Stacked Graphical Learning

Zhenzhen Kou

December 2007
CMU-ML-07-123

Carnegie Mellon

Stacked Graphical Learning

Zhenzhen Kou

December, 2007
CMU-ML-07-123

Machine Learning Department
School of Computer Science
Carnegie Mellon University
Pittsburgh PA 15213

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

Thesis Committee:
William W. Cohen (Chair)
Robert F. Murphy
Tom M. Mitchell
David Jensen, University of Massachusetts Amherst

Keywords: Statistical Relational Learning, Graphical Models, Machine Learning

To my parents for love and support.

Abstract

In reality there are many relational datasets in which both features of instances and the relationships among the instances are recorded, such as hyperlinked web pages, scientific literature with citations, and social networks. *Collective classification* has been widely used to classify a group of related instances simultaneously. Recently there have been several studies on statistical relational learning for collective classification, including relational dependency networks, relational Markov networks , and Markov logic networks. In statistical relational learning models, collective classification is usually formulated as an inference problem over graphical models. Hence the existing collective classification methods are expensive due to the iterative inference procedure required for general graphical models. Procedures that learn collective classifiers are also expensive, especially if they are based on iterative optimization of an expensive iterative inference procedure. Our goal is to develop an efficient model for collective classification for relational datasets.

In my thesis, I have studied a learning scheme called *stacked graphical learning*. In stacked graphical learning, a base learner is augmented by providing the predicted labels of relevant instances. That is, first, a base learner is applied to the training data to make predictions using a cross validation-like technique. Then we expand the features by adding the predictions of relevant examples into the feature vector. Finally the base learner is applied to the expanded feature sets to make the final predictions. The intuition behind stacked graphical learning is that combining the predictions on the neighbors with local features can capture the dependencies among examples; hence we can rely on the base learner to classify the instances using the expanded feature set.

We have applied stacked graphical learning to many real problems including collective classification, sequential partitioning, information extraction, and multi-task problems in an information extraction system. Stacked graphical learning has been demonstrated to achieve competitive performance to the state-of-art relational graphical models with much less inference time.

In addition to exploring many applications of stacked graphical learning in real problems, we formally analyze an idealized version of the algorithm, which can be formulate as an inhomogeneous Gibbs sampling process with parameters learned in a greedy manner, provide proof of convergence of the idealized version of stacking, and discuss the conditions under which the algorithm of stacked graphical learning is nearly identical to the idealized stacked graphical learning.

We also studied an online version of stacked graphical learning, which integrates a single-pass online learning algorithm *Modified Balanced Winnow* with stacked learning. Online stacked graphical learning can save

training time and is capable to handle large streaming datasets with minimal memory overhead. We analyze the time and memory cost of online stacked graphical learning and applied it to several real problems.

Acknowledgments

I owe a lot to my advisors William W. Cohen and Robert F. Murphy for their great guidance and timely support. I am grateful that they gave me the invaluable opportunity to join the SLIF project when I was in the first year. I also appreciate the freedom I was given to work on both SLIF and CALO projects, which was really an eye opener. It has been a great pleasure to have the opportunity to work with such true experts in the field. My advisors taught me how to do research, how to give good talks, and how to write scientific papers. They are always supportive - when I met obstacles, their encouragement and optimism help me to gain more confidence, and to achieve my goal. William is more than just an academic advisor to me: he introduces me to his fellows and encourages me to explore opportunities in industry labs to broaden my horizons.

I would also like to thank my other thesis committee members, Tom Mitchell and David Jensen, for their invaluable assistance, feedback and patience at all stages of this thesis. Their criticisms, comments, and advice were critical in making this thesis more accurate, more complete and clearer to read.

Moreover, I thank all the fellow group members in Murphy's lab and text learning group for providing me inspiration and suggestions during the meetings and discussions. My special thanks go to Juchang, Sam, Xiang, Kai, Richard, Einat, Quinten, Sophie, Yifen, and Andrew for their insightful discussions. I thank Vitor R. Carvalho for the joint work on extending standard stacked graphical learning to the online version.

Thanks to my good friends and fellow graduate students, Yanhua Hu, Jimeng Sun, Huiming Qu, Minglong Shao, Changhao Jiang, Ke Yang, Ning Hu, Yanjun Qi, Ting Liu, Jiaxin Fu, Jie Xu, Jiang Ni, Sichen Sun for their good friendship and support.

I spent one summer at IBM TJ Watson as an intern. The experience provides me a unique opportunity to see how people outside universities do real work and do research. I thank my co-workers there who make the intern experience enjoyable: Yan Liu, Rick Lawrence, Saharon Rosset, Claudia Perlich, Prem Melville, and Naoki Abe.

I am also indebted to Carnegie Mellon University and many people here. CMU has the ideal atmosphere for graduate study: open and relaxing environment, insightful experts in diverse domains, nice and friendly people. My special thanks go to Christos Faloutsos for supporting me to explore research and career opportunities in data mining. I will surely miss many people here: Diane Stidle, Sharon Cavlovich, Monica Hopes, Chenyu Wu, Xiaofeng Wang, Leo Gu, Yan Li, and Yiheng Li.

Last, but definitely not the least, I would also like to thank my family for their love and support, without which I would not have survived the Ph.D. process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Relational Data and Statistical Relational Learning

Traditional machine learning methods assume that instances are independent and identically distributed, i.e., i.i.d.. In reality there are many relational datasets, such as hyperlinked web pages, scientific literature with citations, and social networks. Relational datasets record both features of instances and the relationships among the instances. Usually there are multiple types of objects and relations between objects in relational datasets. The dependencies among data can be complex, i.e., there can be several types of dependencies and there might be attributes associated with links. For example, in the international movie database, the relationship between actors and movies is "act-in", the relationship between movies and directors is "directed-by". And in a social network, people are connected due to "attending the same school", or "sharing same interests", or "having friends in common ". The instances can also have varying structures; for example, papers may have different numbers of authors, and web pages link to different numbers of web pages. Therefore relational datasets

often have structures and dependencies that contradict the "i.i.d." assumption of traditional machine learning algorithms.

Collective classification has been widely used for classification on relational datasets [26, 49]. In collective classification, classes are predicted simultaneously for a group of related instances, rather than predicting a class for each instance separately. Recently there have been several studies on statistical relational learning for collective classification. *Statistical relational learning* addresses the challenge of applying statistical learning algorithms to problems which involve rich collections of objects linked together in complex relational networks. Examples of relational graphical models include relational dependency networks [25], relational Markov networks [58], and Markov logic networks [55]. Statistical relational learning models have the capability of modeling dependencies between examples and provide better predictive accuracy and better understanding of the relational domains.

Collective classification can be formulated as an inference problem over *graphical models* [27], which are a powerful computational tool to represent and analyze complex statistical dependencies with graphs. In graphical models, nodes represent random variables, and edges represent conditional independencies. Graphical models define a joint probability distribution over all the variables as a product of the local functions at the nodes of the graph and inference queries are answered by marginalization. Consider collective classification in the context of Markov random fields (MRFs) [27]. Markov random fields are undirected graphical models which define independence among data as follows: two (sets of) nodes A and B are conditionally independent given a third set, C, if all paths between nodes in A and B are separated by a set of nodes in C. In MRFs, the meaning of "locality" is defined in terms of *maximal cliques*, cliques that can not be extended to include more nodes without losing the property of being fully connected. The local function is defined upon maximum cliques C and referred to as a *potential function* ψ. Following the setup for a classification task, the

conditional probability of the labels $\mathbf{Y}$ given the observations $\mathbf{x}$ can be written as follows:

$$P(Y_1, \ldots, Y_n | \mathbf{x}_1, \ldots, \mathbf{x}_n) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(x_C)$$

,

in which $\mathcal{C}$ represents a collection of cliques, C refers to a clique in $\mathcal{C}$, and Z is a normalization factor, obtained by summing the product with respect to x and usually referred to as *partition function*.

Inference in MRFs is intractible in the general case [28]. Common approximate inference approaches includes *Gibbs sampling*[23] and *loopy belief propagation* [28], which are based on iterative updating schemes. Learning the parameters for graphical models from a dataset is typically expensive, as it requires probabilistic inference (usually iterative procedures) in order to assess the likelihood of the data, and typically, this inference is the inner loop of some optimization procedure for training.

An alternative approach which is often used is to maximize the *pseudo-likelihood* of the data. Pseudo-likelihood approximates the likelihood function of a graphical model with a product of conditional distributions of each variable given its neighbors. As such, using a pseudo-likelihood method avoids the need to calculate the full partition function. This is quite tractible, as it simply requires learning separate conditional models for each variable. Gibbs sampling for an MRF with parameters learned to maximize pseudo-likelihood is closely related to *conditional dependency networks* [23].

In relational graphical models, the graph is often massive since each example is a node and each link in the dataset is an edge in the graph. In addition, Gibbs sampling usually takes many iterations to converge and thus graphical models are usually expensive, especially when exact inference is infeasible. To summarize, the

existing collective classification methods are expensive due to the iterative inference procedure required for general graphical models. Procedures that learn collective classifiers are also expensive, especially if they are based on iterative optimization of an expensive iterative inference procedure. Hence our goal is to develop an efficient model for collective classification for relational datasets.

1.2 Stacked Graphical Learning

Cohen and Carvalho introduced *stacked sequential learning* [5]. In a stacked sequential model, an extended instance is obtained by expanding each instance's features with the predicted labels of the nearby instances. To obtain labels predicted by a base learner, a cross validation-like technique is applied during training. On several sequential learning tasks the stacked sequential learning achieved better performance than other sequential algorithms such as conditional random fields (CRFs)[36].

Stacked sequential learning is capable of capturing long-range dependencies easily and can be constructed with any kind of base learner. It is also very efficient, having a limited number of rounds of learning. Also the inference is not iterative: instead, only one iteration is needed.

In my thesis, I have extended the stacked sequential model to a more general case, where relational data is considered as the application. I proposed a learning scheme called *stacked graphical learning*. In stacked graphical learning, a base learner is augmented by providing the predicted labels of related instances. That is, first, a base learner is applied to the training data to make predictions using a cross validation-like technique. Then we expand the features by adding the predictions of relevant examples into the feature vector. Finally the base learner is applied to the expanded feature sets to make the final predictions. The intuition behind stacked graphical

learning is that combining the predictions on the neighbors with local features can capture the dependencies among examples; hence we can rely on the base learner to classify the instances using the expanded feature set.

One advantage of stacked graphical learning is that dependencies can be captured easily using a *relational template* C, which finds instances related to a given example. Stacked graphical learning can be constructed based on any base learning algorithm, i.e., the base learner does not have to be a graphical model. Stacked graphical learning is easy to implement, and the training and inference are efficient.

We have applied stacked graphical learning to many real problems including collective classification, information extraction, and multi-task problems in an information extraction system. Stacked graphical learning has been observed to achieve competitive performance to the state-of-art relational graphical models with much less inference time.

1.3 Thesis Outline

In this thesis, our primary goal is to develop an efficient statistical relational model for inference over relational data. In particular, we hope to design models to scale up statistical relational learning to massive data sources. We organize the rest of the thesis as follows:

In Chapter 2, we introduce the algorithm of standard stacked graphical learning, and give an example to demonstrate how the algorithm works and its performance. Next, we formally analyze an idealized version of the algorithm, compare the stacking procedure to Gibbs samping, and provide theoretical proof for the convergence, in order to better understand its performance.

In Chapter 3, we describe the application of stacked graphical learning to three do-

mains: collective classification, sequential partitioning, and inter-related subtasks in multi-task systems. We evaluated stacked graphical learning with eleven real datasets and compared the performance with state-of-art relational graphical models. The experimental results demonstrated that stacked graphical learning can improve the performance of the base learner statistically significantly and are competitive in real applications: achieving comparable accuracy to state-of-art models with much less inference time. We also discuss the inference efficiency of stacked graphical learning compared to other relational models. The experimental results demonstrate that many iterations in not necessary in the inference of stacked graphical learning.

In Chapter 4, we provide in-depth study the application of stacked graphical learning to four cases: two applications in an information extraction system, Subcellular Location Image Finder (SLIF), another application to WebMaster system, and an application to the large-scale job title prediction. In the case study, we not only provide the performance of stacked graphical learning to solve a collective classification problem, but also analyze how to tune the setup to further improve the accuracy and explore the application in multi-task systems to multi-task learning.

In Chapter 5, we extend the standard stacked graphical learning to an online version, to save training time and to handle large streaming datasets with minimal memory overhead. We briefly describe a single-pass online learning algorithm *Modified Balanced Winnow* (MBW) [4] and introduce a scheme to integrate MBW with stacked learning. We analyze the time and memory cost of online stacked graphical learning and propose an implementation with limited data. In the application to real problems, we also provide empirical solutions to split relational datasets into disjoint subsets.

In Chapter 6, we survey recent progress in graphical models and their relational extensions. We reviewed related work on relational template design, efficient graphical models, and a sampling based scheme to analyze large-scale relational datasets.

In Chapter 7, we summarize the thesis work, state its major contributions as well as limitations, and discuss future directions.

Chapter 2

Stacked Graphical Learning

2.1 Notation

We consider here collective classification tasks, in which the goal is to "collectively" classify some set of instances in a relational dataset. In this section we introduce the notations first.

We denote a random variable by a capitalized letter (e.g., V or W), and the state or value of a corresponding variable by that same letter in lower case (e.g., v or w). We denote a list of variables by a bold-face capitalized token (e.g., $\mathbf{V}$ or $\mathbf{W}$). We use a corresponding bold-face lower-case token (e.g., $\mathbf{v}$ or $\mathbf{w}$) to denote an assignment of state or value to each variable in a given set. The probability of $\mathbf{W} = \mathbf{w}$ given $\mathbf{V} = \mathbf{v}$ is denoted as $p(\mathbf{W} = \mathbf{w}|\mathbf{V} = \mathbf{v})$. Let $\mathbf{w}_i^j$ denote an instance and let $\mathbf{x}_i$ denote a collection of instances $\mathbf{w}_i$'s: $\mathbf{x}_i =< \mathbf{w}_i^1, ..., \mathbf{w}_i^{N_i} >$. Let $\mathbf{y}_i$ denote the corresponding list of labels for $\mathbf{x}_i$, and $(\mathbf{x}_i, \mathbf{y}_i)$ is a labeled collection of instances. For example, in a relational dataset containing linked webpages, $\mathbf{w}_i$ can be a bag-of-word representation of a webpage and y_i is the category of $\mathbf{w}_i$. In a sequential classification task (e.g., extracting names from sentences), $\mathbf{x}_i$ can be a sequence of $\mathbf{w}_i^j$'s, where $\mathbf{w}_i^j$ is the

feature vector for token j of $\mathbf{x}_i$.

We use $p(\mathbf{Y} = \mathbf{y}|\mathbf{X} = \mathbf{x})$ (or $p(\mathbf{y}|\mathbf{x})$ as a shorthand) to denote the probability that $\mathbf{Y} = \mathbf{y}$ given $\mathbf{X} = \mathbf{x}$. The random variables $\mathbf{X}$ and $\mathbf{Y}$ are jointly distributed, but $\mathbf{Y}$ is usually inferred from $\mathbf{X}$ and in most cases people are interested in $p(\mathbf{Y}|\mathbf{X})$ directly (e.g., in a discriminative framework).

In our notation, a dataset is $D = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$. Let $\mathbf{x}$ denote a set of instances and $\mathbf{y}$ denotes the corresponding labels for $\mathbf{x}$.

In classification tasks, the inference problem is to estimate $p(\mathbf{Y}|\mathbf{X})$ given a training set D. In traditional machine learning with the i.i.d. assumption, the model for inference is usually in the following format:

$$p_\theta(\mathbf{Y}|\mathbf{X}) \propto \prod_{i=1}^{n} p_\theta(\mathbf{y}_i|\mathbf{x}_i)$$

where θ denotes the parameters in the model.

In relational datasets, dependencies among examples can not be ignored. Collective classification can be formulated as an inference problem over Markov random fields (MRFs), where nodes represent random variables and edges represent dependencies. MRFs define a joint probability distribution over all the variables in which each label y_i depends on the labels $y_{i_1}, \ldots, y_{i_L}$ of some set of "neighboring" or "related" instances, as well as the instance x_i. We let $\mathbf{Y}_{-i}$ denote the set of all variables with indices $\{j : j \neq i\}$. We will let MB_i denote variables in the Markov blanket (set of related instances) for y_i, i.e., the set such that $p(Y_i|\mathbf{Y}_{-i}, \mathbf{X}) = p(Y_i|\mathrm{MB}_i, \mathbf{X})$. When it is necessary to make a distinction, MB_i will denote a set of random variables, and $\mathrm{MB}_i(\mathbf{y}')$ will denote the projection of the concrete assignment $\mathbf{y}'$ onto the variables in MB_i. Hence in MRFs, we have

$$p(\mathbf{Y}|\mathbf{X}) = \prod_{i=1}^{n} p(Y_i|\mathbf{Y}_{-i}, \mathbf{x}, \theta)$$

.

Table 2.1: Summary of notations.

Symbol	Description
V, W	Random variables
$\mathbf{V}$, $\mathbf{W}$	A set of random variables
v, w	Assignments of V, W
$\mathbf{v}$, $\mathbf{w}$	Assignments of $\mathbf{V}$, $\mathbf{W}$
$\mathbf{x}_i$	A collection of instances, $\mathbf{x}_i =< \mathbf{w}_i^1, ..., \mathbf{w}_i^{N_i} >$
$\mathbf{y}_i$	The corresponding list of labels for $\mathbf{x}_i$
$D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$	Dataset
$p(\mathbf{Y} = \mathbf{y} \mid \mathbf{X} = \mathbf{x})$	Probability of $\mathbf{Y} = \mathbf{y}$ given $\mathbf{X} = \mathbf{x}$
MB_i	Random variables in the Markov blanket for y_i

Table 2.1 summarizes the notations defined here.

2.2 The Algorithm

In classifying relational datasets, the simplest approach of approximating $P(\mathbf{Y}|\mathbf{X}) = \prod_{i=1}^{n} p(\mathbf{y}_i|\mathbf{x}_i)$ with the *i.i.d.* assumption via "ordinary" machine learning algorithms can not achieve satisfactory accuracy. Hence we aim to develop a model in which $p(Y_i|\text{MB}_i, \mathbf{x})$ is estimated. However, in general the values in MB_i are not known and they are updated iteratively during inference over MRFs. We propose to approximate Y's in MB_i with a simpler model in stacked graphical learning, i.e., predict Y's with simple classification models. In our stacked graphical learning, we model the joint probability in a scheme similar to the pseudo-likelihood measure, i.e., if X_i and X_j are not directly connected, X_i is conditionally independent of X_j given X_i's neighbors. Hence we model the local conditional probability distribution with a base learner given features built upon predicted Y's in MB_i. And the joint probability is modeled

Given a training set $D = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$ and a base learner A, construct cross-validated predictions as follows:

1. Split D into J equal-sized disjoint subsets D_1 , ..., D_J.

2. For $j = 1$, ..., J, let $f_j = A(D - D_j)$. That is, train an ordinary classifier f_j, based on all the data from D except the subset D_j.

3. For $\mathbf{x}_t \in D_j$, let $\hat{\mathbf{y}}_t = f_j(\mathbf{x}_t)$. That is, for ($\mathbf{x}_t$, $\mathbf{y}_t$) in D_j, apply the classifier f_j to $\mathbf{x}_t$ to obtain a set of predictions, $\hat{\mathbf{y}}$.

Figure 2.1: A cross-validation-like technique to obtain predictions for training examples

by the local CPD while treating the data as *i.i.d.*.

