$ by

$$\{\hat{n}_{u,m,x}\} \sim \text{Multi}\left(\frac{\left\{\mu^{(*)}_{u,m,x}\right\}}{\lambda_{u,m,x}}, n_{u,m,x}\right). \tag{8.9}$$

The result is $n_{u,m,x} = \sum_{i\in\mathcal{L}_{u,m}} \hat{n}^{(i)}_x$. That is, if we observe word x, $n_{u,m,x}$ times in review (u,m), we break this count to a set of $\hat{n}^{(i)}_x$ counts, each of which are credited to the corresponding Poisson distribution i, where i indexes over the set of involved Poisson distributions. For example, if the word "delicious" is used three times in a review, we may consider one use of the word to be "from" the base language model and two uses of the word to be "from" the item-specific language model.

By sampling these count allocations, we can now tractably sample our Poisson rates and later our cluster assignments. To sample a particular $\mu^{(i)}_x$, we consider $\mathcal{R}_i$, the set of reviews (u,m) partially sampled form a Poisson distribution with rate $\mu^{(i)}$. Therefore, we can sample $\mu^{(i)}_x$ by:

$$\mu^{(i)}_x \sim \Gamma\left(\alpha^{(i)}, \beta^{(i)}\right) \prod_{(u,m)\in\mathcal{R}_i} \text{Poi}\left(\hat{n}^{(i)}_{u,m,x} \middle| \mu^{(i)}_x\right) \tag{8.10}$$

$$\sim \Gamma\left(\alpha^{(i)} + \sum_{(u,m)\in\mathcal{R}_i} \hat{n}^{(i)}_{u,m,x}, \beta^{(i)} + |\mathcal{R}_i|\right) \tag{8.11}$$

This is trivially parallelized across all words in a particular language model μ, which we will expand on later.

8.4.2 Sampling Cluster Assignments

For each stencil, we need to sample the cluster assignment for each user and item. To do this for users, we need to calculate the posterior distribution $p(\mathbf{c}^{(\ell)}_u = a|\text{rest})$ for each cluster a. This probability is composed of three main terms: (1) the CRP prior, (2) the likelihood of the user ratings and (3) the likelihood of the user review texts. This can be written as

$$\begin{aligned} p(\mathbf{c}^{(\ell)}_u = a|\text{rest}) &\propto \text{CRP}(a) \\ &\times \prod_{(u,m)\in\mathcal{R}_u} p(\mathbf{R}_{u,m}|\text{rest}) \prod_x p(n_{u,m,x}|\text{rest}) \end{aligned}$$

where $\mathcal{R}_u$ denotes the set of reviews from user u. In log-space this becomes:

$$\begin{aligned} &\log p(\mathbf{c}^{(\ell)}_u = a|\text{rest}) \propto \log\left(\text{CRP}(a)\right) \\ &+ \sum_{(u,m)\in\mathcal{R}_u} \left(\log p(\mathbf{R}_{u,m}|\text{rest}) + \frac{1}{|n_{u,m}|}\sum_x \log p(n_{u,m,x}|\text{rest})\right) \end{aligned} \tag{8.12}$$

The details for calculating the CRP term and rating terms can be found in Chapter 7. Here, we focus on how to efficiently calculate the term that corresponds to probability of the reviews.

To calculate the probability of the text, we in fact focus on the $\hat{n}_{u,m,x}$ rather than $n_{u,m,x}$. Specifically, when sampling the user cluster assignment, we must calculate on

$$\begin{aligned}\Delta_{u,a} := &\sum_{(u,m)\in\mathcal{R}_u}\sum_x \log\ \text{Poi}\left(\hat{n}^{(\ell)}_{u,m,x}\middle|\,\mu^{(\ell)}_{a,\mathbf{d}^{(\ell)}_m,x}\right)\\ &\quad + \log\ \text{Poi}\left(\hat{n}^{(u,\ell)}_{u,m,x}\middle|\,\mu^{(u,\ell)}_{a,x}\right)\\ = &\sum_{(u,m)\in\mathcal{R}_u}\sum_x \hat{n}^{(\ell)}_{u,m,x}\log(\mu^{(\ell)}_{a,\mathbf{d}^{(\ell)}_m,x}) - \mu^{(\ell)}_{a,\mathbf{d}^{(\ell)}_m,x}\\ &\quad + \hat{n}^{(u,\ell)}_{u,m,x}\log(\mu^{(u,\ell)}_{a,x}) - \mu^{(u,\ell)}_{a,x} \end{aligned} \tag{8.13}$$

For k clusters, to naively calculate $\Delta_{u,a}$ for each user would require $O(k|\mathcal{W}|)$ logarithm operations, which are significantly slower than the other simple addition and multiplication operations. However, we can significantly speed this up.

First, let's define the following terms:

$$\tilde{\mu}^{(\ell)}_{a,b} = \sum_x \mu^{(\ell)}_{a,b,x} \qquad \tilde{\mu}^{(u,\ell)}_{a} = \sum_x \mu^{(\ell)}_{a,b,x}$$

$$n_{u,b} = \sum_{\substack{m|\mathbf{d}^{(\ell)}_m=b,\\(u,m)\in\mathcal{R}_u}} 1 \qquad \hat{n}^{(\ell)}_{u,b} = \sum_{\substack{m|\mathbf{d}^{(\ell)}_m=b,\\(u,m)\in\mathcal{R}_u}} \hat{n}^{(\ell)}_{u,m}$$

$$\hat{n}^{(u,\ell)}_{u} = \sum_{(u,m)\in\mathcal{R}_u} \hat{n}^{(u,\ell)}_{u,m}$$

All of these terms can be precalculated and cached for sampling all cluster assignments for stencil ℓ. Now we can rearrange the terms of (8.13) to achieve the following simplified equation:

$$\begin{aligned}\Delta_{u,a} = &-\sum_{b=1}^{k} n_{u,b}\left(\tilde{\mu}^{(\ell)}_{a,b} + \tilde{\mu}^{(u,\ell)}_{a}\right)\\ &+ \langle\hat{n}^{(\ell)}_{u,b}, \log(\mu^{(\ell)}_{a,b})\rangle + \langle\hat{n}^{(u,\ell)}_{u}, \log(\mu^{(u,\ell)}_{a})\rangle\end{aligned} \tag{8.15}$$

With this formulation we can cache the logarithm of each $\mu_{a,b,x}$ and $\mu_{a,x}$ and reuse them for sampling cluster assignment of each user. As such, we only need to take one pass over the original data and thus minimize the number of logarithm computations in each iteration. Sampling item cluster assignments can be done by an analogous sampler. We will show in Section 8.4.3 that this approach results in a fast sampling procedure.

Now we are ready to describe our full Gibbs sampling algorithm. For each stencil, we first update the rating model, following the algorithm in Chapter 7, and then update the text model using aforementioned techniques. Specifically, we resample $\hat{n}$ following (8.9), and then resample the stencil's μ terms (i.e. the Poisson language models associated with that stencil) based on $\hat{n}$, following (8.11). Finally, we update the cluster assignments for a given stencil following (8.12). The full procedure is given in Algorithm 12.

Algorithm 12: PACO Sampler

```
Run K-Means ACCAMS for initial cluster assignments.
Initialize all $\mu^{(i)}$ to a vector $\mathbf{1}$
while not converged do
    Resample $\{\hat{n}_{u,m,x}\}$ for all $u, m, x$ by (8.9)
    Sample $\mu^{(0)}$ by (8.11)
    for Stencil $\ell = 1, \ldots, S$ do
        Update predicted ratings as described in Ch. 7
        if Not first iteration then
            Resample $\{\hat{n}_{u,m,x}\}$ for all $u, m, x$ by (8.9)
        end if
        Sample $\mu^{(\ell)}, \mu^{(m,\ell)}, \mu^{(u,\ell)}$ by (8.11)
        Sample $\mathbf{c}_u^{(\ell)}$ by (8.12)
        Sample $\mathbf{d}_m^{(\ell)}$ analogous to (8.12)
    end for
    Sample $\mu^{(m)}$ by (8.11)
end while
```

8.4.3 Implementation

The sampler is written in C++11. The implementation is very efficient and uses the following techniques. Sampling $\{\hat{n}_{u,m,x}\}$ is embarrassingly parallelizable across reviews. With μ fixed, this involves simply parallel sampling from the re-parameterized multinomials[1]. Sampling Poisson rate μ is also embarrassingly parallelizable across μ and across words. We use C++11 implementation of Gamma sampler. Review likelihood can be efficiently calculated when sampling user/item cluster assignments. With $\tilde{\mu}_{a,b}^{(\ell)}$, $\tilde{\mu}_a^{(u,\ell)}$, $\log(\mu_{a,b,x}^{(\ell)})$ and $\log(\mu_{a,x}^{(u,\ell)})$ cached, review likelihood can be calculated with one pass over non-zero words. Again, this can be parallelized across users/items.

Speed: We generally found the above optimizations to make our sampler sufficiently fast for the large datasets we tested on. On the Amazon fine foods dataset described below, sampling all cluster assignments for one stencil takes 3.66 seconds on average, sampling all $\hat{n}$ for half million reviews takes 45 seconds, and sampling all μ for one stencil takes less than one second on a single machine.

8.5 Experiments

We now test our model in a variety of settings both to understand how it models different types of data and to demonstrate its performance against similar, recent models.

[1]GNU Scientific Library (GSL) is used for multinomial sampling. OpenMP is used to parallelize independent sampling.

8.5.1 Experimental Setup

Datasets To extensively test our model, we select four datasets about movies, beer, businesses, and food. All four datasets come from different websites and communities, thus capturing different styles and patterns of online ratings and reviews (imdb.com, ratebeer.com, yelp.com and amazon.com respectively). In all of these datasets, one observed (user,item) pair is associated one rating and one review. We randomly select 80% of data as training set and 20% as testing set while making sure every user/item in testing set has at least one example in training set. Infrequent words, standard stop words and words shorter than 3 characters are removed. We rescale ratings to the same range during training and center all ratings based on the global average in the dataset. The resulting datasets are summarized in Table 8.2.

Dataset	Yelp	Food	RateBeer	IMDb
Number of items	60,785	74,257	110,369	117,240
Number of users	366,715	256,055	29,265	452,627
Number of observations	1,569,264	568,447	2,924,163	1,462,124
Number of unigrams	9,055	9,088	8,962	9,182
Average review length	45.20	31.55	28.57	88.30

Table 8.2: **Four datasets used in experiments**. Infrequent words, standard stop words and words shorter than 3 characters are removed during pruning.

Metrics To evaluate our review model, we examine its ability to predict held-out testing ratings and reviews.

RMSE: When comparing predictions of held-out ratings, we use the root mean squared error (RMSE) to compare prediction quality. For PACO, we average predictions over many samples from the posterior, as is customary in evaluating sampling-based algorithms.

Perplexity: When evaluating the ability to predict held-out reviews, we compare perplexity of review text with other models. In order to have a comparable definition of perplexity, following the usual intuition as the average number of bits necessary to encode a particular word of review, we transform our model as follows. We use $\lambda_{u,m}$ as defined in (8.1) as the vector of expected counts for each word to be in a review from user u about item m. We transform this vector to a multinomial with probabilities $\theta_{u,m}$ where

$$\theta_{u,m,x} = \frac{\lambda_{u,m,x}}{\sum_x \lambda_{u,m,x}} \tag{8.16}$$

Note that when we average over multiple samples from the posterior, we average $\lambda_{u,m}$ over those samples and use the averaged rate of the Poisson in (8.16). With this multinomial, we calculate perplexity of testing set D_{test} as

$$-\log \mathrm{PPX}(D_{test}) = \frac{1}{N_w} \sum_{(u,m)\in D_{test}} \sum_{x} n_{u,m,x} \log \theta_{u,m,x} \tag{8.17}$$

where N_w is the total number of words in held-out testing reviews. Note that this now views our Poisson distributions in their expected state, but makes our quite different model more easily comparable to previous techniques.

Joint Negative Log-Likelihood: To jointly evaluate the models in terms of both rating and text prediction, we compare the joint negative log-likelihood. Per-review joint negative log-likelihood is defined as

$$-\log(\mathrm{PPX}) - \frac{1}{|D_{test}|} \sum_{(u,m)\in D_{test}} \log\left(\mathcal{N}\left(\mathbf{R}_{u,m}|\hat{\mathbf{R}}_{u,m}, \sigma^2\right)\right) \tag{8.18}$$

The text likelihood is normalized by number of words in review to equally weight the importance of predicting ratings and text. σ is taken from the training of each model.

Baseline methods We compare PACO with the following models:

PMF Probabilistic matrix factorization (PMF) [159] factorizes ratings into latent factors. It is simple in structure but usually effective. Number of latent factors Rank $\in \{10, 20\}$ were tested.[2]

BPMF Bayesian probabilistic matrix factorization (BPMF) [188] takes a more directly Bayesian approach to matrix factorization. Number of latent factors Rank $\in \{10, 20\}$ were tested.[2]

HFT Hidden factors with topics (HFT) [155] is one of the state-of-the-art models that jointly model ratings and reviews. It builds connection between topic distribution of reviews and latent factors. It shows significant improvement on rating prediction over traditional latent factor models on a variety of datasets. We use the implementation from the authors' website and run it with parameters recommended in the original paper.

JMARS JMARS [64] is another state-of-the-art model in joint-review-rating modeling. It explicitly models aspects, sentiments, ratings and reviews and provides interpretable and accurate recommendation. Similarly, parameters recommended in the original paper are used.[3]

Note, PMF and BPMF only optimize for prediction accuracy on ratings, thus only focusing on half of the problem we are attacking. However, we include them for completeness.

We test PACO with different priors and combinations of language models, where we include per-block language models, per-user cluster language models, or per-item cluster language models, or some combination thereof. For all datasets we use these joint text and rating stencils for the first S_0 stencils and then a series of rating-only stencils. All results from PACO are reported based for the best RMSE.

[2]Implementation at www.cs.toronto.edu/~rsalakhu/BPMF.html is used in experiments.

[3]The implementation is in Java.

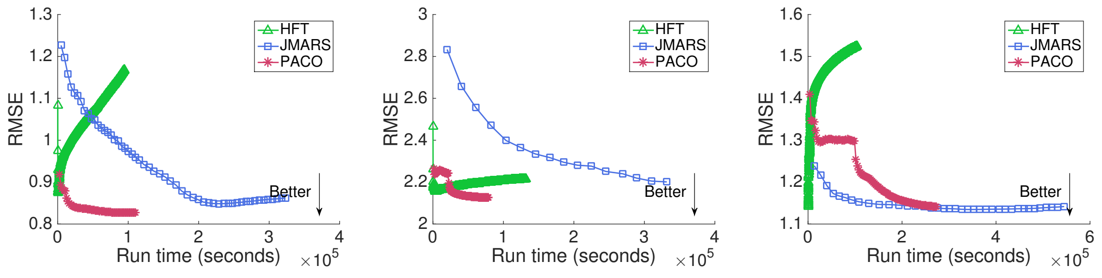
	HFT	JMARS	PACO
Yelp	20.1377	13.3171	8.9300
RateBeer	52.3546	30.2174	8.5994
Amazon fine foods	14.5827	10.1129	10.0904
IMDb	57.4515	33.7715	31.2567

Table 8.3: **Joint prediction accuracy for text reviews and ratings**, as given by joint negative log-likelihood (8.18), for all datasets. Lower is better.

8.5.2 Quantitative evaluation

Joint Predictive Ability

We compare the ability of PACO to predict jointly reviews and ratings against that of HFT and JMARS. In particular, we track the joint negative log-likelihood by runtime in Figure 8.1 and give the detailed numbers for best results are presented in Table 8.3. We observe that PACO converges rapidly and has superior performance to both competitors on all four datasets. We note that while HFT very quickly reaches reasonable accuracy on both text and ratings, it very quickly overfits its model of ratings, causing the surprising curve in Figure 8.1. However, results reported in Table 8.3 for all models are based on the best RMSE so as to prevent skew from overfitting. While we clearly offer high quality joint performance, we now look more closely at our prediction accuracy for ratings and for text separately.

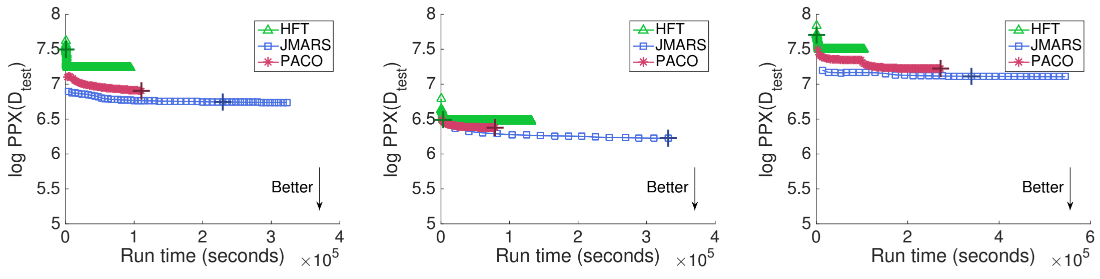


(a) Amazon fine foods (b) RateBeer (c) Yelp

Figure 8.3: **Rating prediction accuracy (RMSE)** compared by runtime to other joint modeling systems.

Rating prediction

We evaluate performance of rating prediction based on RMSE. Figure 8.3 shows RMSE over runtime, and Table 8.4 presents detailed results. A number of interesting patterns



(a) Amazon fine foods (b) RateBeer (c) Yelp

Figure 8.4: **Log perplexity of review text predictions** on three datasets. Plus sign markers indicate the values corresponding to the best RMSE. (Lower perplexity and time are both better.)

	PMF	BPMF	HFT	JMARS	PACO
Yelp	1.2649	1.1346	1.1408	1.1347	1.1407
RateBeer	2.1944	2.1164	2.1552	2.1675	2.1273
Amazon fine foods	0.8752	0.8193	0.8809	0.8486	0.8292
IMDb	2.5274	2.1622	2.2328	2.2947	2.1877

Table 8.4: **Rating prediction accuracy (RMSE)** across all four datasets. We observe that PACO generally outperforms the other joint learning models (HFT and JMARS) as well as PMF.

emerge in these results. In Figure 8.3, we observe more clearly that HFT converges extremely quickly before overfitting, but again results reported in Table 8.4 are from best RMSE before overfitting. For PACO we observe reasonable RMSE before burn-in and then quickly improved RMSE after burn-in. Finally we observe that JMARS is generally slower to converge.

In Table 8.4 we observe that PACO has superior performance to both HFT and JMARS on RateBeer, Amazon fine foods, and IMDb; JMARS performs slightly better on Yelp. While PACO generally outperforms the competing joint models, in these experiments it achieves slightly worse accuracy than BPMF. Here we observe a general trade-off between size and performance. As discussed in Chapter 7, using a sum of co-clusterings is far more succinct than bilinear models like BPMF. As a result, the ratings model in PACO is far smaller than that of BPMF, while we consistently achieve nearly as high of an accuracy.

Review Text Prediction

The second component of the predictive ability of our model is predicting review text, as measured by perplexity. We give the perplexity over runtime in Figure 8.4. We observe an apparent trade-off in perplexity for speed, simplicity and accuracy in rating prediction. PACO is efficient and outperforms HFT on all datasets. Note, in [155], HFT is described to primarily use text to improve rating predictions and not for predicting review. JMARS gives slightly better perplexity at the cost of significantly more complex model. Precise perplexity of HFT, JMARS, and PACO are given in Table 8.5. Since our primary goal is recommendation, the perplexity reported correspond to the points that obtain the best RMSE.

	HFT	JMARS	PACO
Yelp	7.7031	7.1112	7.2223
RateBeer	6.4891	6.2098	6.3779
Amazon fine foods	7.5015	6.7450	6.8759
IMDb	8.1747	7.4610	7.5540

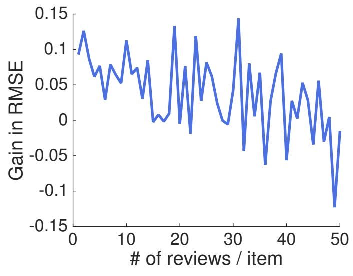
Table 8.5: **Perplexity** on all four datasets. The perplexity values reported here correspond to the points that obtain the best RMSE.

Cold-start

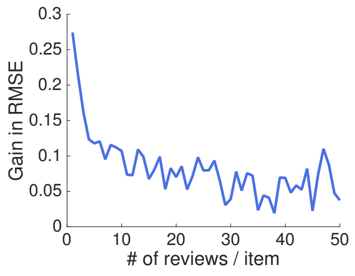
For the sake of thoroughness, we looked at where does our joint model excel at rating prediction over the rating-only PMF model. We generally found PACO to help alleviate cold-start challenges. We show improvement in RMSE over PMF for items and users with different number of training examples in Figure 8.5 and 8.6. We observe that for items with fewer observed ratings we achieve a greater improvement in rating prediction accuracy. This suggests that PACO is able to extract rich information from reviews and provide benefits especially when items have scarce signals. For users with few ratings, we can see similar trends except for Amazon fine foods. One hypothesis is that the quality of food is less subjective than movies or beers. It is thus harder to learn user preference from the review in this case.

8.5.3 Interpretability

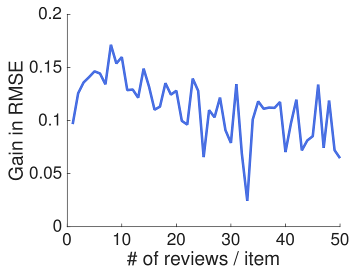
In addition to quantitatively evaluating our method, we also want to empirically demonstrate that the patterns surfaced and review predictions would be useful to the human eye. However, because reviews from each dataset follow very different patterns, we



(a) Amazon fine foods

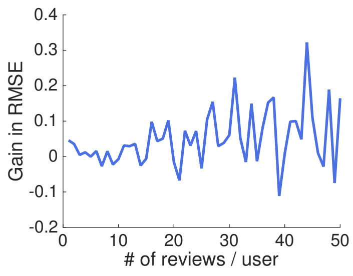


(b) RateBeer

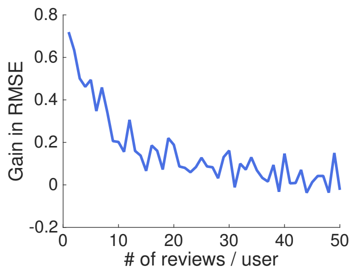


(c) Yelp

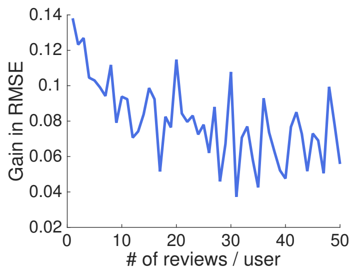
Figure 8.5: Gain of PACO over PMF in RMSE demonstrates the benefits for items with few observed ratings.



(a) Amazon fine foods



(b) RateBeer



(c) Yelp

Figure 8.6: Gain of PACO over PMF in RMSE demonstrates the benefits for users with few observed ratings.

expect PACO to model them differently. In this effort, we analyzed our learned models to understand which parts of the models were effective in understanding each of these datasets.

Sentiment word extraction: We first checked to make sure that blocks in our co-clustering that had high rating predictions found appropriately positive or negative words. In Table 8.6 we present the top words in the highest-rating block and lowest-rating block in the first stencil for each dataset. We can see in general, in that high-rating blocks, there are more positive sentiment words, and in lower-rating blocks, more negative words. This is less clear in the RateBeer and Amazon fine foods datasets where reviews more strongly focus on describing the item. However, we still note the strong association with "good" for a positive rating and in some cases "bad" for a negative rating.

Item-specific words: In Table 8.7, top item-specific words associated with some popular items are presented. In general these words provide basic descriptions unique to that particular item. We do observe some overfitting in cases where there are fewer reviews, but generally the item-specific language model improves predictive performance.

Item-clusters and cluster-specific words: In Table 8.8 we present 4 clusters of items and the words associated with them from Amazon fine foods and IMDb. For Amazon fine

Dataset	Rating	Words
IMDb	0.022	great, love, movies, story, life, watch, time, people, character, characters, best, films, scene, watching
	-0.018	bad, good, plot, like, worst, money, waste, acting, script, movies, minutes, horrible, boring, thought
Fine foods	0.024	good, love, great, like, product, amazon, store, time, find, eat, price, years, buy, coffee, taste, stores
	-0.048	like, buy, taste, time, product, bought, purchased, good, pretty, thought, reviews, smell, purchase
Yelp	0.51	highly, recommend, professional, amazing, job, customer, work, best, appointment, staff, great, needed
	-0.76	told, manager, customer, called, call, rude, asked, horrible, worst, phone, minutes, order, money, hotel
RateBeer	0.15	nice, pours, hops, flavor, hop, citrus, color, taste, finish, tap, good, sweet, bitterness, white, malt, light
	-0.22	taste, bad, beer, like, color, good, decent, weak, drink, pours, beers, boring, special, watery, dont, bottled

Table 8.6: **Blocks predict words** matching the sentiment of the predicted rating.

Item	Item-specific words
The Dark Knight	batman, joker, dark, ledger, knight, heath, nolan, best, performance, bale
Silent Hill	game, silent, hill, games, horror, video, rose, town, like, played, plot, scary
Purina Pro Plan Canned Kitten Chicken Liver Food	food, time, kittens, kitty, liver, cats, feral, white, plan, female, guys, kitten
Peace Cereal Low Fat Clusters & Flakes Cereal Raisin Bran	cereal, sugar, fat, wheat, calories, color, barley, raisins, ingredients, listed
Luxury Nail & Spa	hair, appointment, stylist, cut, salon, desk, paying, stylists, bid, grille, color
Enrico's Tazza D'oro Cafe & Espresso Bar	coffee, espresso, pittsburgh, pastries, park, baristas, cappuccino, atmosphere
Barley Island Sheet Metal Blonde	wheat, orange, wit, coriander, clove, bubblegum, slice, chamomile, witbier
Rock Bottom Braintree Boston Fog Lager	peaches, dishwater, lager, perfumey, disipating, fog, outspoken, component

Table 8.7: **Item-specific words** capture concepts highly specific to the individual item.

foods, we see items with similar categories are clustered. For example, the coffee cluster and the snack cluster are learned effectively, as presented in the first and second row of the table. On the other hand, PACO learns relatively general clusters for IMDb. For example, while one presented cluster captures exciting or action movies, the other one groups generally lower-rated movies, and the associated words are general negative sentiment words that would be in a movie review.
Review prediction: Finally, for RateBeer we give examples of generally well-predicted held-out reviews and the top words the entire PACO model predicts for them. Because RateBeer reviews are largely descriptive of the item, we find that PACO is effective in predicting the properties of the items, particularly focused on the type of beer.

Subset of items in cluster	Cluster words
Melitta Cafe de Europa Gourmet Coffee, Flavored, Coffee People Black Tiger, Dark Roast, K-Cup for Keurig Brewers, K-Cup Portion Pack for Keurig K-Cup Brewers, Lavazza Super Crema Espresso - Whole Bean Coffee (Amazon fine foods)	like, coffee, good, taste, flavor, cup, drink, nice, product, thought, great, tastes, tasting, drinking, best, full, time, buy, recommend, enjoy, brand, love, strong, blend, black, regular, bit, bad, recommended, size
Ice Breakers Ice Cubes Sugar Free Gum, Kiwi Watermelon, Bell Plantation PB2 Powdered Peanut Butter, PB2 Powdered Peanut Butter, Ella's Kitchen Organic Smoothie Fruits, The Red One, Blue Diamond Almonds Bold Lime n Chili (Amazon fine foods)	taste, snack, like, good, eat, eating, buy, bag, love, diet, healthy, fat, great, flavor, store, sweet, salty, healthier, price, amount, bags, find, case, crunchy, size, tasty, ate, packs, texture, yummy
Entrapment, Mission: Impossible III, Zombie, Snake Eyes, Starsky & Hutch, New England Patriots vs. Minnesota Vikings, I Am Legend, Chaos (IMDb)	action, good, character, thought, story, plot, scene, expected, average, movies, game, scenes, lack, massive, destruction, entertained, suspenseful, audience, seats, batman, pulls, mistakes, steel, effect, shopping, richardson, atmosphere, ford, genetic, horrific
Gargantua, Random Hearts, Chocolate: Deep Dark Secrets, Blackout, The Ventures of Marguerite, Irresistible, Ghosts of Girlfriends Past, Youth Without Youth (IMDb)	like, good, bad, time, movies, people, acting, plot, watch, horror, watching, worst, scenes, pretty, awful, effects, scene, characters, thought, story, actors, worse, films, terrible, special, lot, fun, give, stupid, guy

Table 8.8: **Discovered clusters of items** and associated topics for Amazon fine foods and IMDb.

Real review	Predicted words (ordered by likelihood)
poured from the bottle pitch black with a caramel head smells like a great espresso with a little bit of oatmeal in there great creamy mouthfeel tastes is strong of very bitter coffee and oatmeal the booze is pretty well hidden this is one tasty stout	coffee, head, aroma, beer, roasted, sweet, light, malt, bitter, bottle, taste, stout, flavor, dark, like, finish, thick, white, brown, nice, creamy, good, tan, medium, pours, smooth, chocolate, body, caramel, great
tap at pour is hazy orange gold with a white head aroma shows notes of wheat tangerine orange yeast and coriander flavor shows the same with light vanilla	orange, white, head, citrus, aroma, light, wheat, sweet, hazy, flavor, malt, yeast, finish, beer, coriander, spice, medium, bottle, body, nice, taste, hops, good, lemon, pours, cloudy, bitter, notes, color, caramel
this is a pale ale it is a pale orangish color and it is an ale hops dominate the nose but there is a more than ample malt backbone good medium mouthfeel clean hoppy follow through beer as it ought to be	head, hops, aroma, ipa, nice, good, flavor, beer, citrus, hop, hoppy, taste, sweet, finish, bottle, malt, white, pours, light, color, medium, pine, golden, bitter, like, grapefruit, body, amber, floral, caramel

Table 8.9: **Predicted words for held-out reviews** on RateBeer.

8.6 Summary

We presented PACO, an additive co-clustering algorithm for explainable recommendations. We offer three primary contributions:

- **Joint Model:** A Poisson additive co-clustering model enabling joint interpretable modeling of ratings and reviews.
- **Efficient Sampler:** A novel algorithm for sampling from a sum of Poisson random variables.
- **Empirical evidence** that PACO models both ratings and text well on a variety of datasets.

Part IV

Scalable Machine Learning

Based primarily on work previously published in
SDM 2014 [30], *AISTATS* 2014 [127].
Additional segments based on [32].

Introduction

How can we efficiently learn user behavior models over many machines?
How can our learning algorithms adapt to the messy realities of "the cloud," such as stragglers?

Online services that depend on user-generated content are made useful by aggregating the knowledge of *many* users about *many* items in the world. As a result, these services capture large graphs that provide great insight into the world. On Facebook, there are now over 1.5 billion users [4], 350 million photos uploaded each day [5], billions of likes per day, and many billions of friendships [215]; on Amazon there are hundreds of millions of products; on Twitter there are over 300 million users [6]. In order for powerful algorithms, like those discussed in Parts II and III, to be useful for these services, the algorithms must scale to their large graphs. This challenge includes scaling to large datasets, with relations among multiple types of data, and creating models of many different items in the world.

In the following chapters we focus on developing distributed platforms for scalable user behavior modeling. While we have worked to developed novel behavior models, factorization techniques have been the bread and butter of behavior modeling of many years, and as such we focus on scaling latent factor models of this form. We offer the following contributions:

- **Distributed Modeling of Attributed Hypergraphs:** In Chapter 9, we offer the FlexiFaCT system to scale factorization of attributed hypergraphs through coupled tensor factorization. We demonstrate that our system can scale to billions of parameters and is up to 190 times faster than previous methods.
- **Fast in the face of Stragglers:** In practice, due to imbalanced machines or concurrent programs, some machines in real-world clouds are slower than others, resulting in bottlenecks to distributed learning. In Chapter 10, we offer a simple technique to learn efficiently, even in the face of stragglers. Our system, Fugue, is up to 26 times as fast as competitors in topic modeling, dictionary learning, and community detection.

[4]https://newsroom.fb.com/company-info/
[5]https://newsroom.fb.com/products/
[6]https://about.twitter.com/company

IV.1 Related Work

We begin with a survey of related work for scalable and distributed machine learning for user behavior modeling.

IV.1.1 Big Data Processing

Distributed file systems like the Hadoop Distributed Filesystem (HDFS) [75] have made it economical to store large quantities of data for extended periods of time. Variants of MapReduce [61, 235] are often used to analyze this data, and the value derived from it accelerated the overall trend of keeping ever increasing amounts of data with the expectation of eventual value extraction. Because MapReduce is not well suited to iterative computations, a number of alternative frameworks have been developed [238], some with fundamentally different programming models [148, 152]. We go into greater detail on these systems below.

IV.1.2 Distributed Learning

Machine learning algorithms are fundamentally different from other big data challenges, as they are often iterative, stochastic algorithms with complex dependencies. These novel properties have led to a wide variety of systems to parallelize and distribute machine learning, each trying to handle the unique challenges and exploit the unique properties of the learning.

Some papers focus on parallelization across the dataset to handle the often big datasets associated with machine learning. Most of these papers exploit data point independence to construct stochastic distributed optimization schemes with little need for inter-machine synchronization. For example, the PSGD algorithm [243] is completely data parallel and requires absolutely no inter-machine communication, therefore making it trivial to implement. However, in practice, one can almost always obtain faster convergence with some inter-machine communication, as our experiments will show in the following chapters.

For learning large models, the general strategy is to exploit the fact that each model variable usually depends on only a small number of other variables, and then partition model variables in a way that limits the number of dependencies that cross machines. The GraphLab system [149] is a good example of this concept, but it requires machine learning algorithms to be rewritten into "vertex programs," which can be awkward or even difficult for some algorithms. Furthermore, there has been little theoretical analysis on machine learning algorithms running under GraphLab. The Google Brain project [60] provides another example of the need for model partitioning, this time for a custom-built deep network meant for image and video feature extraction. However, the paper does not provide general partitioning strategies for arbitrary models.

Another class of big machine learning systems is parameter servers, which provide a distributed shared-memory interface for large numbers of model parameters [13, 140, 141, 182], but perform no variable scheduling themselves. Recent work on parameter servers has led to new, theoretically-sound computational models such as Stale Synchronous Parallelism (SSP) [54, 96].

IV.1.3 Matrix Factorization

Because of their importance to user behavior modeling, we primarily focus on factorization techniques. As described in previous parts of this dissertation, a wide variety of matrix factorization formulations have been proposed [134]. In addition to the variety of matrix factorization models, many other models follow related bilinear structures, such as topic modeling [35], dictionary learning [125], and mixed-membership stochastic blockmodels [14].

A variety of learning algorithms have been used to learn the parameters in matrix factorizations, such as alternating least squares (ALS) [237] and stochastic gradient descent (SGD) [37].

Because of its pervasiveness, significant research has focused on fast matrix factorization and specialized systems for distributed learning of matrix factorizations. [104] demonstrated how to efficiently use ALS for matrix factorization. More recently, both Hogwild [167] and DSGD [77] exploited the patterns of stochastic gradient descent for parallelized matrix factorization. Of particular note, Gemulla *et al.* [77] made a breakthrough in scaling matrix factorization, by proposing a distributed version of SGD for matrix factorization that elegantly exploits the factorization structure of the problem, breaking the matrix into blocks and process them in parallel. Follow-up works [185, 209] have successfully extended this idea to the matrix completion problem using parallel stochastic updates. Since the publication of our work, other systems also built on these insights for asynchronous matrix factorization, such as [241] for shared memory systems and [236] for distributed learning.

IV.1.4 Tensor Factorization

As was outlined previously in this dissertation, tensor factorization is a powerful tool for modeling multimodal data [119]. The most popular tensor factorization is the so called Canonical Polyadic (CP) or PARAFAC decomposition [93]. As with matrix factorization, many variants of tensor factorization have been introduced; e.g., PARAFAC with ℓ_1 constraints is introduced in [177].

Like matrix factorization, significant research has focused on making tensor factorization efficient. For in-memory single-machine computations, the state of the art is the Tensor Toolbox [25], which provides an implementation of the ALS algorithm. Besides ALS there exist first order optimization techniques, such as [4]. In [205], the authors propose algorithms for incremental computation of the tensor decomposition, where the tensor is a stream. For large datasets residing on HDFS, the state-of-the-art at the time of publication was GigaTensor [116], which solves the PARAFAC decomposition using Alternating Least Squares (ALS). Later work has achieved additional improvements for scalability [51, 195].

As described previously, in some cases we have a tensor and a matrix, or two tensors, or two matrices, that share one dimension. Singh *et al.* [197] jointly factorize two related matrices for better decomposition and provide stochastic updates for this coupled optimization. For coupled matrix-tensor factorization, recent work [5] used first order optimization techniques.

Chapter 9

Distributed Modeling of Attributed Hypergraphs

Given multiple data sets of relational data that share a number of dimensions, how can we efficiently decompose our data into the latent factors? Factorization of a single matrix or tensor has attracted much attention, as, e.g., in the Netflix challenge, with users rating movies. However, we often have additional, side, information, like, e.g., demographic data about the users, in the Netflix example above. Incorporating the additional information leads to the *coupled factorization* problem. Previously, it was only solved for relatively small datasets.

In this chapter, we provide a distributed, scalable method for decomposing matrices, tensors, and coupled data sets through stochastic gradient descent on a variety of objective functions. We offer the following contributions: (1) *Versatility:* Our algorithm, FlexiFaCT, can perform matrix, tensor, and coupled factorization, with flexible objective functions including parameter constraints, such as non-negative factorization and ℓ_1 induced sparsity. (2) *Scalability:* FlexiFaCT scales to unprecedented sizes in both the data and model, with up to *billions* of parameters. (3) *Usability:* our implementation is open-source and runs on *stock* Hadoop.

9.1 Introduction

How can we efficiently mine data that capture relations between different entities? Suppose, for instance, that we are given a time-evolving social network, such as Facebook, and we have information about who messages whom, or who becomes friends with whom, and when. This data may be formulated as a three mode tensor. Suppose now that we also have some side information pertaining to the users, e.g., demographic information. This problem can be formulated as an instance of a so-called *coupled factorization*, where the two pieces of data, a three-mode (user, user, time) tensor and a

Data/Model	FlexiFaCT	DSGD [77]	PSGD [243]	Matlab	GigaTensor [116]
Matrix	✓	✓	$\sim$	✓	
Tensor	✓		$\sim$	✓[25]	✓
Coupled Tensor/Matrix	✓		$\sim$	✓[5, 6]	
Obj. Function					
Frobenius norm	✓	✓	✓	✓	✓
Frobenius norm + ℓ_1 penalty	✓		✓	✓	
Non-negativity constraints	✓		✓	✓	
Handles missing data	✓	✓	✓	✓	
Scalability					
in number of non-zeros	✓	$\sim$	✓		✓
in data dimensions	✓	$\sim$			✓
in decomposition rank	✓	$\sim$	✓		$\sim$
Proof of convergence					
Matrix Factorization	✓	✓			
Tensor/Coupled Factorization	✓	$\sim$			✓
Projections (ℓ_1 & non-negativity)	✓		$\sim$		

Table 9.1: **Feature comparison of proposed FlexiFaCT** vs previous methods. ($\sim$ represents unknown or not directly applicable.) FlexiFaCT contains existing methods as special cases.

(user, demographic) matrix share a common dimension. Even without the presence of the (user, demographic) matrix, efficient tensor decomposition of truly large datasets can be challenging, attracting increasing interest.

Most prior work has either focused on a specific type of factorization or a specific loss function (e.g., Frobenius norm), thus having a limited range of potential applications. Here we propose FlexiFaCT, a flexible and highly scalable distributed factorization algorithm which attacks a very broad spectrum of problems: FlexiFaCT can handle matrices, tensors, coupled tensor-matrix settings, *cross product* a variety of loss functions, including Frobenius norm, KL divergence, ℓ_1 regularization, and non-negativity constraints.

Moreover, FlexiFaCT is very fast and scalable; we show how to implement it on Hadoop, and we show how to achieve high speeds, by distributing both the data as well as the parameters. In Table 9.1 , we provide a comprehensive overview of the previous methods. In short, FlexiFaCT uniquely distributes and scales coupled tensor factorization across a wide variety of objectives.

In summary, our main contributions are:

1. **Versatility**: FlexiFaCT can operate under a wide spectrum of settings, including

plain matrix factorization, tensor factorization, as well as coupled decompositions. Thus, FlexiFaCT includes several recent methods [77, 116], as *special cases*.

2. **Scalability**: FlexiFaCT scales very well both with the input size, as well as with the number of model parameters.
3. **Usability and Reproducibility**: Our implementation runs on *stock* Hadoop, as opposed to other recent methods [77]. We also open-source our code.

9.2 FlexiFaCT Approach

Symbol	Description
$\boldsymbol{\mathcal{X}}$	Data tensor
$\mathbf{Y}$	Data matrix
$\mathbf{U}, \mathbf{V}, \mathbf{W}$	Factor matrices of $\boldsymbol{\mathcal{X}}$
$\mathbf{U}, \mathbf{A}$	Factor matrices of $\mathbf{Y}$
$I \times J \times K$	Dimensions of data tensor $\boldsymbol{\mathcal{X}}$
$I \times M$	Dimensions of data matrix $\mathbf{Y}$
θ	Any parameter in the objective
σ	The parameter being updated
∇	Symbol for derivative
η_t	Step size at iteration t
R	Rank of decomposition
$\otimes$	Outer product

Table 9.2: Notation used in this chapter

As mentioned previously, we take on the problem of matrix, tensor and coupled factorization. In this section we will explain the variety of loss functions used in these tasks, the Stochastic Gradient Descent (SGD) update rules, and our partitioning scheme allowing for distribution of the SGD work. Although much of our description of the *matrix* factorization work is similar to [77], we will explain it here for completeness and clarity. A list of our commonly used symbols can be seen in Table 9.2.

9.2.1 Optimization Objectives

We begin by explaining how stochastic gradient descent works for our variety of objective functions. We will briefly go over the objective functions for simpler cases like the Frobenius norm of matrices before expanding to more complex objectives.

Matrix Factorization For matrix factorization we would like to approximate our $I \times J$ data matrix $\mathbf{X}$ by $\mathbf{U}\mathbf{V}^T$, where $\mathbf{U}$ is of size $I \times R$ and $\mathbf{V}$ is of size $J \times R$. Therefore, we can have a loss function using the Frobenius norm as follows:

$$L(\mathbf{U}, \mathbf{V}) = \|\mathbf{X} - \mathbf{U}\mathbf{V}^T\|_F^2 = \sum_{i,j \in \mathbf{X}} L_{\mathbf{X}_{i,j}}(\mathbf{U}, \mathbf{V}) \tag{9.1}$$

where $L_{\mathbf{X}_{i,j}}(\mathbf{U}, \mathbf{V}) = (\mathbf{X}_{i,j} - \sum_{r=1}^{R} \mathbf{U}_{i,r}\mathbf{V}_{j,r})^2$. As seen above, we divide our loss function into its component pieces $L_{\mathbf{X}_{i,j}}$ based on each observed point $\mathbf{X}_{i,j}$. This is necessary to use stochastic gradient descent.

Tensor Factorization For tensor factorization we would like to approximate our $I \times J \times K$ tensor $\boldsymbol{\mathcal{X}}$ by an outer product $\sum_{r=1}^{R} \mathbf{U}_{*,r} \otimes \mathbf{V}_{*,r} \otimes \mathbf{W}_{*,r}$ where $\mathbf{U}$ is of size $I \times R$, $\mathbf{V}$ is of size $J \times R$ and $\mathbf{W}$ is of size $K \times R$ and we are performing an outer product between these three matrices. We can analyze the loss in a few different ways. Following the standard Frobenius norm, as is common in PARAFAC, the loss is:

$$\begin{aligned} L(\mathbf{U}, \mathbf{V}, \mathbf{W}) &= \|\boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \mathbf{U}_{*,r} \otimes \mathbf{V}_{*,r} \otimes \mathbf{W}_{*,r}\|_F^2 \\ &= \sum_{(i,j,k) \in \boldsymbol{\mathcal{X}}} (\boldsymbol{\mathcal{X}}_{i,j,k} - \sum_{r=1}^{R} \mathbf{U}_{i,r}\mathbf{V}_{j,r}\mathbf{W}_{k,r})^2 \\ &= \sum_{(i,j,k) \in \boldsymbol{\mathcal{X}}} L_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\mathbf{U}, \mathbf{V}, \mathbf{W}) \end{aligned}$$

where

$$L_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\mathbf{U}, \mathbf{V}, \mathbf{W}) = (\boldsymbol{\mathcal{X}}_{i,j,k} - \sum_{r=1}^{R} \mathbf{U}_{i,r}\mathbf{V}_{j,r}\mathbf{W}_{k,r})^2.$$

Similarly we can induce sparsity in our parameter space with an ℓ_1 penalty:

$$\begin{aligned} \mathcal{L}(\mathbf{U}, \mathbf{V}, \mathbf{W}) &= \|\boldsymbol{\mathcal{X}} - \sum_{r=1}^{R} \mathbf{U}_{*,r} \otimes \mathbf{V}_{*,r} \otimes \mathbf{W}_{*,r}\|_F^2 \\ &\quad + \lambda(\|\mathbf{U}\|_1 + \|\mathbf{V}\|_1 + \|\mathbf{W}\|_1) \\ &= L(\mathbf{U}, \mathbf{V}, \mathbf{W}) + \lambda(\|\mathbf{U}\|_1 + \|\mathbf{V}\|_1 + \|\mathbf{W}\|_1) \end{aligned} \tag{9.2}$$

or add a constraint that $\mathbf{U}, \mathbf{V}, \mathbf{W} \geq 0$ as is common in non-negative matrix factorization (NNMF). These terms are not as clearly separable in the loss function, but as we will see the update rules are still separable as is necessary for SGD. We make a distinction here between $\mathcal{L}$ and L: the objective $\mathcal{L}$ is obtained by adding ℓ_1 or non-negativity constraints to the loss L.

Objective ($\mathcal{L}$)	**Formulation**
Frobenius	$\sum_{(i,j,k)\in\boldsymbol{\mathcal{X}}} \mathcal{L}_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\mathbf{U},\mathbf{V},\mathbf{W}) + \sum_{(i,j)\in\mathbf{Y}} \mathcal{L}_{\mathbf{Y}_{i,j}}(\mathbf{U},\mathbf{A})$
Frobenius + ℓ_1	$\sum_{(i,j,k)\in\boldsymbol{\mathcal{X}}} \mathcal{L}_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\mathbf{U},\mathbf{V},\mathbf{W}) + \sum_{(i,j)\in\mathbf{Y}} \mathcal{L}_{\mathbf{Y}_{i,j}}(\mathbf{U},\mathbf{A})$ $+ \lambda(\|U\|_1 + \|V\|_1 + \|W\|_1 + \|A\|_1)$
Frobenius + ℓ_1 + NN	$\sum_{(i,j,k)\in\boldsymbol{\mathcal{X}}} \mathcal{L}_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\mathbf{U},\mathbf{V},\mathbf{W}) + \sum_{(i,j)\in\mathbf{Y}} \mathcal{L}_{\mathbf{Y}_{i,j}}(\mathbf{U},\mathbf{A})$ $+ \lambda(\|U\|_1 + \|V\|_1 + \|W\|_1 + \|A\|_1)\ s.t.\ \Theta_{i,r} \geq 0$

Table 9.3: **Objective functions handled by FlexiFaCT: $\boldsymbol{\Theta}$** denotes any of the factors U, V, W or A.

Coupled Matrix-Tensor Factorization In this case our data tensor $\boldsymbol{\mathcal{X}}$ is approximated by $\sum_{r=1}^{R} \mathbf{U}_{*,r} \otimes \mathbf{V}_{*,r} \otimes \mathbf{W}_{*,r}$ and our data matrix $\mathbf{Y}$ is simultaneously approximated by $\mathbf{U}\mathbf{A}^T$. Note here we use the same component $\mathbf{U}$ in both approximations. As such, our objective function is merely a sum of the losses on each data set:

$$\mathcal{L}(\mathbf{U},\mathbf{V},\mathbf{W},\mathbf{A}) = \sum_{(i,j,k)\in\boldsymbol{\mathcal{X}}} \mathcal{L}_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\mathbf{U},\mathbf{V},\mathbf{W}) + \sum_{(i,j)\in\mathbf{Y}} \mathcal{L}_{\mathbf{Y}_{i,j}}(\mathbf{U},\mathbf{A})$$

Table 9.3 denotes the loss objectives for different coupled cases. For each of these we use SGD to minimize our loss and thus approximate our data.

9.2.2 SGD Updates

For SGD we perform updates to our parameters $\mathbf{U}, \mathbf{V}, \mathbf{W}, \mathbf{A}$, which we will collectively refer to as $\boldsymbol{\Theta}$ matrix whereas θ are the individual components of the matrix. This definition of $\boldsymbol{\Theta}$ and θ will come in handy for parameter updates based on the gradient at individual data points. For example, the update for tensor $\boldsymbol{\mathcal{X}}$ are:

$$\theta^{(t+1)} = \theta^{(t)} - \eta_t \nabla \mathcal{L}_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\theta^{(t)}) \tag{9.3}$$

For these update rules, we list below the differentials for each component σ of θ where $(\nabla L_{X_{i,j,k}}(\theta))_\sigma = \frac{\partial L_{X_{i,j,k}}}{\partial \sigma}$:

$$(\nabla L_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\theta))_\sigma = \begin{cases} -2(\boldsymbol{\mathcal{X}}_{i,j,k} - \sum_r \mathbf{U}_{i,r}\mathbf{V}_{j,r}\mathbf{W}_{k,r})\mathbf{V}_{j,\ell}\mathbf{W}_{k,\ell} & \text{if} \sigma = \mathbf{U}_{i,\ell} \\ 0 \quad \text{if} \sigma = \mathbf{U}_{i',\ell},\ i \neq i' \end{cases}$$

similarly for $\sigma = \mathbf{V}_{j,l}$ or $\sigma = \mathbf{W}_{k,l}$. From this we observe that SGD update for $\mathbf{U}_{i,l}$ at a particular entry $\boldsymbol{\mathcal{X}}_{i,j,k}$ (for a tensor $\boldsymbol{\mathcal{X}}$) depends only on previous $\mathbf{U}_{i,r}, \mathbf{V}_{j,r}, \mathbf{W}_{k,r}$ where

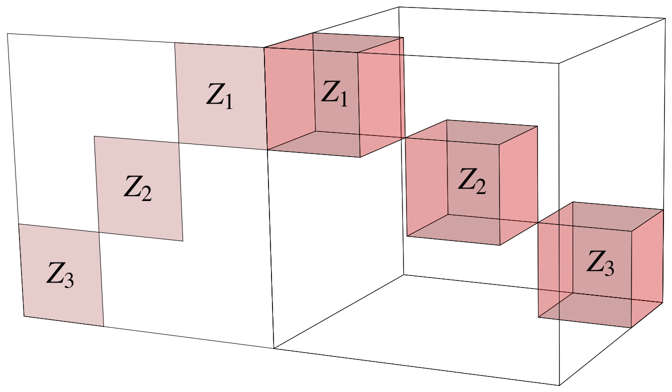


Figure 9.1: **Partitioned matrix and tensor for parallelization:** Dividing the paired matrix and tensor into blocks such that no two of them share any row or a column or a third dimension in case of tensor. Through multiple different configurations of the blocks we can use all data available.

$r \in 1, \ldots, R$ and R is the rank we chose. The updates for each component are similar for the paired cases.

In the case of additional components such as an ℓ_1 penalty or a non-negativity constraint on our parameters, we add a projection to our update rule. For example, for an ℓ_1 penalty, the update rule is

$$\theta^{(t+1)} = S_\lambda(\theta^{(t)} - \eta_t \nabla \mathcal{L}_{\boldsymbol{\mathcal{X}}_{i,j,k}}(\theta^{(t)})) \tag{9.4}$$

$$S_\lambda(x) = \begin{cases} x - \lambda & \text{if } x > \lambda \\ x + \lambda & \text{if } x < -\lambda \\ 0 & \text{if } -\lambda \leq x \leq \lambda \end{cases} \tag{9.5}$$

Here we see that S_λ is the soft thresholding operator. We can similarly use the following projection for the non-negativity constraint:

$$NN(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{9.6}$$

9.2.3 Blocking for Parallelization

Given this understanding of our optimization objective and SGD update rules, we would like to segment our data in such a way that certain blocks Z_b can be run in parallel, where we define $Z_b \subseteq \boldsymbol{\mathcal{X}}$. Gemulla *et al.* provide a partitioning algorithm for matrices [77]; here, we generalize their approach to tensors and coupled tensors and matrices, where independence between data points is more restrictive and more complex. Figure 9.1 is a pictorial representation of the way we segment our simple matrix or a coupled tensor/matrix to enable parallelization. In order to run SGD on our blocks in parallel, we divide them such that no two blocks share common rows or columns. To be more precise, we say that a point $x \in Z_b$ is the coordinates in the data, such as $x = (x_i, x_j, x_k) \in \boldsymbol{\mathcal{X}}$. Two blocks Z_b and $Z_{b'}$ are non-overlapping if for all $x \in Z_b$ and $x' \in Z_{b'}$, $x_i \neq x'_i$ and $x_j \neq x'_j$ and $x_k \neq x'_k$. (We will prove later that this allows us to run the blocks in parallel.) We see that in the division shown in Figure 9.1 no two blocks share common rows or columns. More interestingly, we note that blocks in the tensor $\boldsymbol{\mathcal{X}}$ and the matrix $\mathbf{Y}$ share coordinates in the i dimension, and as a result, data points in the same i range must be in the same block across both data sets.

Given this intuition, we provide a detailed description of our partitioning algorithm. We call one set of independent blocks a stratum, and we denote the number of blocks in each stratum by d. In order to cover all regions of $\boldsymbol{\mathcal{X}}$, we need multiple strata. For a matrix we require d strata, and for 3-mode tensors we require d^2 strata. (One can easily see the generalization to a k-mode tensor requiring d^{k-1} strata.) For a stratum s we have blocks $Z_i^{(s)}$ for $i = 0 \ldots d-1$. Each block $Z = (b_i, b_j, b_k)$ where b_i, b_j, b_k are ranges in I, J, and K: $b_i = (i\lceil I/d \rceil, (i+1)\lceil I/d \rceil), b_j = (j\lceil J/d \rceil, (j+1)\lceil J/d \rceil), b_k = (k\lceil K/d \rceil, (k+1)\lceil K/d \rceil)$. With this we define the blocks for stratum s as

$$Z_i^{(s)} = (b_i, b_{j_{s,i}}, b_{k_{s,i}}) \tag{9.7}$$

$$j_{s,i} = (j + s) \bmod d \tag{9.8}$$

$$k_{s,i} = \lfloor (j + s)/d \rfloor \bmod d \tag{9.9}$$

for $i = 0 \ldots d-1$.