We consider a model that captures the dependency by expanding the features of w_i^j with "predicted" labels for the *related instances*, i.e., variables in the Markov blanket. We use predicted labels instead of true labels since during inference there is no way to get true labels. We use a *relational template* C to pick up the related instances. A relational template is a procedure that finds all the instances related to a given example and returns their indices. For w_i^j, $C(w_i^j)$ retrieves the indices $i_1, ..., i_L$ of instances $w_i^{i_1}, ..., w_i^{i_L}$ that are related to w_i^j. Given predictions $\hat{\mathbf{y}}_\mathbf{i}$ for a set of instances $\mathbf{x}_i$, $C(w_i^j, \hat{\mathbf{y}}_i)$ returns the predictions on the related instances, i.e., $\hat{y}_{i_1}, ..., \hat{y}_{i_L}$. Since the relation between w_i^j and w_i^k might be one-to-many, for example, webpages link to different numbers of webpages, we allow aggregation functions to combine predictions on a set of related instances into a single feature. In the webpage example, a COUNT aggregator can be applied to retrieve the number of outgoing and incoming links in each category, given one webpage.

One practical difficulty to obtain predictions for training examples is that, while some learning methods produce reasonably well-calibrated probability estimates on

- Parameters: a relational template C.

- Learning algorithm: Given a training set $D = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$ and a base learner A:

  - Learn the local model, i.e., when $k = 0$:

    Let $f^0 = A(D^0)$. Please note that $D^0 = D$.

  - Learn the stacked models, for $k = 1...K$:

    1. Construct predictions $\hat{\mathbf{y}}_i^{k-1}$ for $\mathbf{x}_i^{k-1} \in D^{k-1}$ in a cross-validation-like way, as shown in Figure 2.1.
    2. Construct an extended dataset $D^k = \{(\mathbf{x}_1^k, \mathbf{y}_1), ..., (\mathbf{x}_n^k, \mathbf{y}_n)\}$ by converting $\mathbf{x}_i$ to $\mathbf{x}_i^k$ as follows: let $\mathbf{x}_i^k =< (w_i^1)^k, ..., (w_i^{N_i})^k >$, where $(w_i^j)^k =< w_i^j, C(w_i^j, \hat{\mathbf{y}}_i{}^{k-1}) >$, $C(w_i^j, \hat{\mathbf{y}}_i{}^{k-1})$ will return the predictions for examples related to w_i^j.
    3. Let $f^k = A(D^k)$.

- Inference algorithm: given a set of testing instances $\mathbf{x}$:

  I $\hat{\mathbf{y}}^0 = f^0(\mathbf{x})$.

  For $k = 1...K$,

  II Carry out Step 2 above to produce $\mathbf{x}^k$.

  III $\mathbf{y}^k = f^k(\mathbf{x}^k)$.

  Return $\mathbf{y}^K$.

Figure 2.2: Standard Stacked Graphical Learning and Inference. K: the level of stacking; $\mathbf{x}_i^k$: the instance expanded from $\mathbf{x}_i$; $\hat{\mathbf{y}}_i^k$: the level-k prediction of $\mathbf{x}_i$; and f^k: the learned classifier, at k^{th} level of stacking.

unseen test data, their probability estimates on *training* data are biased. Thus, to obtain the "predictions" for training examples, we apply a cross-validation-like technique suggested by a meta-learning scheme, *stacking* [61]. The procedure to obtain the predictions for training examples is shown in Figure 2.1. That is, we split the training set $D = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$ into J disjoint subsets D_1 , ..., D_J, train an ordinary classifier f_j for D_j based on all the data from D except the subset D_j, and apply f_j to examples in D_j to obtain their predictions.

Finally we end up with the inference and learning methods of Figure 2.2 for collective classification. In stacked graphical learning, we first ignore the dependencies and obtain the prediction for the training set via a cross-validation-like technique based on pure local features and an ordinary classifier. Then we expand the feature vector for each example with features calculated based on the predictions of its related examples. Finally we learn a model based on the expanded feature set, which contains both local attributes and information about dependencies.

The relational template can be extended to include aggregation functions based on $\hat{\mathbf{y}}$ and $\mathbf{x}_i$. We will demonstrate the use of aggregations in the next chapter.

2.3 An Example

Consider a collection of webpages with hyperlinks among them, as an example to demonstrate how stacked graphical learning works. In our example, we consider the WebKB dataset [11] which contains webpages from four computer science departments and the task of classifying webpages in a website into six categories: course, faculty, student, staff, research projects, or other. In this example, $\mathbf{x}_j$ is a site, $\mathbf{y}_j$ is the corresponding label, w_j^i is the bag-of-words representation of a webpage, and the relational template C encodes hyperlinks. We use Cr_i to denote a webpage labeled

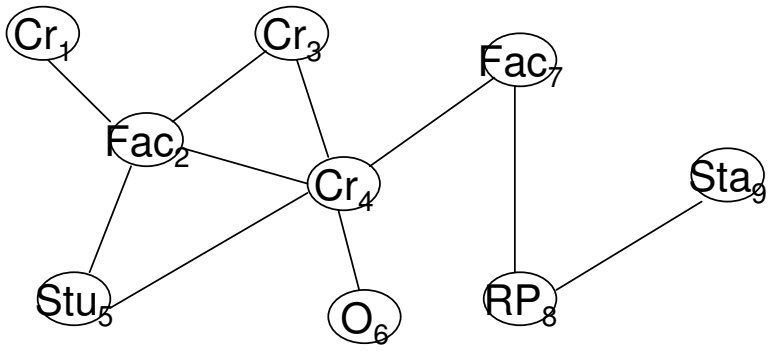

Figure 2.3: An example: the WebKB dataset

"course", Fac_j for a webpage labeled "faculty", Stu_k for a webpage labeled "student", Sta_l for a webpage labeled "staff", RP_m for a webpage labeled "research project", and O_n for a webpage labeled "other". Figure 2.3 shows examples with dependencies.

Define the webpages with direct hyperlinks between them to be relevant, e.g., in Figure 2.3, Cr_4 is relevant to Fac_2, Cr_3, Stu_5, O_6, and Fac_7. The relational template C selects the predictions of the relevant webpages, e.g., $C(w_4, \hat{\mathbf{y}}) = (\hat{v}_2, \hat{v}_3, \hat{v}_5, \hat{v}_6, \hat{v}_7)$. In this example, we define the aggregation function to COUNT the number of related instances in each category. For example, denote the categories course, faculty, student, staff, research projects, and other with 0, 1, 2,...,5 respectively. If $\hat{v}_2 = 1,\ \hat{v}_3 = 0,\ \hat{v}_5 = 2,\ \hat{v}_6 = 5,\ \hat{v}_7 = 1,\quad C(w_4, \hat{\mathbf{y}})$ will produce additional features (1,2,1,0,0,1).

In the WebKB problem, we split the webpages into four subsets by departments, hence $J = 4$. We use MaxEnt as the base learner (more details about the base learner and features are described in Chapter 3.1.1) and let A denote the base MaxEnt learner. The stacked graphical learning works in the following manner:

- To learn the stacked model:
  - Construct predictions $\hat{v}_i$ for each webpage w_i as follows:
    1. Split D into 4 subsets D_1, D_2, D_3, D_4.

2. For $j = 1, ..., 4$, let $f_j = A(D - D_j)$. For example, $f_1 = A(D_2 + D_3 + D_4)$. f_j is trained with a bag-of-words representation of webpages.

3. For $w_i \in D_j, \hat{v}_i = f_j(w_i)$. Here w_i is the bag-of-words representation of a webpage.

– Construct an extended dataset D' of instances (w_i', v_i) , where $w_i' = (w_i, C(w_i, \hat{\mathbf{y}}))$. That is, the extended feature vector contains the bag-of-words representation of a webpage and the additional features returned by the relational template.

– Return $f' = A(D')$, where f' is trained with the extended dataset D'.

- To learn the local model: Return $f = A(D)$.

- Inference algorithm: given a website (or a collection of webpages) $\mathbf{x}$:

I $\hat{\mathbf{y}}^0 = f^0(\mathbf{x})$.

For $k = 1...K$,

II Carry out Step 2 above to produce $\mathbf{x}^k$.

III $\mathbf{y}^k = f^k(\mathbf{x}^k)$.

Return $\mathbf{y}^K$.

Table 2.2 shows the performance of stacked graphical models with 1 and 2 iterations compared to the local MaxEnt model. On the WebKB dataset, stacked graphical learning improves the accuracy of the base learner and there is no significant difference in accuracy for k=1 and k=2 in stacking. We did not include the performance of stacking with more iterations in the table here. Usually with more iterations, the performance of stacking keeps at the same level, which suggests that stacking converges very quickly and does not require many iterations. We will study further into the performance of stacked graphical models in Chapter 3.

Table 2.2: Evaluation of stacked graphical models with the WebKB dataset. The accuracy for "webpage classification" is reported. We compared stacked graphical model to a local model, a MaxEnt model.

	WebKB
Local model	57.6
Stacked model (k=1)	72.9
Stacked model (k=2)	72.1

2.4 Theoretical Insight

In this section we formally analyze an idealized version of the algorithm, in order to better understand its performance. We consider a Markov chain to generate the data to be modeled. Starting with a homogeneous process defined under the scheme of dependency networks and Gibbs sampling, we introduce an idealized version of stacked graphical learning, which can be formulate as an inhomogeneous Gibbs sampling process with parameters learned in a greedy manner. Finally we discuss the conditions under which the algorithm of stacked graphical learning described in this chapter is nearly identical to the idealized stacked graphical learning.

2.4.1 Gibbs sampling and dependency networks

We will assume that the data to be modeled, $D = \{(\mathbf{x}_i, \mathbf{y}_i)\}$, can be generated by a homogeneous Markov chain, i.e., a Markov chain with time-invariant conditional probabilities. In other words, we assume that each $\mathbf{y}_i$ is drawn from a distribution $\pi(\mathbf{Y}|\mathbf{X} = \mathbf{x}_i)$, where $\pi(\mathbf{Y}|\mathbf{X})$ can be defined as the limit of the following process as $T \to \infty$:

1. for $i = 1 \ldots n$, pick $y_i^0 \sim p(Y_i|\mathbf{X} = \mathbf{x}, \theta^0)$

2. for $t = 1 \ldots T$

   (a) for $i = 1 \ldots n$,

   pick $y_i^t \sim p(Y_i | \mathbf{Y}_{-i} = \mathbf{y}_{-i}^{t-1}, \mathbf{X} = \mathbf{x}, \theta^+)$

Under relatively mild conditions [30, 63], this limit will exist, and will be independent of $\mathbf{y}^0$ (and hence θ^0).

In the algorithm above, θ^0 is some set of parameters that define the initial choice of values for y_i, and θ^+ is a set of parameters that defines a process for incrementally updating Y_i given estimated values for $\mathbf{Y}_{-i}$ and $\mathbf{x}$. We will assume that θ^+ and θ^0 are shorthand for a set of n probabilistic classifiers, one that predicts each Y_i.

To simplify our notation let us introduce these abbreviations:

$$\begin{aligned}
P_i(Y_i|\mathbf{x};\theta) &\equiv p(Y_i|\mathbf{X} = \mathbf{x}, \theta) \\
P_i(Y_i|\mathbf{x},\mathbf{y};\theta) &\equiv p(Y_i|\mathrm{MB}_i(\mathbf{Y}) = \mathrm{MB}_i(\mathbf{y}), \mathbf{X} = \mathbf{x}, \theta) \\
P(\mathbf{Y}|\mathbf{x};\theta) &\equiv \prod_i P_i(Y_i = y_i|\mathbf{x};\theta) \\
P(\mathbf{Y}|\mathbf{x},\mathbf{y}';\theta) &\equiv \prod_i P_i(Y_i = y_i|\mathbf{x},\mathbf{y}';\theta)
\end{aligned}$$

The process above can now be re-written as follows.

Definition 1 *Gibbs sampling is the following stochastic process:*

1. *for* $i = 1 \ldots n$, *pick* $y_i^0 \sim P_i(Y_i|\mathbf{x};\theta)$

2. *for* $t = 1 \ldots T$

   (a) *for* $i = 1 \ldots n$, *pick* $y_i^t \sim P_i(Y_i|\mathbf{x}, \mathbf{y}^{t-1}; \theta^+)$

(The limit of) Gibbs sampling is one means of approximating inference in a conditionally-defined Markov random field. It is intuitively appealing, as it simply

requires iteratively applying and re-applying a set of conditional probability models—such as could be learned by logistic regression, probabilistic decision trees, probabilistic SVMs, etc—each of which predicts a single variable Y_i from $\mathbf{x}$ and some set of "related" variables, as defined by MB_i.

To make these definitions more concrete, let us use them to define the following well-known learning method.

Definition 2 *The pseudo-likelihood learning method [1] for a Gibbs sampler and a class of models $\mathcal{M}$ is defined as follows.*

$$\begin{aligned}
\hat{\theta}^0 &= argmax_{\theta \in \mathcal{M}} \prod_{(\mathbf{x},\mathbf{y}) \in D} P(\mathbf{y}|\mathbf{x}, \theta) \\
\hat{\theta}^+ &= argmax_{\theta \in \mathcal{M}} \prod_{(\mathbf{x},\mathbf{y}) \in D} P(\mathbf{y}|\mathbf{x}, \mathbf{y}, \theta) \\
&= argmax_{\theta \in \mathcal{M}} \prod_{(\mathbf{x},\mathbf{y}) \in D} \prod_{i} P_i(y_i|\mathbf{x}, \mathbf{y}, \theta)
\end{aligned}$$

The "argmax" here means that $\hat{\theta}^+$ will be the maximum-likelihood (ML) model in $\mathcal{M}$ for each Y_i. The optimization over $P(y_i|\text{MB}_i(\mathbf{y}), \mathbf{x}, \theta)$ means that in the ML optimization used to train each probabilistic classifier for Y_i, the values for the "related" values MB_i will be taken from values for $\mathbf{y}$ seen in the training data D. The model learned by this method is sometimes called a *dependency network* [23].

2.4.2 An idealized version of stacked graphical learning

Pseudo-likelihood/dependency net learning approximates the unknown Gibbs sampling parameters θ^0, θ^+ with estimates $\hat{\theta}^0$, $\hat{\theta}^+$ that will be used in a Gibbs sampler of the same form. We propose to approximate the unknown Gibbs sampler with a different samping procedure:

Definition 3 *Inhomogeneous Gibbs sampling is the following stochastic process:*

1. *for* $i = 1 \ldots n$, *pick* $y_i^0 \sim P_i(Y_i|\mathbf{x}; \theta^0)$

2. *for* $k = 1 \ldots K$

   (a) *for* $i = 1 \ldots n$, *pick* $y_i^k \sim P_i(Y_i|\mathbf{x}, \mathbf{y}^{k-1}; \theta^k)$

Note that this sampling procedure is defined by $K+1$ sets of parameters, $\theta^0, \theta^1, \ldots, \theta^K$, rather than only two, as in the Gibbs sampling described in Definition 1. We will use $Q^k(\mathbf{x}; \theta^0, \ldots, \theta^k)$ to denote the distribution produced after k iterations of inhomogeneous Gibbs sampling.

Clearly, inhomogeneous Gibbs samplers include all ordinary Gibbs samplers, since it could be that K is large and $\theta^1 = \theta^2 = \ldots = \theta^+$. Hence inhomogeneous samplers produce a larger class of distributions. There are several potential advantages of considering this larger class. First, there may be practical problems that are better approximated by the larger class. Second, it is possible that even distributions generated by a long homogeneous Gibbs sampler can be well-approximated by a *short* inhomogeneous Gibbs sampler. Notice that a short inhomogeneous Gibbs sampler can be executed quickly.

Finally, the larger class of inhomogeneous samplers may be computationally more efficient to learn. This may seem counter-intuitive, but recall that there are many instances of learning tasks which are made computationally easier by expanding the class of possible models (e.g., [56]).

To learn the parameters of an inhomogeneous Gibbs sampler, we begin by posing the question: can one efficiently learn an ML estimate of the parameters $\theta^0, \theta^1, \ldots, \theta^K$ above? We claim that the answer to this question is "no", even if the Markov blanket of every Y_i is small. This can be demonstrated by viewing the series of models at $k = 1 \ldots K$ as a dynamic Bayes network (DBN) with $t = 1 \ldots K$, in which $y_i^t \sim P(Y_i|\mathbf{Y}_{-i} = \mathbf{y}_{-i}^{t-1}, \mathbf{X} = \mathbf{x}, \theta^t)$ specifies the state transition probability. In general,

learning an ML estimate of parameters in DBNs is expensive [21]. Therefore we propose the following method for approximating the ML estimate.

Definition 4 *Idealized stacked graphical learning is the following learning method.*

1. *Let* $\hat{\theta}^0 = argmax_{\theta \in \mathcal{M}} \prod_{(\mathbf{x},\mathbf{y}) \in D} P(\mathbf{y}|\mathbf{x}, \theta)$

2. *For* $k = 1, \ldots, K$*:*

   (a) *Create* D^k *by replacing each* $(\mathbf{x}, \mathbf{y}) \in D$ *with* $(\mathbf{x}, \mathbf{y}')$ *where* $\mathbf{y}'$ *was drawn from* $Q^{k-1}(\mathbf{x}; \hat{\theta}^0, \ldots, \hat{\theta}^{k-1})$*. Let* $\mathbf{y}'^{k-1}_{\mathbf{x}}$ *denote the* $\mathbf{y}'$ *so drawn.*

   (b) *Set*

$$\hat{\theta}^k = argmax_{\theta \in \mathcal{M}} \prod_{(\mathbf{x},\mathbf{y}) \in D} Q(\mathbf{y}|\mathbf{x}, \mathbf{y}'^{k-1}_{\mathbf{x}}, \theta)$$

The optimization over $P(y_i|\mathrm{MB}_i(\mathbf{y}), \mathbf{x}, \theta)$ means that in the ML optimization used to train the k-th probabilistic classifier for Y_i, the values for the "related" values MB_i will be taken from values for $\mathbf{y}'$ constructed in Step 2a.

2.4.3 Stacked graphical learning as greedy learning of an inhomogeneous sampler

To understand the idealized stacked graphical learning method better, first notice that the learning algorithm picks $\hat{\theta}^0$ as in pseudo-likelihood training. Let us now consider how $\hat{\theta}^1$ is chosen. We will show that the method picks $\hat{\theta}^1$ so as to force the distribution $Q^1(\mathbf{x})$ to be as close as possible to the unknown $P(\mathbf{Y}|\mathbf{X})$ on the data D with regard to KL-divergence.

Intuitively, in the unknown target inhomogeneous Gibbs sampler that we are trying to learn, θ^1 will be applied to values $(\mathbf{x}, \mathbf{y}')$ where $\mathbf{y}'$ is generated according to

$Q^0(\mathbf{x};\theta^0)$. We do not know θ^0, but we can approximate θ^0 with $\hat{\theta}^0$, and pick θ^1 by maximizing the empirical probability of $P(\mathbf{y}|\mathrm{MB}_i(\mathbf{y}'),\mathbf{x},\theta)$ where $\mathbf{y}'$ is sampled from $Q^0(\mathbf{x};\hat{\theta}^0)$:

$$\hat{\theta}^1 = \mathrm{argmax}_{\theta\in\mathcal{M}} P(\mathbf{y}|\mathrm{MB}_i(\mathbf{y}'),\mathbf{x},\theta)$$

The idealized stacked graphical learning method does exactly this, and then continues this process for K further iterations.

One can formalize the claim that idealized stacked graphical learning is a greedy learner for homogeneous Gibbs-sampler distributions. Let $\mathcal{M}$ be some set of parameter values θ, and let $\mathcal{P}^k_{\mathcal{M}}$ be the set of all distributions defined by inhomogeneous Gibbs samplers using parameters $\theta^0,\ldots,\theta^k$ from $\mathcal{M}$. Let $\mathcal{P}^k_{\mathcal{M}}(\tilde{\theta}^0,\ldots,\tilde{\theta}^{k-1},*)$ be the set of all distributions defined by inhomogeneous Gibbs samplers using parameters $\theta^0=\tilde{\theta}^0,\ldots,\theta^{k-1}=\tilde{\theta}^{k-1}$ and $\theta^k\in\mathcal{M}$. We have the following claim.

Theorem 1 *Let $\pi\in\mathcal{P}^K_{\mathcal{M}}$, and let dataset D be generated by picking each $\mathbf{x}$ from a fixed distribution, and picking the associated $\mathbf{y}$ according to $\pi(\mathbf{Y}|\mathbf{x})$. Let $\hat{\theta}^0$, ..., $\hat{\theta}^K$ be the parameters learned by idealized stacked graphical learning from dataset D, and consider the limit as $|D|\to\infty$. For all $i: 0\leq i\leq K$,*

$$Q^i(\hat{\theta}^0,\ldots,\hat{\theta}^i) = argmin_{Q\in\mathcal{P}^i_{\mathcal{M}}(\hat{\theta}^0,\ldots,\hat{\theta}^{i-1},*)} KL(Q||\pi)$$

where $KL(Q||P)$ is the KL-divergence of Q and P.

Proof. Every distribution in $\mathcal{P}^i_{\mathcal{M}}(\hat{\theta}^0,\ldots,\hat{\theta}^{i-1},*)$ is defined by an inhomogeneous Gibbs sampler with parameters $\hat{\theta}^0,\ldots,\hat{\theta}^{i-1}$ and some $\theta^i\in\mathcal{M}$. Let $\tilde{\pi}_{i-1}$ denote $Q^i(\hat{\theta}^0,\ldots,\hat{\theta}^{i-1})$, and let $\tilde{\pi}_i\in\mathcal{P}^i_{\mathcal{M}}(\hat{\theta}^0,\ldots,\hat{\theta}^{i-1},*)$. For any such $\tilde{\pi}_i$ and any $\mathbf{x}$, $\tilde{\pi}_i(\mathbf{y}|\mathbf{x})$ is defined by

$$\tilde{\pi}_i(\mathbf{y}|\mathbf{x}) = \tilde{\pi}_{i-1}(\mathbf{y}'|\mathbf{x})\cdot P(\mathbf{y}|\mathbf{x},\mathbf{y}';\theta^i)$$

In the limit as $|D|\to\infty$, minimizing KL-divergence to P is equivalent to maximizing the probability of a dataset drawn from the distribution P. Thus minimizing

$KL(\pi_i||\pi)$ can be accomplished by choosing θ^i to maximize probability of

$$\prod_{(\mathbf{x},\mathbf{y},\mathbf{y}')\in D'} P(\mathbf{y}|\mathbf{x},\mathbf{y}';\theta^i)$$

where $\mathbf{y}$ is drawn from $\pi(\mathbf{Y}|\mathbf{x})$ and $\mathbf{y}'$ is drawn from $\tilde{\pi}_{i-1}(\mathbf{Y}|\mathbf{x})$. Notice that Step 2a of idealized stacked graphical learning constructs the appropriate $\mathbf{y}'$ for $\mathbf{x}$, and Step 2b performs the appropriate optimization.

Notice that idealized stacked graphical learning is nearly identical to the algorithm introduced in this chapter, if we assume that the relational template $C(x_i, \mathbf{y})$ returns $\mathrm{MB}_i(\mathbf{y})$. The main differences are the use of most-likely predictions from cross-validation rather than sampling to produce values $\mathbf{y}'^k_{\mathbf{x}}$, and the use of aggregation functions in our implementation.

2.5 Conclusions

In this chapter, we describe the algorithm of stacked graphical learning, illustrate how it works with the WebKB example, and provide theoretical analysis on an idealized version of the algorithm to show its convergence and better understand its performance.

We formulate the idealized version of stacked graphical learning as an inhomogeneous Gibbs sampling process with parameters learned in a greedy manner. The intuition behind the quick convergence can be understood in the following way: we expand the concept space such that it may be easier for the learning algorithm to produce a nearly correct answer from a rich set of alternatives more quickly than from a small concept set.

Chapter 3

Applications

In this chapter, we discuss the performance of stacked graphical model. We evaluated stacked graphical learning on tasks from several domains, including collective classification over relational datasets, sequential partitioning [5], and named entity extraction.

3.1 Collective Classification Over Relational Datasets

3.1.1 The problems and datasets

One straightforward application of stacked graphical learning is collective classification of relational data. We evaluated stacked graphical learning on three document classification corpus: the WebKB dataset, the Cora dataset, and the Citeseer dataset.

We consider webpage classification on the WebKB dataset [11], which contains webpages from four computer science departments, and paper classification on the Cora dataset and the CiteSeer dataset [39]. The webpages in WebKB dataset were manually labelled with one of six categories: course, faculty, student, staff, research projects, or other. The WebKB data contains 3818 webpages and 8073 hyperlinks.

The relational template applies the COUNT aggregator and returns the number of outgoing and incoming links in each category, given one webpage.

The Cora dataset [42] contains 2708 machine learning papers and citations among them, and each paper was manually labelled into seven topics: Case Based, Genetic Algorithms, Neural Networks, Probabilistic Methods, Reinforcement Learning, Rule Learning, and Theory. The Cora data contains abstracts of 2708 papers and 5429 citations. If paper A cites paper B, we consider there is a link from paper A to paper B. The relational template applies the COUNT aggregator and returns the number of outgoing and incoming links in each category, given one paper.

The Citeseer dataset [22] is another collection of papers from six categories: Agents, Artificial Intelligence, Databases, Human Computer Interaction, Machine Learning, and Information Retrieval. The Citeseer data contains 3312 papers and 4732 citations. The relational template is the same as the template for Cora data and WebKB data, i.e., returning the number of outgoing and incoming links in each category.

For the document classification tasks, the initial feature representation is a bag-of-word feature set giving the counts of each token in a document.

3.1.2 Methods

We use a maximum entropy learner [53] as the base learner in stacked graphical learning for document classification.

Maximum Entropy is widely used for inducing probabilistic classifications. The principle of maximum entropy states that the least biased distribution that encodes certain given information is that which maximizes the information entropy. In a maximum entropy classifier, the probability distribution of a possible class label v

given an instance w, $p(v|w)$, is modeled as follows:

$$P(v|w) = \frac{1}{Z(w)} \exp(\sum_i \lambda_i f_i(w, v))$$

In this definition, usually each feature $f_i(w, v)$is expressed as a binary function based on w and its proposed classification v, λ_i is the corresponding feature weight, and $Z(w)$ is a normalization factor. The unknown parameters λ_i's are usually estimated by maximum likelihood.

We compare our stacked graphical models to two state-of-art statistical relational learning models: relational dependency networks [25] and relational Markov networks [58].

Relational dependency networks (RDNs) [25] are a statistical learning model capable of expressing and reasoning with dependencies in relational datasets. RDNs learn the local conditional probability distribution (CPD) independently using pseudo-likelihood learning techniques. Gibbs sampling is applied to estimate the full joint distribution and extract probabilities of interest in RDNs. Ideally any relational learning approach can be used in a relational dependency network [25]. Here we implement a naive relational method to model the local CPD. Our implementation is a Max-Ent learner with relational features. The features include the local feature for each instance, and a set of relational features calculated with the relational template. In the training set, the relational features are calculated with true values for the related instances. In the test set, relational features are calculated once the predicted labels are updated during Gibbs sampling.

Relational Markov networks (RMNs) [58] are another state-of-art relational graphical model. Relational Markov networks correspond to undirected graphical models. Hence a RMN specifies a set of cliques and potentials between attributes of related entities at a template level. Usually the cliques are instantiated by a certain relational template and the corresponding clique potentials are also in a relational setting up.

Unlike RDNs, there is no pseudo-likelihood technique in RMN and the joint probability has to be computed in a complex way. Loopy belief propagation is usually applied for the inference over RMNs. Our implementation is based on a naive relational potential function: a MaxEnt learner with relational features. Similar to our implementation of RDNs, the features include the local feature for each instance, and a set of relational features calculated with the relational template. In the training set, the relational features are calculated with true values. In the test set, relational features are calculated once the predicted labels are updated during loopy belief propagation.

We also consider a naive relational model. This is similar to stacked graphical learning except that in the naive model the predictions for the training examples are obtained by applying the model learned with all training data instead of applying the cross-validation-like trick.

The last model is an upper-bound for the stacked graphical model: i.e., we use the stacked graphical model but allow $\widehat{\text{true}}$ labels of related instances to be used during the feature extension at both training and testing time. This can not be implemented in practice, but gives some idea of what performance is theoretically achievable for collective classification with our model.

3.1.3 Experimental results

To evaluate the effectiveness of stacked graphical learning for collective classification, we compare five models. The first model is a local model, i.e., the model trained with the base learner. For document classification, the local model is a MaxEnt model.

The second model is a "competitive" collective classification model. For document classification, we compare to relational dependency networks (RDNs) [25] and relational Markov Networks(RMNs). The RDN model uses the same features as the stacked model, but learns via a pseudo-likelihood method and inference with Gibbs

sampling. RMNs also use the same features but performs inference with loopy belief propagation.

The third model is a naive relational model described in Section 3.1.2.

The fourth model is stacked graphical models. We study two cases here, with one and two iterations respectively.

The fifth model is an upper-bound (noted as ceiling model in Table 4.1) for the stacked graphical model as described in Section 3.1.2.

Table 3.1: Evaluation on five models. The accuracy for "Document classification" is reported. We compared stacked graphical model to a local model, other relational models, and its probabilistic ceiling. The local models are MaxEnt models. The competitive relational models are RDN models and RMN models.

	Document classification		
	WebKB	Cora	CiteSeer
Local model	57.6	63.1	54.9
RDNs	**73.1**	72.3	57.9
RMNs	72.9	71.3	56.1
Naive relational model	70.6	65.9	56.3
Stacked model (k=1)	72.9	**73.0**	**59.3**
Stacked model (k=2)	72.1	**73.0**	**59.3**
Ceiling for stacked model	73.4	76.1	61.8

Table 4.1 shows the performance for each of the five models on three real-world datasets. We used 5 fold cross validation for all datasets, except for WebKB data, where we used 4 fold cross validation by departments. We use paired *t*-tests to assess the significance of the changes in accuracy. The t-tests compare the stacked graphical models with k=1 to each of the other four models. The null hypothesis is that there is

no difference in the accuracy of the two models. The differences that are statistically significant at a $p < .05$ level are reported below. Table 3.2 shows the summary of t-test. Each element in Table 3.2 records how many times (out of the three tasks in this section) model i in the row achieves higher accuracy than model j in the column. Detailed t-test evaluation results on each task are described in Appendix A.

On all of the three datasets, stacked graphical learning improves the performance of the base learner statistically significantly. On all the tasks, stacked graphical learning achieves statistically indistinguishable results to the competitive models. On the WebKB dataset, stacked graphical learning achieves comparable results to the ceiling models. Usually there is no significant difference in accuracy for k=1 and k=2 in stacking. We did not include the performance of stacking with more iterations in the table here. Usually with more iterations, the performance of stacking stays at the same level, which suggests that stacking converges very quickly and does not require many iterations. We will study further the convergence rate of stacked graphical learning in Chapter 3.4.

Table 3.2: Summary of evaluation with t-test on collective classification: this table shows how many times model i in the row achieves higher accuracy than model j in the column.

	Local	RDN	RMN	Naive	Stack (k=1)	Stack (k=2)
Local						
RDNs	3		2			
RMNs	3		2			
Naive model	3					
Stacked (k=1)	3	1	1	3		
Stacked (k=2)	3	1	1	3		

3.2 Sequential Partitioning

3.2.1 The problems, datasets, and methods

Sequential partitioning tasks are sequential classification tasks characterized by long runs of identical labels: examples of these tasks include document analysis, video segmentation, and gene finding [5]. In this work we consider three datasets.

The *signature* dataset is originated from the problem of recognizing the "signature" section of an email message. Each line of an email message [3] is labels as either *positive* or *negative*. A *positive* label indicates that a particular line in the message was part of a signature section, and *negative* otherwise. This dataset contains 33,013 labeled lines from 617 email messages. About 10% of the lines are labeled "positive".

One set of tasks involved classifying lines from FAQ documents with labels like "header", "question", "answer", and "trailer". We used the features adopted by McCallum *et al* [40] and the ai-general task adopted by Dietterich *et al* [13]. The data consists of 7 long sequences, each sequence corresponding to a single FAQ document; the task contains 10909 labeled lines. Our current implementation only supports binary labels, so we considered the label "answer" (A) for the FAQ dataset.

Another task was video segmentation task, in which the goal is to take a sequence of video "shots" (a sequence of adjacent frames taken from one camera) and classify them into categories such as "anchor", "news" and "weather". This dataset contains 12 sequences, each corresponding to a single video clip. There are a total of 406 shots, and about 700 features, which are produced by applying LDA to a 5x5, 125-bin RGB color histogram of the central frame of the shot. We constructed a video partitioning task, corresponding to the most common label [5].

We use MaxEnt as the local model, and split the datasets according to sequences. The relational templates returns the predictions of ten adjacent examples (five pre-

ceding examples and five following examples).

We compared to a state-of-art conditional random fields (CRFs) [36] model for tagging sequences. CRFs give the conditional probability of a possible label sequence $\mathbf{y} = (v_1, ..., v_n)$ given the input sequence $\mathbf{x} = (w_1, ..., w_n)$:

$$P(\mathbf{y}|\mathbf{x}) = \frac{\exp(\sum_j \sum_i \lambda_i f_i(v_j, \mathbf{x}, j))}{Z(\mathbf{x})}$$

In this definition, each f_i is a function that measures a feature relating the state v_j at position j with the input sequence around position j, λ_i is the corresponding feature weight, and $Z(\mathrm{x})$ is the normalization factor, usually referred to as the partition function. CRFs are generally expensive computationally during training. Learning CRFs is usually an iterative optimization procedure and in each step it requires to calculate the partition function, which is in general an iterative inference procedure.

The experimental results are summarized in Table 3.3.

Table 3.3: Accuracy comparison of stacked graphical learning for sequential partitioning. The local model is MaxEnt. We compared to conditional random fields (CRFs) and the naive relational model described in section 3.1.

	Sequential Partitioning		
	FAQ	signature	video
Local model	67.3	96.3	80.9
Competitive CRFs model	85.6	**98.1**	83.0
Naive relational model	70.8	96.5	80.9
Standard Stacked model (with MaxEnt, k=1)	**87.1**	**98.1**	**85.8**
Ceiling for stacked model	93.1	99.7	91.3

3.3 Named Entity Extraction

3.3.1 Problems and datasets

We also study collective named entity extraction(NER) with stacked graphical learning. We cast the name extraction task as a binary classification problem, i.e., tokens corresponding to name entities are labeled as "positive" and non-name tokens are labeled as "negtive". One NER problem is protein named entity extraction from Medline abstracts. We used three datasets to evaluate our method for protein name extractions. The University of Texas, Austin dataset contains 748 labeled abstracts[2]; the GENIA dataset contains 2000 labeled abstracts[9]; and the YAPEX dataset contains 200 labeled abstracts[14].

We use conditional random fields as the base learner, as discussed in Section 3.2.1. The feature set contains hand-coded features and part-of-speech (POS) tags. The hand-code features are calculated with rules defined according to protein naming conventions. Details about the hand-coded rules are summarized in Table 3.4. Each rule generates corresponding binary features. For example, one feature associated with the rule of "has suffix -in" is:

$$f_i(x,y) = \begin{cases} 1 & if\ current\ token\ ends\ with\ -in\ and\ x\ is\ labelled\ as\ protein \\ 0 & otherwise \end{cases}$$

The relational template will retrieve the predictions for the nearby words (with window size 3) and for the same word appearing in one abstract, apply the COUNT aggregator, and return the number of words in each category, given one word. For example, let w_i be the word in a document. For words $w_j = w_i$ in the same document, we count the number of times w_j appearing with label y and use it as one of the stacked features for w_i. In the experimental part, we will also explore another simpler relational template as a comparison.

Table 3.4: Hand-coded rules to generate features for protein name extraction.

Rules/templates	Example token
Initial Caps	There
All Caps	CRF
Caps mix-case 1	SthSth
Caps mix-case 2	sthSthsth
Caps mix-case 3	sthSth
Caps mix-case 4	SthsthSth
Char digit mix 1	Sth-123
Char digit mix 2	Sth-123-Sth
Greek letter	alpha
Suffix	-in, -ase
Single Digit	1
Double Digit	12
More Digits	123
Caps+Punctuation1	Aname,sth
Caps+Punctuation2	aname,sth

We also studied person name extraction for the CSpace email corpus. The CSpace corpus we used contains 216 email messages collected from a management course at Carnegie Mellon University [45]. We use conditional random fields as the base learner and the feature set described in the previous work [7], which includes a history of length one, plus the lower-cased value of the token, letter cases, and letter-case patterns for all tokens in a window of size three centered at the $i-th$ token. The relational template is the same as the template for protein name extraction.

3.3.2 Experimental results

Five models are evaluated for the NER tasks. The first model, a local model, is a CRF model.

The second model, a competitive model, is a stacked sequential model [5]. Stacked sequential models are similar to stacked graphical models, except that the model can only retrieve dependencies along the sequence, i.e., dependencies among adjacent words in a sentence. Here we use a stacked CRF model.

The third model is the naive relational model, with an advance relational template as described in the next paragraph. The fourth model is stacked graphical models with one iteration, but different relational templates. The fifth model is a probabilistic upper-bound (noted as ceiling model in Table 3.5) for the stacked graphical model.

As shown in Table 3.5, we explored different relational templates for NER. A simple relational template will just retrieve the predictions for the adjacent words (with window size 5), denoted as *sequential stacking* in Table 3.5. Another more "advanced" relational template will retrieve the predictions for the adjacent words (with window size 5) and for the same word appearing in one document, apply the COUNT aggregator, and return the number of words in each category, given one word. In Table 3.5 we denote this relational template as *relational stacking*.

Table 3.5 shows the F1 for each of the five models on NER task. We used 5 fold cross validation and t-tests to evaluate our results. Again, we observe that stacked graphical learning improves the performance of the base learner significantly and is competitive to the state-of-art models.

Table 3.6 shows the summary of t-test for NER tasks. Each element in Table 3.6 records how many times (out of the three tasks in this section) model i in the row achieves higher accuracy than model j in the column. Detailed t-test evaluation

Table 3.5: Evaluation on five models. The F1-measure for NER is reported. We compared stacked graphical models (with different relational templates) to a local model, other relational models, and its probabilistic ceiling. The local models are CRFs. The competitive relational models are stacked CRFs.

	Protein NER			Email NER
	UT	Yapex	Genia	CSpace
Local CRF model	73.1	65.7	72.0	80.3
Stacked sequential CRFs	76.8	66.8	77.1	81.2
Naive relational models	74.5	67.3	77.0	82.3
Stacked model (sequential stacking)	76.5	66.9	77.1	81.4
Stacked model (relational stacking)	**78.3**	**69.3**	**77.9**	**82.5**
Ceiling for stacked model	80.5	70.5	80.3	84.6

results on NER task are described in Appendix A.

There are usually two ways to improve the performance of a classifier: 1. feature engineering, or 2. a better (usually more sophisticated) model. In the NER applications, we demonstrate that stacked graphical learning can easily do feature engineering to improve the performance. In the next chapter, we will show that stacked graphical learning with simple features can achieve performance competitive to a model with highly engineered features.