In our algorithm, we run the strata sequentially, but for each stratum we run SGD on the blocks in parallel. We consider running SGD on one stratum to be a subepoch in our algorithm, and running it on all strata an epoch. (Note, the order in which you run the strata does not matter, as long as they are each run once per epoch.) We can do this repeatedly, iteratively updating our parameters θ, until the algorithm converges. A more formal write up of the distributed stochastic gradient algorithm for a tensor (which can easily be generalized to matrices and coupled factorizations) is shown in Algorithm 13. We next offer a proof that this converges appropriately.

9.3 Proof Sketch

The FlexiFaCT approach is described in Algorithm 13. We provide here a proof sketch for the convergence of Algorithm 13 to a local optima; the complete proof can be found in Appendix C. We first prove that two blocks in a stratum are interchangeable. We use this to prove that sequence of strata forms a regenerative process, defined later in this section. We use this to prove that our FlexiFaCT approach for tensor and coupled case

Algorithm 13: FlexiFaCT for tensor factorization

```
Input : $\boldsymbol{\mathcal{X}}, \mathbf{U}_0, \mathbf{V}_0, \mathbf{W}_0$,sub-epoch size $d$
$\mathbf{U} \leftarrow \mathbf{U}_0$, $\mathbf{V} \leftarrow \mathbf{V}_0$, $\mathbf{W} \leftarrow \mathbf{W}_0$
Block $\boldsymbol{\mathcal{X}}, \mathbf{U}, \mathbf{V}, \mathbf{W}$ into corresponding $d$ blocks
while not converged do
    Pick step size $\eta$
    for $s = 0, ..., d^2 - 1$ do
        Pick $d$ blocks($Z_1^{(s)}, ..., Z_d^{(s)}$) to form stratum $Z^{(s)}$
        for $b = 0, \dots, d-1$ in parallel do
            Run SGD on the training points $Z_b^{(s)}$
```

converges. Note, this goes beyond the theoretical results provided in [77] by covering projections, and thus enabling constraints and regularization.

Our generic constrained loss function for a tensor case is

$$\begin{aligned} \mathcal{L} = L(U, V, W) + \lambda_u \|U\|_1 + \lambda_v \|V\|_1 + \lambda_w \|W\|_1 \\ s.t. \;\; U_{i,r}, V_{j,r}, W_{k,r} \geq 0. \end{aligned} \tag{9.10}$$

Here θ is a parameter to be updated as defined in previous section (equation 9.3). The gradient based on equation 9.10 is:

$$\nabla_\theta \mathcal{L} = \nabla_\theta L + p(\theta), \quad p(\theta) \in C(\theta) \tag{9.11}$$

where θ is defined in Table 9.2 and $\mathcal{L}$ and L are defined in equation 9.2. Function $p()$ is the projection or constraint term of the gradient. The set $C(\theta)$ is the union of the subgradients at θ.

Definition 7 *Two blocks Z_i and $Z_{i'}$ in a given stratum are independent if for each $x \in Z_i$ and $x^{'} \in Z_{i'}$ we have*

$$\begin{aligned} \nabla L_x(\theta) = \nabla L_x(\theta - \eta \nabla L_{x'}(\theta)) \\ \text{and } \nabla L_{x'}(\theta) = \nabla L_{x'}(\theta - \eta \nabla L_x(\theta)) \end{aligned} \tag{9.12}$$

where $\nabla L_x(\theta)$ is the partial differential of L_x w.r.t. θ and θ is the parameter we are updating.

Theorem 3 *If blocks Z_i and $Z_{i'}$ in a given stratum S_n are non-overlapping then they are independent (as defined previously in Definition 7).*

Proof. For any two points $x \in Z_i$ and $x' \in Z_{i'}$ their rows or columns or any other coordinate do not overlap. From equation (9.4) we see that x does not modify θ in positions for which $i \neq x_i$, $j \neq x_j$ and $k \neq x_k$. Therefore, because x and x' are not equal in any dimension, an update from $\nabla L_{x'}$ will update different values than ∇L_x, where ∇L_x is the gradient at point x

Additionally from equation (9.4) we see that any updates on ∇L_x only use values from $U_{x_i,*}$, $A_{x_j,*}$ and $B_{x_k,*}$, and thus do not use any values that would be updated by $\nabla L_{x'}$. Because updates from x only effect parameters in x's coordinates, updates from x are only based on parameters in x's coordinates, and x and x' have no overlapping coordinates, we know that

$$\begin{aligned} \nabla L_x(\theta) &= \nabla L_x(\theta - \eta \nabla L_{x'}(\theta)) \\ \text{and } \nabla L_{x'}(\theta) &= \nabla L_{x'}(\theta - \eta \nabla L_x(\theta)) \end{aligned}$$

Therefore, Z_i and $Z_{i'}$ are independent. By a similar argument

$$\begin{aligned} \nabla \mathcal{L}_x(\theta) &= \mathcal{L}_x'(\theta - \eta \nabla \mathcal{L}_{x'}(\theta)) \\ \text{and } \nabla \mathcal{L}_{x'}(\theta) &= \nabla \mathcal{L}_{x'}(\theta - \eta \nabla \mathcal{L}_x(\theta)) \end{aligned}$$

$\square$

Based on this independence, and under certain conditions of continuity and step size similar to those in [77, 129], we can prove that Algorithm 13 converges. Particularly of note, we must prove that adding projections, caused by additional terms like ℓ_1 regularization, do not prevent convergence. As proven in detail in Appendix C, FlexiFaCT converges as long as the projections are Lipschitz-continuous. Following the arguments of [129] (Theorem 2.1, part 2) and with the assumption that updates to θ are bounded (based on the aforementioned conditions), the algorithm will converge.

9.4 MapReduce Implementation of FlexiFaCT

We implemented our algorithm within the MapReduce framework [61]. To do this we used the open source Hadoop [75] version of MapReduce. The challenge is to turn the factorization problem into `map()` and `reduce()` functions, of the kind that Hadoop is designed to handle.

In our implementation we pass the data matrix or tensor as input to the mappers in the form $(i, j, k, \boldsymbol{\mathcal{X}}_{i,j,k})$. We also store our current parameters $\theta^{(s)}$, which could include $\mathbf{U}$, $\mathbf{V}$, $\mathbf{W}$, and $\mathbf{A}$ on the Hadoop File System (HDFS).

FlexiFaCT Mapper Mappers take individual observations (cells) in the tensor in the form of $(i, j, k, \boldsymbol{\mathcal{X}}_{i,j,k})$ at random. The Mapper function, shown in Algorithm 14, emits the block index and subepoch for each observation.As such, the mappers split the entire tensor into the appropriate blocks and determines the order they should be processed in within each reducer. We also overload the default Hadoop partitioner, which typically just partitions on unique KEY values, and now partition only on b_i so that each reducer

Algorithm 14: FlexiFaCT Mapper (for tensor)

```
Input: $I, J, K, d$
1: for all $(i, j, k, \mathcal{X}_{i,j,k})$ do
2:     $b_i = \lfloor \frac{i}{\lceil \frac{I}{d} \rceil} \rfloor$, $b_j = \lfloor \frac{j}{\lceil \frac{J}{d} \rceil} \rfloor$, $b_k = \lfloor \frac{k}{\lceil \frac{K}{d} \rceil} \rfloor$ // Get block index
3:     $subepoch = d \times ((b_k - b_i + d) \bmod d) + ((b_j - b_i + d) \bmod d)$
4:     emit $\langle (b_i, b_j, b_k, subepoch), (i, j, k, \mathcal{X}_{i,j,k}) \rangle$
5: end for
```

Algorithm 15: FlexiFaCT Reducer (for tensor)

```
Given: $\mathbf{U}, \mathbf{V}, \mathbf{W}, I, J, K, d, \eta$
Input: Values $\mathcal{V}$
$s_{\text{old}}$ =Pointer to final parameters from last epoch
for $(i, j, k, \mathcal{X}_{i,j,k}) \in \mathcal{V}$ do
    $b_i = \lfloor \frac{i}{\lceil \frac{I}{d} \rceil} \rfloor$, $b_j = \lfloor \frac{j}{\lceil \frac{J}{d} \rceil} \rfloor$, $b_k = \lfloor \frac{k}{\lceil \frac{K}{d} \rceil} \rfloor$
    $t = d \times ((b_k - b_i + d) \bmod d) + ((b_j - b_i + d) \bmod d)$
    if $t \neq t_{\text{old}}$ then
        Write $\mathbf{V}^{(s_{\text{old}})}_{b_{j_{\text{old}}}}$ and $\mathbf{W}^{(s_{\text{old}})}_{b_{k_{\text{old}}}}$ to HDFS
        Wait for $\mathbf{V}^{(s_{\text{old}})}_{b_j}$ and $\mathbf{W}^{(s_{\text{old}})}_{b_k}$ to be available on HDFS
        Save $\mathbf{U}^{(s)}_{b_i} = \mathbf{U}^{(s_{\text{old}})}_{b_i}$ Save $\mathbf{V}^{(s)}_{b_j} = \mathbf{V}^{(s_{\text{old}})}_{b_j}$ and $\mathbf{W}^{(s)}_{b_k} = \mathbf{W}^{(s_{\text{old}})}_{b_k}$ from HDFS
        $b_{i_{\text{old}}} = b_i$, $b_{j_{\text{old}}} = b_j$, $b_{k_{\text{old}}} = b_k$, $t = t_{\text{old}}$
    end
    $\theta^{(s)} = \theta^{(s)} - \eta \nabla \mathcal{L}_{\mathcal{X}_{i,j,k}}(\theta^{(s)})$ (where $\theta^{(s)}$ is the concatenation of $\mathbf{U}^{(s)}_{b_i}, \mathbf{V}^{(s)}_{b_j}, \mathbf{W}^{(s)}_{b_k}$)
end
Write $\mathbf{U}^{(s_{\text{old}})}_{b_{i_{\text{old}}}}$, $\mathbf{V}^{(s_{\text{old}})}_{b_{j_{\text{old}}}}$, and $\mathbf{W}^{(s_{\text{old}})}_{b_{k_{\text{old}}}}$ to HDFS
```

represents a unique set of i in I or a unique set of rows in $\mathbf{U}$. We additionally override the default Comparator, allowing us to sort our (KEY,VALUE) pairs within each reducer by the *subepoch* term calculated in the Mapper. We see here while the Mapper is quite simple, the calculation of the blocks and the order within each reducer captures our partition function that allows us to perform SGD in this distributed fashion.

FlexiFaCT Reducer We create d reducers, one for each block in a straum. Our Reducer function is shown in Algorithm 15. As explained before, each reducer gets all points for a given range of values i ordered by the subepochs s. The reducer iterates over the points in $\mathcal{V}$ in order, each time updating $\theta^{(s)}$. Each reducer only stores the components of θ that correspond to its current block in the current stratum. As such, when a new subepoch is reached, it must write its updated θ values to disk (for another reducer to retrieve) and read the most current θ values for its next block in the subsequent subepoch.

We run the MapReduce jobs iteratively. Each MapReduce job is one epoch using all points in $\boldsymbol{\mathcal{X}}$ to update the full parameter space θ and ultimately to save it to HDFS. We then use the updated parameters θ in the subsequent epoch (another run of the MapReduce algorithm). We do this for a constant number of steps or until the algorithm converges.

Reproducibility and Usability This is a high level overview of our implementation, but captures the general method we use to both distribute our work and optimize our speed within the MapReduce framework. While this is not the typical way Hadoop is programmed, it requires no modification of the Hadoop framework and can be run on *any* standard Hadoop cluster. Our code is open-sourced, and available at cs.cmu.edu/~abeutel/flexifact. It can run for all of the data types and loss functions described in this chapter.

Because FlexiFaCT creates long-running reducers that communicate with each other, it breaks the typical mold for Hadoop MapReduce algorithms. More recently, new platforms designed for such paradigms are available and can be easily programmed for the FlexiFaCT algorithm. For example, [32] designs a similar algorithm on top of the REEF platform [52].

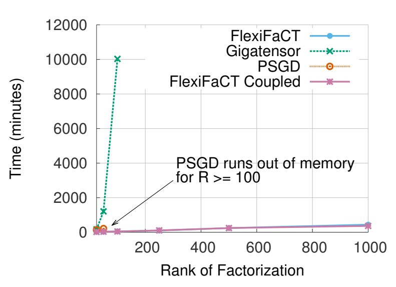
9.5 Experiments

9.5.1 Performance Evaluation

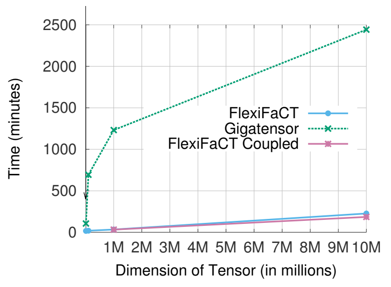
In order to assess how scalable and fast FlexiFaCT is, we conducted a series of experiments in order to measure the running time of FlexiFaCT with respect to 1) increasing number of data points, 2) increasing dimensions of the data and thus model, and 3) increasing rank of the factorization. The first aspect has to do with scalability in terms of data size, whereas the two latter aspects refer to scalability with respect to parameter space size; FlexiFaCT is able, as we demonstrate in the following experiments, to scale easily in all three aspects. As a baseline for tensor decomposition, we use GigaTensor [116]. We also compared against PSGD [243], however, the solutions obtained achieved much worse RMSE, and the algorithm was not able to scale for very large number of parameters (either rank or dimensions).

FlexiFaCT was implemented in Java, with Hadoop 0.20.1 [61, 75]. We ran the experiments on the OCC-Y cluster[1]. For the sake of experimentation, we created a series of synthetic datasets wherein we were able to control the three aspects we were testing: data size, data dimensions, and rank. Additionally, we validate our method's correctness by

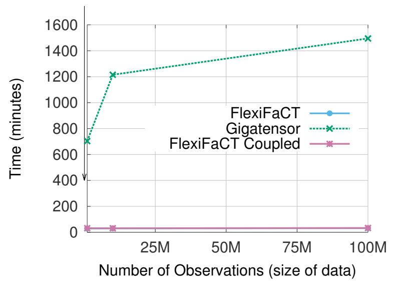
[1]http://opencloudconsortium.org/tag/occ-y/



(a) Time vs. rank



(b) Time vs. dimensions



(c) Time vs. observations

Figure 9.2: **Scalability of FlexiFaCT** in terms of: (a) rank, (b) data dimensions, and (c) number of observations. We observe that FlexiFaCT scales very well with respect to all aspects. PSGD can be seen in sub-figure (a) before it runs out of memory. FlexiFaCT was applied to both a tensor, and a matrix-tensor couple, whereas GigaTensor was only applied to a tensor.

presenting experimental evidence (on top of our theoretical guarantees), for monotone convergence. In all cases, number of reducers was constant and equal to 24.

Synthetic Data Generation To generate data we first generate randomly matrix factors $\mathbf{U}, \mathbf{V}, \mathbf{W}$ of the specified dimension D (where $I = J = K = D$) and rank $R = 30$. We then randomly select data points (i, j, k) and add their value $\sum_r \mathbf{U}_{i,r}\mathbf{V}_{j,r}\mathbf{W}_{k,r}$ to the dataset. We do this until we have the desired number of data points for each dataset. Unless otherwise stated, we set $D = 1$ million and the number of data points to 10 million.

9.5.2 Scalability

We now test FlexiFaCT on all three types of scalability to demonstrate that it scales in all dimensions and to unprecedented sizes.

Rank Scalability In testing the scalability with respect to rank, we ran FlexiFaCT, GigaTensor, and PSGD on a tensor and varied the rank from 25 up to 1000. Figure 9.2(a) shows the running time for FlexiFaCT, both for tensor and coupled factorizations, as the rank (i.e. one of the parameter dimensions) increases. We can see that FlexiFaCT scales linearly as the rank of the factorization increases, having similar timing behaviour both for plain tensor and coupled factorizations. GigaTensor, on the other hand, due to the fact that it is inverting an $R \times R$ matrix, locally, at every iteration slows down significantly for large ranks. We were unable to test GigaTensor on ranks larger than 100 due to the slow down. Although barely visible in the plot, we also ran PSGD on the data. However, it too couldn't scale in rank due to the size of the parameter space. For rank above 50 the factor matrices could no longer fit in memory on each machine and thus

PSGD could not run. This demonstrates FlexiFaCT's unique scalability in the parameter size. Additionally, we note that with $R = 1000$ and $D = 1$ million, the coupled FlexiFaCT factorization scales to a total parameter space of *4 billion parameters*.

Data Dimensions Scalability To test more directly the scalability as the dimension D of the data tensor grows, we created a variety of tensors with varying dimension from 10,000 to 10 million. We decompose each tensor with $R = 50$. When testing the coupled FlexiFaCT decomposition, we add an additional coupled matrix with 100,000 data points and the same dimensionality as the main tensor.

In Figure 9.2(b) we show how coupled factorization using FlexiFaCT scales, as the dimensions of the data increase. We observe that FlexiFaCT runs much faster than the baseline, GigaTensor. A likely explanation for the degree to which FlexiFaCT is faster than GigaTensor is that FlexiFaCT only focuses on the observed data points, where GigaTensor has to convert unobserved data points to zeros, thus slowing down the computation. We can see that FlexiFaCT has a very smooth behaviour, scaling linearly with the dimension size. Again, when we are performing the coupled decomposition, we see that as the dimension scales our total parameter space reaches 2 billion parameters.

Data Size Scalability Last, for data scalability, we vary the number of observed data points from 1 million to 10 million. Figure 9.2(c) shows FlexiFaCT's running time as a function of the data size, i.e. the number of observations. We can see that FlexiFaCT has, again, very smooth behaviour, and scales linearly with the number of observed elements. Again, we are significantly faster than GigaTensor, though the degree of difference is likely because it is must make unobserved points zeros for it to run.

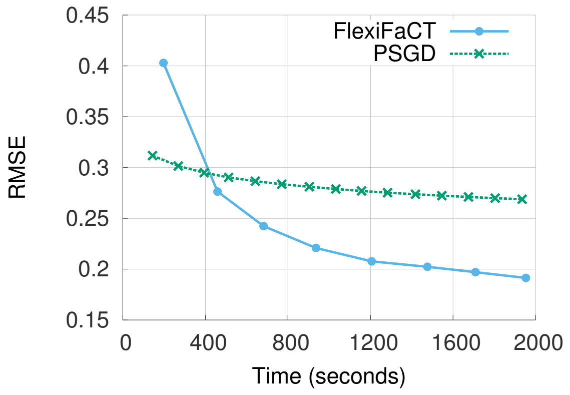


Figure 9.3: **Convergence:** RMSE vs. time, for tensor factorization comparing FlexiFaCT and PSGD [243]. Note, Zinkevich *et al.* [243] do not claim to work on this problem because it is not convex.

9.5.3 Correctness & Monotone Convergence

Besides speedup, we experimentally validate that FlexiFaCT indeed decreases monotonically the objective function that it is minimizing. To test this we run on a small synthetic data set with $D = 10,000$ and 10 million data points, making the dataset very dense. We run a factorization with $R = 50$ using our implementation of PSGD and FlexiFaCT with both ℓ_1 sparsity and non-negativity constraints. We then monitor the root mean squared error (RMSE).

Figure 9.3 shows that FlexiFaCT decreases the RMSE as expected, and at a much quicker pace than PSGD. It is important to note that the slow convergence of PSGD is because the problem (tensor factorization) is not convex, and thus Zinkevich *et al.* do not claim that their method works on such problems. However, we use PSGD as a comparison because it is not possible to track the RMSE with GigaTensor and thus PSGD is the closest competitor.

9.6 Summary

In this work we have introduced FlexiFaCT, a highly flexible, efficient, and scalable factorization tool. Our main contributions are

1. **Versatility:** FlexiFaCT, can operate under numerous factorization scenarios, including matrix, tensor, and coupled factorization as well as with non-negativity and sparsity constraints.
2. **Scalability:** FlexiFaCT scales very well with the input size, as well as with the number of parameters.
3. **Reproducibility and Usability:** Our implementation works on *stock* Hadoop; furthermore, we ensure the reproducibility and outreach of FlexiFaCT by making our implementation publicly available.

Chapter 10

Fast in the face of Stragglers

How can we learn perform distributed learning in the face of stragglers? In this chapter, we present a scheme for *fast*, distributed learning on a wide variety of big (i.e. high-dimensional) models applied to big datasets. Building on the partitioning scheme of Gemulla *et al.* [77] and projections of Chapter 9, we show how to significantly scale the learning of a wide variety of models, such as topic models and dictionary learning, particularly by including additional constraints and projections, such as distributed normalization. We demonstrate that this approach not only leads to faster learning on distributed clusters, but also enables machine learning applications where both data and model are too large to fit within the memory of a single machine. Most significantly, we offer a novel solution to the straggler problem, common in distributed systems, by allowing worker machines to perform *additional updates while waiting for slow workers* to finish, which provides users with a tunable synchronization strategy that can be set based on learning needs and cluster conditions. We present empirical results for latent space models such as topic models, which demonstrate that our method scales well with large data and model sizes, while beating learning strategies that fail to take both data and model partitioning into account.

10.1 Introduction

Machine Learning applications continue to grow rapidly, in terms of both input data size (big data) as well as model complexity (big models). The big data challenge is already well-known—with some estimates putting the amount of data generated on the internet at 5 exabytes every two days[1]—and much effort has been devoted toward learning models on big datasets, particularly through stochastic optimization techniques that randomly partition the data over different machines. Such techniques have been the subject of much theoretical scrutiny [7, 97, 242, 243]. On the other hand, the big model is-

[1]http://techcrunch.com/2010/08/04/schmidt-data/

sue, which is about learning models with an extremely large number of variables and/or parameters—such as the Google Brain deep network with over 1B parameters [60]—has just started to gain greater attention, and recent papers on this subject have primarily focused on intelligent partitioning of the model variables in order to minimize network synchronization costs [60, 149], albeit not always with theoretical backing.

Although big data and big models are both crucial research foci, there are few distributed machine learning efforts that explicitly consider both aspects in conjunction, with a notable example being the partitioned matrix factorization algorithm of Gemulla *et al.* [77] and later our partitioned tensor factorization algorithm in Chapter 9. More generally, in the big data, big model setting, the model variables may not all fit into a single machine's memory, which in turn imposes additional constraints on how the data is partitioned. Furthermore, careless partitioning of model variables across distributed machines imposes significant network synchronization costs [149], which are required whenever dependent variables or datapoints are placed on separate machines. If we are to effectively partition both data and model, it follows that we must carefully examine and exploit the interdependencies between data and model variables.

In this chapter, we investigate the practical algorithmic challenges for learning big latent space models on big data over a realistic distributed cluster. Building on our results in Chapter 9, we offer distributed projections so as to support a wider variety of models expressible in a matrix form, such as topic modeling and dictionary learning. Our proposed algorithm, *Fugue*, exploits the structures of the model in question to partition the input data and model variables over the cluster, in a manner that automatically balances inter-machine network synchronization costs with performing useful computational work, even when the worker machines are not equally capable (e.g., because of different hardware or other concurrently running programs), or the data cannot be evenly partitioned (e.g., due to the dependency structure of the model).

Most significantly, Fugue solves the "last reducer" or "straggler" issue in distributed systems, in which some worker machines can be much slower than others (because of cluster heterogeneity, or because other jobs are running on the same machine), causing faster workers to waste computational cycles waiting for them. Instead, our algorithm ensures that the faster workers will continue to perform useful work until the last reducer finishes by performing additional updates on cached data; if the last reducer is unable to finish, it can be restarted without affecting program correctness, while the faster workers continue doing work. Because Fugue has modest synchronization and communication requirements, it is easy to implement on top of common distributed computing systems such as Hadoop—yet it can also be applied to specialized systems for big Machine Learning, such as parameter servers [13, 54, 96, 182], in order to further improve their performance. Finally, theoretical and empirical analysis confirms that Fugue can provide faster convergence on a variety of latent space problems in ML; furthermore, careful control of inter-worker synchronization can lead to even faster convergence, which opens the door to intelligent exploitation of fine-grained synchronization schemes (such as the aforementioned parameter servers).

10.2 Related Work

Most existing literature is focused on learning under either big data or big model conditions, but rarely both together. As discussed in Section IV.1, many algorithms that are designed to overcome large datasets design schemes with little inter-machine synchronization. PSGD [243] is completely data parallel with no inter-machine communication, but, as we demonstrated in the previous chapter, we can obtain faster convergence with some inter-machine communication. Another important class of methods are the fixed-delay algorithms [7, 242] in which machines communicate with a central server (or each other) in a fixed ordering. This fixed ordering is a serious practical limitation, because all machines will be held up by slowdowns or failures in just one machine. In contrast, our algorithm ensures that all machines continue to do useful work even under such conditions. Most importantly, unlike these big data algorithms, our algorithm can partition model variables (and not just datapoints) across machines, which is absolutely critical for massive models that cannot fit onto a single machine.

There are some papers that tackle big data and big model issues together, such as the partitioned matrix factorization algorithm of Gemulla *et al.* [77], to which our work is most closely related. Unlike Gemulla *et al.*, our algorithm allows worker machines to perform useful variable updates continuously, without blocking or waiting for any other machine to complete its assigned task. This property is exceptionally beneficial on very large clusters, where machines often fail or slow down for any number of reasons. Thus, our algorithm is not bottlenecked by the slowest machine, unlike [77]. Furthermore, we include substantially richer theoretical analysis, including convergence and variance bounds with constraints and analysis of the effect of non-blocking workers. In particular, work by Murata [162] lays the foundation for variance analysis of SGD algorithms, by providing variance bounds over datapoint selection.

Additionally, the recent research on parameter servers [13, 96, 140] is fundamentally related to Fugue in that both allow faster workers to perform additional work, without being held up by slower workers. The main difference between SSP and Fugue is that SSP is employed to reduce inter-machine communication regardless of how updates are scheduled, whereas Fugue is used to intelligently schedule updates while assuming limited capacity for inter-machine communication. Combining these perspectives is an interesting direction for future research.

10.3 Fugue: Slow-Worker Agnostic Learning

We will now dive into the Fugue algorithm. We begin with an explanation of the different models Fugue covers, and then explain the algorithm for distributed learning of these models.

10.3.1 Latent Factor Models

The key principle behind Fugue is to exploit independent or weakly-dependent blocks of data, variables and parameters. For example, a probabilistic graphical model can contain many latent variables and parameters that capture the modeler's generative assumptions about large datasets. In order to tackle problems of such scale, we need to exploit independence structures present in the data and model, so as to partition both over a distributed cluster.

Before describing our partitioning strategy, we motivate our method with examples of popular latent space models recently gaining much attention. Consider **Topic Modeling (TM)** [35]: given a *document by vocabulary* data matrix Y (with the rows normalized to sum to 1), we want to decompose it into two matrices: *docs by topics* π (which are model variables) and *topics by vocabulary* β (which are parameters). We formulate this task as an optimization problem with simplex and non-negativity constraints:

$$\begin{aligned} &\operatorname*{argmin}_{\pi,\beta} L(Y,\pi,\beta) = ||Y - \pi\beta||_p^p \\ &\text{s.t. } \forall i,j,k \quad \sum_k \pi_{i,k} = 1, \sum_j \beta_{k,j} = 1, \quad \pi_{i,k} \geq 0, \beta_{k,j} \geq 0, \end{aligned} \tag{10.1}$$

where $\|\cdot\|_P^P$ is an ℓ_p norm, typically ℓ_2. Fugue exploits structure in the form of *blocks* of document-word pairs $Y_{\mathcal{I},\mathcal{J}}$, in order to partition the topic model. We note that other matrix-decomposition-based algorithms for topic modeling also exist, such as the spectral decomposition method of Anandkumar *et al.* [19].