3.4 Inference Efficiency of Stacked Graphical Learning

In this section, we study the inference efficiency of stacked graphical models. We evaluate the inference of standard stacked graphical learning and show that stacked

Table 3.6: Summary of evaluation with t-test for named entity extraction: this table shows how many times model i in the row achieves higher accuracy than model j in the column. "Stack (seq.)" denotes the stacked model with *sequential* relational template, "Stack (rel.)" denotes the stacked model with *relational* relational template.

	CRFs	Stacked CRFs	Naive (rel.)	Stack (seq.)	Stack (rel.)
CRFs					
Stacked CRFs	4		1		
Naive model (rel.)	4	2		2	
Stacked (seq.)	4			3	
Stacked (rel.)	4		2	3	

graphical learning is 40 to 80 times faster than Gibbs sampling during inference.

We compared the performance and computational cost of inference in stacked graphical model with one iteration to that of Gibbs sampling in RDNs with 50 iterations and 100 iterations, evaluating on the SLIF problem and the document classification problems. Table 3.7 shows the speedup - in the table "39.6" means the CPU time used by Gibbs sampling is 39.6 times that used by the inference of stacked graphical learning, i.e., the inference in stacked graphical learning is 39.6 times faster than Gibbs sampling. If the accuracy of stacked graphical learning is statistically significantly better than the accuracy of Gibbs sampling, there is a "+" marked by the number indicating the speedup; otherwise there is a "-" mark. If there is no significant difference, there is no mark.

Table 3.7 shows that compared to Gibbs sampling with 50 iterations, stacked graphical learning generally achieves better accuracy but is about 40 times faster during inference. Compared to Gibbs sampling with 100 iterations, stacked graphical learning can achieve competitive or better accuracy but is more than 80 times faster

during inference.

Here we also include the experimental results on SLIF Task 1 - more details about the dataset and the collective classification task are described in the next Chapter.

Table 3.7: Comparison on performance and efficiency. "39.6" means that the inference in stacked graphical learning is 39.6 times faster than Gibbs sampling. "+" means that the accuracy of stacked graphical learning is statistically significantly better than the accuracy of Gibbs sampling.

	Gibbs 50	Gibbs 100
SLIF Task 1	39.6^{+}	79.3^{+}
WebKB	43.4^{+}	87.0
Cora	42.7^{+}	85.4
Citeseer	43.6^{+}	87.3
Average speed-up	42.3	84.8

Figure 3.1 and Figure 3.2 show the convergence rate of stacking compared to Gibbs sampling on RDNs. The plots were generated using SLIF Task 1 data and the document classification datasets: the WekKB dataset, the Cora dataset, and the Citeseer dataset, for the collective classification task. We created the plots with a natural logarithm of k, where $ln(k) = -1$ corresponds to $k = 0$, $ln(k) = 0$ corresponds to $k = 1$, and $ln(k) = 1$ corresponds to $k = e$. In addition to the Gibbs sampling with random starting points, we also evaluated Gibbs sampling starting with same $\mathbf{y}^0$ as the corresponding stacked graphical models.

We observe that stacked models converge more quickly than Gibbs sampling and achieve a satisfactory performance much faster, even if the Gibbs sampling starts with same $\mathbf{y}^0$ as the corresponding stacked graphical models. Stacked graphical models can achieve significant improvement over the base learner after the first iteration. More

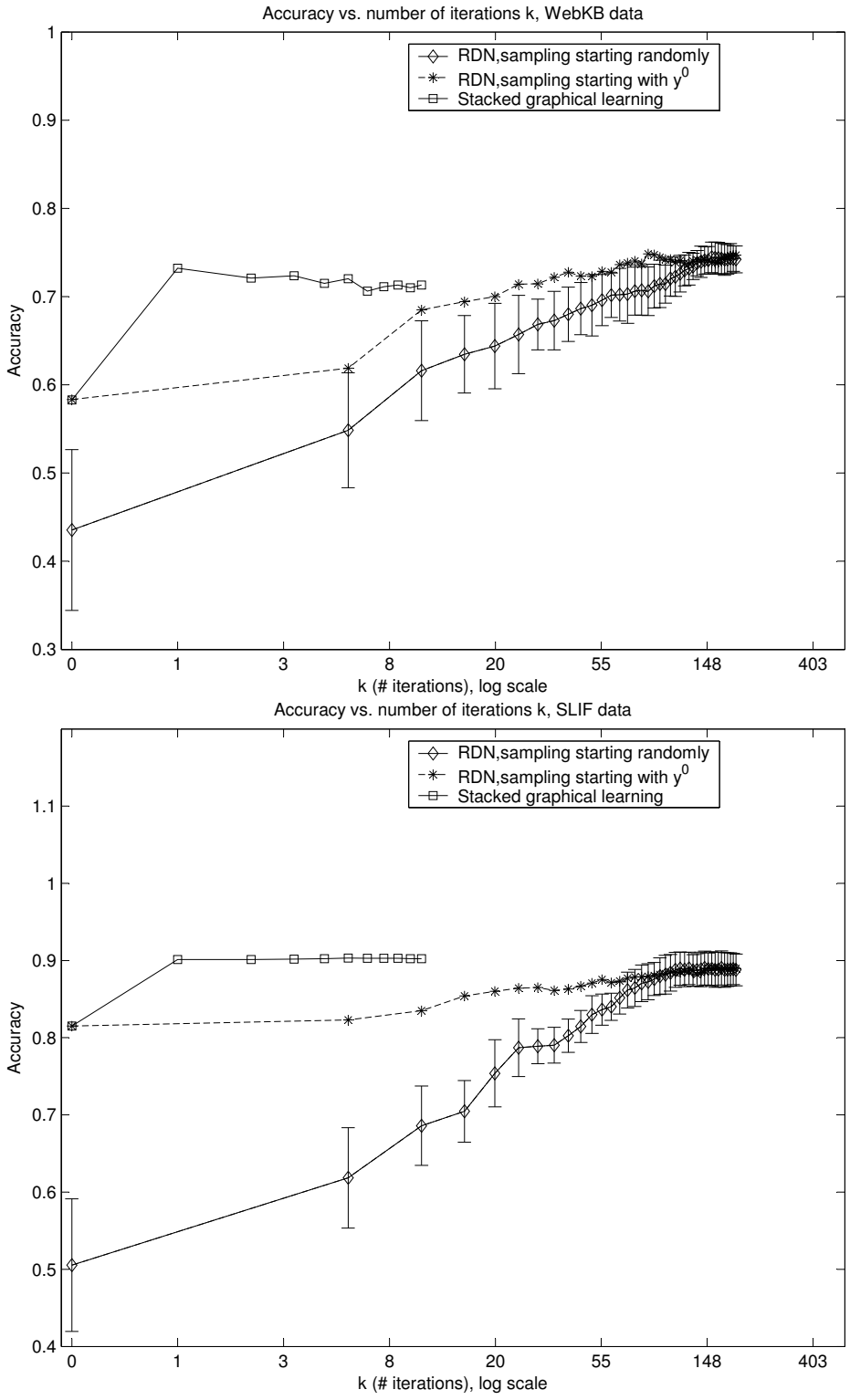

Figure 3.1: Convergence rate of stacking and Gibbs sampling, on SLIF Task 1 and WebKB task

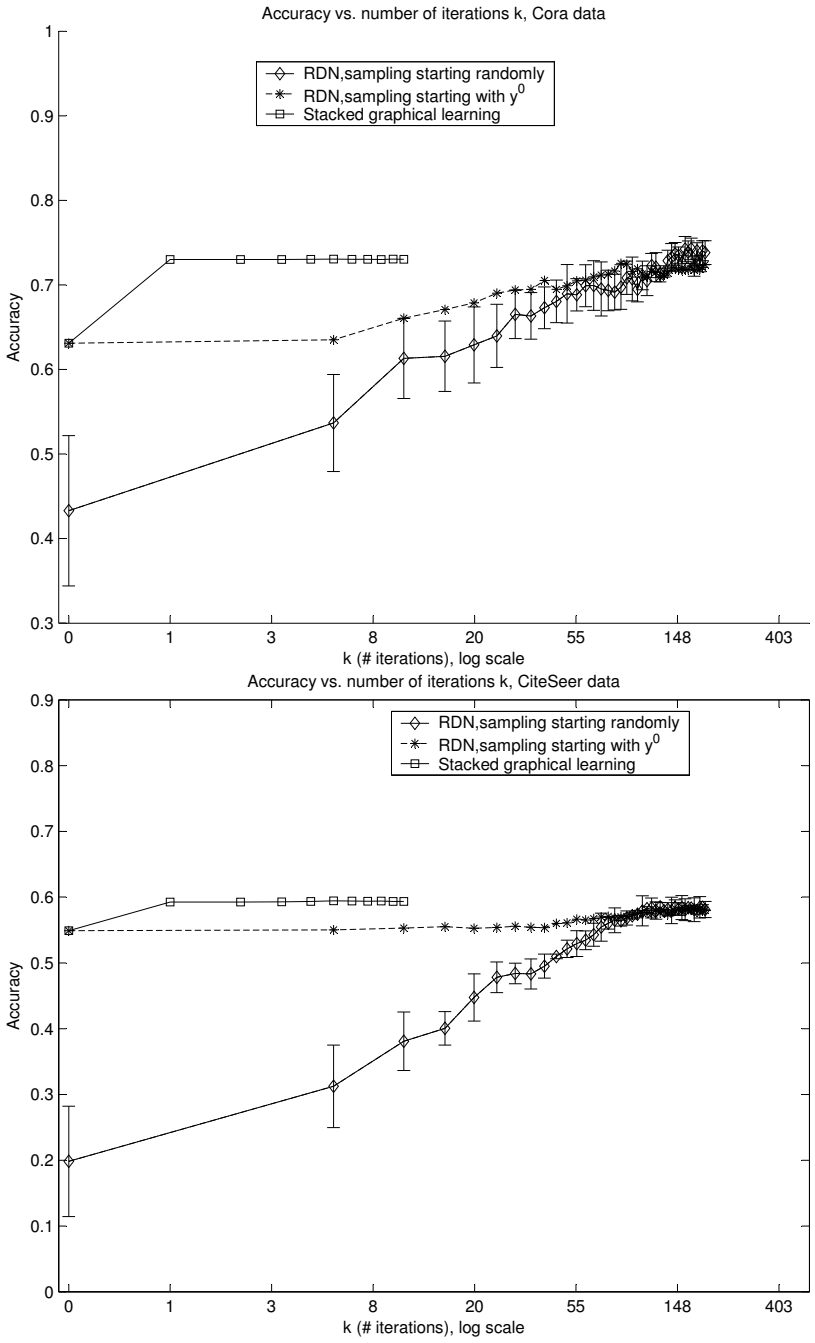

Figure 3.2: Convergence rate of stacking and Gibbs sampling, on Cora and Citeseer datasets

iterations of stacking do not seem to be more helpful, with the performance staying at about the same level. We observe that Gibbs sampling converges to a same level after many more iterations and the convergence rate when k is small depends heavily on the starting points. We plot error bars along the curve for Gibbs sampling with random starting points. The error bars are calculated over 5 randomly initial samples, i.e., in each fold, Gibbs sampling is run 5 times with random initials.

3.5 A further approximation to the stacked model

In this section, we study a further approximation to stacked graphical models. As discussed in Chapter 2, in stacked graphical models, parameters $\theta^0, \theta^1, \ldots, \theta^K$ are learned in a greedy fashion to sample $\mathbf{y}^1$, $\mathbf{y}^2$, $\ldots$, $\mathbf{y}^K$. A further approximation is to set $\mathbf{y}^1$, $\mathbf{y}^2$, $\ldots$, $\mathbf{y}^K$ to all be equivalent, and all equal to $\mathbf{y}^0$. This approximation is conceptually useful, as it allows us to "unroll" the functions defined by θ^1, θ^2, $\ldots$, θ^K, and enumerate the features that contribute to each. Let MB_i^2 be the set of all variables Y_ℓ such that $Y_\ell \in \mathrm{MB}_j$ for some $Y_j \in \mathrm{MB}_i$—i.e., MB_i^2 is the Markov blanket of the Markov blanket of Y_i. Likewise, let MB_i^k be the "order k Markov blanket of i", or set of all variables Y_ℓ such that $Y_\ell \in \mathrm{MB}_j$ for some $Y_j \in \mathrm{MB}_i^{k-1}$. Intuitively, MB_i^K is a diameter-K subgraph centered around Y_i—a sort of generalized "sliding window".

Letting $\tilde{Q}^k$ be approximation to Q^k (recall that Q^k is the distribution over Y^k defined by the inhomogeneous Markov process) that is obtained in this way, it is easy

to see that

$$\begin{array}{lll} \tilde{Q}^1(Y_i = y_i|\mathbf{x}) & \text{depends only on the values of} & \mathbf{x}, y_i, \mathrm{MB}_i(\mathbf{y}^0) \\ \tilde{Q}^2(Y_i = y_i|\mathbf{x}) & \text{depends only on the values of} & \mathbf{x}, y_i, \mathrm{MB}_i^2(\mathbf{y}^0) \\ \ldots & \ldots & \ldots \\ \tilde{Q}^K(Y_i = y_i|\mathbf{x}) & \text{depends only on the values of} & \mathbf{x}, y_i, \mathrm{MB}_i^K(\mathbf{y}^0) \end{array}$$

This suggests a new approximate learning method, in which we set θ^0 as in the greedy method above; construct, in some way, an expanded set of features that relate the value of y_i to the values of $\mathbf{x}$ and $\mathrm{MB}_i^K(\mathbf{y^0})$; and finally find parameters $\tilde{\theta}$ that are MAP estimates for the dataset D using these features. The further approximation suggests that a multi-stage stacking is related to a single-stage stacking with features from an order-k "window". Therefore this approximation leads to an inhomogeneous chain that is only two steps long.

In Cohen and Carvalho's work [5], this sort of approximation was found to be effective for certain sequential classification problems—a special case of the collective classification task considered here in which the Markov network is a linear chain. However, there are reasons to believe that this "window" approximation will be less appropriate for general graphs. Consider a simple case, where the expanded feature set includes only a single edge between y_i and each $y_j \in \mathrm{MB}_i$, the Markov network has a maximum clique size of 2, and every $|\mathrm{MB}_i|$ is bounded by some small constant b. Let n be the number of parameters in the model θ^+. It is easy to see that the number of features used in this "unrolled" 2-step chain can grow rapidly with K: because $|\mathrm{MB}_i^K|$ can grow as b^K, the 2-step chain can have up to $n \cdot b^K$ parameters. This means that learning and evaluating the classifiers used in the second step of the chain will be expensive, and that the learner will be prone to overfit. While with linear chains, such as sequential classification problems, MB_i^K contains only $O(K)$ variables, which is probably why overfitting is less of a problem in [5].

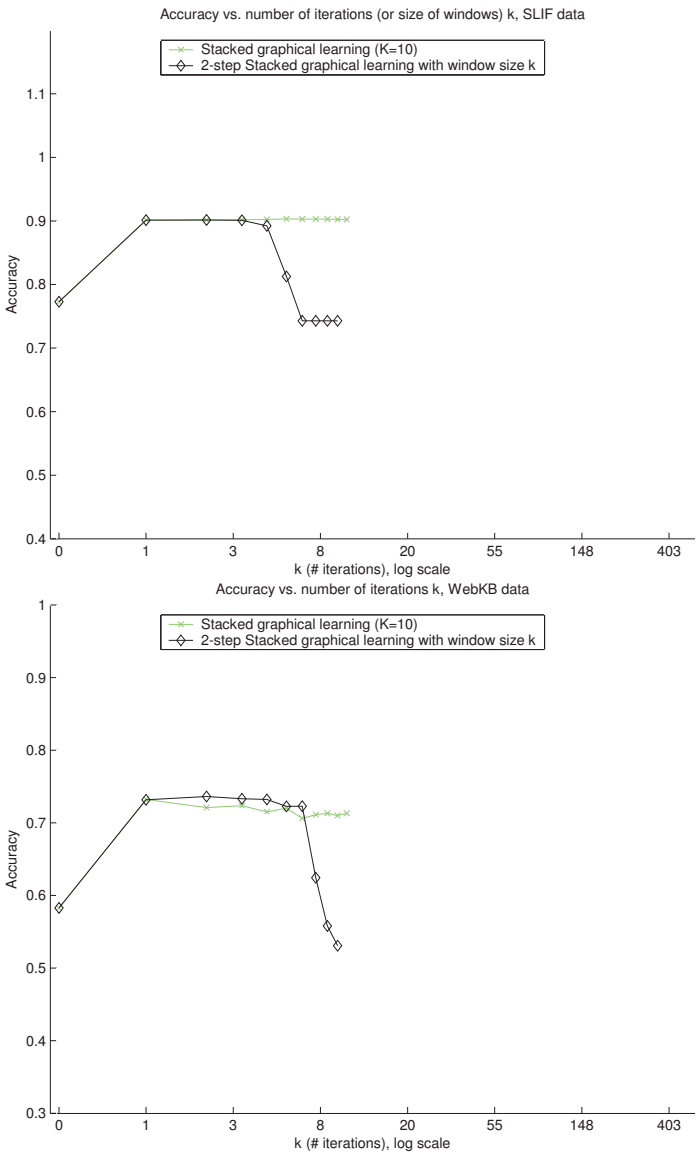

Figure 3.3: Performance of the further approximation

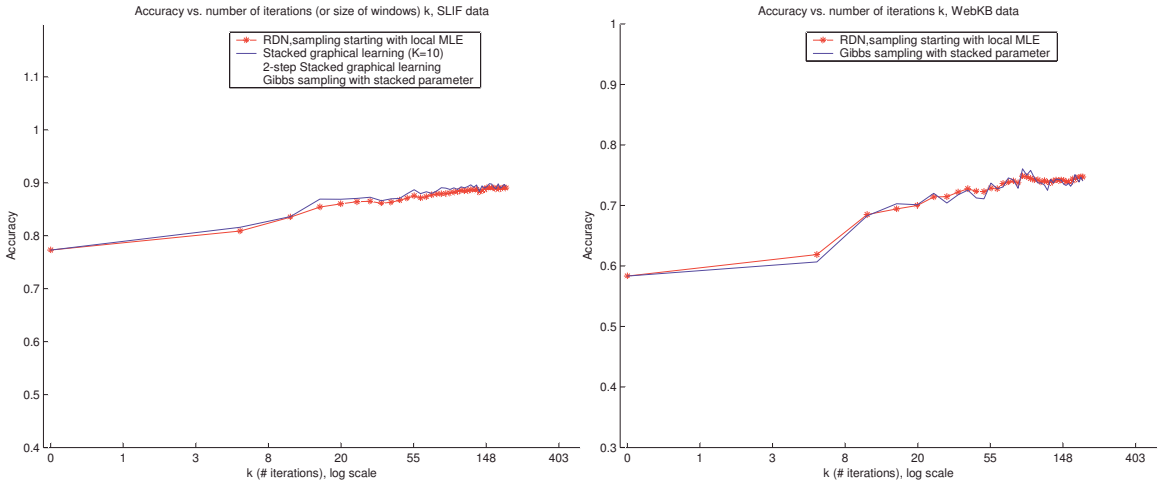

Figure 3.4: Performance of the alternative Gibbs sampling methods

Under the same assumptions, the method of stacked graphical models will use only $n \cdot K$ parameters: even though $Q^K(Y_i = y_i|\mathbf{x})$ also depends on the values of $\mathbf{x}, y_i, \mathrm{MB}_i^K(\mathbf{y}^0)$, this dependency is "funnelled through" small Markov blankets involving variables $\mathbf{y}^1, \ldots, \mathbf{y}^{K-1}$.

We evaluated this approximation with the SLIF data and WebKb data. The further approximation suggests that a multi-stage stacking is related to a single-stage stacking with features from an order-k "window". Figure 3.3 shows that this is true when k is small, while the two-step approximation tends to overfit when the window size k grows (black curves).