Another example is **Dictionary Learning (DL)** [125], in which the goal is to decompose a signal matrix Y into a dictionary D and a sparse reconstruction α:

$$\begin{aligned} &\operatorname*{argmin}_{\alpha,D} L(Y,\alpha,D) = \frac{1}{2}||Y - D\alpha||_2^2 + \lambda||\alpha||_1 \\ &\text{s.t. } \forall j, D_j^T D_j \leq 1. \end{aligned} \tag{10.2}$$

Here, blocks of sample-feature pairs $Y_{\mathcal{I},\mathcal{J}}$ form the primary exploitable structure.

A third example is multi-role or **Mixed-Membership Network Decomposition (MMND)**, where an $N \times N$ adjacency matrix Y is decomposed into an $N \times K$ matrix θ, whose i-th row is the normalized role-vector for node i, and a $K \times K$ role matrix B. Together, θ, B

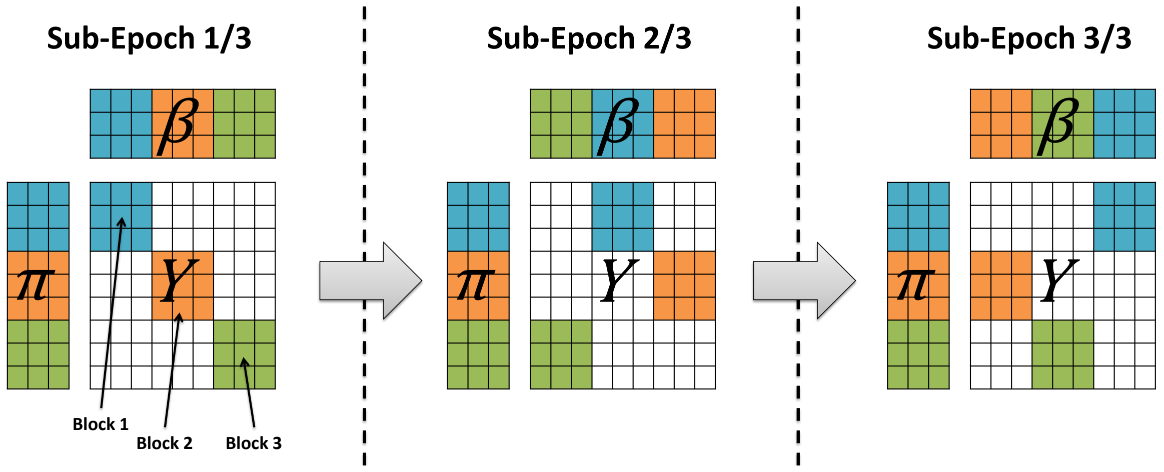


Figure 10.1: **Partitioning strategy** for data Y, model variables π, and parameters β. We show one epoch broken into multiple sub-epochs (3 in this case). Each sub-epoch is further divided into (colored) blocks, such that the data $Y_{i,j}$ (with its associated variables $\pi_{i,\cdot}$ and parameters $\beta_{\cdot,j}$) from one block do not share rows/columns with data $Y_{a,b}$ from another block. Taken together, all blocks from all sub-epochs cover every element of Y, π, β.

characterize the behavior of every node in the network, and the optimization problem is:

$$\begin{aligned} &\underset{\theta,B}{\operatorname{argmin}}\, L(Y,\theta,B) = \frac{1}{2}||Y - \theta B \theta^{\top}||_2^2 \\ &\text{s.t. } \forall i, \quad \sum_j \theta_{ij} = 1, \quad \theta_{i,j} \geq 0. \end{aligned} \tag{10.3}$$

The subgraphs $Y_{\mathcal{I},\mathcal{J}}$ between node sets $\mathcal{I}$ and $\mathcal{J}$ are the basic unit of partitioning used by Fugue.

The main difference between such latent space models and the matrix factorization problems (unconstrained or non-negative) of [77] and the tensor factorization methods of Chapter 9 is that these latent space models have more complex constraints: simplex constraints in the case of topic modeling and mixed-membership network decomposition, and bounded norm in the case of dictionary learning. To handle these, our distributed learning algorithm supports *distributed projection* steps to ensure the final solution always satisfies the constraints. Some distributed algorithms have explicit theoretical guarantees under projection [7, 242], but involve complex synchronization schemes that are inefficient on large clusters and difficult to deploy on common systems like Hadoop. Others, such as Parallel Stochastic Gradient Descent (PSGD) [243], lacks explicit projection theory for parallel learning but works well in practice; furthermore, they have simple synchronization requirements, making them suitable for large cluster deployments.

10.3.2 Fugue Approach

We now explain at a high level how Fugue works. We stress that Fugue is platform-agnostic: it can be implemented on top of Hadoop, MPI, or even parameter servers and distributed key-value stores—essentially, any platform with programmer control[2] over how data, variables and parameters are partitioned and scheduled for updates. Our experiments demonstrate that Fugue is highly efficient even when built on Hadoop, which, as was explained in the previous chapter, was not designed for algorithms of this sort. We will discuss later in this chapter how Fugue could likely scale even better on specialized big ML platforms.

To learn latent space models effectively on a distributed cluster, we must exploit the interdependence of parameters and variables. As a running example, consider the topic modeling objective $L(Y, \pi, \beta)$: we can divide the data matrix Y into a sequence of sub-epochs, where each epoch consists of blocks that do not overlap on parameters β and variables π, and where the union over all epochs covers the entire matrix (Figure 10.1). This blocking strategy is attributed to Gemulla *et al.* [77], and it permits multiple machines to perform stochastic gradient descent (SGD) on different blocks in parallel, on a Hadoop cluster. However, it requires all workers to process a roughly equal number of data-points per block, which leads to problems with slow workers. In this chapter, we intend to address this limitation. Our proposed algorithm allows faster workers to process extra data-points in their assigned block, which maximizes the cluster's computational efficiency.

At a high level, our algorithm proceeds one epoch at a time, performing SGD on all blocks within an epoch in parallel. In order to satisfy the problem constraints, we must interleave projection steps with the SGD algorithm. In this respect, the parameters β and variables π must be handled differently: while the simplex projection for variables π can be performed by each worker independently of others, the simplex projection for the parameters β requires workers to repeatedly synchronize with each other. This cannot be done through the standard MapReduce programming model, so our Hadoop implementation builds on the synchronization strategy in Section 9.4 and deviates from the MapReduce scheme by allowing workers (reducers) to write to the Hadoop Distributed File System (HDFS) in order to communicate projection information with each other. This is of course more expensive than the independent projections for π, but even these more expensive projections are computationally cheap relative to the typical sub-epoch synchronization. We find that this scheme works well in practice, while dedicated, memory-based synchronization systems such as parameter servers [13, 54, 182] have the potential to perform even better.

[2]Systems that do not provide programmer control over partitioning/scheduling, e.g., GraphLab, are unsuitable for Fugue.

10.3.3 Partitioning strategy

More formally, let $\mathbf{\Psi}$ collectively refer to the variables and parameters π, β, and let ψ refer to individual elements of $\mathbf{\Psi}$. These definitions will make the subsequent analysis easier to understand. Thus, we rewrite the topic modeling objective L as:

$$\psi^{(t+1)} = \psi^{(t)} - \eta_t \nabla \mathcal{L}_{Y_{i,j}}(\psi^{(t)}), \tag{10.4}$$

and we apply parameter/variable projections each time we execute Eq. (10.4). Assuming the ℓ_2 norm, the differential of ψ with respect to π at a single data point $Y_{i,j}$ is

$$(\nabla L_{Y_{i,j}}(\psi))_\sigma = \begin{cases} -2(Y_{i,j} - \sum_k \pi_{i,k}\beta_{k,j})\beta_{\ell,j} & \text{if } \sigma = \pi_{i,\ell} \\ 0 & \text{if } \sigma = \pi_{i',\ell},\ i \neq i' \end{cases} \tag{10.5}$$

where σ is the element of π being differentiated, and $(\nabla L_{Y_{i,j}}(\psi))_\sigma = \frac{\partial L_{Y_{i,j}}}{\partial \sigma}$. The differentials with respect to $\sigma = \beta_{j,\ell}$ are similar. From these equations, we observe that the SGD update for $\pi_{i,\ell}$ at a particular datapoint $Y_{i,j}$ depends only on a small subset of the variables and parameters: specifically, $\pi_{i,k}, \beta_{k,j}$ where $k \in 1, \ldots, K$ and K is the number of topics we chose. Notice that the π all come from the same row i as $Y_{i,j}$, while the β all come from the same column j. Furthermore, the SGD updates for $\pi_{i,\ell}$ are zero for any datapoint $Y_{a,b}$ where $a \neq i$. A similar observation holds for the parameters: the SGD update for $\beta_{r,j}$ is zero for any datapoint $Y_{a,b}$ where $b \neq j$.

These observations lead to the following key insight: we can perform SGD on two datapoints $Y_{i,j}$ and $Y_{a,b}$ at the same time, provided $i \neq a$ and $j \neq b$—in other words, as long as the datapoints do not overlap on their rows or columns. An intuitive proof goes like this: SGD updates on $Y_{i,j}$ only touch the variable row $\pi_{i,\cdot}$ and the parameter column $\beta_{\cdot,j}$, and both of them do not overlap with $Y_{a,b}$'s variable row $\pi_{a,\cdot}$ and parameter column $\beta_{\cdot,b}$. In other words, the SGD updates on $Y_{i,j}$ and $Y_{a,b}$ touch disjoint sets of variables π and parameters β. Furthermore, each $\pi_{i,\cdot}$ in row i only ever depends on other π in the same row i, and similarly for $\beta_{\cdot,j}$ and other β in column j. We therefore conclude that all quantities associated with row i and column j, namely $Y_{ij}, \pi_{i\cdot}, \beta_{\cdot j}$, are completely independent of the quantities from row a and column b, namely $Y_{ab}, \pi_{a\cdot}, \beta_{\cdot b}$. Hence, datapoints with disjoint rows and columns can be simultaneously used for SGD updates [77].

From the perspective of data, model and parameter partitioning, we have essentially partitioned datapoints Y, model variables π and parameters β into independent "collections" $\mathcal{C}_{ij}$, where i is a row index and j is a column index. In other words, the collection $\mathcal{C}_{ij}$ contains $Y_{ij}, \pi_{i\cdot}, \beta_{\cdot j}$, and can be "processed" (meaning that we run SGD on Y_{ij} to update $\pi_{i\cdot}$ and $\beta_{\cdot j}$) in parallel with any other collection $\mathcal{C}_{ab}$ where $a \neq i$, $b \neq j$.

We note that the above scheme applies to Dictionary Learning with only slight modification. For Mixed-Membership Network Decomposition, the presence of the symmetric term $\theta B \theta^T$ presents additional challenges. Instead, we replace $\theta B \theta^\top$ with θC where $C := B\theta^\top$, and could recover B post-optimization via pseudoinversion: $B = C\theta(\theta^\top\theta)^{-1}$. The inversion cost is reasonable since $\theta^\top\theta$ is $K \times K$, where $K = 1000$ would be considered a large model in practice.

10.3.4 Distributed update scheduling

Since collections $\mathcal{C}_{ij}$ with disjoint rows/columns can be processed simultaneously, let us consider grouping them into multiple blocks $S_b \subseteq Y$, such that the blocks have disjoint rows/columns (Figure 10.1). While we cannot process collections $\mathcal{C}_{ij}$ within the same block in parallel (because they might share rows/columns), we can process collections from *different* blocks in parallel, as they are guaranteed to be non-overlapping. Thus, if we managed to construct P non-overlapping blocks, we can spawn P workers to perform SGD in parallel. Although it is impossible to find a set of non-overlapping blocks that covers all of Y, we can find multiple disjoint sets of non-overlapping blocks that, taken together, cover Y completely (Figure 10.1). We call these sets *sub-epochs*, which are processed sequentially (while the blocks within a sub-epoch are processed in parallel). An *epoch* is a sequence of sub-epochs that fully covers Y.

Fugue differs most significantly from Gemulla *et al.* and FlexiFaCT (Ch. 9) in that within a sub-epoch, we allow different worker-blocks to perform varying numbers of SGD updates per collection $\mathcal{C}_{ij}$ (whereas previously equal numbers of updates were required per worker). This makes our algorithm much more efficient whenever there are slow worker machines (which are common in large clusters), since faster workers can keep running until synchronization, rather than wasting computational time waiting for slower workers to catch up. The full algorithm is shown in Algorithm 16.

10.4 Proof Sketch

We now give a high-level analysis of Fugue (Algorithm 16), and give a proof sketch showing that our strategy of allowing multiple SGD iterations on each data/variable/parameter collection $\mathcal{C}_{ij} = \{Y_{ij}, \pi_{i\cdot}, \beta_{\cdot j}\}$ leads to faster convergence (under reasonable conditions). The complete analysis and proof can be found in Appendix D. We analyze the variance of the model state Ψ under Fugue, and show that it remains bounded under certain assumptions. Briefly, the variance can be attributed to two aspects of our partitioning strategy: (1) running multiple blocks within a sub-epoch in parallel, and (2) splitting the data matrix into a sequence of sub-epochs. These variance bounds distinguish the analysis from that of Gemulla *et al.* [77] and the proof in Appendix C, where we do not have variance bounds for the given blocking and learning strategies. We show that allowing additional iterations on fast workers is better than waiting for slow workers,

Algorithm 16: Fugue, our slow-worker agnostic learning algorithm, applied to topic modeling.

```
Input : $Y, \beta, \pi$, sub-epoch size $d$
$\pi \leftarrow \pi_0$, $\beta \leftarrow \beta_0$
Block $Y, \pi, \beta$ into corresponding $w$ blocks
while not converged do
    Pick step size $\eta_S$
    Pick $w$ blocks($S_1, ..., S_w$) to form sub-epoch $S$
    for $b = 0, \ldots, w-1$ in parallel do
        Run SGD on the training points $Y_{ij} \in S_b$
        // (until every block is ready to synchronize)
        // Workers can use datapoints in $S_b$ multiple times while waiting
        Apply appropriate projections
        // (e.g., on variables $\pi$ in topic modeling)
    end
    Apply appropriate projections
    // (e.g., on parameters $\beta$ in topic modeling)
end
```

by proving that the model state Ψ's variance converges to zero under an appropriate step-size sequence.

Assume we have w worker processors, and that in each sub-epoch, every processor is assigned to a distinct block i—henceforth, we shall use index i to refer interchangeably to processors or blocks. We now define the following terms:

Definition 8 : Key states and parameters

- n_i, κ_i and N_w: Let n_i be the number of datapoints that worker i touches (with repetition) in its assigned block, before transitioning to the next sub-epoch. In other words, if worker i was assigned n datapoints, and touches each point $\kappa_i \geq 1$ times on average, then

$$n_i = \kappa_i n \qquad \text{and} \qquad N_w = \sum_{i=1}^{w} n_i \tag{10.6}$$

- η_t: SGD step size at iteration t. An iteration is defined as one SGD update on one datapoint.
- $\nabla\mathcal{L}(\psi^{(t)})$: Exact gradient at iteration t.
- $\delta L^{(t)}(V^{(t)}, \psi^{(t)})$: Stochastic gradient at iteration t, i.e. $\nabla\mathcal{L}_{Y_{i,j}}(\psi^{(t)})$ for some i, j.
- ε_t: Error due to stochastic update at iteration t, $\left[\nabla\mathcal{L}(\psi^{(t)}) - \delta L^{(t)}(V^{(t)}, \psi^{(t)})\right]$.
- $\psi^{(t)}$: Model state ψ (see Eq. 10.4) at iteration t.

We now introduce $V^{(t)}(\psi^{(t+1)}, \psi^{(t)})$, a state potential function defined over a previous state $\psi^{(t)}$, a future state $\psi^{(t+1)}$, and the data points $y^{(t)}$ picked at iteration t. $V^{(t)}$ encodes the probability that $\psi^{(t)}$ will be updated to $\psi^{(t+1)}$ when the algorithm performs the stochastic update over datapoint $y^{(t)}$.

Next, assumptions on the error terms ε_t and step sizes η_i:

Assumption 1 : Errors and Step-sizes

- **Martingale difference error** ε_t: The error terms ε_t form a martingale difference sequence.
- **Variance bound on** ε_t: For all t, we have $E[\varepsilon_t^2] < D$.
- **Step size** η_t **assumption**: $\sum \eta_t^2 < \infty$.

The condition that error terms are a martingale difference sequence is weaker (easier to satisfy) than assuming error terms ε_t are independent of each other. The martingale difference assumption means that the stochastic gradient $\delta L^{(t)}(V^{(t)}, \psi^{(t)})$, when conditioned on the initial model state $\psi^{(0)}$ and previous gradients $\delta L^{(i)}(V^{(i)}, \psi^{(i)})$ for all $i < t$, depends only on the current model state $\psi^{(t)}$. Fugue satisfies this martingale assumption because our blocking strategy ensures that parallel parameter updates (from different blocks) never overlap on the same elements of ψ. For models with more complex dependency structures (e.g., arbitrary graphical models), the martingale assumption may allow for parallelization opportunities even when such non-overlapping structure is absent.

Convergence of Fugue Because the errors ε_t are a martingale difference sequence, we know that ε_t has zero expectation when conditioned on previous errors. With this we can provide a convergence guarantee:

Theorem 4 *The stochastic updates $\psi^{(t+1)} = \psi^{(t)} - \eta_t \nabla \mathcal{L}_{Y_{i,j}}(\psi^{(t)})$ from algorithm 16 and the exact gradient descent updates $\psi^{(t+1)} = \psi^{(t)} - \eta_t \nabla \mathcal{L}(\psi^{(t)})$ converge to the same set of limit points asymptotically, given that the error terms ε_t are a martingale difference sequence, and $E[\varepsilon_i^2] < D$ (bounded variance), and $\sum \eta_i^2 < \infty$.*

This theorem says that, asymptotically, the error terms cancel each other out, and therefore our stochastic algorithm will find the same set of optima as an exact gradient algorithm. The theorem also easily extends to cover projections, by applying the Arzela-Ascoli theorem and considering the limits of converging sub-sequences of our algorithm's SGD updates [129]. The proof for Theorem 4 can be found in Appendix D.

Variance decrease despite slow-workers We find that the variance of ψ decreases when it is updated inside block i in a sub-epoch, and therefore, due to the independence of blocks, the variance between sub-epochs decreases. We find that this depends on the number of datapoints in each block n_i and the step size η_t, where having more datapoints

n_i decreases the variance as long as the step size is sufficiently small (see Appendix D for details).

Based on these properties, we find that that using additional datapoints in any block can decrease the variance of Ψ, while Theorem 4 tells us that the algorithm converges asymptotically, regardless of the number of processors and the number of datapoints assigned to each processor. Because the algorithm converges, decreasing the variance will only move Ψ towards an optimum, and therefore it makes sense for faster processors to perform more updates (with appropriate choice of step size η_{S_n}) and decrease the variance of Ψ, rather than wait for slow processors to finish. However, if the step size η_{S_n} is set incorrectly, then using too many updates n_i could increase the variance of Ψ. In total, reusing the same datapoints will improve convergence but with diminishing returns on each reuse; however, we experimentally find that these additional updates significantly improve the speed of Fugue.

It is important to note that Fugue is *not* equivalent to fully asynchronous computation (e.g., Hogwild [167]), in which every worker can proceed to arbitrary data points, variables and parameters, without regard to what other workers are doing. Fugue requires that all workers advance to the next sub-epoch at the same time, in order to preserve independence between parallel blocks (and thus the martingale condition). Bounded-synchronization schemes like Stale Synchronous Parallel [54, 96] may allow Fugue to advance workers to different sub-epochs at different times, thus increasing the effectiveness of fast workers without losing convergence guarantees. Through careful scheduling of blocks to workers, we could combine and trade-off the diminishing returns of reused datapoints with the staleness of parameters. This is an interesting and promising direction for future research.

10.5 Experiments

We now test Fugue in a wide variety of settings to demonstrate its success and to understand its improvements.

10.5.1 Experimental Setup

We compare Fugue implemented on Hadoop to three self-implemented baselines: (a) BarrieredFugue on Hadoop (the same system as Fugue, but *without* the additional updates for fast workers when waiting at a barrier to synchronize), (b) Parallel SGD [243] on Hadoop, and (c) constrained Matrix Factorization on distributed GraphLab [149], a modification of the default GraphLab matrix factorization toolkit that regularly projects variables to maintain constraints (without altering the input graph). We test all methods on our 3 latent space models: topic modeling, dictionary learning, and mixed-membership network decomposition. All methods were tuned to their optimum parameters.

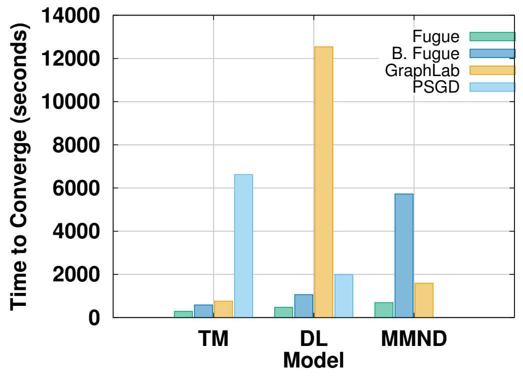
Compared to the baselines, our method has several theoretical and practical advantages: unlike BarrieredFugue, our algorithm allows fast workers to continue doing work while waiting for slow workers to synchronize, and unlike PSGD, we explicitly partition the data/variables/parameters into collections $\mathcal{C}_{ij} = \{Y_{ij}, \pi_{i\cdot}, \beta_{\cdot j}\}$ instead of averaging updates over all data points. Finally, we note that GraphLab is poorly-suited for implementing the simplex and inner-product constraints required by our topic modeling, dictionary learning and mixed-membership network decomposition models. This is because the constraints are over entire matrix rows, creating dependencies over all row variables—which is especially problematic for topic modeling, because the vocabulary matrix β has V columns, and V can be $\geq 100K$ words in practice. As we shall show, such long-range dependencies hurt GraphLab's performance, because GraphLab picks sets of variables for updating without regard to the constraints—whereas a better strategy is to schedule as many dependent elements together as possible.

Cluster Hardware The Hadoop algorithms (ours, BarrieredFugue, PSGD) were run on a Hadoop cluster with the following machine specifications: 2x Intel Xeon E5440 @ 2.83GHz (8 cores per machine), 16GB RAM, 10Gbit Ethernet. Because the Hadoop cluster did not support MPI programs, we ran GraphLab on a newer cluster with the following machine specifications: 2x Intel Xeon E5-2450 @ 2.1-2.9GHz (16 cores per machine), 128GB RAM, 10Gbit Ethernet. Thus, the GraphLab experiments have significantly more memory, but slightly slower processors.

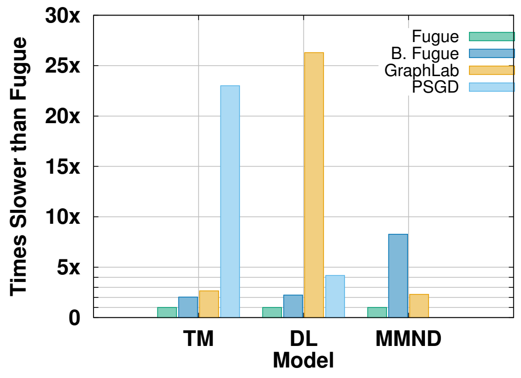
Dataset	**Dimensions**	**Nonzeros**	**Size (GB)**
ImageNet	$0.63 * 10^6 \times$1,000	$0.63 * 10^9$	7.99
WebGraph	$0.28 * 10^6 \times 0.28 * 10^6$	$0.31 * 10^9$	4.46
NyTimes	$0.3 * 10^6 \times$102,660	$0.1 * 10^9$	1.49
NyTimes4	$1.2 * 10^6 \times$102,660	$0.4 * 10^9$	6.08
NyTimes16	$4.8 * 10^6 \times$102,660	$1.6 * 10^9$	25.12
NyTimes64	$19.2 * 10^6 \times$102,660	$6.4 * 10^9$	103.4
NyTimes256	$76.8 * 10^6 \times$102,660	$25.6 * 10^9$	421.42

Table 10.1: Dimension, filesize and nonzero statistics for our datasets. The biggest dataset (NyTimes256) is approximately 0.4 terabytes. Note that the ImageNet dataset is 100% dense.

Datasets Statistics for all datasets are summarized in Table 10.1. For topic modeling, we simulated datasets of various sizes, using the 300K-document NY Times dataset as a



(a) Absolute time



(b) Time relative to Fugue

Figure 10.2: **Time taken by all methods to converge** on the three ML models, on an absolute scale (left) as well as a relative scale (right). The methods plateau at these values in the respective plots shown in Figure 10.3. The bar for PSGD is absent in the figure as it never reaches 0.059 and stops around objective value 0.092.

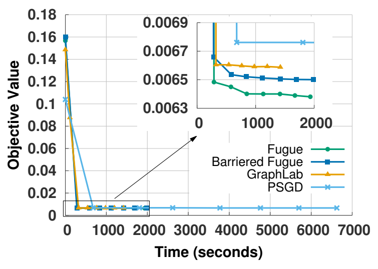
building block. The resulting β matrices contain 102,660 columns (words), and between 1.2M to 76.8M rows (documents); zero word counts are treated as missing entries. For dictionary learning, we used a 630,716-image (rows) sample from ImageNet [62], with 1000 features per image (columns). The resulting matrix is fully dense (no missing entries). For mixed-membership network decomposition, we used the Stanford web graph [139], with 281,903 vertices (rows and columns). The resulting adjacency matrix contains all edges, as well as 0.5% of the non-edges (all other non-edges are treated as missing entries).

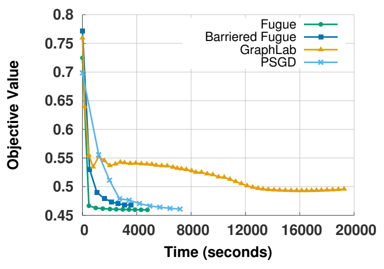
Stopping Criteria and Parameter Tuning We stop each method when its objective value reaches 0.0065 (TM), 0.059 (MMND), or 0.49 (DL). These are the values at which all methods were observed to plateau (Figure 10.3).

10.5.2 Empirical Results

We now compare Fugue to previous methods in terms of convergence, scalability, resource utilization, and overall speed.

Convergence We first compare the convergence of Fugue to the other three methods. As we observe in Figure 10.3, Fugue converges faster and to a better solution than all three baselines, on all three ML models. We note that GraphLab sometimes takes more than twice as long as Fugue to converge (noting that the GraphLab machines had a slightly slower average clock-speed). Moreover, GraphLab sometimes oscillates


(a) Topic Modeling


(b) Dictionary Learning

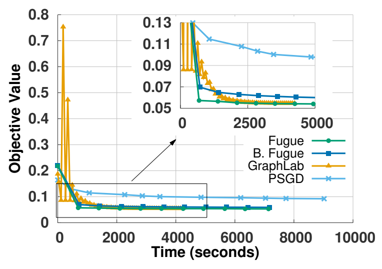

(c) MMND

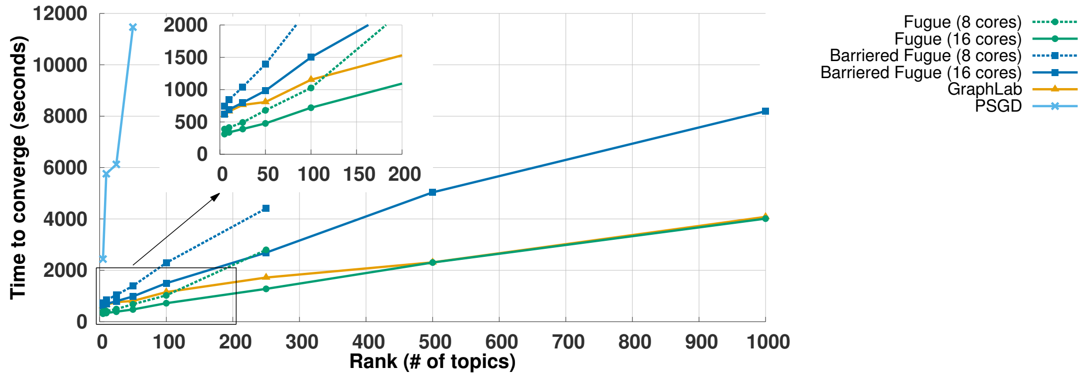
Figure 10.3: **Convergence** plots for the three models (TM, DL, MMND), under our Fugue and baselines (BarrieredFugue, PSGD, GraphLab). Unless otherwise stated in the plot, all methods were run with 16 cores and rank $K = 25$. We observe objective trajectory and final value of each method.

because its local vertex synchronization prevents it from applying the global projections frequently enough.