3.6 An alternative Gibbs sampler

Viewed from a high level, relational dependency networks use pseudo-likelihood learning and Gibbs sampling and stacked graphical models use conditional learning and very few iterations of stacking. In this section, we explore an alternative strategy, where the parameters learned via conditional learning (level 1 of the stacked model)

is used during Gibbs sampling, i.e., the parameters learned in stacked models and Gibbs sampling are enforced together. We evaluated this idea with SLIF task 1 data and WebKb data, the results are shown in Figure 3.4.

We observe that the performance of this alternative strategy tracks the Gibbs sampler in relational dependency networks closely - i.e., unlike the previously-described version stacking, this variant does not improve over pseudo-likelihood plus Gibbs sampling. Further, the normal Gibbs sampler tends to converge after about 100 iterations, while the Gibbs sampler with "stacked parameters" seems to take longer to converge.

3.7 Conclusions

In this chapter, we evaluated stacked graphical models on tasks from several domains and demonstrate the accuracy and inference efficiency of stacked graphical models. Our experiments show that stacked graphical models can achieve competitive accuracy compared to state-of-art relational learning models, via much less inference time.

We also discussed a further approximation to stacked models, where a single-stage stacking with features from an order-k "window" is used instead of a multi-stage stacking. Our study suggested that the two-step approximation tends to overfit when the window size k grows. An alternative Gibbs sampling procedure with parameters learned from stacked graphical models is also studied. This Gibbs sampler tends to track the ordinary Gibbs sampler, yet harder to converge.

Chapter 4

In-depth Case Studies

In this chapter, we study four applications in more depth and demonstrate more strengths of stacked graphical learning.

4.1 SLIF Task 1: Collective Classification of Panel Labels

4.1.1 SLIF overview

Subcellular Location Image Finder (SLIF) is a system which extracts information from both figures and the associated captions in biological journal articles [33, 47, 48]. SLIF applies both image analysis and text interpretation to figures. Figure 4.1 and Figure 4.2 are typical figures that SLIF can analyse. SLIF is an information extraction system involving relational data and many inter-related tasks.

Figure 4.3 shows an overview of the steps in SLIF system with references to publications in which they are described in more details.

SLIF analyses both images and text. Image processing includes several steps:

Fig. 5. Double immunofluorescence confocal microscopy using mouse mAb against cPABP and affinity-purified rabbit antibodies against mrnp 41. Methanol-permeabilized and fixed HeLa cells were incubated with affinity-purified rabbit anti-mrnp 41 antibodies (a) and with monoclonal anti-cPAPB antibodies (b), and the bound antibodies were visualized with fluorescently labeled secondary antibodies. (Bar = 10 um.)

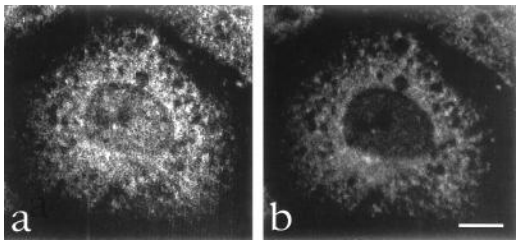

Figure 4.1: A figure caption pair reproduced from the biomedical literature.

Fig. 1. Kinase inactive Plk inhibits Golgi fragmentation by mitotic cytosol. (A) NRK cells were grown on coverslips and treated with 2mMthymidine for 8 to 14 h. Cells were subsequently permeabilized with digitonin, washed with 1M KCl-containing buffer, and incubated with either 7 mgyml interphase cytosol (IE), 7mgyml mitotic extract (ME), or mitotic extract to which 20 mgyml kinase inactive Plk (ME + Plk-KD) was added. After a 60-min incubation at 32C, cells were fixed and stained with anti-mannosidase II antibody to visualize the Golgi apparatus by fluorescence microscopy. (B) Percentage of cells with fragmented Golgi after incubation with mitotic extract (ME) in the absence or the presence of kinase inactive Plk (ME + Plk-KD). The histogram represents the average of four independent experiments.

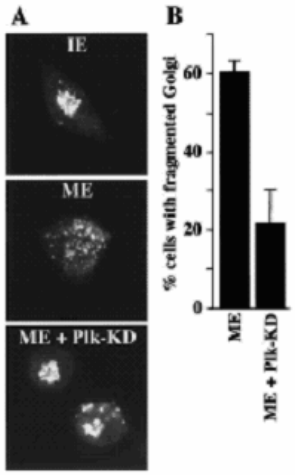

Figure 4.2: Another figure caption pair reproduced from the biomedical literature.

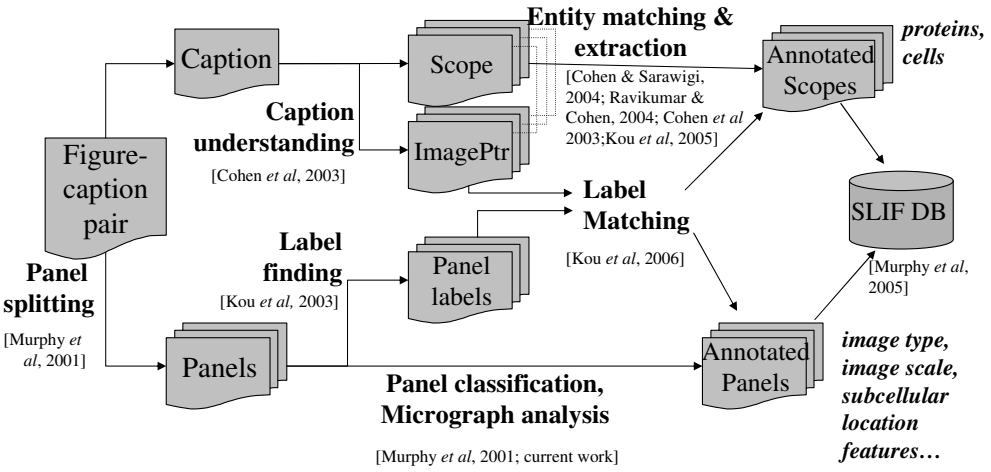

Figure 4.3: Overview of the image and text processing steps in SLIF.

Decomposing images into panels. For images containing multiple *panels* (independently meaningful sub-figures), the individual panels are recovered from the image.

Identifying fluorescence microscope images. Panels are classified as whether they are fluorescence microscope images, so that appropriate image processing steps can be performed.

Image preprocessing and feature computations. Firstly the annotations such as labels, arrows and indicators of scale contained within the image are detected, analyzed, and then removed from the image. In this step, *panel labels* are recognized to by Optical Character Recognition (OCR). Panel labels are textual labels which appear as annotations to images, for example, "a" and "b" printed in panels in Figure 4.4. Recognizing panel labels is very challenging. Even after careful image preprocessing and enhancement the F1 accuracy is only about 75%. The OCR results are used as *candidate panel labels* and a set of heuristic rules are applied to filter out some candidates [33]. After filtering candidates an F1 of 78% is obtained.

Secondly, the *scale bar* is extracted. Automated analysis of fluorescence microscope images requires knowing the scale of an image since some of the previously

developed subcellular location features (SLF) directly depend on the scale of the images. Microscope images published in journals often have scale information included in the figures. The images usually contain a *scale bar* and the respective caption contains the number of micrometers corresponding to the scale bar. Imaging processing techniques are used to locate the scale bar associated with a panel. The size of the scale bar is extracted from the accompanying caption. Scale bar extraction is currently done with a precision of 76% and a recall of 50%.

Finally subcellular location features (SLFs) are produced and the localization pattern of each cell is determined.

Caption Processing is done as follows.

Entity name extraction. In the current version of SLIF we use an extractor trained on conditional random fields and an extractor trained on Dictionary-HMMs to extract protein names [34].

Image pointer extraction. The linkage between the panels and the text of captions is usually based on textual labels which appear as annotations to the images (i.e., panel labels), and which are also interspersed with the caption text. We call these textual labels appearing in text *image pointers*. In the example shown in Figure 4.1, "(a)" appearing in the sentence "... 41 antibodies (a) and with monoclonal ..." is an image pointer. In our analysis, image pointers are classified into four categories according to their linguistic function: *Bullet-style image pointers*, *NP-style image pointers*, *Citation-style image pointers*, and *other*. Bullet-style image pointers function as compressed versions of bulleted lists; the strings "(A)" and "(B)" in Figure 4.2 are bullet-style image pointers. NP-style image pointers are used as proper names in grammatical text; an example is the string "(A)" in the text: "Following a procedure similar to that used in (A), ...". Citation-style image pointers are interspersed with grammatical caption text, in the same manner that bibliography

citations are interspersed with ordinary text; the remaining image pointers in Figure 4.2 are citation-style. The image-pointer extraction and classification steps are done via a machine learning method [8].

Entity to image pointer alignment The *scope* of an image pointer specifies what text should be associated with that image pointer. The *scope* of an image pointer is the section of text (sub-caption) that should be associated with it. The scope is determined by the class assigned to an image pointer [8]. For example, the scope of a NP-style image pointer is the set of words that (grammatically) modify the proper noun it serves as. The scope of a bullet-style image pointer is all the text between it and the next bullet-style image pointer. The scope of a citation-style image pointer is some sequence of tokens around the image pointer, usually corresponding to a nearby noun phrase.

4.1.2 Collective classification in SLIF

The task of text region detection in SLIF is a collective classification problem. Usually there are multiple panels within one figure. Finding the text regions, i.e., the regions in panels containing their labels, is one important task in SLIF. The problem studied here is to classify if the candidate regions found via image processing are text regions or not. The text region detection dataset contains candidate regions found in 1396 panels from 207 figures. The dataset contains 4129 connections among the examples. The problem studied here is to classify if the candidate regions are text regions or not.

There are dependencies among the locations of candidate regions. Intuitively, if after image processing a candidate text region was found at the upper-left corner of panel B and two candidate regions were found in panel A, one located at the upper-left corner, another in the middle, it is more likely the candidate region at the upper-left

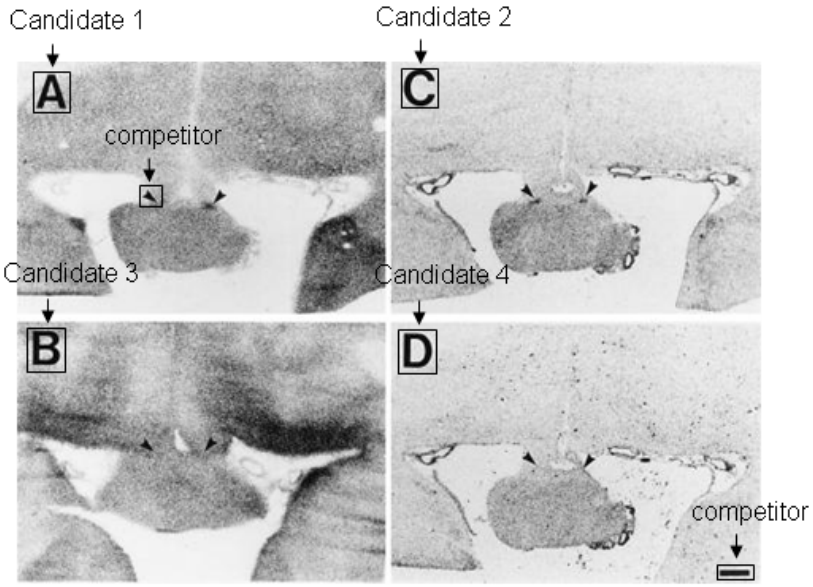

Figure 4.4: An example figure in SLIF

of panel A is the real text region. We define the *neighbor* of a candidate text region to be the region located in the "same" position (i.e., the regions with same the x-y coordinates in panels) in adjacent panels in the same figure and consider the neighbors on four directions - left, right, upper, and lower. We also consider the dependency among candidate regions within the same panel, called *competitors*. Figure 4.4 is an example figure in SLIF which demonstrates candidate regions, neighbors and competitors.

The relational template returns the predictions on one candidate region's neighbors and competitors. Since one candidate region can have several competitors from the same panel, we apply an EXISTS aggregator to the competitors, i.e., as long as there is one competitor which is predicted to be a text region, we assign 1 to the corresponding feature added during stacking. We use a maximum entropy learner as the base learner. The features for the base learner are obtained via image processing and contain binary features indicating whether Optical Character Recognition(OCR) extracts a character or not from the candidate region and its neighbors.

Figure 4.5 shows the dependency among instances in SLIF Task 1. The horizontal

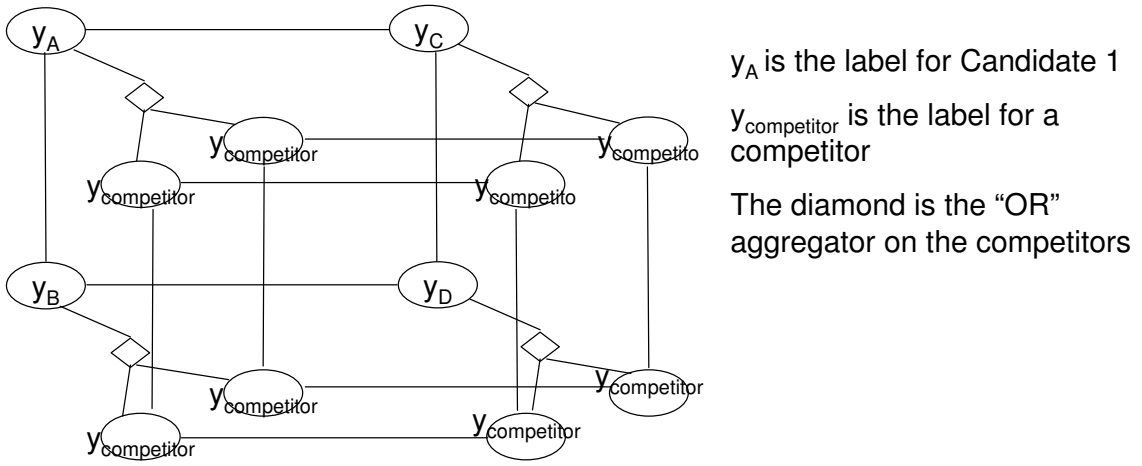

Figure 4.5: A graph for dependencies in SLIF Task 1

and vertical edges represent the neighboring information along the horizontal and vertical direction respectively. The diamond represents the EXISTS aggregator on the competitor.

4.1.3 Experimental results

Accuracy of Stacked Graphical Learning

Similar to the previous chapter, we compare five models. The competitive models here are RDNs and RMNs. The local model is a MaxEnt model.

Table 4.1 shows that stacked graphical learning achieves the best performance on text region detection.

Because of the practical importance of this problem to our work and our familiarity with the problem, in addition to the standard setup for stacked graphical learning, we explored more settings to study more aspects of stacked graphical learning, in particular, interaction between stacked graphical learning and feature engineering.

Table 4.1: Evaluation on six models. We compared stacked graphical model to a local model, another relational model, and its probabilistic ceiling. The local models for "SLIF" is MaxEnt. The competitive relational model is RDNs.

	SLIF
Local model	81.5
RDNs	86.7
RMNs	87.1
Naive relational model	83.9
Stacked model (k=1)	**90.1**
Stacked model (k=2)	**90.1**
Ceiling for stacked model	96.3

Improvements to stacked graphical learning

We can refine the model by passing a numeric value indicating the probability of being that class – the predicted confidence of being that class. The results of using the confidence as stacked features are shown in Table 4.2. Adding the confidence instead of the class labels gave an improvement to the performance of the stacked model. Yet we keep passing the class labels as the default setting and the results presented are based on using the predicted class labels if not specified.

	F1(%)
With class labels as stacked features	92.1
With confidence as stacked features	93.7(+1.6)

Table 4.2: Using confidence from local models as stacked features.

We noticed that the text region detection dataset contains many disjoint subgraphs in it, i.e., the information from every figure is disjoint from other figures' information.

	F1 (%)	
	Local model	Stacked model
Data splitting by figure	81.5	92.1 (+10.6)
Random data splitting	81.5	85.3 (+3.8)

Table 4.3: Performance of a stacked graphical model with random data splitting.

This characteristic was used to split the dataset during training for the stacked model. If we split the dataset totally randomly, and do not keep the text regions from one figure in the same subset, we obtained the result shown in Table 4.3.

Table 4.3 shows that the dependency among the data is very important in a stacked graphical model. When splitting the dataset, a random splitting might break the relevant instances into disjoint subsets. Therefore some relevant stacked features are missed and the performance of the stacked model drops.

Sensitivity to the feature set

Besides the output of GOCR, we tried to include more features, such as the region size, the mean of the gray values, the standard deviation of gray values, whether the candidate region is considered clean or not (based on whether binarization is needed before running GOCR). Four additional features were added to each candidate region and its neighbors and now the feature vector contains 26 features. The values of size and gray values are continuous and in a wide scale compared to the binary features.

Directly adding the additional features helped the local model yet did not help the stacked model, as shown in Table 4.4. Therefore we applied discretization by introducing 10 binary features for each continuous feature: we divide the values of a feature into 10 bins and the 10 binary features indicate into which bin the feature

	F1(%)	
	Local model	Stacked model
With basic features	81.5	92.1 (+10.6)
With extended features		
Before discretization	89.3	89.8 (+0.5)
After discretization	89.3	91.2 (+0.9)
After normalization (to [0,1])	88.2	91.6 (+3.4)
After feature selection (top 6)	89.5	91.9 (+2.4)

Table 4.4: Performance of a stacked graphical model based on a MexEnt Learner with expanded feature set (here we used the predicted class labels as stacked features).

falls. For example, the range of mean gray values might be (12∼245). For a mean gray value of 240, we get a vector of (0, 0, 0, 0, 0, 0, 0, 0, 0, 1) and append it to the feature vector. The performance on the expanded feature sets is summarized in Table 4.4. Normalization and feature selection based on information gains were also applied to observe the influence of feature engineering.

Table 4.4 shows that there is a complex relationship between stacked models and feature engineering. The stacked model always performs better than the local one; but unlike the local model, the performance of the stacked model degrades when more features are added directly– even features that improve the local model can degrade the stacked model. One conjecture is that MaxEnt degrades with many redundant features, and new features are redundant for stacked learning. Thus, while stacking improves performance, the feature engineering is still important. Simply putting the features together to get an extended instance might not as work well as choosing them carefully.

One possible reason for the necessity of feature engineering could be that the base

	F1 (%)	
	Local model	Stacked model
With basic GOCR features	82.3	91.4
With expanded features	88.7	91.0

Table 4.5: Performance of a stacked graphical model based on a boosted decision tree learner.

	F1 (%)	
	Local model	Stacked model
With basic GOCR features	81.5	91.4
With circular features	85.0	93.1

Table 4.6: Performance of a stacked graphical model with circular features.

learner we used, the MaxEnt learner, is biased to give equal or nearly equal importance to all features, and also sensitive to feature scales. Therefore when there are more features, the added stacked features tend to play a weaker role in the classification. We also applied a boosted decision tree as the base learner, and achieved similar performance with the basic GOCR feature set and the expanded feature set without feature engineering, as shown in Table 4.5. The performance of a decision tree is not affected by the scale of features. Table 4.5 shows that without any feature engineering, the performance of the stacked model based on a boosted decision tree does not change when adding more features, while the local model achieves a better performance with more features.

It suggests that the stacked model can achieve a satisfactory performance even with fewer features. Also, when there are more features, stacking becomes sensitive to the feature selection and the base learner selectioin.

As another experiment in feature engineering, we also defined the neighbor of a candidate region circularly, for example, the Candidate 2 in Figure 3 is also the "left" neighbor of Candidate 1, and the Candidate 3 is also the "upper" neighbor of Candidate 1. With these features, the stacked graphical model achieves better performance, as shown in Table 4.6. This suggests that a relational template that gives a more uniform set of features might be better than one with a feature set that varies.