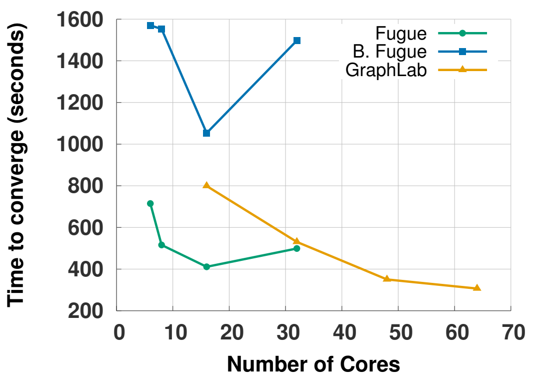
Scalability We next compare the ability of Fugue to scale topic modeling relative to GraphLab and PSGD. As can be seen in Figure 10.4, Fugue converges faster (on topic modeling) for any number of topics and documents, and generally requires fewer processors to converge quickly.

Resource Utilization In Figure 10.5, we observe the minimum number of machines Fugue and GraphLab require for topic modeling on fixed a number of documents. The primary reason for needing more machines is memory, and we see that GraphLab requires additional machines at a faster rate (despite having 128GB per 16-core machine), whereas Fugue scales much more gently (even with only 16GB per 8-core machine). This is because Fugue uses Hadoop and HDFS, so it never loads the entire model into memory (unlike GraphLab). In fact, GraphLab runs out of memory so quickly, that it cannot scale past 20 million documents on a 128GB machine (see Figure 10.4(c)).

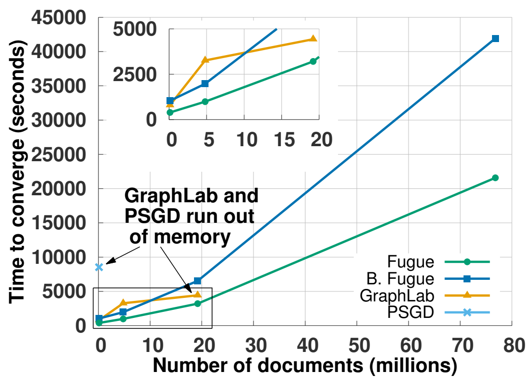
Overall Speed Comparison Last, we compare the total wall-clock time for all of the methods to converge on all of the problems. As can be seen in Figure 10.2, Fugue consistently outperforms competing methods over all three models: topic modeling, mixed-membership network decomposition, and dictionary learning. Fugue is faster by anywhere between $2.6\times$ (versus GraphLab on TM) to $26.2\times$ (versus GraphLab on Dictionary Learning).



(a) Scaling in number of topics



(b) Scaling in number of cores



(c) Scaling in number of documents

Figure 10.4: **Scalability** plots for TM under our Fugue and baselines (BarrieredFugue, PSGD, GraphLab). Unless otherwise stated in the plot, all methods were run with 16 cores, rank $K = 25$, and with the NyTimes4 dataset (Table 10.1). We observe how well each method fares as we increase the (a) problem rank, (b) number of processor cores, and (c) data size.

10.5.3 Why Fugue Succeeds

Compared to BarrieredFugue and PSGD, Fugue succeeds because it (1) it partitions both data and model variables to account for dependencies (which PSGD lacks), while (2) it allows faster workers to do more work before synchronization (which BarrieredFugue lacks). In particular, PSGD must hold the whole model on each machine due to non-partitioning (a memory bottleneck for rank $K \geq 100$). Moreover, PSGD needs one machine to average all models (a computational bottleneck), hence it does not scale well with additional cores (and does not even finish on MMND).

Table 10.2 provides insights about slow and fast workers in Fugue. For example, dense input matrices (e.g., our dictionary learning dataset) result in balanced workloads, thus every worker is equally balanced. On the other hand, the topic-word matrix constraints

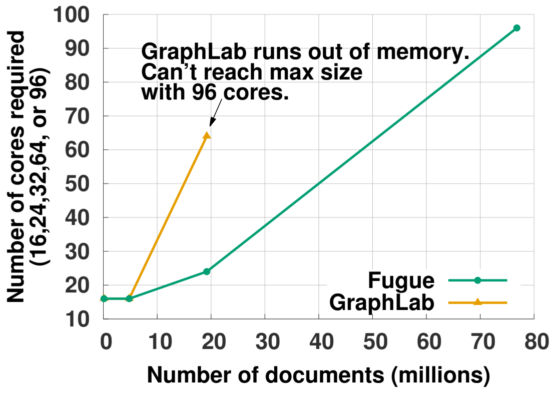


Figure 10.5: **Comparing the minimum number of machines needed** for a given topic modeling dataset for Fugue and GraphLab.

Fugue Model	Wait/Sync Time	Epoch Time
Dictionary Learning	32.6	471
Topic Modeling	152.7	391
MM Network Decomp.	182	692

Table 10.2: **Comparison of synchronization time vs epoch time**, for each ML model under Fugue. Dictionary learning has the smallest wait:epoch ratio, because the input matrix is fully dense (hence every worker has the same workload). In contrast, Topic Modeling has the highest wait:epoch ratio, because the normalization constraints on the topic-word matrix impose additional synchronization costs. More waiting means faster workers perform more updates, so TM converges in fewer iterations than DL.

in topic modeling increase inter-epoch synchronization times, providing an opportunity for workers to conduct extra updates.

Although Fugue exhibits good performance, there remain areas for improvement—for example, Fugue exhibits diminishing returns going from 16 to 32 cores on topic modeling. This is due to increased synchronization costs, which dominate the computational benefits from using additional cores. We expect that moving from Hadoop to a parameter server system [96] will alleviate the synchronization bottleneck, thus allowing Fugue to harness more machines. Furthermore, while Fugue's row/column-wise partitioning strategy for data/variables/parameters works well for the latent space models we have presented, it does not apply to all possible ML models: for example, graphical models and deep networks can have arbitrary structure between parameters and variables, while problems on time-series data will have sequential or autoregressive dependencies between datapoints. In such cases, a row/column-wise partitioning will not work. Nevertheless, the *idea* and basic theoretical analysis of grouping data, variables and parameters into independent collections still holds; only the partitioning strategy needs to be changed. This opens up rich possibilities for future work on general-purpose partitioning algorithms under our framework.

10.6 Generality and Applicability to Bayesian models

We have thus far demonstrated how to parallelize and distribute learning latent factor models with stochastic gradient descent, and have demonstrated the usefulness of these algorithms across a wide variety of model structures, from coupled tensor factorization to dictionary learning. These same techniques for distributed learning can be applied to using Bayesian learning algorithms to learn Bayesian collaborative filtering models. As was discussed in Part III, there are wide variety of successful Bayesian collaborative filtering models, e.g., Matchbox [202], TrueSkill [95], Bayesian Probabilistic Matrix Factorization [188], and our CoBaFi model (Ch. 6). These models obey a very similar dependency structure as the other models discussed in Part IV, but use Bayesian learning algorithms to infer the model parameters.

We demonstrate the generality and applicability of these techniques to distributed learning of Bayesian collaborative filtering models in [32]. There, we use the expectation propagation (EP) algorithm [158], where each observed rating sends a message (in the form of a Gaussian distribution) to each parameter involved in generating that rating. These messages form a fixed-point system that needs to be iterated until convergence. The fixed-point updates in EP exhibit the same natural parallelism exploited in the past two chapters. As such, we can perform a similar partitioning of our data and model, allowing us to run updates on each block within a stratum simultaneously without interference. In the case of parts of the model that are shared across blocks, such as conjugate priors, we can exploit the SSP ideas from parameter servers [96, 141]. [32] provides additional contributions such as distributing a probabilistic programming language and techniques for improving resource utilization when using Bayesian inference algorithms as compared to SGD, as described here.

10.7 Summary

In conclusion, we have developed Fugue, a slow-worker agnostic algorithm for distributed learning of latent space models such as topic modeling, dictionary learning and mixed-membership network decomposition. Our method takes advantage of both data/model partitioning and extra computational time on fast workers to achieve faster empirical convergence. In terms of empirical performance, our method combines the best ideas from DSGD and FlexiFaCT (data and model variable partitioning) and PSGD (additional work before synchronization) to yield performance better than either baseline, and is able to tackle highly constrained problems, which GraphLab falters on. Although our method is implemented in Hadoop, it could likely be improved by being implemented on newly-emerging Big ML systems, such as parameter servers [96, 141], to reach higher performance and data scales.

Part V

Concluding Remarks

Chapter 11

Conclusion

In this dissertation, we have taken a graph-based perspective to understanding, modeling, and predicting user behavior. Taken together, the work has pushed forward the frontier of user behavior modeling in multiple directions, with both academic contributions and broader impact. We give an overview of these points below.

11.1 Contributions

In this dissertation we have offered a variety of novel algorithms, models, and systems to overcome user behavior modeling challenges:

- **Modeling Abnormal Behavior:** In Part II we develop stronger fraud detection algorithms by modeling how fraudsters work:
  - **Detect Fraud in Static Graphs:** CatchSync (Ch. 3), using only the structure of graphs, detects fraud on both Twitter and Tencent Weibo. We demonstrate that it can scale to 3 billion edges and improves detection accuracy by up to *36%*.
  - **Detect Fraud in Graphs with Time:** By modeling the temporal patterns of fraudsters, CopyCatch (Ch. 4) detects illegitimate Page Likes on Facebook. Through careful design and implementation, CopyCatch is distributed on Hadoop, runs on Facebook's *10-billion edge* Page Like graph, and detects both fake and hijacked accounts.
  - **Detect Fraud in Graphs with Multiple Attributes:** Multiple attributes can be indicative of suspicious behavior. By flexibly taking into account all attributes, including time and IP addresses, CrossSpot (Ch. 5) finds retweet boosting and hashtag hijacking on Tencent Weibo and improves over previous techniques by up to *68%*.
- **Modeling Normal Behavior:** In Part III, we develop novel models of user behavior

that provide better understanding of recommendations:

  - **Flexible Models for Normal and Abnormal Behavior:** To capture the wide variety of behavior online, CoBaFi (Ch. 6) gives a highly flexible model of user behavior. We demonstrate that CoBaFi can cluster spammers, finds both bimodal and Gaussian rating distributions, and improves prediction accuracy by up to *17%*.
  - **Interpretable Recommendations:** To make recommender systems interpretable, we design a succinct additive co-clustering model of user behavior. ACCAMS (Ch. 7) matches the accuracy of state-of-the-art methods, while having a *4 times smaller* model.
  - **Explaining Recommendations:** To actually explain recommendation to users, we build on Chapter 7 and jointly model text reviews along with ratings. In Chapter 8, we offer PACO, a Poisson additive co-clustering model with an efficient Gibbs sampler. PACO predicts what words an individual would use in describing his feelings for a particular item.

- **Scalable Machine Learning:** In Part IV, we design distributed systems to scale user behavior modeling:

  - **Distributed Modeling of Attributed Hypergraphs:** To model attributed hypergraphs, FlexiFaCT (Ch. 9) distributes coupled tensor factorization. FlexiFaCT handles a variety of objective functions and scales to *billions of parameters*.
  - **Fast in the face of Stragglers:** Fugue (Ch. 10) overcomes the bottlenecks of slower machines called stragglers by exploiting the randomness of stochastic optimization. We demonstrate Fugue's effectiveness in topic modeling, dictionary learning, and community detection, improving convergence time by up *26 times* over previous methods.

11.2 Impact

In addition to the intellectual contributions, this work has already had broad impact across academia and industry:

- **Academic Recognition:**
  - CatchSync [108], from Chapter 3, was a Best Paper Finalist in *KDD* 2014, and was invited to the TKDD special issue for "Best of KDD 2014" [111].
  - FlexiFaCT [30], from Chapter 9, is the most cited paper of SDM 2014[1].
- **Industry adoption:**
  - CopyCatch (Ch. 4) was used for years by Facebook to detect fraud.
  - Other researchers have extended CopyCatch to detect fraud at both Instagram [42] and YouTube [143].
  - We built on the ideas of Part II also to detect fraud for Flipkart [100]
  - We were granted a patent for CopyCatch [34], which was assigned to Facebook.
- **Open-source code:**
  - We open-sourced ACCAMS (Ch. 7), and it has been accessed over 150 times from over 15 universities.
  - We open-sourced FlexiFaCT (Ch. 9), and it has been forked seven times on Github.
- **Education:**
  - CopyCatch (Ch. 4) was included in Carnegie Mellon University's "Multimedia Databases and Data Mining" (15-826) course as well as the University of Florida's "Social Network Computing" (CIS 6930) course.
  - FlexiFaCT (Ch. 9) and Fugue (Ch. 10) have been taught in Carnegie Mellon University's "Machine Learning with Large Datasets" (10-805).

[1]Based on scholar.google.com/scholar?cites=7979229435915048938, as of January 1, 2016.

Chapter 12

Overarching Vision and Future Work

We conclude by taking a step back to see the broad trends developed in this dissertation, and to follow these trends into the future.

12.1 Big Picture—Thesis Statements

Beyond the individual contributions offered by each chapter independently, this dissertation offers the following overarching research principles:

T1. Making use of Interaction Context: We believe that the contextual data acquired when a user interacts with the world around them, which is captured as edge attributes in our graphs, is interesting and valuable. We develop new techniques that make use of these edge attributes and demonstrate that we can move beyond asking "who" and "what" and instead now ask "how" and "why:"

- **Modeling *how* fraudsters work:** In Chapters 4 and 5, we model *how* fraudsters work, considering the time of interactions and IP addresses used, and as a result we better detect and limit fraud.
- **Understanding *why* users like certain items:** In Part III we design simple models of both ratings and text reviews, and as a result, we can understand not just what items a user would like, but also predict how he would describe *why* he likes or dislikes them.
- **Model attributed hypergraphs at scale:** By viewing attributed hypergraphs as coupled matrices and tensors, FlexiFaCT in Chapter 9 efficiently learns models of them at scale.

T2. Normal and Abnormal Behavior—Two Sides of the Same Coin: In the past, work on fraud detection and recommender systems have typically resided in separate communities. We believe that unifying these perspectives is valuable, and demonstrate that insights from one domain can be useful to the other:

- **Co-clustering for understanding:** For example, we demonstrate that co-clustering can be a highly useful technique for both detecting fraud, as in Chapters 4 and 5, as well as for recommender systems, as in Chapters 6, 7, and 8.
- **Finding fraudsters when making recommendations:** Additionally, we find cases where these perspectives can be directly merged. In Chapter 6, we build on insights from fraud detection literature [110, 112, 183, 190] to design a recommender system capable of detecting groups of fraudulent users.

T3. Bridging Models and Scalable Systems: Throughout this dissertation, we believe that it is important to scale behavior models, and demonstrate that by understanding the dependency structure of user behavior models and the properties of learning algorithms, we can design highly scalable user behavior models:

- **Scalable Fraud Detection:** In Chapter 4, we scale our CopyCatch algorithm to Facebook's massive graph with over 10 billion edges and 300 gigabytes of data.
- **Scalable Coupled Tensor Factorization:** In Chapter 9, by making use of the structure of tensor factorization, we distribute coupled tensor factorization with FlexiFaCT, scaling learning to billions of parameters.
- **Scaling Machine Learning in Chaotic Clouds:** In Chapter 10 we distribute factorization models in cloud environments where not all machines are equally fast and, as a result, there are stragglers. By exploiting the stochastic properties of machine learning algorithms, Fugue outperforms competitors by up to 26 times.

12.2 Future Directions

Building on theses success for graph-based user behavior modeling, there are three main challenges going forward to make hypergraphs broadly useful: (1) solve modeling challenges in new applications, focused on where our prior research can have the greatest impact and where new tools need to be developed, (2) develop new modeling techniques that address the practical limitations of classic modeling techniques but are general enough to be used across a wide variety of applications, (3) develop systems for scalable learning of models of large, complex networks. Through bridging these three challenges, we can develop new insights and holistic solutions to maximize real-world impact. We discuss the research opportunities and challenges for each of these areas below.

12.2.1 Application Outreach

First and foremost, it is important to expand the use of graph modeling to new applications. As described above, all relational databases can be considered large hypergraphs. As a result, there are fascinating opportunities in healthcare, education, security, distributed systems, financial systems, and many other fields. For example, in education can we predict what resources make a student enter or leave a field? What factors cause a student to misunderstand a concept, and what resources resolve that confusion? In healthcare, what combinations of unrelated diseases or treatments develop together? Can we predict when a treatment plan will fail, based on the circumstances of the patient? Through modeling graphs, we can change the way these fields develop, enabling new types of questions to be asked and giving actionable answers. By closely collaborating with researchers and practitioners in these fields, we can understand the unique constraints of their problems and develop custom, useful solutions.

12.2.2 Modeling and Algorithms

The second challenge is to develop general techniques for understanding attributed hypergraphs. For example, how do we find which relation types and attributes are important? In this regard, deep learning, which has been remarkably successful for modeling images and text and remarkably unused for modeling graphs, presents exciting potential. At a higher level, parts of the graph, edges or attributes, may have relevant meaning that we want to include. How can we include the insights of economics or biology in our models? To be concrete and to generalize on the work in this dissertation: how can we include arbitrary incentive functions and economic models, such as the cost and profit opportunities for fraudsters, in our models of graphs? In other fields, can we find behavior that overfits to economic incentives rather than predictive models, such as risk-averse doctors over-medicating or over-testing? Designing algorithms to understand attributed graphs and incorporate our intuition on these attributes will lead to more accurate and powerful models.

12.2.3 Scalable Machine Learning

The strength and opportunity in modeling graphs comes from the ability to jointly understand many complex interactions. Therefore, by improving the scalability of our learning algorithms and models, we can directly increase the impact of these models. In scaling graph models, there are multiple ongoing challenges: How can we learn more complex models, of more entities and more attributes, with more data? How can we learn these models quickly? When should we trade off accuracy for speed? Building scalable, flexible, and expressive systems for modeling large complex hypergraphs will improve the development of new models and the usefulness of those models in practice.

Appendices

Appendix A

CatchSync Proofs

Here we provide the proof of Theorem 1 from Chapter 3, which defines a parabolic lower limit in the synchronicity-normality relationship. This proof was originally presented in [108], with major contributions by Meng Jiang, and is included for completeness.

Proof 14 *In order to find the lower limit of synchronicity when given a normality, we define the problem as follows.*

Given *G grids, f_g and b_g counts of points ($f_g \leq b_g$) from the foreground/background cloud in grid g ($g = 1, \ldots, G$), the normality n of $\vec{f}$,* **find** *values of $\vec{f}$ to* minimize *the synchronicity s.*

Remind that (1) the synchronicity s is the synchronicity of foreground points, i.e., the dot product of (unit sum) foreground with the same foreground: $s = \sum_g \frac{f_g^2}{F^2}$; (2) the normality n is the dot product of (unit sum) foreground with (unit sum) background: $n = \sum_g \frac{f_g b_g}{FB}$.

Let $B(F)$ be the total counts: $\sum f_g = F$ and $\sum b_g = B$. Let $\hat{b}_g = b_g/B$ and similarly $\hat{f}_g = f_g/F$. Thus, the resulting vectors sum up to one ("probability vectors"). The problem definition is updated as follows.

Given *a (probability) vector $\vec{\hat{b}}$ with M entries ($M \leq G$),* **find** *a (probability) vector $\vec{\hat{f}}$ with given normality $n = \vec{\hat{f}} \cdot \vec{\hat{b}} = \sum(\hat{f}_g * \hat{b}_g)$ and minimum synchronicity $s = \vec{\hat{f}} \cdot \vec{\hat{f}} = \sum \hat{f}_g^2$, and* **report** *both the optimal such vector $\vec{\hat{f}}_{opt}$, as well as the minimum synchronicity $s_{\min}$.*

The method of Lagrange multipliers is a well-known strategy for finding the local minima (maxima) of a function subject to equality constraints. Here the Lagrange function is

$$\mathcal{F}(\hat{f}_g, \lambda, \mu) = (\sum_g \hat{f}_g^2) + \lambda(\sum_g \hat{f}_g - 1) + \mu(\sum_g (\hat{f}_g * \hat{b}_g) - n) \quad \text{(A.1)}$$

The gradients of the function are

$$\partial \mathcal{F}/\partial \hat{f}_g = 2\hat{f}_g + \lambda + \mu \hat{b}_g = 0 \quad g = 1, \ldots, M \quad \text{(A.2)}$$

and the two initial conditions are

$$\partial \mathcal{F}/\partial \lambda = \sum_g \hat{f}_g - 1 = 0 \tag{A.3}$$

$$\partial \mathcal{F}/\partial \mu = \sum_g (\hat{f}_g * \hat{b}_g) - n = 0 \tag{A.4}$$

From Eq A.2 we have, after summing them all up:

$$2 + M\lambda + \mu = 0 \tag{A.5}$$

From Eq A.2 we have, after multiplying each with $\hat{b}_g$ and summing them all up:

$$2 * n + \lambda + \mu s_b = 0 \tag{A.6}$$

where we call s_b the synchronicity of the background: $s_b = \sum_g \hat{b}_g^2 = \sum_g \frac{b_g^2}{B^2}$. Solving for μ we get

$$\mu = -2 - M\lambda \tag{A.7}$$

and substituting μ and for λ we get

$$\lambda = 2(s_b - n)/(1 - M * s_b) \tag{A.8}$$

We can substitute the values of μ and λ into Eq A.2 and solve for each $\hat{f}_g$, or, even faster, we multiply each of Eq A.2 with the corresponding $\hat{f}_g$ and we add, obtaining:

$$2 * s + \lambda + \mu n = 0 \tag{A.9}$$

which gives that the (optimal) s_{opt} satisfies

$$s_{opt} = 1/2(-\lambda - \mu n) \tag{A.10}$$

If the Hessian matrix is positive definite at a point, then the function is a convex *function and it attains a local* minima *at the point. Here the Hessian is a diagonal matrix with "2" in the first M positions, and zeros everywhere else. So eventually the minimum synchronicity is*

$$\boxed{s_{min} = (-Mn^2 + 2n - s_b)/(1 - Ms_b)} \tag{A.11}$$

That is, the minimum synchronicity s_{min} for a given normality n, is a quadratic function of n.

Appendix B

ACCAMS Approximation Guarantee

We provide here the formal proof for Theorem 2, with primary contributions by Alexander J. Smola.

Definition 9 (Covering Number) *Denote by $\mathcal{B}$ a Banach Space. Then for any given set $B \in \mathcal{B}$ the covering number $\mathcal{N}_\epsilon(B)$ is given by the set of points $\left\{b_1, \ldots b_{\mathcal{N}_\epsilon(B)}\right\}$ such that for any $b \in B$ there exists some b_j with $\|b - b_j\| \leq \epsilon$.*

Of particular interest for us are covering numbers $\mathcal{N}_\epsilon$ of unit balls and their functional inverses ϵ_n. The latter are referred to as entropy number and they quantify the approximation error incurred by using a cover of n elements [199, Chapter 8]. A key tool for computing entropy numbers of scaling operators is the theorem of Gordon, König and Schütt [83], relating entropy numbers to singular values.

Theorem 5 (Entropy Numbers and Singular Values) *Denote by D a diagonal scaling operator with $D : \ell_p \to \ell_p$ with scaling coefficients $\sigma_i \geq \sigma_{i+1} \geq 0$ for all i. Then for all $n \in \mathbb{N}$ the entropy number $\epsilon_n(D)$ is bounded via*

$$\epsilon_n(D) \leq 6 \sup_{j \in \mathbb{N}} \left(n^{-1} \prod_{i=1}^{j} \sigma_i\right)^{\frac{1}{j}} \leq 6\epsilon_n(D) \tag{B.1}$$

This means that if we have a matrix with rapidly decaying singular values, we only need to focus on the leading largest ones in order to approximate all elements in the space efficiently. Here the trade-off between dimensionality and accuracy is obtained by using the harmonic mean.

Corollary 1 (Entropy Numbers of Unit Balls) *The covering number of a ball $\mathcal{B}$ of radius r in ℓ_2^d is bounded by*

$$rn^{-\frac{1}{d}} \leq \epsilon_n(\mathcal{B}) \leq 6rn^{-\frac{1}{d}}. \tag{B.2}$$

Proof 15 *This follows directly from using the linear operator $x \to rx$ for $x \in \ell_2^d$. Here the scaling operator has eigenvalues $\sigma_i = r$ for all $1 \leq i \leq d$ and $\sigma_i = 0$ for $i > d$. The maximum in (B.1) is always $j = d$.*

With this we can now prove Theorem 2. This theorem states that we can approximate $\mathbf{R}$ up to a multiplicative constant at any step, provided that we pick a large enough clustering. It also means that we get linear convergence, i.e. convergence in $O(\log \epsilon)$ steps to $O(\epsilon)$ error, since the bound can be applied iteratively.

Theorem 6 (Approximation Guarantees) *Denote by $\sigma_1, \dots \sigma_n$ the singular values of $\mathbf{R}$. Then using l clusters for rows and columns respectively the matrix $\mathbf{R}$ can be approximated with error at most*

$$\|\mathbf{R} - \mathbf{R}'\|_\infty \le 2 \|\mathbf{R}\|^{\frac{1}{2}} \epsilon_l \left(\mathbf{\Sigma}^{\frac{1}{2}}\right)$$
$$\|\mathbf{R} - \mathbf{R}'\|_2 \le (\sqrt{N} + \sqrt{M}) \|\mathbf{R}\|^{\frac{1}{2}} \epsilon_l \left(\mathbf{\Sigma}^{\frac{1}{2}}\right)$$

Here ϵ_l is given by Theorem 5 and Corollary 1 respectively.