4.2 SLIF Task 2: Application in SLIF as a Multi-task System

The original idea of stacking is to take advantage of the dependencies among instances. In many systems, there are several learned modules that interact with each other, for example, extracting named entities from emails and recognizing the request type of the email are closely related. SLIF is such an information extraction system with many inter-related subtasks. We applied the idea of stacking to SLIF as a multi-task system. Without the stacking idea, each subtask was accomplished independently. After applying the idea of stacking to SLIF system, we will use the predictions of relevant subtasks as features for the other subtask.

In our application to SLIF system, we conjecture the panel label extraction and image pointer extraction are inter-related, and design a stacked model that combines them.

4.2.1 A stacked model for mapping

In the previous version of SLIF, we map panel labels to image pointers by finding the equal-value pair. Below we first review the previously implemented models for

solving these subtasks. Then we apply the idea of stacked graphical models to map panel labels to image pointers.

In SLIF the image pointer finding was done as follows. Most image pointers are parenthesized, and relatively short. We thus hand-coded an extractor that finds all parenthesized expressions that are (a) less than 15 characters long and (b) do not contain a nested parenthesized expression. We replace X-Y constructs with the equivalent complete sequence. (E.g., constructs like "B-D" are replaced with "B,C,D".) We call the image pointers extracted by this hand-coded approach *candidate image pointers*. The hand-coded extractor has high recall but only moderate precision. Using a classifier trained with machine learning approaches, we then classify the candidate image pointers as *bullet-style, citation-style, NP-style, or other*. Image pointers classified as "other" are discarded, which compensates for the relatively low precision of the hand-coded extractor[8].

In SLIF the panel label extraction was done as follows. Image processing techniques and OCR techniques are applied to find the labels printed within the panel. That is, first *candidate text regions* are computed via image processing techniques, and OCR is run on these candidate regions to get *candidate panel labels*. This approach has a relatively high precision yet low recall. We call the panel labels recognized by image processing and OCR *candidate panel labels*. Then *grid analysis*, i.e., finding the number of total columns and rows, and also determine the row and column position of each panel, is applied to figure out how the panels are ranged. For example, in Figure 4.1, the grid contains 1 row and 2 columns. Panel "a" is located in row 1 and column 1. A strategy based on *grid analysis* is applied to the candidate panel labels to get a better accuracy [33].

The task of matching panels labels to image pointers can be formulated as a classification problem. We construct a set of pairs $< o_i, p_j >$ for all candidate panel

labels o_i's and candidate image pointers p_j's from the same figure. That is, for a panel with l_i representing the real label, o_i representing the panel label recognized by OCR, and p_j's representing the image pointers in the same figure, we construct a set of pairs $< o_i, p_j >$. We label the pair $< o_i, p_j >$ as positive only if $l_i = p_j$, otherwise negative.

For example, in Figure 4.1, the real label l_i for panel a is "a". If OCR recognizes o_i where o_i ="a", image pointers for the figure are "a" and "b", we construct two pairs, $< a, a >$ labelled as *positive* and $< a, b >$ labeled as *negative*. Please notice that the pair is labelled according to the real label and the image pointers. If unfortunately, OCR recognizes o_i incorrectly for panel a in Figure 4.1, for example o_i ="o", we have two pairs, $< o, a >$ stilled labelled as positive (while the features for this pair will be different, as described in the next paragraph) and $< o, b >$ labeled as negative.

We designed features based on the o_i's and p_j's. In the base feature set, there are 3 binary features: one boolean value indicating whether $o_i = p_j$, one boolean value indicating whether $o_{i\_left} = p_j - 1$ or $o_{i\_upper} = p_j - 1$, and another boolean value indicating whether $o_{i\_right} = p_j + 1$ or $o_{i\_down} = p_j + 1$. Here i_left is the index of the left panel of panel i in the same row, i_upper is the index of the upper panel of panel i in the same column, $p_j + 1$ is the successive letter of p_j and $p_j - 1$ is the previous letter of p_j. This feature set takes advantage of the context information by comparing o_{i_left} to $p_j - 1$ and so on. The second and third features capture a sort of short-range dependency. That is, if the *neighboring panel* (an adjacent panel in the same row or the same column) is recognized as the corresponding "adjacent" letter, there is a higher chance that o_i is equal to p_j.

In the inference step for the base learner in the stacked model, if a pair $< o_i, p_j >$ is predicted as positive, i.e., f^0 predicts that o_i matches p_j. We apply a hand-coded rule to update the feature: if f^0 predicts that o_i matches p_j, let $\hat{o}_i = p_j$, otherwise $\hat{o}_i$

remains as o_i for a negative pair. With $\hat{o}_i$, we recalculate the features via comparing $\hat{o}_i$'s and p_j's. We call the procedure of predicting $< o_i, p_j >$, assigning $\hat{o}_i$, and re-calculating features based on $< \hat{o}_i, p_j >$ "stacking". We choose MaxEnt as the base learner to classify $< o_i, p_j >$ and in our experiments we implement one iteration of stacking.

Besides the basic features, we also include another feature that captures the "long-range context", i.e., consider the spatial dependency among all the "sibling" panels, even though they are not adjacent. In general the arrangement of labels might be complex: labels may appear outside panels, or several panels may share one label. However, in the majority of cases, panels are grouped into grids, each panel has its own label, and labels are assigned to panels either in column-major or row-major order. The "panels" shown in Figure 4.6 are typical of this case. For such cases, we analyze the locations of the panels in the figure and reconstruct this grid, i.e., the number of total columns and rows, and also determine the row and column position of each panel. We compute the long-range feature as follows: for a panel located at row r and column c with label o, as long as there is a panel located at row $r^{'}$ and column $c^{'}$ with label $o^{'}$ ($r^{'} \neq r$ and $c^{'} \neq c$) and according to either row-major order or column-major order the label assigned to panel ($r^{'}$, $c^{'}$) is $o^{'}$ given the label for panel (r, c) is o, we assign 1 to the long-range feature. For example, in Figure 4.6, recognizing the panel label "a" at row 1, column 1 would help to recognize "e" at row 2, column 2 and "h" at row 3, column 2.

With the first order-features and long-range features from neighboring panels, it increases the chance of a missing or mis-recognized label to be matched to an image pointer.

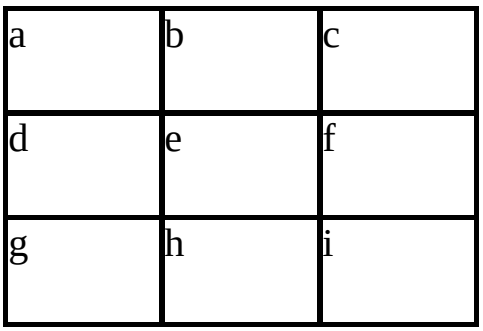

Figure 4.6: Long-range dependency.

4.2.2 Experiments

Dataset

To evaluate the stacked model for panel label and image pointer matching, we collected a dataset of 200 figures which includes 1070 panels. This is a random subsample of a larger set of papers from the Proceedings of the National Academy of Sciences. Our current approach can only analyse labels contained within panels, (internal labels) due to the limitations on the image processing stage therefore in our dataset we only collected figures with internal labels. Though our dataset does not cover all the cases, panels with internal labels are the vast majority in our corpus.

We hand-labeled all the image pointers in the caption and the label for each panel. The match between image pointers and panels was also assigned manually.

Baseline algorithms

In the previous version of SLIF, the match between image pointers and panel labels was done via comparing their values and finding the equal-value pair.

The approaches to find the candidate image pointers and panel labels have been described in Section 4.2.1. In this work, we take the hand-code approach and machine learning approach as the baseline algorithms for image pointer extraction. The OCR-

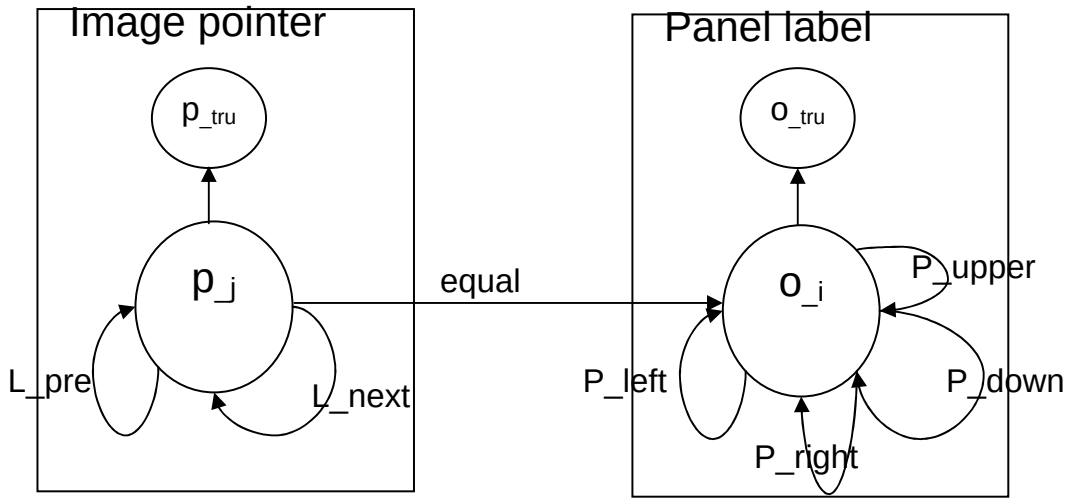

Figure 4.7: An RDN model

based approach and grid analysis approach[33] are baseline algorithms for panel label extraction.

We also compare the stacked model to relational dependency networks (RDNs)[25]. We build an RDN model as shown in Figure 4.7.

In the RDN model there are two types of entities, image pointer and panel label. For a image pointer, the attribute p_j is the value of the candidate image pointer and o_i is the candidate panel label. $p_{\_true}$ and $o_{\_true}$ are the true values to be predicted. The linkage *L_pre* and *L_next* capture the dependency among the sequence of image pointers: *L_pre* points to the previous letter and *L_next* points to the successive letter. *P_left*, *P_right*, *P_upper*, and *P_down* point to the panels to the left, right, upper and down direction respectively. The RDN model takes the candidate image pointers and panel labels as input and predict their true values. The match between the panel label and the image pointer is done via finding the equal-value pair.

Table 4.7: Accuracy of image pointer to pane label matching.

	Image pointer to panel label matching (accuracy)
Baseline algorithm 1	48.7%
Baseline algorithm 2	64.3%
RDN	70.8%
Stacked model (short-range)	75.1%
Stacked model (long-range)	81.3%

Experimental Results

We used 5-fold cross validation to evaluate the performance of the stacked graphical model for image pointer to panel label matching. The evaluation was reported in two ways; the performance on the matching and the performance on image pointer and panel label extraction. To determine the matching is the "real" problem, i.e., what we really care about are the matches, not getting the labels correctly. Evaluation on the image pointer and panel label extraction is a secondary check on the learning technique.

Table 4.7 shows the accuracy of image pointer to pane label matching. For the baseline algorithms, the match was done via finding the equal-value pair. Baseline algorithm 1 was done via comparing the candidate image pointers to the candidate panel labels, i.e., predict a match if two labels are the same. Baseline algorithm 2 was done via comparing the image pointers extracted by the learning approach to the panel labels obtained after grid analysis. The stacked graphical model takes the same input as Baseline algorithm 2, i.e., the candidate image pointers extracted by the hand-coded algorithm and the candidate panel labels obtained by OCR. We observe

that the stacked graphical model improves the accuracy of matching. Both the short-range dependency and long-range dependency help to achieve a better performance. RDN also achieved a better performance than the two baseline algorithms.

Our stacked model achieves a better performance than RDN, because in stacking the dependency is captured and indicated "strongly" by the way we design features. That is, the stacked model can model the matching as a binary classification of $< o_i, p_j >$ and capture the short-range dependency and long-range dependency directly according to our feature definition. However, in RDNs, the data must be formulated as types of entities described with attributes and the dependency is modeled with links among attributes. Though RDNs can model the dependency among data, the matching problem is decomposed to a multi-class classification problem and a matching procedure. Besides that, the long-range dependency can not be modeled explicitly.

Also the stacked models here correspond to a complicated graph, e.g., the model with short-range features includes additional dependencies among L_{pre} to P_{left}/P_{upper}, L_{next} to P_{right}/P_{down}. The model with long-range features is even more sophisticated. Hence stacking can easily cast a complicated model to capture more dependencies in a flexible way.

Table 4.8 shows the performance on the sub-tasks of image pointer extraction and panel label extraction. The results are reported with F1-measurement. Since during the stacked model we update the value of o_i and set it to be p_j when finding a match, the stacking also improves the accuracy of panel label extraction. The accuracy for image pointer extraction remains the same since we do not update the value of p_j. Baseline algorithm 1 is the approach of finding candidate image pointers or candidate panel labels. Baseline algorithm 2 for image pointer extraction is the learning approach, and the grid analysis strategy for panel label extraction. The inputs for

Table 4.8: Performance on image pointer extraction and panel label extraction.

	Image pointer extraction	Panel label extraction
Baseline algorithm 1	60.9%	52.3%
Baseline algorithm 2	89.7%	65.7%
RDN	85.2%	73.6%
Stacked model with first order dependency	-	77.8%
Stacked model with second order dependency	-	83.1%

the stacked graphical model are candidate image pointers and candidate panel labels. We observe that by updating the value of o_i, we can achieve a better performance of panel label extraction, i.e., provide more "accurate" features for stacking. RDN also helps to improve the performance yet the best performance is obtained via stacking.

4.3 WebMaster: Another Multi-task System

4.3.1 Problem formulization and WebMaster dataset

Another application we studied is understanding website update requests [6]. Minkov et al. describe an intelligent system that can process email requests from users to update an organizational database. Some example web site update requests include "Add the following contact to the Staff list. Arthur Scott, ascott@ardra.com, Rm 7992, 412 281 1914" and "Please delete Freddy Smith's phone number".

The information extraction system contains several sub-tasks:

Request type classification As suggested by preliminary analysis of real web-

master request logs, factual-update requests are in one of the following forms: add a new tuple to the database; delete an existing tuple; delete a value from an existing tuple; or alter (add or replace) a value of an existing tuple. One step of the analysis is thus determining the type of request. This is casted as a text classification problem.

Named entity recognition Another important step is to identify all entity names in a request.

Target relationship classification To execute the user's request, the relation associated with each request needs to be determined. For any fixed database schema, there is a fixed set of possible relations, so this is a text classification task.

Target attribute classification Usually the "deleteValue" request needs to specify the attribute to be deleted. For example, for request "Please delete Freddy Smith's phone number", consider the previous analysis tells us we should delete some attribute value from the tuple of the "person" relation with the key value of "Freddy Smith", but does not specify the value to be deleted. Hence, to complete the analysis for deleteValue requests, it is necessary to determine the attribute that needs to be deleted. This is also a text classification task.

Knowing the answer to one sub-task could improve the accuracy on other subtasks. For instance, it was shown in Minkov's paper that knowing the entity type will improve the performance of target relationship classification.

We applied the idea of stacking to several subtasks of the information extraction system. Without stacking, each subtask was accomplished independently. With stacking, we use the predictions for relevant subtasks as features for one subtask.

The datasets and features we used for each task are as in Minkov's paper. The base learner for each subtask was chosen according to Minkov's paper, i.e., VPHMMs for named entity recognition, SVMs for target relationship classification, request type classification, and target attribute classification. In the stacked model, we include the

predictions from other subtasks to expand the feature sets for each task.

4.3.2 Experimental results

We evaluated the idea of applying stacking to an information extraction system with the webmaster data. In stacking, we used two relational templates: (1). a universal relational template, (i.e., including the predictions from all other subtasks to expand the feature sets), or (2). an engineered relational template, which includes only the predictions from the subtasks believed to be relevant.

The performance is summarized in Table 4.9. F1 is used for the evaluation on entity extraction and accuracy is used for the evaluation on other tasks. Table 4.9 shows the performance of including predictions from all other subtasks (a universal template), and the performance of including predictions from relevant subtasks (an engineered template). The relevant subtasks were selected through experience, i.e., for each subtask, we provided the true labels of another subtask. If adding the true labels from subtask T1 improves the performance of T2, we define T1 to be relevant to T2.

Table 4.9 shows that in an information extraction system, applying the stacked idea to break the system into a set of subtasks and use the predictions for subtasks as additional features for relevant subtasks, can improve the performance of the whole system. Request type classification is the only subtask where stacking hurts. One thing we want to mention here is that the qualities of the subtasks are different, i.e., they are quite different classification problems. The last column in Table 4.9 shows the result using the true labels, i.e., the upper bound a stacked model can achieve. We observe that the interrelated subtasks can help improve each other. The reason why stacking did not help much in some cases is probably that there is not much room to improve consider the upper bound obtained by using the true labels.

		Stacked models		
	local model	all subtasks	selected subtasks	ceiling
Entity recognition				
Date	89.6	90.3(+ .7)	90.1(+ .5)	91.5(+1.9)
Time	85.1	85.6(+ .5)	85.5(+ .4)	86.0(+ .9)
Room	90.7	92.8(+2.1)	93.3(+2.6)	95.3(+4.6)
Phone	91.3	92.3(+1.0)	91.6(+ .3)	98.0(+6.7)
Person Name	81.5	81.5	82.9(+1.4)	84.6(+3.1)
Amount	89.3	90.2(+ .9)	89.8(+ .5)	88.9(- .9)
Overall F1 (%)	85.6	86.1(+ .5)	**86.6(+1.0)**	88.3(+2.7)
Target relation classification				
accuracy	97.9	98.1(+.2)	**98.1(+.2)**	99.2(+1.3)
Request type classification				
accuracy	96.1	95.5(-.5)	95.5(-.5)	97.1(+1.0)
Target attribute classification				
accuracy	87.8	90.3(+2.5)	**90.3(+2.5)**	91.9(+4.1)

Table 4.9: Evaluating stacked graphical model on webmaster data. The local model was chosen according to Minkov's paper. Two relational templates for stacked graphical models were explored. The ceiling was obtained by using true labels from relevant subtasks.

	local model with tuned features	local model with basic features	stacked model with basic features
Target relation classification			
accuracy	97.9	96.4	95.9
Request type classification			
accuracy	96.1	88.0	89.3
Target attribute classification			
accuracy	87.8	83.5	86.1

Table 4.10: Evaluating stacked graphical model on webmaster data with a base feature set.

Also there are few examples of some tasks, for example, there are only 167 examples (111 of category 'deltuple' and 56 in category 'delval') for request type classification. For target relation classification, there are 378 examples in category 'D_people', 62 examples in category 'D_buget', 140 examples in category 'D_events' and 37 examples in category 'D_sponsors'.

Since the features used in Minkov et al's paper are well tuned, we also explored the performance of stacking given a weaker feature set. Table 4.10 shows the performance using a *base feature set* that contains bag-of-words feature and capitalization templates over a window including the word to be classified and the three adjacent words to each side. It is observed that with a weaker feature set, the improvement that stacking can achieve is larger. However, there is still more to explore when applying stacking to a system with many variables and complex dependencies to figure out where it works well and where it does not.

4.4 Job Title Prediction

In this section, we consider the task of job-title classification using data obtained from online employee directories. These data include organizational reporting structures as well as information provided by employees about their own job responsibilities. Assigning accurate job-categories to a large employee population is an essential step in enabling state-of-the-art workforce management practices. Combined with employee skills assessments, job categorization is required to match employees to specific job openings, and form well-balanced, multi-disciplinary project teams. Assigning job categories to a large population of employees is a time-consuming task, and hence automation of this process can play a role in improving overall workforce management. It can be particularly important in the case where a company inherits a large number of new employees via a merger or acquisition.

Job title prediction is a challenging classification task for several reasons:

1. the available textual information is very limited, because most people do not provide a detailed description of the job responsibilities
2. the population of employees is very large
3. the distribution over different job categories is very skewed - in our application about 80% employees fall into a few dominant classes.

We applied stacked graphical models for large-scale job title prediction and compare stacked models to a Markov network model and several non-relational machine learning approaches, such as Support Vector Machines, naive Bayes, and MaxEnt, with different feature sets.

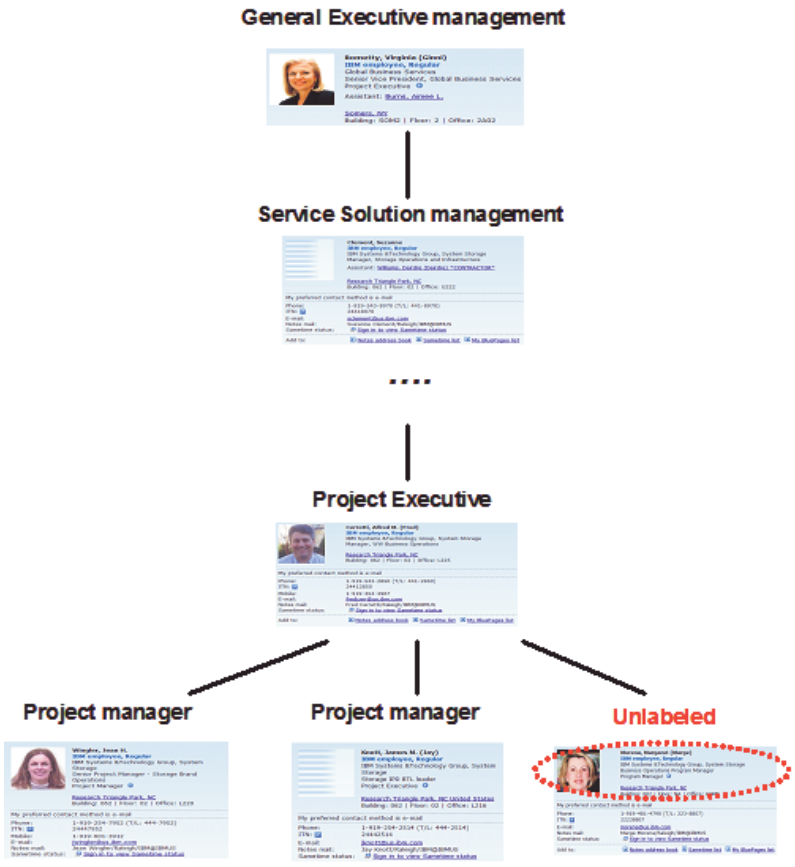

Figure 4.8: The reporting structure

4.4.1 Task description

Like most large companies today, IBM maintains an online directory consisting of profiles for all employees. Known internally as "Blue Pages", this directory contains the usual contact information, as well as other sources of information relevant to the task of inferring the job-classification label for each employee. These data include

1. the name of the employee's department (*e.g.* "Predictive Modeling")
2. a relatively short, employee-entered description of the employee's job responsibilities (*e.g.* "Research in machine learning and data mining")

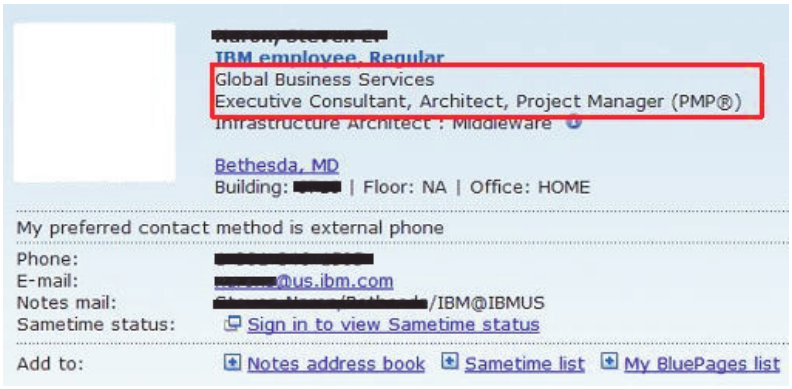

Figure 4.9: A challenging example

3. a separate section where an employee can provide a longer description of their experience, and attach a resume (this section is typically completed by only a small fraction of employees)

4. the organizational reporting tree, *i.e.* links to the profile for each manager above the employee in the organization tree

5. a list of peers of this employee, *i.e.* links to the profile for each employee reporting to the same manager

Hence, we have profiles for each employee, for all managers in the reporting structure above them, and for each of their peers. Figure 4.8 shows an example of a Blue Pages reporting structure.

The machine-learning problem is to predict the job category label for each employee using the data described above. Each employee in IBM is assigned to a job category – in our data, there are 23 unique job-category labels. Figure 4.10 shows the distribution of these labels for one of our datasets described in detail in Section 5. Note that the 3 most common labels, "Consultant", "IT Specialist", and "Project

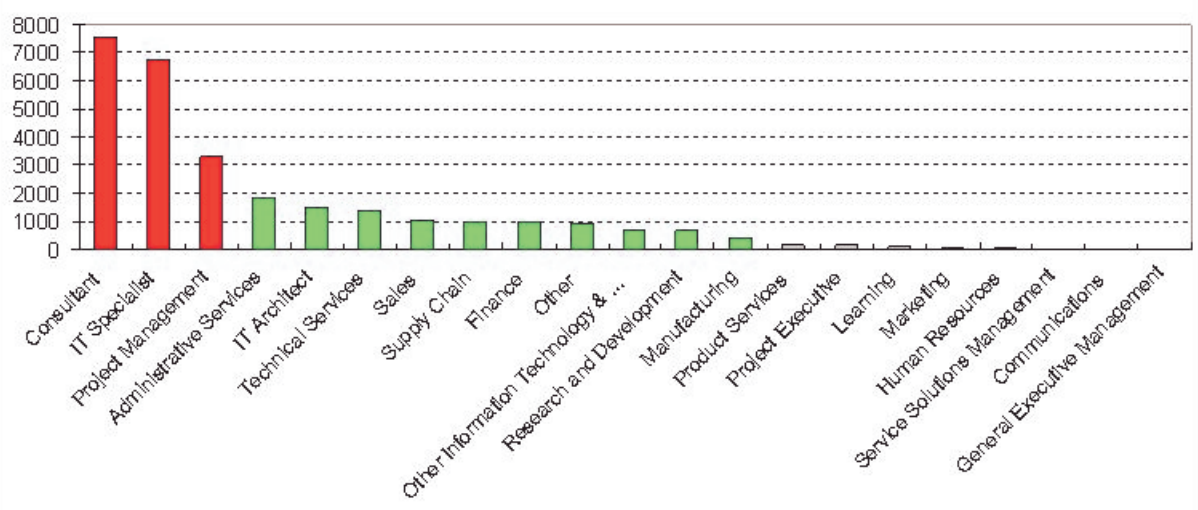

Figure 4.10: The distribution of job categories

Management" account for over 80% of the population. [1] We ignore any labels on a Blue Page that are provided by Human Resources and which can align closely with the job category label we wish to predict.

Figure 4.9 illustrates a practical challenge encountered in this specific application. The correct label for this employee is "IT Specialist", yet the description provided by the employee includes the terms "Consultant", "Architect", and "Project Manager" which are very similar to other labels. One explanation may be that the job-category label is out of date, and does not reflect the employee's current responsibilities. Another possibility is that this employee has a different view of his responsibilities than implied by the label. In any event, this issue introduces some incorrect labels into the process.

[1]This distribution is for a specific sample of IBM employees and is not indicative of the complete IBM employee population.

4.4.2 Methods

From the data mining perspective, we can solve the job title prediction task via multiple approaches: (1) we can treat it as a simple classification problem. Similar to text classification, we can convert the textual features into vectors using the "bag-of-words" representation and feed them into classifiers; (2) we can cast the task as a multiview learning: for one employee, there are two disjoint sub-sets of features (or views), *i.e.* the employees' profiles as well as their managers' profiles. Therefore the multiview learning algorithms can be applied if we assume each feature is sufficient to learn the job title; (3) we can also approach the task as a stacking problem: the reporting tree provides rich information about the relationships between employees, including their job titles. A computational efficient way to make use of these relations is stacking, that is, generating additional "local" features, such as the predicted labels from their managers or peers, and incorporating them into the models; (4) finally, we can solve the problem using the graphical models: we construct a graph with the nodes representing the employees and the edges denoting their reporting structures so that the "global" relations between employees can be captured. There are two variables associated with each node: the job title, either observed or hidden and to be inferred, and the employees profile features. Using the inferencing algorithms, we can predict the job title for unlabeled employees. As we can see, those four methods progress from simple classifiers without modeling any dependencies between employees to graphical models that capture the whole reporting trees.

Simple Classification

In our simple classification model, a bag-of-word feature set is used. The base bag-of-word feature set contains only the textual information available on the employee's webpage. An enriched bag-of-word feature set contains textual information from both

the employee's and the manager's webpages. Here we consider different vocabularies for employees and managers, *i.e.*, if the word "business" appears both in the employee's and the manager's webpage, it is considered as different words.

Support vector machines (SVMs), naive Bayes, MaxEnt, and boosted Decision Tree classifier are chosen as baselines. Support vector machines are discriminative classifiers that simultaneously minimize the empirical classification error and maximize the geometric margin [60]. SVMs have been very successful in many applications and are chosen as one of the baselines in our approach. Naive Bayes (NB) classifier is a probabilistic classifier based on Bayesian theorem with naive independence assumptions [41]. Naive Bayes often works well in many real-world applications and is particularly suited when the dimensionality of the inputs is high. Maximum Entropy (MaxEnt) models have been successfully applied to many fields, *e.g.*, computer vision and natural language processing [54]. We apply MaxEnt to our job title prediction task. Decision tree classifier is another popular classification algorithm in data mining, which uses information gain to find predictive input attributes and can generate concise and meaningful models. In our practise, we also apply AdaBoost algorithm to Decision Tree to improve the performance. Some of the baseline algorithms are defined as binary classifiers, *e.g.*, Naive Bayes and SVMs. The one-vs-all strategy is used to build a multi-class classifier using a binary model.

Stacked Models

Besides the information from the manager, there is also rich information among peer workers (the peers are defined as people working with the same manager). Instead of using the textual information on the peers' webpages, we consider the job title distribution among peers. To estimate this distribution, we need to estimate the job titles first.

Stacking was applied to obtain the "predictions" for training examples,i.e., we first run the simple classification model with the enriched feature set in a cross-validation way to obtain the predicted job titles, estimate the job distribution of peer workers, and add the distribution to the feature set. Finally we run a learning model with the feature set containing the estimated distribution of peers.

In contrast to using the predictions to estimate the job title distribution among peers, true labels can be used to estimate the distribution and the accuracy achieved is the upper bound for the stacked model, *i.e.*, the accuracy that can be achieved if there is a perfect prediction on job titles.

Graphical Models

The job title classification can be formulated as an inference problem over a graphical model, in which the nodes denote individual employees and the edges represent the "reporting to" relationships. We build an undirected graphical model, *i.e.*, a Markov network, with each node associated with observed bag-of-word features based on the employee's profiles and hidden labels for the job titles. The potential function Φ, is defined to include the bag-of-word features in connected nodes. Following the idea of conditional random fields and discriminative relational Markov networks, we have the conditional probability of the labels given the observation as follows:

$$\begin{aligned}
&P(Y_1, \ldots, Y_n | \mathbf{x}_1, \ldots, \mathbf{x}_n) \\
&= \frac{1}{Z} \prod_{i=1}^{n} \Phi(\mathbf{x_1}, \ldots, \mathbf{x_n}, y_i) \Phi(\mathbf{x_1}, \ldots, \mathbf{x_n}, y_i, y_i^{\uparrow}) \\
&= \frac{1}{Z} \exp(\sum_{i=1}^{n} \sum_{k=1}^{K} \lambda_k f_k(\mathbf{x}_1, \ldots, \mathbf{x}_n, y_i, y_i^{\uparrow})),
\end{aligned}$$

in which $y_i^{\uparrow}$ represent the neighbor of y_i on the upper depth of the reporting tree, *i.e.* the manager of y_i; f_k are the features defined over the pair of labels and the

observations. We further factorize f_k as

$$f_k(\mathbf{x}_1, \ldots, \mathbf{x}_n, y_i, y_i^{\uparrow}) = f'_k(\mathbf{x}_i, \mathbf{x}_i^{\uparrow})\delta(\langle y_i, y_i^{\uparrow}\rangle, \langle y, y'\rangle),$$

where δ is the indicator function and $\langle y, y'\rangle$ is the pair of label values. λ_k is the feature weight, which is usually learned by maximizing the likelihood of the training data. We do not have a fully labeled set on the whole graph to learn the parameters. Therefore we use an alternative estimation that trains one simple logistic regression by assuming each pair of employee and manager as independent and the use the corresponding learned weights in the graphical models.

To predict the labels for each node in the graph, we use the marginal probability, *i.e.*

$$\tilde{y_i} = \arg\max_y \ P(Y_i = y|\mathbf{x}_1, \ldots, \mathbf{x}_n).$$

The marginal probability can be estimated using the belief propagation (BP), given that our graph topology is simply a tree [50]. The algorithm maintains a message $m_{j,i}(y_i)$ from node y_j to node y_i. The update from y_j to y_i is given by:

$$m_{j,i}(y_i) \leftarrow \sum_{y_j} \Phi(\mathbf{x}, y_j)\Phi(\mathbf{x}, y_i, y_j) \prod_{k \in \mathcal{N}(j)/i} m_{k,j}(y_j),$$

where $\mathcal{N}(j)$ is the neighbor of node y_j. Given the message vector m, the marginal probability can be computed as

$$p(y_i) \leftarrow \frac{1}{Z'_1}\Phi(\mathbf{x}, y_i) \prod_{j \in \mathcal{N}(i)} m_{j,i}(x_i),$$

where Z'_1 is the local normalization term over node y_i.

One practical difficulty of applying graphical models is the computational expense and the scale of the dataset that can be handled. In our practise, the implemented Markov networks can process a graph with about 300 nodes at a time. To handle the computational constraints, we split the original reporting tree into sub-trees empirically, *i.e.*, we only allow to cut the edge from a non-leaf node to its manager,

so as to keep the peers in one sub-graph. Borrowing the idea of generalized belief propagation [62], we first run belief propagation on each subtree, then update the messages between the nodes where an edge was cut, and run BP again to pass the new information to other nodes in the subgraph.

4.4.3 Experimental results

Datasets

We examined our models with 3 datasets. The first dataset contains 979 examples randomly selected from whole US employee population, maintaining the original ratio of the 21 different job categories. The second dataset contains 27,626 examples that were selected based on the employment year, i.e before a particular year cut-off. The third data set contains 20,320 US IBM employees within a particular department labeled with one of the 23 categories. For the first two datasets, there are employee profiles associated with the reporting structure, while the labels for managers and peers are not available in general. The third dataset includes the whole employee population in a particular department and there are both profiles and labeling information available along the reporting chain, *i.e.*, for each example in the third dataset, there is the label for the employee's manager. The distribution of the job categories are skewed in all of these three datasets, For example, in the third dataset, there are 23 job categories yet about 80% employees fall into the 3 dominant classes. Figure 4.10 shows the distribution of the job categories in the third dataset. All the experiment results are reported in prediction accuracy on 10-way cross-validation.

As described in the previous section, we examined the simple classification models on two feature sets: (1) a base feature set with "bag-of-word" representation containing only information from the employee's profiles; (2) an enriched feature set with bag-of-word features from both the employee's and the manager's profiles. We exam-

ined the simple models on all of the three datasets to evaluate the impact of different feature sets.

The stacked models and graphical models can capture the dependencies between employees along the reporting chain and among peers. Therefore we study their effectiveness on the third dataset with the whole the whole reporting structure and profiles available.

Random samples

Table 4.11: Accuracies on the first dataset (979 examples) and second dataset (27,626 examples), with simple models.

Algorithm	1st dataset		2st dataset	
	base feature	enriched feature	base feature	enriched feature
Naive Bayes	55.5	61.6	67.3	69.5
SVMs	52.2	58.5	67.3	74.6
Decision Tree	**61.4**	59.2	65.8	71.4
Boosted DTree	57.6	**64.3**	70.7	76.6
MaxEnt	60.1	64.2	**71.2**	**78.2**

The first and second datasets are random samples with different sizes. We applied simple models on the two random samples to evaluate the impact of the size of training data and feature sets. The results are summarized in Table 4.11. As we can see, the additional feature set with the managers' profiles helps to improve the accuracy; a larger training set also boosts the performance. Among the simple models, MaxEnt and Boosted Decision Tree yields the best accuracy in general.

Department-level experiments

The third dataset we collected contains 20,320 employees from one whole business department. It is essentially a complete relational data that can be represented via a tree, considering the employee as nodes and reporting chains as edges. We examine both simple classification models and the two relational models - stacked models and graphical models to the third dataset.

Table 4.12 summarizes the results of "simple classification", with two feature sets. Similar to the performance on the 1st and 2nd datasets, MaxEnt and Boosted Decision Tree with enriched feature sets give best results in general.

Table 4.12: Accuracy on the 3rd data set, with simple classification models

Algorithm	Accuracy	
	base feature	enriched feature
Naive Bayes	72.7	76.0
SVMs	72.2	80.5
Decision Tree	70.2	77.2
Boosted Decision Tree	75.6	82.2
MaxEnt	**76.5**	**82.6**

Table 4.13: Accuracy on the data in global business service, stacked models, the upper bound is obtained using the dist. estimated with true labels

Algorithm	Accuracy	
	stacked feature	upper bound
MaxEnt	82.9	84.9
Boosted Decision Tree	**84.6**	85.9

Table 4.14: Comparison of non-relational learning and Markov networks, on the 3rd dataset.

	Solution	Accuracy
non-relational learning	Baseline 1: base feature	76.5
	Baseline 2: enriched feature	82.6
Relational Learning	Stacked model: enriched + stacked feature	84.6
	Markov network	83.1

Stacked models can capture the dependencies among employees in a natural way and are easy to implement based on any simple learning algorithm. We explore the stacked models built upon MaxEnt and Boosted Decision Trees, with the enriched feature set. An upper bound of the performance by stacked models can be achieved when the generated features are reliable and noise-free, that is, they are built upon the true labels of the managers and the peers. Table 4.13 summarizes the results of stacked models. Compared with the best performer of simple classification in Table 4.12, we observe an improvement of 2.4% via the stacked models which capture the local dependencies. Compared to the upper bound that can be achieved via stacked models, there is still room for further improvement.

The graphical models that can capture the global dependencies is also applied to the third dataset. The tree representing the whole dataset with 20,320 nodes can not be processed via the inference algorithm. Therefore we split the whole graph into sub-graphs by cutting the edges between a non-leaf node and its parent, as described in Section 4.4.2. Table 4.14 shows the accuracy achieved by the graphical model. The best performance of other models are also summarized in Table 4.14. We can see: (1) the graphical model achieves better performance than the baseline with

simple classification, but worse than the stacking algorithms; (2) With well-tuned features, simple models can obtain satisfactory accuracies. There are two potential reasons for less competitive performance of graphical models: (1) the approximation algorithm we take in the experiment lose important relational information; (2) the local dependencies dominate the job title assignment. However, there is room for graphical models to improve, for example, we can design potential functions that include information from peers directly, rather than passing through the managers, or develop more robust approximation algorithms to handle a larger dataset at a time.

In this section, we explore models for large-scale job title classification. The experiment results show that the graphical model is able to achieve comparable results to other classification methods using well-engineered features as input. Stacking can easily capture more dependencies and achieved the best performance so far.