Proof 16 *Using the singular value decomposition of $\mathbf{R}$ into $\mathbf{R} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$ we can factorize $\mathbf{R} = \mathbf{Q}^\top \mathbf{P}$ where $\mathbf{Q} = \mathbf{\Sigma}^{\frac{1}{2}}\mathbf{U}$ and $\mathbf{P} = \mathbf{\Sigma}^{\frac{1}{2}}\mathbf{V}$. By construction, the singular values of $\mathbf{Q}$ and $\mathbf{P}$ are $\mathbf{\Sigma}^{\frac{1}{2}}$. We now cluster the rows of $\mathbf{Q}$ and $\mathbf{P}$ independently to obtain an approximation of $\mathbf{R}$.*

For $\mathbf{Q}$ we know that its rows can be approximated by l balls with error $\epsilon_l\left(\mathbf{\Sigma}^{\frac{1}{2}}\right)$ as per Theorem 5. Also note that its row vectors are contained in the image of the unit ball under $\mathbf{Q}$—if they were not, project them onto the unit ball and the approximation error cannot increase since the targets are within the unit ball, too. Hence the ϵ_l-cover of the latter also provides an approximation of the row-vectors of $\mathbf{Q}$ by $\mathbf{Q}'$ with accuracy ϵ_l, where $\mathbf{Q}'$ contains at most l distinct rows. The same holds for the matrix $\mathbf{P}$, as approximated by $\mathbf{P}'$. Hence we have

$$\begin{aligned}\left|\mathbf{R}_{ij} - \left\langle \mathbf{Q}'_{i:}, \mathbf{P}'_{j:}\right\rangle\right| &= \left|\left\langle \mathbf{Q}_{i:}, \mathbf{P}_{j:}\right\rangle - \left\langle \mathbf{Q}'_{i:}, \mathbf{P}'_{j:}\right\rangle\right| \\ &= \left|\left\langle \mathbf{Q}_{i:} - \mathbf{Q}'_{i:}, \mathbf{P}_{j:}\right\rangle + \left\langle \mathbf{Q}'_{i:}, \mathbf{P}_{j:} - \mathbf{P}'_{j:}\right\rangle\right| \\ &\le \|\mathbf{Q}_{i:} - \mathbf{Q}'_{i:}\| \|\mathbf{P}_{j:}\| + \|\mathbf{Q}'_{i:}\| \left\|\mathbf{P}_{j:} - \mathbf{P}'_{j:}\right\| \\ &\le 2 \|\mathbf{R}\|^{\frac{1}{2}} \epsilon_l \left(\mathbf{\Sigma}^{\frac{1}{2}}\right)\end{aligned}$$

This provides a pointwise *approximation guarantee.If we only have a bound on the rank and on $\|\mathbf{R}\|$, this yields*

$$|\mathbf{R}_{ij} - \langle q_i, r_j \rangle| \le 12 \|\mathbf{R}\| \, l^{\frac{1}{2d}}$$

Moreover, since each row in $\mathbf{Q}$ and $\mathbf{P}$ respectively will be approximated with residual bounded by ϵ_l we can bound $\|\mathbf{Q} - \mathbf{Q}'\| \le \sqrt{m}\epsilon_l$ and $\|\mathbf{P} - \mathbf{P}'\| \le \sqrt{n}\epsilon_l$ respectively. This yields a bound on the matrix norm of the residual via

$$\begin{aligned}\|[\mathbf{Q}\mathbf{P} - \mathbf{Q}'\mathbf{P}']\, x\| &\le \|\mathbf{Q}(\mathbf{P} - \mathbf{P}')x\| + \|(\mathbf{Q} - \mathbf{Q}')\mathbf{P}'x\| \\ &\le (\sqrt{N} + \sqrt{M}) \|\mathbf{R}\|^{\frac{1}{2}} \epsilon_l \left(\mathbf{\Sigma}^{\frac{1}{2}}\right) \|x\|\end{aligned}$$

This bounds the matrix norm of the residual.

Appendix C

FlexiFaCT Proof of Convergence

We provide here the complete proof for that FlexiFaCT, as given in Algorithm 13, converges. The proof was originally given in [30], with major contributions by Abhimanu Kumar; we provide the proof here for completeness. In Section 9.3 we first proved that two blocks in a stratum are interchangeable. Here, we use this to prove that sequence of strata are a regenerative process, defined later in this section. We use this to prove that our FlexiFaCT approach for tensor and coupled case converges.

As describe previously, our generic constrained loss function for a tensor case is

$$\mathcal{L} = L(U, V, W) + \lambda_u \|U\|_1 + \lambda_v \|V\|_1 + \lambda_w \|W\|_1 \\ s.t.\ \ U_{i,r}, V_{j,r}, W_{k,r} \geq 0.$$

In the above projected loss equation 9.10 the parameter is always in a set, $\mathcal{P}$, constrained by the ℓ_1 and non-negativity constraints. The set $\mathcal{P}$ is a hyperrectangle defined as $\exists\, a_i < b_i, i = 1 \dots r$, such that $\mathcal{P} = \{\theta : a_i \leq \theta_i \leq b_i\}$ where $a_i, b_i \in (-\infty, \infty)$. Here θ is a parameter to be updated as defined in previous section (equation 9.3). The gradient based on equation 9.10 is:

$$\nabla_\theta \mathcal{L} = \nabla_\theta L + p(\theta), \quad p(\theta) \in C(\theta) \tag{C.1}$$

where θ is defined in Table 9.2 and $\mathcal{L}$ and L are defined in equation 9.2. Function $p()$ is the projection or constraint term of the gradient. The set $C(\theta)$ is the union of the subgradients at θ. When $\theta \in$ interior of $\mathcal{P}, C(\theta)$ contains only the zero elements and contains the convex cone generated by the subgradients at θ when $\theta \in \partial\mathcal{P}$, boundary of $\mathcal{P}$.

Based on Definition 7 and Theorem 3, we can begin to view our algorithm as a regenerative process.

Definition 10 *a process $P(t), t \geq 0$ is regenerative if there exist time points $0 \leq T_0 < T_1 < T_2 < \dots$ such that the the remainder of the process after T_k $P(T_k + t) : t \geq 0$, for $k \geq 1$: (a) has* *the same distribution as the remainder of the process after T_0, and (b) process $P(T_0+t) : t \geq 0$ is independent of the process prior to T_k $P(t) : 0 \leq t < T_k$.*

In other words a stochastic process with certain time points such that from a probabilistic view the process restarts itself at these time points is called a regenerative process. Intuitively this means a regenerative process can be split in to i.i.d. cycles [24].

Based on equation 9.10 the projected SGD updates can be written as:

$$\theta^{t+1} = \Pi_{\mathcal{P}}(\theta^{(t)} + \eta_t \nabla L_x(\theta^{(t)})) \tag{C.2}$$

where $\Pi_{\mathcal{P}}()$ is the projection of the updated gradient with respect to the original loss $L(U, V, W)$. The projection step can be further broken up into

$$\begin{aligned} \theta^{t+1} &= \theta^{(t)} + \eta_t \nabla L_x^{s_i}(\theta^{(t)}) + \eta_t p(\theta^{(t)}) \;\; (using \;\; (9.11)) \\ &= \theta^{(t)} + \eta_t \nabla L^0(\theta^{(t)}) + \eta_t \delta M_t + \eta_t \beta_t + \eta_t p(\theta^{(t)}) \end{aligned} \tag{C.3}$$

where $L_x^{s_i}(\theta^{(t)})$ is the loss function at stratum s_i at a point x in iteration t given parameter value in previous iteration $\theta^{(t)}$. $\nabla L^0(\theta^{(t)})$ is the exact gradient in iteration t given previous parameter value $\theta^{(t)}$. And

$$\delta M_t = \nabla L_x^{s_i}(\theta^{(t)}) - \nabla L^0(\theta^{(t)}) - \beta_t \tag{C.4}$$

where β_t is the "error" before projection i.e. the error by which the update is outside $\mathcal{P}$.

To prove the convergence of the method we define the following conditions, similar to the ones defined in [77, 129]:

Condition 1. $\nabla L^0(\theta)$ is continuous.

Condition 2. $\nabla L^0(\theta^{(t)})$ is bounded in second moment: $E[(\nabla L^0(\theta^{(t)}))^2] < \infty$ for all θ.

Condition 3. The squared sum of the step sizes η_t is bounded i.e. $\sum_t \eta_t^2 < \infty$.

Condition 4. The noise is martingale difference: $E[\delta M_{(t+1)} | \delta M_i, i \leq t] = \delta M_t$.

Condition 5. $E[\eta_t \beta_t] < \infty$ with probability 1. Note that this is a condition on step-size. It implicitly says that the projection must not wander off infinitely outside the set $\mathcal{P}$ over the iterations.

Theorem 7 *The distributed SGD algorithm for tensor decomposition with projections, as presented in algorithm 13, converges.*

Proof. The primary equations being updated each time in our iterations is equation C.3. Rewriting it here we have:

$$\theta^{t+1} = \theta^{(t)} + \eta_t \nabla L^0(\theta^{(t)}) + \eta_t \delta M_t + \eta_t \beta_t + \eta_t p(\theta^{(t)}) \tag{C.5}$$

From theorem 3 we can see that the individual blocks in a given stratum are independent of each other's updates and are interchangeable. We can also observe from

Algorithm 13 that every stratum out of d strata is picked exactly once in one cycle i.e. one epoch (outer while loop). Moreover two different cycles of strata i.e. iterations of the while loop are identical and independent. In other words the while loop forms an i.i.d cycle, and thus a regenerative process. The time-period of cycles is finite and bounded consequently that of the regenerative process too. Besides given all the conditions 1 to 5 as defined above, we have

$$\left[\sum_{i=0}^{\kappa(t+t_0)-1} (\eta_i \delta M_i + \eta_i \beta_i)\right] \to 0 \tag{C.6}$$

for any arbitrary κ. The proof is similar to [129] and is valid due to the fact that noise is a martingale difference sequence and $\eta_i \delta M_i$ and $\eta_i \beta_i$ are an equicontinuous sequence ([129] Theorem 2.1, part 1, chapter 5; [77] follows a similar proof up to this point). We can now use this to analyze the updated with a projected loss. We find that equation C.3 has the same set of stable points as

$$\theta^{t+1} = \theta^{(t)} + \eta_t \nabla L^0(\theta^{(t)}) + \eta_t p(\theta^{(t)}) \tag{C.7}$$

Now we show that equation C.7 converges. Through few algebraic manipulations it can be verified that the projection functions $p(\cdot)$ we have, ℓ_1 soft threshold and non-negativity constraint project, are Lipschitz continuous. Following the arguments of [129] (theorem 2.1, part 2) and with the assumption that updates $\theta^{(t)}$ are bounded (follow from the conditions 1 to 5 assumed earlier), equation C.7 converges to a set of stationary points.
□

Appendix D

Fugue Proofs

We now provide the proofs of convergence as well as variance bounds for Fugue, as described in Chapter 10. The proofs were originally published in [127], with primary contributions by Abhimanu Kumar, and are provided here for completeness.

We begin with some new notation.

Definition 11 : State Potential Function (V)

$V^{(t)}$ encodes the probability that $\psi^{(t)}$ will be updated to $\psi^{(t+1)}$ when the algorithm performs the stochastic update over datapoint $y^{(t)}$. We also define an n_i-dimensional state potential $V = \left(V^{(t+1)}, V^{(t+2)} \dots V^{(t+n_i)}\right)$, which encodes the probability distribution of updates caused by all n_i iterations in block i of a sub-epoch (assuming that block i starts at iteration $t+1$).

Convergence Proof: Theorem 4

First, using the definition of V^t, we obtain the relation

$$p(\psi^{(t+1)}|\psi^{(t)})d\psi^{(t)} = p(V^{(t)}(\psi^{(t+1)}, \psi^{(t)}))dV^{(t)}(\psi^{(t+1)}, \psi^{(t)}). \tag{D.1}$$

We can interpret this equation as follows: fix an particular update event $\psi^{(t)} \to \psi^{(t+1)}$, then $V^{(t)}(\psi^{(t+1)}, \psi^{(t)})$ represents the event that some datapoint $y^{(t)}$ gets chosen, while $dV^{(t)}(\psi^{(t+1)}, \psi^{(t)})$ is the probability that said choice leads to the update event $\psi^{(t)} \to \psi^{(t+1)}$. The intuition here is that $V^{(t)} = V^{(t)}(\psi^{(t+1)}, \psi^{(t)})$ is a function that keeps track of the state of $\psi^{(t)}$ and $\psi^{(t+1)}$, and that depends on the datapoint ${Y_{i,j}}^{(t)}$ chosen by the SGD update.

Next, by definition of a martingale difference sequence:

$$\begin{aligned} E\left[\nabla\mathcal{L}(\psi^{(t)}) - \delta L^{(t)}(V^{(t)}, \psi^{(t)}) \mid \right.& \\ \left.\delta L^{(i)}(V^{(i)}, \psi^{(i)}), \psi^{(i)}, i < t, \psi^{(t)}\right] &= 0, \\ E\left[\varepsilon_t | \varepsilon_i, i < t\right] &= 0. \end{aligned} \tag{D.2}$$

In other words, the error term ε_t has zero expectation when conditioned on previous errors. We now have the necessary tools to provide a convergence guarantee, as given in Theorem 4.

From equation 10.4 we have

$$\begin{aligned} \psi^{(t+1)} &= \psi^{(t)} - \eta_t \delta L^{(t)}(V^{(t)}, \psi^{(t)}) \\ &= \psi^{(t)} - \eta_t \nabla \mathcal{L}(\psi^{(t)}) + \eta_t \left[\nabla \mathcal{L}(\psi^{(t)}) - \delta L^{(t)}(V^{(t)}, \psi^{(t)})\right] \\ &= \psi^{(t)} - \eta_t \nabla \mathcal{L}(\psi^{(t)}) + \eta_t \varepsilon_t \end{aligned} \tag{D.3}$$

Using n_i and N_w as defined in equation 10.6

$$\psi^{(t+(\sum_1^w n_i)m)} = \psi^{(t)} + \sum_{i=t}^{t+m(\sum_1^w n_i)} -\eta_i \nabla \mathcal{L}(\psi^{(i)}) + \sum_{i=t}^{t+m(\sum_1^w n_i)} \eta_i \varepsilon_i$$

$$\implies \psi^{(t+mN_w)} = \psi^{(t)} + \sum_{i=t}^{t+mN_w} -\eta_i \nabla \mathcal{L}(\psi^{(i)}) + \sum_{i=t}^{t+mN_w} \eta_i \varepsilon_i$$

$$\text{assuming} \sum_1^w n_i = N_w$$

$$\implies \psi^{(t+mN_w)} = \psi^{(t)} + \sum_{i=t}^{t+mN_w} -\eta_i \nabla \mathcal{L}(\psi^{(i)}) + M_{mN_w} \tag{D.4}$$

where $M_{mN_w} = \sum_{i=t}^{t+mN_w} \eta_i \varepsilon_i$ is a martingale sequence since it is a sum of martingale difference sequence. mN_w captures the m whole sub-epochs of work done as a whole by all the workers combined. From Doobs martingale inequality (Friedman, 1975, ch. 1, Thm 3.8)

$$P\left(\sup_{t+mN_w \geq r \geq t} |M_r| \geq c\right) \leq \frac{E\left[\left(\sum_{i=t}^{t+mN_w} \eta_i \varepsilon_i\right)^2\right]}{c^2} \tag{D.5}$$

where $M_r = \sum_{i=t}^{r} \eta_i \varepsilon_i$. Lets look at the RHS of equation D.5 above:

$$E\left[\left(\sum_{i=t}^{t+mN_w} \eta_i \varepsilon_i\right)^2\right] = E\left[\sum_{i=1}^{mN_w} (\eta_i \varepsilon_i)^2\right]$$

$$(\text{equation D.2} \implies E[\varepsilon_i \varepsilon_j] = 0 \text{ if } i \neq j)$$

$$= \sum_{i=1}^{mN_w} \eta_i^2 E[\varepsilon_i^2] \leq \sum_{i=1}^{mN_w} \eta_i^2 D \to 0$$

$$\text{where } E[\varepsilon_i^2] < D\ \forall i \text{ and assuming} \sum \eta_i^2 < \infty$$

$$\lim_{t\to\infty} \implies P\left(\sup_{i \geq t} |M_i| \geq c\right) = 0 \ \ as \ \ t \to \infty \tag{D.6}$$

From equation D.6 we have

$$\psi^{(t+mN_w)} = \psi^{(t)} + \sum_{i=t}^{t+mN_w} -\eta_i \nabla \mathcal{L}(\psi^{(i)})$$

asymptotically.

Note that we do a theoretical analysis of the algorithm without projection steps. Extending the proof to include projection can be done by using Arzela-Ascoli theorem and the limits of converging sub-sequence of our algorithm's SGD updates [129]. ■

Within block variance bound

We now bound the variance of the model state ψ, when it is updated inside block i in a sub-epoch.

Assumption 2 Assume for simplicity that the parameter being updated in block i is univariate. The analysis can be easily extended to multivariate updates.

Theorem 8 *Within block i, suppose we update the model state ψ using n_i datapoints (Eq. 10.6). Then the variance of ψ after those n_i updates is*

$$\begin{aligned} Var(\psi^{t+n_i}) = {} & Var(\psi^t) - 2\eta_t n_i \Omega_0 (Var(\psi^t)) - 2\eta_t n_i \Omega_0 CoVar(\psi_t, \bar{\delta}_t) + \eta_t^2 n_i \Omega_1 \\ & + \underbrace{\mathcal{O}(\eta_t^2 \rho_t) + \mathcal{O}(\eta_t \rho_t^2) + \mathcal{O}(\eta_t^3) + \mathcal{O}(\eta_t^2 \rho_t^2)}_{\Delta_t} \end{aligned}$$

Constants Ω_0 and Ω_1 are defined in theorems 9 and 10 respectively.

Proof. We start with analyzing $\mathbb{E}^V[u(\psi^{(t+ni)})]$ term from lemma 10

$$\mathbb{E}^V[u(\psi^{(t+ni)})] = \mathbb{E}^V[u(\psi^t + \underbrace{(-\sum_{i=1}^{n_i}\eta_{t+i}\delta L^{t+i}(v^{t+i}, \psi^{t+i}))}_{\nabla})]$$

$$= \mathbb{E}^V[u(\psi^t) - \frac{du(\psi^t)}{d\psi^t}\nabla + \frac{1}{2}\frac{du^2(\psi^t)}{d(\psi^t)^2}\nabla^2 + \mathcal{O}(\eta_t^3)]$$

$$= u(\psi^t) - \eta_t\frac{du(\psi^t)}{d\psi^t}\mathbb{E}^V[\sum_{i=1}^{n_i}\delta L^{t+i}(v^{t+i}, \psi^{t+i})] + \eta_t^2\frac{1}{2}\frac{du^2(\psi^t)}{d(\psi^t)^2}\mathbb{E}^V[(\sum_{i=1}^{n_i}\delta L^{t+i}(v^{t+i}, \psi^{t+i}))^2]$$

$$+ \mathcal{O}(\eta_t^3) \qquad \left(\text{since } \eta_t = \eta_{t+i} \text{ within a block and expanding } \nabla\right)$$

$$= u(\psi^t) - \eta_t\frac{du(\psi^t)}{d\psi^t}\sum_{i=1}^{n_i}\frac{d\mathbb{E}^V[L^{t+i}(v^{t+i}, \psi^{t+i})]}{d\psi^{t+i}} + \eta_t^2\frac{1}{2}\frac{du^2(\psi^t)}{d(\psi^t)^2}\mathbb{E}^V[(\sum_{i=1}^{n_i}\frac{dL^{t+i}(v^{t+i}, \psi^{t+i})}{d\psi^{t+i}})^2]$$

$$+ \mathcal{O}(\eta_t^3)$$

$$= u(\psi^t) - \eta_t\frac{du(\psi^t)}{d\psi^t}(\sum_{i=1}^{n_i}\frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i}, \psi^{t+i})]}{d\psi^{t+i}})$$

$$\left(\text{using Lemma 12}\right)$$

$$+ \eta_t^2\frac{1}{2}\frac{du^2(\psi^t)}{d(\psi^t)^2}\left[\mathbb{E}^V[\sum_{i=1}^{n_i}(\frac{dL^{t+i}(v^{t+i}, \psi^{t+i})}{d\psi^{t+i}})^2] + \mathbb{E}^V[\sum_{i\neq j}\frac{dL^{t+i}(v^{t+i}, \psi^{t+i})}{d\psi^{t+i}}\frac{dL^{t+j}(v^{t+j}, \psi^{t+j})}{d\psi^{t+j}}]\right]$$

$$+ \mathcal{O}(\eta_t^3)$$

$$= u(\psi^t) - \eta_t\frac{du(\psi^t)}{d\psi^t}(\sum_{i=1}^{n_i}\frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i}, \psi^{t+i})]}{d\psi^{t+i}}) + \eta_t^2\frac{1}{2}\frac{du^2(\psi^t)}{d(\psi^t)^2}\left[\sum_{i=1}^{n_i}\mathbb{E}^{v^{t+i}}[(\frac{dL^{t+i}(v^{t+i}, \psi^{t+i})}{d\psi^{t+i}})^2]\right.$$

$$\left.+ (\sum_{i\neq j}\frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i}, \psi^{t+i})]}{d\psi^{t+i}}\frac{d\mathbb{E}^{v^{t+j}}[L^{t+j}(v^{t+j}, \psi^{t+j})]}{d\psi^{t+j}})\right] + \mathcal{O}(\eta_t^3)$$

$$\left(\text{using Lemma 13}\right) \tag{D.7}$$

From equation D.7 and lemma 10

$$\mathbb{E}^{\psi^{(t+n_i)}}[u(\psi^{(t+n_i)})] = \mathbb{E}^{\psi^{(t)}}\Bigg[u(\psi^t) - \eta_t \frac{du(\psi^t)}{d\psi^t}(\sum_{i=1}^{n_i} \frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i}, \psi^{t+i})]}{d\psi^{t+i}}) + \eta_t^2 \frac{1}{2}\frac{du^2(\psi^t)}{d(\psi^t)^2}\Bigg[\sum_{i=1}^{n_i}\mathbb{E}^{v^{t+i}}[(\frac{dL^{t+i}(v^{t+i}, \psi^{t+i})}{d\psi^{t+i}})^2] + (\sum_{i\neq j}\frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i}, \psi^{t+i})]}{d\psi^{t+i}}\frac{d\mathbb{E}^{v^{t+j}}[L^{t+j}(v^{t+j}, \psi^{t+j})]}{d\psi^{t+j}})\Bigg]\Bigg] + \mathcal{O}(\eta_t^3) \tag{D.8}$$

From equation above the variance of ψ^{t+n_i} is

$$\begin{aligned}
&Var(\psi^{t+n_i}) = \mathbb{E}^{\psi^{(t+n_i)}}[(\psi^{(t+n_i)})^2] - \Big(\mathbb{E}^{\psi^{(t+n_i)}}[\psi^{(t+n_i)}]\Big)^2 \\
&= \mathbb{E}^{\psi^t}[(\psi^t)^2] - \eta_t n_i \mathbb{E}^{\psi^t}[2\psi^t(\Omega_0(\psi^t - \psi_* + \bar{\delta}_t) + \mathcal{O}(\rho_t^2))] \\
&\left(\text{using theorem 9 and defining } \bar{\delta}_t = \frac{\sum_{i=1}^{n_i}\delta_i}{n_i}\right) \\
&+ \eta_t^2 \frac{1}{2}\mathbb{E}^{\psi^t}[2\{n_i(\Omega_1 + \mathcal{O}(\mathbb{E}[\rho_t]) + \mathcal{O}(\rho_t^2)) \\
&+ \sum_{i\neq j}(\Omega_0(\psi^{t+i} - \psi_*) + \mathcal{O}(\rho_t^2))(\Omega_0(\psi^{t+j} - \psi_*) + \mathcal{O}(\rho_t^2))\}] \\
&- \Big(\mathbb{E}^{\psi^t}[\psi^t] - \eta_t n_i \mathbb{E}^{\psi^t}[(\Omega_0(\psi^t - \psi_* + \bar{\delta}_t) + \mathcal{O}(|\psi^t - \psi_*|^2))]\Big)^2 \\
&= \mathbb{E}^{\psi^t}[(\psi^t)^2] - 2\Omega_0\eta_t n_i \mathbb{E}^{\psi^t}[(\psi^t)^2] + 2\Omega_0\eta_t n_i \psi_* \mathbb{E}^{\psi^t}[\psi^t] - 2\Omega_0\eta_t n_i \mathbb{E}^{\psi^t}[\psi^t\bar{\delta}_t] - \mathcal{O}(\eta_t\rho_t^2) \\
&+ \eta_t^2 n_i \Omega_1 + \mathcal{O}(\eta_t^2\rho_t) + \mathcal{O}(\eta_t^2\rho_t^2) + \mathcal{O}(\eta_t^2\rho_t^3) + \mathcal{O}(\eta_t^2\rho_t^4) \\
&- \Big(\mathbb{E}^{\psi^t}[\psi^t]\Big)^2 + 2n_i\eta_t\mathbb{E}^{\psi^t}[\psi^t](\mathbb{E}^{\psi^t}[\Omega_0\psi^t] - \Omega_0\psi_* + \mathbb{E}^{\psi^t}[\Omega_0\bar{\delta}_t] + \mathcal{O}(\rho_t^2)) - \mathcal{O}(\eta_t^2\rho_t^2) + \mathcal{O}(\eta_t^3) \\
&= Var(\psi^t) - 2\eta_t n_i\Omega_0(Var(\psi^t)) - 2\eta_t n_i \Omega_0 CoVar(\psi_t, \bar{\delta}_t) + \eta_t^2 n_i \Omega_1 \\
&+ \underbrace{\mathcal{O}(\eta_t^2\rho_t) + \mathcal{O}(\eta_t\rho_t^2) + \mathcal{O}(\eta_t^3) + \mathcal{O}(\eta_t^2\rho_t^2)}_{\Delta_t}
\end{aligned} \tag{D.9}$$

■

Intra sub-epoch variance

Let us consider the variance of ψ across entire sub-epochs. First, note that within a sub-epoch, two blocks Z_i and $Z_{i'}$ are independent if for each datapoint $y \in Z_i$ and $y^{'} \in Z_{i'}$, the following holds:

$$\begin{aligned} \nabla L_y(\psi) &= \nabla L_y(\psi - \eta \nabla L_{y'}(\psi)) \\ \text{and} \quad \nabla L_{y'}(\psi) &= \nabla L_{y'}(\psi - \eta \nabla L_y(\psi)) \end{aligned} \tag{D.10}$$

In other words, even when the model state ψ is perturbed by the stochastic gradient on y', the stochastic gradient on y must not change (and vice versa). By our earlier argument on collections $\mathcal{C}_{ij} = \{Y_{ij}, \pi_{i\cdot}, \beta_{\cdot j}\}$, this condition holds true for any pair of points from distinct blocks. Thus, within a sub-epoch, distinct blocks Z_i and $Z_{i'}$ operate on disjoint subsets of Ψ, hence their update equations are independent of each other. At the end of a sub-epoch S_{n+1}, the algorithm synchronizes the model state $\Psi_{S_{n+1}}$ by aggregating the non-overlapping updates $\delta\psi^i_{S_{n+1}}$ from all blocks S^i_{n+1}. Therefore, we can write the variance $Var(\Psi_{S_{n+1}})$ at the end of sub-epoch S_{n+1} as

$$\begin{aligned} Var(\Psi_{S_{n+1}}) &= \sum_{i=1}^{w} Var(\psi^i_{S_{n+1}}) \qquad \text{(D.11)} \\ =&\sum_{i=1}^{w} \Bigg[Var(\psi^i_{S_n}) - 2\eta_{S_n} n_i \Omega^i_0 (Var(\psi^i_{S_n})) \\ &- 2\eta_{S_n} n_i \Omega^i_0 (CoVar(\psi^i_{S_n}, \bar{\delta}^i_{S_n})) + \eta^2_{S_n} n_i \Omega^i_1 + \Delta_{S^i_n} \Bigg] \\ =&Var(\Psi_{S_n}) - 2\eta_{S_n} \sum_{i=1}^{w} n_i \Omega^i_0 Var(\psi^i_{S_n}) \\ &- 2\eta_{S_n} \sum_{i=1}^{w} n_i \Omega^i_0 CoVar(\psi^i_{S_n}, \bar{\delta}^i_{S_n}) + \eta^2_{S_n} \sum_{i=1}^{w} n_i \Omega^i_1 + \mathcal{O}(\Delta_{S_n}), \end{aligned}$$

where the second line is proven below. The interpretation is similar to Theorem 8: when far from an optimum, the negative terms dominate and the variance shrinks; when close to an optimum, the positive Ω^i_1 term dominates but is also shrinking because of the step sizes $\eta^2_{S_n}$. Again, we can ignore the higher-order terms $\mathcal{O}(\Delta_{S_n})$.

We now prove the above relation:

$$\psi^{(t+1)} = \psi^{(t)} - \delta\psi^{(t)}(V^{(t)}, \psi^{(t)}) \tag{D.12}$$

$$\text{where} \ \ \delta\psi^{(t)}(V^{(t)}, \psi^{(t)}) = \eta^{(t)} \delta L^{(t)}(V^{(t)}, \psi^{(t)}) \Rightarrow \psi^{(t+1)} = \psi^t - \eta_t \delta L^t(V^t, \psi^t)$$

Summing equation D.12 over n_i, the number of points updated in block i of a sub-epoch

$$\psi^{t+n_i} = \psi^t - \sum_{i=1}^{n_i} \eta_{t+i} \delta L^{t+i}(V^{t+i}, \psi^{t+i}) \tag{D.13}$$

As defined earlier, V denotes the joint potential for all the n_i points encountered in block i. The equation D.1 for V is

$$p(\psi^{(t+n_i)}|\psi^t)d\psi^{(t+n_i)} = p(V(\psi^{(t+n_i)}, \psi^t))dV$$
$$\Rightarrow p(\psi^{(t+n_i)})d\psi^{(t+n_i)} = \int_{\psi^t} p(\psi^{(t+n_i)}|\psi^t)p(\psi^t)d\psi^t d\psi^{(t+n_i)} = \int_{\psi^t} p(V(\psi^{(t+n_i)}, \psi^t))dV p(\psi^t)d\psi^t \tag{D.14}$$

Lemma 10 *Let $u(\psi^{(t+n_i)})$ be a function of $\psi^{(t+n_i)}$ then*

$$\mathbb{E}^{\psi^{(t+n_i)}}[u(\psi^{(t+n_i)})] = \mathbb{E}^{\psi^t}[\mathbb{E}^V[u(\psi^{(t+n_i)})]]$$

Proof. From equation D.14

$$\begin{aligned}
\mathbb{E}^{\psi^{(t+n_i)}}[u(\psi^{(t+n_i)})] &= \int_{\psi^{(t+n_i)}} u(\psi^{(t+n_i)})p(\psi^{(t+n_i)})d\psi^{(t+n_i)} \\
&= \int_{\psi^{t+i}} u(\psi^{(t+n_i)})P(\psi^{(t+n_i)})d\psi^{(t+n_i)} \\
&= \int_V \int_{\psi^t} u(\psi^{(t+n_i)})P(V(\psi^{(t+n_i)}, \psi))dV P(\psi^t)d\psi^t \\
&= \mathbb{E}^{\psi^t}[\mathbb{E}^V[u(\psi^{(t+n_i)})]]
\end{aligned}$$

■

Lemma 11

$$\mathbb{E}^V[\delta L^{t+i}(v^{t+i}, \psi^{t+i})] = \frac{d\mathbb{E}^V[L^{t+i}(v^{t+i}, \psi^{t+i})]}{d\psi^{t+i}}$$

Proof. Due to randomness in picking the point to be updated in iteration $t+i$ We have

$$\mathbb{E}^V[L^{t+i}(v^{t+i}, \psi^{t+i})] = \int L(y, \psi^{t+i})dy$$
$$\Rightarrow \frac{d\mathbb{E}^V[L^{t+i}(v^{t+i}, \psi^{t+i})]}{d\psi^{t+i}} = \mathbb{E}^V[\frac{dL^{t+i}(v^{t+i}, \psi^{t+i})}{d\psi^{t+i}}] = \mathbb{E}^V[\delta L^{t+i}(v^{t+i}, \psi^{t+i})]$$

■

Lemma 12

$$\mathbb{E}^V[L^{t+i}(v^{t+i}, \psi^{t+i})] = \mathbb{E}^{v^{t+i}}[L^{t+1}(v^{t+i}, \psi^{t+i})]$$

Proof.

Using the definition of V^{t+i} in equation D.1, the fact that V is a joint variable of each V^{t+i} and an any iteration $t+i$ the chance of picking any data point is completely random and independent of any other iteration.

$$\mathbb{E}^{V}[L^{t+i}(v^{t+i},\psi^{t+i})] = \mathbb{E}^{v^{t+i}}[L^{t+1}(v^{t+i},\psi^{t+i})]$$

■

Lemma 13

$$\mathbb{E}^{V}[\frac{dL^{t+i}(v^{t+i},\psi^{t+i})}{d\psi^{t+i}}\frac{dL^{t+j}(v^{t+j},\psi^{t+i})}{d\psi^{t+j}}] = \frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i},\psi^{t+i})]}{d\psi^{t+i}}\frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i},\psi^{t+i})]}{d\psi^{t+i}}$$

Proof. Two different data points picked at iteration $(t+i)$ and $(t+j)$ are independent of each other. Using this fact and the definition of potential function V in equation D.14

$$\begin{aligned}\mathbb{E}^{V}[\frac{dL^{t+i}(v^{t+i},\psi^{t+i})}{d\psi^{t+i}}\frac{dL^{t+j}(v^{t+j},\psi^{t+i})}{d\psi^{t+j}}] &= \mathbb{E}^{V}[\frac{dL^{t+i}(v^{t+i},\psi^{t+i})]}{d\psi^{t+i}}]\mathbb{E}^{V}[\frac{dL^{t+j}(v^{t+j},\psi^{t+i})}{d\psi^{t+j}}] \\ \left(\text{using lemma 11}\right) &= \frac{d\mathbb{E}^{V}[L^{t+i}(v^{t+i},\psi^{t+i})]}{d\psi^{t+i}}\frac{d\mathbb{E}^{V}[L^{t+j}(v^{t+j},\psi^{t+j})]}{d\psi^{t+j}} \\ \left(\text{using lemma 12}\right) &= \frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i},\psi^{t+i})]}{d\psi^{t+i}}\frac{d\mathbb{E}^{v^{t+j}}[L^{t+j}(v^{t+j},\psi^{t+j})]}{d\psi^{t+j}}\end{aligned}$$

■

Theorem 9 *We define ψ_* as the global optima and Ω_0 as the Hessian of the loss at ψ_* i.e. $\Omega_0 = \frac{d^2\mathbb{E}[L(\psi_*)]}{d\psi_*^2}$ (assuming that ψ is univariate) then*

$$\frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i},\psi^{t+i})]}{d\psi^{t+i}} = \Omega_0(\psi_t - \psi_* + \delta_i) + \mathcal{O}(\rho_t^2)$$

where $\mathcal{O}(\rho_t^2) = \mathcal{O}(|\psi_{t+i} - \psi_*|^2)$ with the assumption that $\mathcal{O}(\rho_{t+i})$ is small $\forall i \geq 0$ and $\delta_i = \psi_{t+i} - \psi_t$.

Proof. Lets define $\phi(\psi_{t+i}) = \mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i},\psi^{t+i})]$ Using Taylor's theorem and expanding around ψ_*

$$\begin{aligned}\phi(\psi^{t+i}) &= \phi(\psi_*) + \frac{d\phi(\psi_*)}{d\psi_*}(\psi^{t+i} - \psi_*) + \frac{(\psi^{t+1} - \psi_*)^2}{2}\frac{d^2\phi(\psi_*)}{d\psi_*^2} + \mathcal{O}((\psi^{t+i} - \psi_*)^3) \\ &= \phi(\psi_*) + \frac{(\psi^{t+i} - \psi_*)^2}{2}\frac{d^2\phi(\psi_*)}{d\psi_*^2} + \mathcal{O}((\psi^{t+i} - \psi_*)^3)\left(\text{as } \frac{d\phi(\psi_*)}{d\psi_*} = 0 \text{ at optima}\right) \\ \Rightarrow \frac{d\phi(\psi^{t+i})}{d\psi^{t+i}} &= (\psi^{t+i} - \psi_*)\frac{d^2\phi(\psi_*)}{d\psi_*^2} + \mathcal{O}((\psi^{t+i} - \psi_*)^2)\end{aligned}$$

$$\Rightarrow \frac{d\mathbb{E}^{v^{t+i}}[L^{t+i}(v^{t+i},\psi^{t+i})]}{d\psi^{t+i}} = \Omega_0(\psi^t - \psi_* + \delta_i) + \mathcal{O}(\rho_t^2)\left(\text{with the assumption that } \mathcal{O}(\rho_t) \text{ is small we have } \mathcal{O}(\rho_{t+i}^2) = \mathcal{O}(\rho_t^2)\right)$$

■

Theorem 10 *With ψ_* as defined in theorem 9 and assuming that ψ is univariate we have*

$$\mathbb{E}^{v^{t+i}}[(\frac{dL^{t+i}(v^{t+i},\psi^{t+i})}{d\psi^{t+i}})^2] = \Omega_1 + \mathcal{O}(\mathbb{E}[\mathcal{O}(\rho_t)]) + \mathcal{O}(\rho_t^2)$$

where $\mathcal{O}(\rho_t^2)$ and δ_i are as defined in theorem 9 and $\Omega_1 = \mathbb{E}^{v^{t+i}}[(\frac{dL^{t+i}(v^{t+i},\psi_*)}{d\psi_*})^2]$

Proof. Expanding $L^{t+i}(v^{t+i},\psi^{t+i})$ around ψ_* using Taylor's theorem

$$\begin{aligned} L^{t+i}(v^{t+i},\psi^{t+i}) &= L^{t+i}(v^{t+i},\psi_*) + \frac{dL^{t+i}(v^{t+i},\psi_*)}{d\psi_*}(\psi^{t+i}-\psi_*) \\ &+\frac{1}{2}\frac{d^2L^{t+i}(v^{t+i},\psi_*)}{d\psi_*^2}(\psi^{t+i}-\psi_*)^2 + \mathcal{O}((\psi^{t+i}-\psi_*)^3) \\ \Rightarrow \frac{dL^{t+i}(v^{t+i},\psi^{t+i})}{d\psi^{t+i}} &= \frac{dL^{t+i}(v^{t+i},\psi_*)}{d\psi_*} + \frac{d^2L^{t+i}(v^{t+i},\psi_*)}{d\psi_*^2}(\psi^{t+i}-\psi_*) + \mathcal{O}((\psi^{t+i}-\psi_*)^2) \\ \Rightarrow \mathbb{E}^{v^{t+i}}[(\frac{dL^{t+i}(v^{t+i},\psi^{t+i})}{d\psi^{t+i}})^2] &= \mathbb{E}^{v^{t+i}}[(\frac{dL^{t+i}(v^{t+i},\psi_*)}{d\psi_*})^2 \\ +2\frac{dL^{t+i}(v^{t+i},\psi_*)}{d\psi_*}\frac{d^2L^{t+i}(v^{t+i},\psi_*)}{d\psi_*^2}&(\psi^{t+i}-\psi_*) + \mathcal{O}((\psi^{t+i}-\psi_*)^2)] \\ \Rightarrow \mathbb{E}^{v^{t+i}}[(\frac{dL^{t+i}(v^{t+i},\psi^{t+i})}{d\psi^{t+i}})^2] &= \Omega_1 + \mathcal{O}(\mathbb{E}[(\psi_{t+i}-\psi_*]) + \mathcal{O}(\rho_t^2) \\ &= \Omega_1 + \mathcal{O}(\mathbb{E}[\mathcal{O}(\rho_t)]) + \mathcal{O}(\rho_t^2) \end{aligned}$$

■

Slow-worker agnosticism

We now explain why allowing fast processors to do extra updates is beneficial. In Eq. D.11, we saw that the variance after each sub-epoch S depends on the number of datapoints touched n_i and the step size η_{S_n}. Let us choose η_{S_n} small enough so that the variance-decreasing terms dominate, i.e.

$$\begin{aligned} 2\eta_{S_n}\sum_{i=1}^{w} n_i\Omega_0^i Var(\psi_{S_n}^i) + 2\eta_{S_n}\sum_{i=1}^{w} n_i\Omega_0^i CoVar(\psi_{S_n}^i,\bar{\delta}_{S_n}^i) \\ > \eta_{S_n}^2\sum_{i=1}^{w} n_i\Omega_1^i. \end{aligned} \tag{D.15}$$

This implies $Var(\Psi_{S_{n+1}}) < Var(\Psi_{S_n})$. Hence, using more datapoints n_i decreases the variance of the model state ψ, provided that we choose η_{S_n} so that Eq. D.15 holds. This is easy to satisfy: the RHS of Eq. D.15 is $\mathcal{O}(\eta_{S_n}^2)$ while the LHS is $\mathcal{O}(\eta_{S_n})$, so we just set η_{S_n} small enough.

Bibliography

[1] Buy Twitter Followers. http://www.buy-followers.org.

[2] Buy Twitter Accounts. http://buytwitteraccounts.org.

[3] Buy, Sell, and Trade Twitter accounts. http://socialsellouts.com.

[4] Evrim Acar, Daniel M. Dunlavy, Tamara G. Kolda, and Morten Mørup. Scalable tensor factorizations with missing data. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pages 701–712, 2010.

[5] Evrim Acar, Tamara G. Kolda, and Daniel M. Dunlavy. All-at-once optimization for coupled matrix and tensor factorizations. *CoRR*, abs/1105.3422, 2011.

[6] Evrim Acar, Tamara G Kolda, and Daniel M Dunlavy. The Matlab CMTF toolbox. http://www.models.life.ku.dk/joda/CMTF_Toolbox, 2013.

[7] Alekh Agarwal and John C Duchi. Distributed delayed stochastic optimization. In *Decision and Control (CDC), 2012 IEEE 51st Annual Conference on*, pages 5451–5452. IEEE, 2012.

[8] Deepak Agarwal and Bee-Chung Chen. Regression-based latent factor models. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 19–28. ACM, 2009.

[9] Deepak Agarwal and Bee-Chung Chen. fLDA: matrix factorization through latent dirichlet allocation. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 91–100. ACM, 2010.

[10] Deepak Agarwal and Srujana Merugu. Predictive discrete latent factor models for large scale dyadic data. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '07, pages 26–35, New York, NY, USA, 2007. ACM.

[11] Charu C. Aggarwal. *An introduction to social network data analytics*. Springer, 2011.

[12] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 37–46, 2001.

[13] Amr Ahmed, Mohamed Aly, Joseph Gonzalez, Shravan M. Narayanamurthy, and Alexander J. Smola. Scalable inference in latent variable models. In *Proceedings of the Fifth International Conference on Web Search and Web Data Mining, WSDM 2012, Seattle, WA, USA, February 8-12, 2012*, pages 123–132, 2012.

[14] Edoardo M. Airoldi, David M. Blei, Stephen E. Fienberg, and Eric P. Xing. Mixed membership stochastic blockmodels. *Journal of Machine Learning Research*, 9:1981–2014, 2008.

[15] Leman Akoglu, Rishi Chandy, and Christos Faloutsos. Opinion fraud detection in online reviews by network effects. In *Proceedings of the Seventh International Conference on Weblogs and Social Media, ICWSM 2013, Cambridge, Massachusetts, USA, July 8-11, 2013.*, 2013.

[16] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. RTM: laws and a recursive generator for weighted time-evolving graphs. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 701–706, 2008.

[17] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II*, pages 410–421, 2010.

[18] Aris Anagnostopoulos, Anirban Dasgupta, and Ravi Kumar. Approximation algorithms for co-clustering. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '08, pages 201–210, New York, NY, USA, 2008. ACM.

[19] Animashree Anandkumar, Dean P Foster, Daniel Hsu, Sham M Kakade, and Yi-Kai Liu. Two SVDs suffice: Spectral decompositions for probabilistic topic modeling and latent dirichlet allocation. *arXiv preprint arXiv:1204.6703*, 2012.

[20] Reid Andersen. A local algorithm for finding dense subgraphs. *ACM Trans. Algorithms*, 6(4):60:1–60:12, September 2010.

[21] Christophe Andrieu, Nando de Freitas, Arnaud Doucet, and Michael I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50:5–43, 2003.

[22] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[23] Yuichi Asahiro, Kazuo Iwama, Hisao Tamaki, and Takeshi Tokuyama. Greedily finding a dense subgraph. *Journal of Algorithms*, 34(2):203–221, 2000.

[24] Soeren Asmussen. *Applied Probability and Queues*. Wiley, 1987.

[25] Brett W. Bader and Tamara G. Kolda. Matlab tensor toolbox version 2.2. *Albuquerque, NM, USA: Sandia National Laboratories*, 2007.

[26] Oana Denisa Balalau, Francesco Bonchi, TH Chan, Francesco Gullo, and Mauro Sozio. Finding subgraphs with maximum total density and limited overlap. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining*, pages 379–388. ACM, 2015.

[27] Arindam Banerjee, Inderjit S. Dhillon, Joydeep Ghosh, Srujana Merugu, and Dharmendra S. Modha. A generalized maximum entropy approach to bregman co-clustering and matrix approximation. *Journal of Machine Learning Research*, 8:1919–1986, 2007.

[28] Fabrício Benevenuto, Tiago Rodrigues, Virgílio Almeida, Jussara Almeida, and Marcos Gonçalves. Detecting spammers and content promoters in online video social networks. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 620–627. ACM, 2009.

[29] Alex Beutel, Amr Ahmed, and Alexander J. Smola. ACCAMS: Additive Co-Clustering to Approximate Matrices Succinctly. In Aldo Gangemi, Stefano Leonardi, and Alessandro Panconesi, editors, *Proceedings of the 24th International Conference on World Wide Web, WWW 2015, Florence, Italy, May 18-22, 2015*, pages 119–129. International World Wide Web Conferences Steering Committee, ACM, 2015.

[30] Alex Beutel, Abhimanu Kumar, Evangelos E. Papalexakis, Partha Pratim Talukdar, Christos Faloutsos, and Eric P. Xing. FlexiFaCT: Scalable flexible factorization of coupled tensors on hadoop. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 109–117. SIAM, 2014.

[31] Alex Beutel, Kenton Murray, Christos Faloutsos, and Alexander J. Smola. CoBaFi: Collaborative Bayesian Filtering. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 97–108. International World Wide Web Conferences Steering Committee, 2014.

[32] Alex Beutel, Markus Weimer, Tom Minka, Yordan Zaykov, and Vijay Narayanan. Elastic distributed bayesian collaborative filtering. In *NIPS workshop on Distributed Machine Learning and Matrix Computations*, 2014.

[33] Alex Beutel, Wanhong Xu, Venkatesan Guruswami, Christopher Palow, and Christos Faloutsos. CopyCatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 119–130. International World Wide Web Conferences Steering Committee, 2013.

[34] Alexander Beutel and Wanhong Xu. Detection of lockstep behavior, July 7 2015.

US Patent 9,077,744.

[35] David M. Blei and John Lafferty. Topic models. *Text mining: classification, clustering, and applications*, 10:71, 2009.

[36] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022, 2003.

[37] Léon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, LNAI 3176, pages 146–168. Springer Verlag, Berlin, 2004.

[38] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Computer networks*, 33(1), 2000.

[39] Christopher J. C. Burges, Robert Ragno, and Quoc V. Le. Learning to rank with nonsmooth cost functions. In *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*, pages 193–200, 2006.

[40] Jian-Feng Cai, Emmanuel J Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010.

[41] Qiang Cao, Michael Sirivianos, Xiaowei Yang, and Tiago Pregueiro. Aiding the detection of fake accounts in large scale social online services. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 197–210, 2012.

[42] Qiang Cao, Xiaowei Yang, Jieqi Yu, and Christopher Palow. Uncovering large groups of active malicious accounts in online social networks. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 477–488. ACM, 2014.

[43] Deepayan Chakrabarti. Autopart: Parameter-free graph partitioning and outlier detection. In *Knowledge Discovery in Databases: PKDD 2004, 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, Pisa, Italy, September 20-24, 2004, Proceedings*, pages 112–124, 2004.

[44] Deepayan Chakrabarti, Spiros Papadimitriou, Dharmendra S. Modha, and Christos Faloutsos. Fully automatic cross-associations. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 79–88. ACM, 2004.

[45] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Survey*, 41(3), 2009.

[46] Allison JB Chaney, David M Blei, and Tina Eliassi-Rad. A probabilistic model for using social networks in personalized item recommendation. In *Proceedings of the 9th ACM Conference on Recommender Systems*, pages 43–50. ACM, 2015.

[47] Moses Charikar. Greedy approximation algorithms for finding dense components in a graph. In *Approximation Algorithms for Combinatorial Optimization*, pages 84–95. Springer, 2000.

[48] Duen Horng Chau, Shashank Pandit, and Christos Faloutsos. Detecting fraudulent personalities in networks of online auctioneers. In *Knowledge Discovery in Databases: PKDD 2006, 10th European Conference on Principles and Practice of Knowledge Discovery in Databases, Berlin, Germany, September 18-22, 2006, Proceedings*, pages 103–114, 2006.

[49] Jie Chen and Yousef Saad. Dense subgraph extraction with application to community detection. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 24(7):1216–1230, 2012.

[50] Yizong Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790 –799, aug 1995.

[51] Joon Hee Choi and S.V.N. Vishwanathan. DFacTo: Distributed factorization of tensors. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 1296–1304, 2014.

[52] Byung-Gon Chun, Tyson Condie, Carlo Curino, Chris Douglas, Sergiy Matusevych, Brandon Myers, Shravan Narayanamurthy, Raghu Ramakrishnan, Sriram Rao, Josh Rosen, et al. Reef: Retainable evaluator execution framework. *Proceedings of the VLDB Endowment*, 6(12):1370–1373, 2013.

[53] Fan Chung and Linyuan Lu. The average distances in random graphs with given expected degrees. *PNAS*, 99(25), 2002.

[54] James Cipar, Qirong Ho, Jin Kyu Kim, Seunghak Lee, Gregory R. Ganger, Garth Gibson, Kimberly Keeton, and Eric P. Xing. Solving the straggler problem with bounded staleness. In *14th Workshop on Hot Topics in Operating Systems, HotOS XIV, Santa Ana Pueblo, New Mexico, USA, May 13-15, 2013*, 2013.

[55] Diane J. Cook and Lawrence B. Holder. *Mining graph data*. Wiley-Interscience, 2006.

[56] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley and Sons, New York, 1991.

[57] Koby Crammer and Gal Chechik. A needle in a haystack: local one-class optimization. In *Proceedings of the twenty-first international conference on Machine learning*, ICML '04, pages 26–, New York, NY, USA, 2004. ACM.

[58] Cristian Danescu-Niculescu-Mizil, Robert West, Dan Jurafsky, Jure Leskovec, and Christopher Potts. No country for old members: User lifecycle and linguistic change in online communities. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13, pages 307–318, 2013.

[59] Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

[60] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc V. Le, Mark Z. Mao, Marc'Aurelio Ranzato, Andrew W. Senior, Paul A. Tucker, Ke Yang, and Andrew Y. Ng. Large scale distributed deep networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1232–1240, 2012.

[61] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*, pages 137–150, 2004.

[62] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Fei-Fei Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255, 2009.

[63] Inderjit S. Dhillon, Subramanyam Mallela, and Dharmendra S. Modha. Information-theoretic co-clustering. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, August 24 - 27, 2003*, pages 89–98, 2003.

[64] Qiming Diao, Minghui Qiu, Chao-Yuan Wu, Alexander J. Smola, Jing Jiang, and Chong Wang. Jointly modeling aspects, ratings and sentiments for movie recommendation (JMARS). In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 193–202, 2014.

[65] Chris HQ Ding, Xiaofeng He, and Hongyuan Zha. A spectral method to separate disconnected and nearly-disconnected web graph components. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 275–280. ACM, 2001.

[66] William Eberle and Lawrence Holder. Discovering structural anomalies in graph-based data. In *ICDM*, pages 393–398, 2007.

[67] Facebook. Better Security through Software. blog.facebook.com/blog.php?post=248766257130. 2010.

[68] Facebook. Working Together to Keep You Secure. blog.facebook.com/blog.php?post=68886667130, 2009.

[69] Facebook. Staying in Control of Your Facebook Logins. blog.facebook.com/blog.php?post=389991097130, 2010.

[70] Facebook. Amendment no. 8 to form S-1/A. https://www.sec.gov/Archives/edgar/data/1326801/000119312512235588/d287954ds1a.htm, May 16 2012. Filed with the U.S. Securities and Exchange Commission.

[71] Facebook. Improvements to our Site Integrity Systems. facebook.com/10151005934870766, 2012.

[72] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM*, 29(4):251–262, 1999.

[73] Tom Fawcett. An introduction to ROC analysis. *PR letters*, 27(8):861–874, 2006.

[74] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

[75] The Apache Software Foundation. Apache Hadoop nextgen MapReduce (YARN). http://hadoop.apache.org/.

[76] Zoltán Füredi. An upper bound on Zarankiewicz' Problem. *Combinatorics, Probability and Computing*, 5(01):29–33, 1996.

[77] Rainer Gemulla, Erik Nijkamp, Peter J Haas, and Yannis Sismanis. Large-scale matrix factorization with distributed stochastic gradient descent. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 69–77. ACM, 2011.

[78] Thomas George and Srujana Merugu. A scalable collaborative filtering framework based on co-clustering. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM '05, pages 625–628, Washington, DC, USA, 2005. IEEE Computer Society.

[79] Saptarshi Ghosh, Bimal Viswanath, Farshad Kooti, Naveen Kumar Sharma, Gautam Korlam, Fabricio Benevenuto, Niloy Ganguly, and Krishna Phani Gummadi. Understanding and combating link farming in the twitter social network. In *Proceedings of the 21st international conference on World Wide Web*, pages 61–70. ACM, 2012.

[80] Christos Giatsidis, Dimitrios M. Thilikos, and Michalis Vazirgiannis. D-cores: measuring collaboration of directed graphs based on degeneracy. *KAIS*, 35(2):311–343, 2013.

[81] Alex Gittens and Michael W. Mahoney. Revisiting the nyström method for improved large-scale machine learning. In *ICML (3)*, pages 567–575, 2013.

[82] Prem Gopalan, Francisco J. Ruiz, Rajesh Ranganath, and David M. Blei. Bayesian nonparametric poisson factorization for recommendation systems. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014*, pages 275–283, 2014.

[83] Y. Gordon, H. König, and C. Schütt. Geometric and probabilistic estimates for entropy and approximation numbers of operators. *Journal of Approximation Theory*, 49:219–239, 1987.

[84] Dilan Görür, Frank Jäkel, and Carl Edward Rasmussen. A choice model with infinitely many latent features. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 361–368, New York, NY, USA, 2006. ACM.

[85] Lars Grasedyck. Hierarchical singular value decomposition of tensors. *SIAM Journal on Matrix Analysis and Applications*, 31(4):2029–2054, 2010.

[86] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander J. Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13:723–773, 2012.

[87] Thomas L. Griffiths and Zoubin Ghahramani. The indian buffet process: An introduction and review. *Journal of Machine Learning Research*, 12:1185–1224, 2011.

[88] Thomas L. Griffiths and Mark Steyvers. Finding scientific topics. *Proceedings of the National Academy of Sciences*, 101:5228–5235, 2004.

[89] Gunjan Gupta and Joydeep Ghosh. Robust one-class clustering using hybrid global and local search. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pages 273–280, New York, NY, USA, 2005. ACM.

[90] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan O. Pedersen. Combating web spam with trustrank. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases, Toronto, Canada, August 31 - September 3 2004*, pages 576–587, 2004.

[91] Tom S.F. Haines. Gaussian conjugate prior cheat sheet. http://thaines.com/content/misc/gaussian_conjugate_prior_cheat_sheet.pdf, 2011.

[92] N. Halko, P.G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Rev.*, 53(2):217–288, May 2011.

[93] R.A. Harshman. *Foundations of the PARAFAC procedure: Models and conditions for an "explanatory" multimodal factor analysis*. UCLA working papers in phonetics. University of California at Los Angeles, 1970.

[94] J. A. Hartigan. Direct clustering of a data matrix. *Journal of the American statistical association*, 67(337):123–129, 1972.

[95] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill$^{\text{TM}}$: A Bayesian skill

ranking system. In *NIPS*, 2007.

[96] Qirong Ho, James Cipar, Henggang Cui, Seunghak Lee, Jin Kyu Kim, Phillip B. Gibbons, Garth A. Gibson, Gregory R. Ganger, and Eric P. Xing. More effective distributed ML via a stale synchronous parallel parameter server. In *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States.*, pages 1223–1231, 2013.

[97] Matthew D. Hoffman, David M. Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

[98] Matthew D. Homan and Andrew Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, January 2014.

[99] Liangjie Hong, Amr Ahmed, Siva Gurumurthy, Alexander J. Smola, and Kostas Tsioutsiouliklis. Discovering geographical topics in the twitter stream. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 769–778, 2012.

[100] Bryan Hooi, Neil Shah, Alex Beutel, Stephan Gunneman, Leman Akoglu, Mohit Kumar, Disha Makhija, and Christos Faloutsos. Birdnest: Bayesian inference for ratings-fraud detection. In *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 2016.

[101] Jianying Hu, Fei Wang, Jimeng Sun, Robert Sorrentino, and Shahram Ebadollahi. A healthcare utilization analysis framework for hot spotting and contextual anomaly detection. In *AMIA 2012, American Medical Informatics Association Annual Symposium, Chicago, Illinois, USA, November 3-7, 2012*, 2012.

[102] Minqing Hu and Bing Liu. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 168–177. ACM, 2004.

[103] Xia Hu, Jiliang Tang, Yanchao Zhang, and Huan Liu. Social spammer detection in microblogging. In *IJCAI'13*, pages 2633–2639, 2013.

[104] Yifan Hu, Yehuda Koren, and Chris Volinsky. Collaborative filtering for implicit feedback datasets. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 263–272, 2008.

[105] Heng Huang, Chris Ding, Dijun Luo, and Tao Li. Simultaneous tensor subspace selection and clustering: the equivalence of high order svd and k-means clustering. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, pages 327–335. ACM, 2008.

[106] Inah Jeon, Evangelos E. Papalexakis, U. Kang, and Christos Faloutsos. Haten2: Billion-scale tensor decompositions. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1047–1058, 2015.

[107] Meng Jiang, Alex Beutel, Peng Cui, Bryan Hooi, Shiqiang Yang, and Christos Faloutsos. A general suspiciousness metric for dense blocks in multimodal data. In *2015 IEEE International Conference on Data Mining, ICDM 2015, Atlantic City, NJ, USA, November 14-17, 2015*, pages 781–786, 2015.

[108] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Catch-Sync: catching synchronized behavior in large directed graphs. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, pages 941–950, 2014.

[109] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Detecting suspicious following behavior in multimillion-node social networks. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 305–306, 2014.

[110] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Inferring strange behavior from connectivity pattern in social networks. In *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part I*, pages 126–138, 2014.

[111] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Catching synchronized behaviors in large networks: A graph mining approach. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 2015.

[112] Meng Jiang, Peng Cui, Alex Beutel, Christos Faloutsos, and Shiqiang Yang. Inferring lockstep behavior from connectivity pattern in large graphs. *Knowledge and Information Systems (KAIS)*, pages 1–30, 2015.

[113] Marc Yor Jim Pitman. The two-parameter poisson-dirichlet distribution derived from a stable subordinator. *Annals of Probability*, 25(2):855–900, 1997.

[114] Nitin Jindal and Bing Liu. Opinion spam and analysis. In *Proceedings of the International Conference on Web Search and Web Data Mining, WSDM 2008, Palo Alto, California, USA, February 11-12, 2008*, pages 219–230, 2008.

[115] Yohan Jo and Alice H Oh. Aspect and sentiment unification model for online review analysis. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 815–824. ACM, 2011.

[116] U. Kang, Evangelos E. Papalexakis, Abhay Harpale, and Christos Faloutsos. GigaTensor: scaling tensor analysis up by 100 times - algorithms and discoveries. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '12, Beijing, China, August 12-16, 2012*, pages 316–324, 2012.

[117] Suin Kim, Jianwen Zhang, Zheng Chen, Alice H Oh, and Shixia Liu. A hierarchical aspect-sentiment model for online reviews. In *AAAI*, 2013.

[118] Jon M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.

[119] Tamara G. Kolda and Brett W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009.

[120] Tamara G. Kolda and Jimeng Sun. Scalable tensor decompositions for multi-aspect data mining. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 363–372, 2008.

[121] Yehuda Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 426–434, 2008.

[122] Yehuda Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*, pages 447–456, 2009.

[123] Yehuda Koren, Robert M. Bell, and Chris Volinsky. Matrix factorization techniques for recommender systems. *IEEE Computer*, 42(8):30–37, 2009.

[124] Danai Koutra, U. Kang, Jilles Vreeken, and Christos Faloutsos. VOG: summarizing and understanding large graphs. In *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 91–99, 2014.

[125] Kenneth Kreutz-Delgado, Joseph F. Murray, Bhaskar D. Rao, Kjersti Engan, Te-Won Lee, and Terrence J. Sejnowski. Dictionary learning algorithms for sparse representation. *Neural Comput.*, 15(2):349–396, February 2003.

[126] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Trans. Knowl. Discov. Data*, 3(1):1:1–1:58, March 2009.

[127] Abhimanu Kumar, Alex Beutel, Qirong Ho, and Eric P Xing. Fugue: Slow-worker-agnostic distributed learning for big models on big data. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 531–539, 2014.

[128] Ravi Kumar, Jasmine Novak, and Andrew Tomkins. Structure and evolution of online social networks. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 611–617, 2006.

[129] Harold Kushner and George Yin. *Stochastic Approximation and Recursive Algorithms*

and Applications. Springer, 2003.

[130] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.

[131] Aapo Kyrola, Guy E. Blelloch, and Carlos Guestrin. GraphChi: Large-scale graph computation on just a PC. In *10th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2012, Hollywood, CA, USA, October 8-10, 2012*, pages 31–46, 2012.

[132] Angeliki Lazaridou, Ivan Titov, and Caroline Sporleder. A bayesian model for joint unsupervised induction of sentiment, aspect and discourse representations. In *ACL (1)*, pages 1630–1639, 2013.

[133] Daniel Lee and H. Sebastian Seung. Algorithms for non-negative matrix factorization. In *Advances in Neural Information Processing Systems*, NIPS '01, pages 556–562, 2001.

[134] Daniel D. Lee and H. Sebastian Seung. Learning the parts of objects by non-negative matrix factorization. *Nature*, 401(6755):788–791, 1999.

[135] Joonseok Lee, Samy Bengio, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local collaborative ranking. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014*, pages 85–96, 2014.

[136] Joonseok Lee, Seungyeon Kim, Guy Lebanon, and Yoram Singer. Local low-rank matrix approximation. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 82–90, 2013.

[137] Michael D. Lee. On the complexity of additive clustering models. *Journal of Mathematical Psychology*, 45(1):131–148, 2001.

[138] Victor E. Lee, Ning Ruan, Ruoming Jin, and Charu C. Aggarwal. A survey of algorithms for dense subgraph discovery. In *Managing and Mining Graph Data*, pages 303–336. Springer US, 2010.

[139] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123, 2009.

[140] Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *11th USENIX Symposium on Operating Systems Design and Implementation, OSDI '14, Broomfield, CO, USA, October 6-8, 2014.*, pages 583–598, 2014.

[141] Mu Li, David G. Andersen, Alexander J. Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information*

Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada, pages 19–27, 2014.

[142] Mu Li, Gary L. Miller, and Richard Peng. Iterative row sampling. In *FOCS 2013*, pages 127–136, 2013.

[143] Yixuan Li, Oscar Martinez, Xing Chen, Yi Li, and John E. Hopcroft. In a world that counts: Clustering and detecting fake social engagement at scale. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 111–120. International World Wide Web Conferences Steering Committee, 2016.

[144] Ee-Peng Lim, Viet-An Nguyen, Nitin Jindal, Bing Liu, and Hady Wirawan Lauw. Detecting product review spammers using rating behaviors. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 939–948. ACM, 2010.

[145] Chenghua Lin and Yulan He. Joint sentiment/topic model for sentiment analysis. In *Proceedings of the 18th ACM conference on Information and knowledge management*, pages 375–384. ACM, 2009.

[146] Guang Ling, Michael R Lyu, and Irwin King. Ratings meet reviews, a combined approach to recommend. In *Proceedings of the 8th ACM Conference on Recommender systems*, pages 105–112. ACM, 2014.

[147] Chao Liu, Xifeng Yan, Hwanjo Yu, Jiawei Han, and Philip S. Yu. Mining behavior graphs for "backtrace" of noncrashing bugs. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005, Newport Beach, CA, USA, April 21-23, 2005*, pages 286–297, 2005.

[148] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. GraphLab: A new parallel framework for machine learning. In *Conference on Uncertainty in Artificial Intelligence*, 2010.

[149] Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. *PVLDB*, 2012.

[150] Lester W. Mackey, Ameet Talwalkar, and Michael I. Jordan. Divide-and-conquer matrix factorization. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011. Proceedings of a meeting held 12-14 December 2011, Granada, Spain.*, pages 1134–1142, 2011.

[151] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

[152] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A system for large-scale

graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135–146, New York, NY, USA, 2010. ACM.

[153] Ching-Hao Mao, Chung-Jung Wu, Evangelos E. Papalexakis, Christos Faloutsos, Kuo-Chen Lee, and Tien-Cheu Kao. Malspot: Multi2 malicious network behavior patterns analysis. In *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part I*, pages 1–14, 2014.

[154] Koji Maruhashi, Fan Guo, and Christos Faloutsos. Multiaspectforensics: Pattern mining on large-scale heterogeneous networks with tensor analysis. In *International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2011, Kaohsiung, Taiwan, 25-27 July 2011*, pages 203–210, 2011.

[155] Julian McAuley and Jure Leskovec. Hidden factors and hidden topics: understanding rating dimensions with review text. In *Proceedings of the 7th ACM conference on Recommender systems*, pages 165–172. ACM, 2013.

[156] Julian McAuley, Jure Leskovec, and Dan Jurafsky. Learning attitudes and attributes from multi-aspect reviews. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 1020–1025. IEEE, 2012.

[157] Bhaskar Mehta and Wolfgang Nejdl. Unsupervised strategies for shilling detection and robust collaborative filtering. *User Model. User-Adapt. Interact.*, 19(1-2):65–97, 2009.

[158] Thomas P. Minka. Expectation propagation for approximate bayesian inference. In *UAI '01: Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence, University of Washington, Seattle, Washington, USA, August 2-5, 2001*, pages 362–369, 2001.

[159] Andriy Mnih and Ruslan Salakhutdinov. Probabilistic matrix factorization. In *Advances in neural information processing systems*, pages 1257–1264, 2007.

[160] HDK Moonesinghe and Pang-Ning Tan. Outrank: a graph-based outlier detection framework using random walk. *International Journal on Artificial Intelligence Tools*, 17(01):19–36, 2008.

[161] Arjun Mukherjee, Bing Liu, and Natalie S. Glance. Spotting fake reviewer groups in consumer reviews. In *Proceedings of the 21st World Wide Web Conference 2012, WWW 2012, Lyon, France, April 16-20, 2012*, pages 191–200, 2012.

[162] Noboru Murata. A statistical study on on-line learning. In *Online Learning and Neural Networks*. Cambridge University Press, 1998.

[163] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. MIT Press, 2012.

[164] Daniel J. Navarro and Thomas L. Griffiths. Latent features in similarity judgments:

A nonparametric bayesian approach. *Neural Computation*, 20(11):2597–2628, 2008.

[165] Radford M. Neal. Markov chain sampling methods for dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000.

[166] Mark E. J. Newman, Duncan J. Watts, and Steven H. Strogatz. Random graph models of social networks. *Proceedings of the National Academy of Sciences*, 99(suppl 1):2566–2572, 2002.

[167] Feng Niu, Benjamin Recht, Christopher Ré, and Stephen J. Wright. Hogwild!: A lock-free approach to parallelizing stochastic gradient descent. In *NIPS*, 2011.

[168] Caleb C. Noble and Diane J. Cook. Graph-based anomaly detection. In *KDD*, pages 631–636, 2003.

[169] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: bringing order to the web. 1999.

[170] Konstantina Palla, David A. Knowles, and Zoubin Ghahramani. An infinite latent attribute model for network data. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012*, 2012.

[171] Shashank Pandit, Duen Horng Chau, Samuel Wang, and Christos Faloutsos. Netprobe: a fast and scalable system for fraud detection in online auction networks. In *Proceedings of the 16th International Conference on World Wide Web, WWW 2007, Banff, Alberta, Canada, May 8-12, 2007*, pages 201–210, 2007.

[172] Ruoming Pang, Mark Allman, Mike Bennett, Jason Lee, Vern Paxson, and Brian Tierney. A first look at modern enterprise traffic. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 2–2. USENIX Association, 2005.

[173] Spiros Papadimitriou and Jimeng Sun. Disco: Distributed co-clustering with map-reduce: A case study towards petabyte-scale end-to-end mining. In *Data Mining, 2008. ICDM '08. Eighth IEEE International Conference on*, pages 512 –521, dec. 2008.

[174] Evangelos E. Papalexakis, Alex Beutel, and Peter Steenkiste. Network anomaly detection using co-clustering. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, pages 403–410. IEEE Computer Society, 2012.

[175] Evangelos E. Papalexakis, Christos Faloutsos, Tom M. Mitchell, Partha Pratim Talukdar, Nicholas D. Sidiropoulos, and Brian Murphy. Turbo-smt: Accelerating coupled sparse matrix-tensor factorizations by 200x. In *Proceedings of the 2014 SIAM International Conference on Data Mining, Philadelphia, Pennsylvania, USA, April 24-26, 2014*, pages 118–126, 2014.

[176] Evangelos E. Papalexakis and Nicholas D. Sidiropoulos. Co-clustering as multilin-

ear decomposition with sparse latent factors. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2011, May 22-27, 2011, Prague Congress Center, Prague, Czech Republic*, pages 2064–2067, 2011.

[177] Evangelos E. Papalexakis, Nicholas D. Sidiropoulos, and Rasmus Bro. From K -means to higher-way co-clustering: Multilinear decomposition with sparse latent factors. *IEEE Trans. Signal Processing*, 61(2):493–506, 2013.

[178] René Peeters. The maximum edge biclique problem is np-complete. *Discrete Appl. Math.*, 131(3):651–654, September 2003.

[179] Jian Pei, Daxin Jiang, and Aidong Zhang. On mining cross-graph quasi-cliques. In *KDD*, pages 228–238, 2005.

[180] Charles Perez, Marc Lemercier, Babiga Birregah, and Alain Corpel. Spot 1.0: Scoring suspicious profiles on twitter. In *ASONAM'11*, pages 377–381, 2011.

[181] Ian Porteous, Evgeniy Bart, and Max Welling. Multi-HDP: A non parametric bayesian model for tensor factorization. In D. Fox and C.P. Gomes, editors, *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, pages 1487–1490. AAAI Press, 2008.

[182] Russell Power and Jinyang Li. Piccolo: building fast, distributed programs with partitioned tables. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, pages 1–14. USENIX Association, 2010.

[183] B. Aditya Prakash, Ashwin Sridharan, Mukund Seshadri, Sridhar Machiraju, and Christos Faloutsos. EigenSpokes: Surprising patterns and scalable community chipping in large graphs. In *Advances in Knowledge Discovery and Data Mining, 14th Pacific-Asia Conference, PAKDD 2010, Hyderabad, India, June 21-24, 2010. Proceedings. Part II*, pages 435–448, 2010.

[184] Marilyn Price and Donna M Norris. Health care fraud: physicians as white collar criminals? *Journal of the American Academy of Psychiatry and the Law Online*, 37(3):286–289, 2009.

[185] Benjamin Recht and Christopher Ré. Parallel Stochastic Gradient Algorithms for Large-Scale Matrix Completion. *submitted*, 2011.

[186] Steffen Rendle and Lars Schmidt-Thieme. Pairwise interaction tensor factorization for personalized tag recommendation. In *WSDM '10: Proceedings of the third ACM international conference on Web search and data mining*, pages 81–90. ACM, 2010.

[187] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[188] Ruslan Salakhutdinov and Andriy Mnih. Bayesian probabilistic matrix factorization using markov chain monte carlo. In *Proceedings of the 25th international conference on Machine learning*, pages 880–887. ACM, 2008.

[189] Aaron Schein, John Paisley, David M Blei, and Hanna Wallach. Bayesian poisson tensor factorization for inferring multilateral relations from sparse dyadic event counts. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1045–1054. ACM, 2015.

[190] Neil Shah, Alex Beutel, Brian Gallagher, and Christos Faloutsos. Spotting suspicious link behavior with fBox: An adversarial perspective. In *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, pages 959–964. IEEE, 2014.

[191] Hanhuai Shan and Arindam Banerjee. Bayesian co-clustering. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM 2008), December 15-19, 2008, Pisa, Italy*, pages 530–539, 2008.

[192] Hanhuai Shan and Arindam Banerjee. Residual bayesian co-clustering for matrix approximation. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2010, April 29 - May 1, 2010, Columbus, Ohio, USA*, pages 223–234, 2010.

[193] Shashi Shekhar, Chang-Tien Lu, and Pusheng Zhang. Detecting graph-based spatial outliers: algorithms and applications (a summary of results). In *KDD'01*, pages 371–376, 2001.

[194] Roger N. Shepard and Phipps Arabie. Additive clustering: Representation of similarities as combinations of discrete overlapping properties. *Psychological Review*, 86(2):87, 1979.

[195] Kijung Shin and U. Kang. Distributed methods for high-dimensional and large-scale tensor factorization. In *2014 IEEE International Conference on Data Mining, ICDM 2014, Shenzhen, China, December 14-17, 2014*, pages 989–994, 2014.

[196] Matthew S. Shotwell and Elizabeth H. Slate. Bayesian outlier detection with dirichlet process mixtures. *Bayesian Analysis*, 6(4):665–690, 2011.

[197] Ajit Paul Singh and Geoffrey J. Gordon. Relational learning via collective matrix factorization. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, pages 650–658, 2008.

[198] Ajit Paul Singh and Geoffrey J. Gordon. A unified view of matrix factorization models. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML/PKDD 2008, Antwerp, Belgium, September 15-19, 2008, Proceedings, Part II*, pages 358–373, 2008.

[199] Alexander J. Smola. *Learning with Kernels*. PhD thesis, Technische Universität Berlin, 1998. GMD Research Series No. 25.

[200] Nathan Srebro, Noga Alon, and Tommi S. Jaakkola. Generalization error bounds for collaborative prediction with low-rank matrices. In *Advances in Neural Information*

Processing Systems 17 [Neural Information Processing Systems, NIPS 2004, December 13-18, 2004, Vancouver, British Columbia, Canada], pages 1321–1328, 2004.

[201] Tao Stein, Erdong Chen, and Karan Mangla. Facebook immune system. In *Proceedings of the 4th Workshop on Social Network Systems*, SNS '11, pages 8:1–8:8, New York, NY, USA, 2011. ACM.

[202] David H. Stern, Ralf Herbrich, and Thore Graepel. Matchbox: large scale online bayesian recommendations. In *Proceedings of the 18th international conference on World wide web*, pages 111–120. ACM, 2009.

[203] Gilbert Strang. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 3 edition, 1998.

[204] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, 2005.

[205] Jimeng Sun, Dacheng Tao, and Christos Faloutsos. Beyond streams and graphs: dynamic tensor analysis. In *Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006*, pages 374–383, 2006.

[206] Jimeng Sun, Dacheng Tao, Spiros Papadimitriou, Philip S. Yu, and Christos Faloutsos. Incremental tensor analysis: Theory and applications. *TKDD*, 2(3), 2008.

[207] Chenhao Tan, Ed H. Chi, David A. Huffaker, Gueorgi Kossinets, and Alexander J. Smola. Instant foodie: predicting expert ratings from grassroots. In *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, pages 1127–1136, 2013.

[208] David M. J. Tax and Robert P. W. Duin. Outlier detection using classifier instability. In *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, SSPR '98/SPR '98, pages 593–601, London, UK, UK, 1998. Springer-Verlag.

[209] Christina Teflioudi, Faraz Makari, and Rainer Gemulla. Distributed matrix completion. In *12th IEEE International Conference on Data Mining, ICDM 2012, Brussels, Belgium, December 10-13, 2012*, pages 655–664, 2012.

[210] Joshua B Tenenbaum. Learning the structure of similarity. *Advances in neural information processing systems*, pages 3–9, 1996.

[211] Ashish Thusoo, Zheng Shao, Suresh Anthony, Dhruba Borthakur, Namit Jain, Joydeep Sen Sarma, Raghotham Murthy, and Hao Liu. Data warehousing and analytics infrastructure at facebook. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, SIGMOD '10, pages 1013–1020, New York, NY, USA, 2010. ACM.

[212] Ivan Titov and Ryan T. McDonald. A joint model of text and aspect ratings for sentiment summarization. In *ACL*, volume 8, pages 308–316, 2008.

[213] Charalampos E Tsourakakis, Francesco Bonchi, Aristides Gionis, Francesco Gullo, and Maria A Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *KDD*, 2013.

[214] Matthew A. Turk and Alex P. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

[215] Johan Ugander, Brian Karrer, Lars Backstrom, and Cameron Marlow. The anatomy of the facebook social graph. *CoRR*, abs/1111.4503, 2011.

[216] Matthijs van Leeuwen, Jilles Vreeken, and Arno Siebes. Identifying the components. *Data Mining and Knowledge Discovery*, 19(2):176–193, 2009.

[217] Benjamin Van Roy and Xiang Yan. Manipulation-resistant collaborative filtering systems. In *Proceedings of the Third ACM Conference on Recommender Systems*, RecSys '09, pages 165–172, New York, NY, USA, 2009. ACM.

[218] M. N. Volkovs and R.S. Zemel. Boltzrank: Learning to maximize expected ranking gain. In *Proceedings of the International Conference on Machine Learning*, 2009.

[219] Jilles Vreeken, Matthijs van Leeuwen, and Arno Siebes. Krimp: mining itemsets that compress. *Data Mining and Knowledge Discovery*, 23(1):169–214, 2011.

[220] Chong Wang and David M. Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, August 21-24, 2011*, pages 448–456, 2011.

[221] Pu Wang, Kathryn B. Laskey, Carlotta Domeniconi, and Michael I. Jordan. Non-parametric bayesian co-clustering ensembles. In *Proceedings of the Eleventh SIAM International Conference on Data Mining, SDM 2011, April 28-30, 2011, Mesa, Arizona, USA*, pages 331–342, 2011.

[222] Xiang Wang and Ian Davidson. Active spectral clustering. In *ICDM*, pages 561–568, 2010.

[223] Xuerui Wang and Andrew McCallum. Topics over time: a non-markov continuous-time model of topical trends. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 424–433. ACM, 2006.

[224] Xuerui Wang, Natasha Mohanty, and Andrew McCallum. Group and topic discovery from relations and text. In *Proceedings of the 3rd international workshop on Link discovery*, pages 28–35. ACM, 2005.

[225] Markus Weimer, Alexandros Karatzoglou, Quoc V. Le, and Alexander J. Smola. COFI RANK - maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, Vancouver, British*

Columbia, Canada, December 3-6, 2007, pages 1593–1600, 2007.

[226] Jason Weston, Samy Bengio, and Nicolas Usunier. WSABIE: scaling up to large vocabulary image annotation. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 2764–2770, 2011.

[227] Baoning Wu, Vinay Goel, and Brian D. Davison. Propagating trust and distrust to demote web spam. *MTW*, 190, 2006.

[228] Baoning Wu, Vinay Goel, and Brian D. Davison. Topical trustrank: Using topicality to combat web spam. In *Proceedings of the 15th international conference on World Wide Web*, pages 63–72. ACM, 2006.

[229] Chao-Yuan Wu, Alex Beutel, Amr Ahmed, and Alexander J. Smola. Explaining reviews and ratings with PACO: Poisson Additive Co-Clustering. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, pages 127–128. International World Wide Web Conferences Steering Committee, 2016.

[230] Yinqing Xu, Wai Lam, and Tianyi Lin. Collaborative filtering incorporating review text and co-clusters of hidden user communities and item groups. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management, CIKM 2014, Shanghai, China, November 3-7, 2014*, pages 251–260, 2014.

[231] Xifeng Yan and Jiawei Han. CloseGraph: mining closed frequent graph patterns. In *KDD*, pages 286–295, 2003.

[232] Shuang-Hong Yang, Bo Long, Alexander J. Smola, Hongyuan Zha, and Zhaohui Zheng. Collaborative competitive filtering: learning recommender using context of user choice. In *Proceeding of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2011, Beijing, China, July 25-29, 2011*, pages 295–304, 2011.

[233] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information retrieval*, 1(1-2):69–90, 1999.

[234] Zhi Yang, Christo Wilson, Xiao Wang, Tingting Gao, Ben Y. Zhao, and Yafei Dai. Uncovering social network sybils in the wild. *TKDD*, 8(1):2:1–2:29, 2014.

[235] Yuan Yu, Michael Isard, Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, Pradeep Kumar Gunda, and Jon Currey. DryadLINQ: A system for general-purpose distributed data-parallel computing using a high-level language. In *Proceedings of the 8th USENIX conference on Operating systems design and implementation*, pages 1–14, 2008.

[236] Hyokun Yun, Hsiang-Fu Yu, Cho-Jui Hsieh, S. V. N. Vishwanathan, and Inderjit S. Dhillon. NOMAD: nonlocking, stochastic multi-machine algorithm for asyn-

chronous and decentralized matrix completion. *PVLDB*, 7(11):975–986, 2014.

[237] Dave Zachariah, Martin Sundin, Magnus Jansson, and Saikat Chatterjee. Alternating least-squares for low-rank matrix reconstruction. *IEEE Signal Processing Letters*, 19(4):231–234, April 2012.

[238] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *NSDI*, April 2012.

[239] Kazimierz Zarankiewicz. Problem p 101. In *Colloq. Math*, volume 2, page 301, 1951.

[240] Zhe Zhao, Zhiyuan Cheng, Lichan Hong, and Ed H. Chi. Improving user topic interest profiles by behavior factorization. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1406–1416. International World Wide Web Conferences Steering Committee, 2015.

[241] Yong Zhuang, Wei-Sheng Chin, Yu-Chin Juan, and Chih-Jen Lin. A fast parallel sgd for matrix factorization in shared memory systems. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 249–256. ACM, 2013.

[242] Martin Zinkevich, Alexander J. Smola, and John Langford. Slow learners are fast. In *Advances in Neural Information Processing Systems 22: 23rd Annual Conference on Neural Information Processing Systems 2009. Proceedings of a meeting held 7-10 December 2009, Vancouver, British Columbia, Canada.*, pages 2331–2339, 2009.

[243] Martin Zinkevich, Markus Weimer, Alexander J. Smola, and Lihong Li. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems 23: 24th Annual Conference on Neural Information Processing Systems 2010. Proceedings of a meeting held 6-9 December 2010, Vancouver, British Columbia, Canada.*, pages 2595–2603, 2010.

[244] Zhaonian Zou, Jianzhong Li, Hong Gao, and Shuo Zhang. Mining frequent subgraph patterns from uncertain graph data. *TKDE*, 22(9):1203–1218, 2010.