4.5 Conclusions

In this application, we explored four in-depth case studies and demonstrate the accuracy and flexibility of stacked graphical models - the relational template can capture the dependency among data in a very flexible way and is easy to implement.

With case studies, we not only show more successful applications of stacked graphical models, but also demonstrate the flexibility of stacked models, both in the sense of modeling the dependency properly to improve the accuracy and in the sense of capture more dependencies in a natural and easy way.

Chapter 5

Online Stacked Graphical Learning

Existing collective classification methods are usually expensive due to the iterative inference in graphical models, and learning procedures based on iterative optimization. Also, for large dataset, the cost of maintaining large graphs or related instances in memory becomes a problem.

As we have seen in previous chapters, one advantage of stacked graphical learning is that the inference is very efficient. We have shown that stacked graphical learning is 40 to 80 times faster than Gibbs sampling during inference. However, the time and memory cost during training for standard stacked graphical learning can be expensive, relative to using a purely local base learner, since it applies a base learner to the training data in a cross-validation-like way to make predictions.

In this chapter, we described a scheme to integrate a recently-developed single-pass online learning method with stacked learning, to save training time and to handle large streaming datasets with minimal memory overhead. During the learning procedure of an online learner, predictions are made to learn the online model. Thus the predictions for training data can be obtained naturally, and there is no need to apply the base learner several times to the training data to obtain the predictions. Therefore online

stacked graphical models will save training time - obtain in average more than 50 time speed-up compared to standard stacked graphical learning. Also the learner needs to maintain only the classifiers, and does not need to store all the examples in memory; thus online stacked graphical learning will save training time and memory.

5.1 Online Stacked Graphical Learning

5.1.1 Single-pass online learning

Compared to batch methods, online learning methods are often simpler to implement, faster, and require considerably less memory. For these reasons, online techniques are natural ones to consider for large-scale learning problems. Online learning methods, such as Perceptron or Winnow [10, 17], are also naturally suited to stream processing. Unfortunately however, in practice multiple passes over the same training data are required to achieve accuracy comparable to state-of-the-art batch learners.

In order to address this problem, Carvalho & Cohen [4] investigated the performance of different algorithms in the *single-pass online learning* setting, i.e., online learning algorithms restricted to a single training pass over the available data. This setting is particularly relevant when the system cannot afford several passes throughout the training set: for instance, when dealing with massive amounts of data, or when memory or processing resources are restricted, or when data is not stored but presented in a stream.

Their work revealed that some single-pass online learning algorithms can provide batch-level performance on a variety of tasks. More specifically, it was observed that in classification tasks for datasets with sparse features (very common in Natural Language Processing tasks), a modification of the Balanced Winnow algorithm (MBW or Modified Balanced Winnow) [4] presented excellent performance - even comparable

to batch learners. They also observed that a variation on the Perceptron algorithm called the *Voted Perceptron* [16] presented fairly good results on classification tasks when the feature is not sparse.

Voting (a.k.a. averaging) an online classifier is a technique that, instead of using the best hypothesis learned so far, uses a weighted average of all hypotheses learned during a training procedure. The averaging procedure is expected to produce more stable models, which leads to less overfitting [15]. Averaging techniques have been successfully used with the Perceptron algorithm [16] as well as with several other online learning algorithms, including MBW [4].

5.1.2 MBW

MBW is a modification of the Balanced Winnow algorithm, which in turn is an extension of the Winnow algorithm [12, 37]. It is based on multiplicative updates and it assumes the incoming example w_t is a vector of positive features, i.e., $w_{t,j} \geq 0$, $\forall t$ and $\forall j$, where $w_{t,j}$ denotes the j^{th} feature of w_t. This assumption is usually satisfied in NLP tasks, where the $w_{t,j}$ values are typically the frequency of a term, presence of a feature, TFIDF value of a term, etc. The learning algorithm is detailed in Table 5.1.

In general terms, for each new example w_t presented, the current model will make a prediction $\widehat{v}_t \in \{-1, 1\}$ and compare it to the true class $v_t \in \{-1, 1\}$. The prediction will be based on the score function f, on the example w_t and on the current hypothesis. MBW is mistake-driven, i.e., only in the case of a prediction mistake the hypothesis (or model) will be updated.

Like Balanced Winnow, MBW has a promotion parameter $\alpha > 1$, a demotion parameter β, where $0 < \beta < 1$ and a threshold parameter $\theta_{th} > 0$. It also has a margin parameter M, where $M \geq 0$.

After the algorithm is initialized, an *augmentation* and a *normalization* preprocessing step is applied to each incoming example w_t. When learning, the algorithm receives a new example w_t with m features, and it initially *augments* the example with an additional feature(the $(m+1)^{th}$ feature), whose value is permanently set to 1. This additional feature is typically known as "bias" feature. After *augmentation*, the algorithm then *normalizes* the sum of the weights of the augmented example to 1, therefore restricting all feature weights to $0 \leq w_{t,j} \leq 1$.

In MBW, the hypothesis is a combination of two parts: a positive model u_t and a negative model v_t. After *normalization*, the score function is calculated as $score = \langle w_t, u_i \rangle - \langle w_t, v_i \rangle - \theta_{th}$, where $\langle w_t, u_i \rangle$ denote the dot product of vectors x_t and u_i.

If the prediction is mistaken, i.e., $(score \cdot v_t) \leq M$, then the models are updated. The update rule will be based on multiplicative operations on the two models, taking into consideration the promotion and demotion parameters (α and β), as well as the particular feature weight of the incoming example.

Following the parameters suggested by by Carvalho & Cohen [4], our implementation sets the promotion parameter $\alpha = 1.5$, the demotion parameter $\beta = 0.5$, the threshold $\theta_{th} = 1.0$, the "margin" M was set to 1.0, and the initial weights were $\theta_0^+ = 2.0$ and $\theta_0^- = 1.0$.

In testing mode, the *augmentation* step in MBW is the same, but there is a small modification in the *normalization*. Before the normalization of the incoming instance, the algorithm checks each feature in the instance to see if it is already present in the current models (u_i and v_i). The features not present in the current model are then removed from the incoming instance before the normalization takes place.

Table 5.1: Modified Balanced Winnow (MBW).

1. Initialize $i = 0$, and models u_0 and v_0.

2. For $t = 1, 2, ..., T$:

   (a) Receive new example w_t.

   (b) Augmentation: add "bias" feature to w_t.

   (c) Normalize w_t to 1.

   (d) Calculate $score = \langle w_t, u_i \rangle - \langle w_t, v_i \rangle - \theta_{th}$.

   (e) Receive true class v_t.

   (f) If prediction was mistaken, i.e., $(score \cdot v_t) \leq M$:

      i. Update models. For all feature j s.t. $w_t > 0$:

$$u_{i+1,j} = \begin{cases} u_{i,j} \cdot \alpha \cdot (1 + w_{t,j}) & \text{, if } v_t > 0 \\ u_{i,j} \cdot \beta \cdot (1 - w_{t,j}) & \text{, if } v_t < 0 \end{cases}$$

$$v_{i+1,j} = \begin{cases} v_{i,j} \cdot \beta \cdot (1 - w_{t,j}) & \text{, if } v_t > 0 \\ v_{i,j} \cdot \alpha \cdot (1 + w_{t,j}) & \text{, if } v_t < 0 \end{cases}$$

5.1.3 Online stacked graphical learning

The Algorithm

During the learning procedure of an online learner, intermediate predictions for training data are generated as the online model is generated. Thus the predictions for training data can be obtained naturally and there is no need to apply the base learner many times to the training data in a cross-validation-like procedure to obtain the predictions. Therefore combining the online learning scheme with stacked graphical models can save training time.

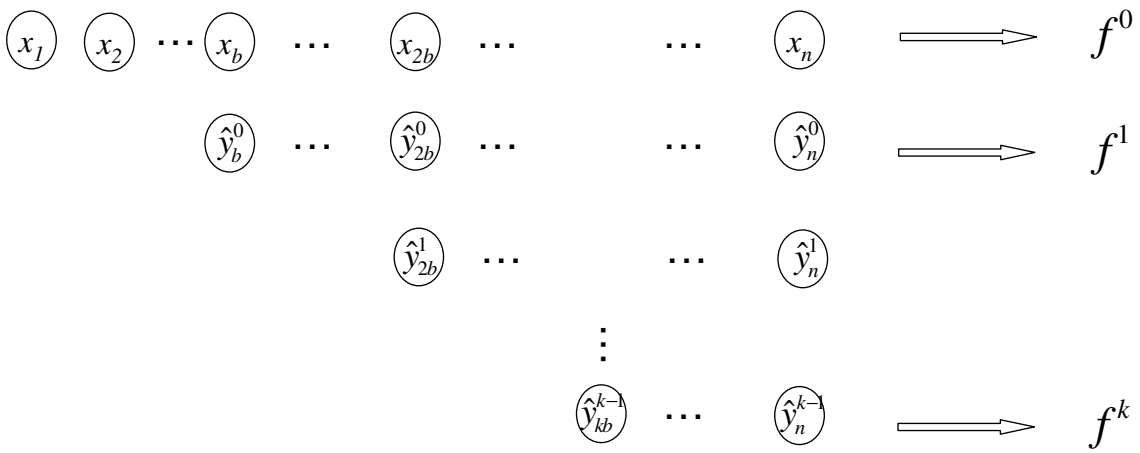

Figure 5.1: Online Stacked Graphical Learning

One practical difficulty is that, while online learning methods produce satisfactory predictions after learning on the whole training set, the intermediate predictions for the training data in the starting stage can be quite inaccurate. Thus, to obtain fair "predictions" for training examples, we define a burn-in data size b. That is, after training on b examples, we start using intermediate predictions from the online learner and expanding features with the predictions, i.e., these predictions are used to train a stacked model online.

The learning procedure of online stacked learning is shown in Figure 5.1. Figure 5.1 shows that in the learning procedure of online stacked graphical models, f^0 is trained on the whole training dataset. Here $\mathbf{x}_i =< (w_i^1), ..., (w_i^{N_i}) >$ is a set of instances, and $\mathbf{y}_i$ is the corresponding labels for $\mathbf{x}_i$. After training on b sets of examples, we start recording the intermediate predictions $\hat{\mathbf{y}}^0_b, ..., \hat{\mathbf{y}}^0_n$, which are generated naturally during the learning of f^0. For the first level of stacking, i.e., $k = 1$, we apply the relational template to expand features (i.e., $(w_i^j)^1 = (w_i^j, C(w_i^j, \hat{\mathbf{y}}^0_i))$, and train f^1 with expanded examples $(\mathbf{x}^1_b, \mathbf{y}_b)..., (\mathbf{x}^1_n, \mathbf{y}_n)$. Similarly, for the k^{th} level of stacking, intermediate predictions $\hat{\mathbf{y}}^{k-1}_{kb}, ..., \hat{\mathbf{y}}^{k-1}_n$ (which are generated naturally during the learning of f^{k-1}) are recorded to expand features and the k^{th} stacked model is trained with expanded

instances $\mathbf{x}_{kb}^k, ..., \mathbf{x}_n^k$.

One thing we would like to point out is that, in stacked graphical learning for collective classification, given an instance w_i, we need to apply the relational template to retrieve the predicted labels for the related instances to extend features. Assume w_i and its neighbors are contained in a subset, we provide the instances in a subset to the online learner as a group and extend the features after the predictions for instances in the whole subset are made. Therefore in general, we provide the instances in groups to the base learner and the burn-in data size b will be chosen to include a few subsets of instances. In practice, the dataset might not be able to be split into disjoint subsets naturally. In Section 5.2 we will demonstrate how to split the dataset into subsets heuristically .

Efficiency Analysis

Theoretically, when there are infinitely many training examples, i.e., $kb << n$, applying the online stacked graphical learning shown in Figure 5.1 only requires single-pass training over the training set. We do not need to apply the cross-validation-like trick to get the predictions for training examples. That is, the complexity of the online scheme is in the order of $O(n)$. Therefore, online stacked graphical learning can save training time. In Section 5.2.3 we will show the speed-up experimentally as well.

In online stacked graphical learning, there are reliable predictions at level k after $(k+1)b$ examples have streamed by, and the learner needs to maintain only k classifiers and does not need to store examples. Therefore, the algorithm can save memory. This becomes extremely important when the size of training data is huge. Also this feature allows online stacked graphical learning to be applied to streaming data.

Given a training set $D = \{(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)\}$, where $x_i =< (w_i^1), ..., (w_i^{N_i}) >$ is a set of instances, and an online learner A, construct predictions as follows:

1. Give the training data,$(\mathbf{x}_1, \mathbf{y}_1), ..., (\mathbf{x}_n, \mathbf{y}_n)$, to the online learner, train a classifier f_1, and record the intermediate predictions from online learning on $\mathbf{x}_j$ for $j = n/2, ..., n$.

2. While $\mathbf{x}_j,\ j = n/2, ..., n$, streaming by, train another online learner f_2 with $(\mathbf{x}_{n/2}, \mathbf{y}_{n/2}), ..., (\mathbf{x}_n, \mathbf{y}_n)$, go back to $\mathbf{x}_1, ..., \mathbf{x}_{n/2}$, keep learning f_2 and record the intermediate predictions for $\mathbf{x}_j,\ j = 1, ..., n/2 - 1$.

Figure 5.2: The procedure to obtain predictions for training examples via an online base learner, with limited data

An Implementation With Limited Data

Theoretically, we assume $kb << n$ and online stacked graphical learning only requires single-pass training over the training set. In practice, the assumption $kb << n$ may not hold. An implementation with limited training data is to let $b = n/2$ and apply a one-and-half-pass procedure shown in Figure 5.2, to obtain the predictions for training examples. Using the procedure shown in Figure 5.2 to obtain the predictions, we end up with a learning and inference method similar to the procedure of standard stacked graphical learning, except that the predictions are no longer obtained in a cross-validation-like way.

In the practical implementation, given n training examples, training a stacked graphical model with level K has the complexity of $O((K + 0.5)N)$.

5.2 Experiments

We evaluated stacked graphical learning on tasks from three domains - collective classification over relational datasets, sequential partitioning [5], and named entity extraction.

5.2.1 Problems and datasets

Relational Datasets

The relational datasets we consider here include text region detection in Subcellular Location Image Finder (SLIF) [33, 47] and document classification. More details about the SLIF dataset and the dependencies defined in the data can be found in Chapter 3.1.1.

We use MBW as the base online learner for SLIF. The features are the same as in Chapter 3.1.1. In the SLIF text region detection task, the candidate regions can be naturally grouped into disjoint subgraphs, i.e., candidate regions from the same figure construct a subgraph. Therefore as long as the prediction for candidate regions from the same figure is obtained, we can apply the relational template to expand features. The relational template is the same as in Chapter 3.1.1.

The document classification includes the webpage classification on the WebKB dataset [11] and paper classification on the Cora dataset and the CiteSeer dataset [39]. More details about the document classification dataset can be found in Chapter 3.1.1.

We use MBW as the base online learner for document classification. The current implementation of MBW only supports binary labels, so we considered the task corresponding to the most common label. The relational template for document classification is the same as in Chapter 3.1.1.

Table 5.2: Performance of online stacked graphical learning for relational datasets: accuracy for "Document classification" and F1-accuracy for "SLIF" are reported. We evaluated two local models: MaxEnt and MBW. We also compared to a competitive relational model - relational dependency networks. The standard stacked model used two-fold-cross-validation predictions. The online stacked graphical model is based on MBW. We used 1 level of stacking, i.e., K=1.

	SLIF	Document classification		
		WebKB	Cora	CiteSeer
Local model				
MaxEnt	81.5	57.6	63.1	54.9
MBW	82.3	58.0	62.8	55.8
Competitive relational model				
Relational Dependency Networks	86.7	73.1	72.3	57.9
Stacked model				
Standard Stacked model (with MaxEnt, k=1)	90.1	72.9	**73.0**	59.3
Standard Stacked model (with MBW, k=1)	92.1	**73.8**	72.9	**60.0**
Online Stacked model (k=1)	**92.3**	73.5	70.7	-

The feature sets and relational templates are the same as in Chapter 3. We use the following heuristic to split the datasets: the WebKB dataset contains webpages from four computer science departments, hence we split them into groups according to departments. We group the papers in Cora dataset by the year of publishing. There is no such year-of-publishing information available for the Citeseer dataset, thus we only applied the implementation shown in Figure 5.2 to Citeseer data.

Table 5.3: Accuracy comparison of online stacked graphical learning for sequential partitioning. We evaluated two local models: MaxEnt and MBW. We compared to a competitive graphical model - conditional random fields. The standard stacked model used two-fold-cross-validation predictions. The online stacked graphical model is based on MBW. We used level 1 of stacking.

	Sequential Partitioning		
	FAQ	signature	video
Local model			
MaxEnt	67.3	96.3	80.9
MBW	64.9	96.5	78.4
Competitive relational model			
CRFs	85.6	98.1	83.0
Stacked model			
Standard Stacked model (with MaxEnt, k=1)	**87.1**	98.1	**85.8**
Standard Stacked model (with MBW, k=1)	84.1	**98.3**	85.5
Online Stacked model (k=1)	86.3	**98.3**	85.7

Table 5.4: Performance of online stacked graphical learning for Named Entity Extraction, F1 accuracy is reported. "Relational template 1" returns predictions of adjacent tokens only, "relational template 2" returns predictions of adjacent and repeated tokens.

	Named Entity Extraction			
	UT	Yapex	Genia	CSpace
Local model				
MaxEnt	69.1	62.1	66.5	74.2
MBW	67.9	62.3	66.9	75.1
Competitive relational model				
CRFs	73.1	65.7	72.0	80.3
Stacked model				
With relational template 1				
Standard Stacked model (with MaxEnt, k=1)	70.1	63.7	70.8	77.9
Standard Stacked model (with MBW, k=1)	72.1	63.9	71.3	79.9
Online Stacked model (with MBW, k=1)	72.6	64.6	72.3	80.0
With relational template 2				
Standard Stacked model (with MaxEnt, k=1)	77.3	68.2	78.5	82.1
Standard Stacked model (with MBW, k=1)	**76.6**	68.9	**78.9**	83.3
Online Stacked model (with MBW, k=1)	**76.6**	**69.1**	**78.9**	**83.4**

Sequential Partitioning Datasets

The Sequential partitioning datasets include a signature dataset, an FAQ dataset, and a video dataset, as decribed in Chapter 3.1.1.

We use a Modified Balance Winnow learner [4] as the base online learner in stacked graphical learning for sequential partitioning. In the sequential partitioning task, the instance is naturally grouped into sequences. Therefore as long as the prediction for a sequence is obtained, we can apply the relational template to expand features. The relational templates returns the predictions of ten adjacent examples (five preceding examples and five following examples).

Named Entity Extraction Datasets

We applied stacked graphical learning to named entity extraction from Medline abstracts and emails. More details about the named entity extraction datasets can be found in Chapter 3.1.1.

The feature sets and relational templates for named entity extraction are the same as in Chapter 3.1.1. The relational template will retrieve the predictions for the adjacent words (with window size 5) and for the same word appearing in one abstract, apply the COUNT aggregator, and return the number of words in each category, given one word. That is, let w_i be the word in a document. For words $w_j = w_i$ in the same document, we count the number of times w_j appearing with label y and use it as one of the stacked features for w_i.

In addition to this relational template, we applied another relational template which just retrieves the predictions for the adjacent words (with window size 5).

5.2.2 Experimental results

Accuracy of Stacked Graphical Learning with efficient training

To evaluate the effectiveness of online stacked graphical learning on the collective classification task, in Table 5.2 we compare local models, stacked models, and a state-of-art competitive model. We evaluated two local models, MaxEnt and MBW. We considered a standard stacked model based on MaxEnt (with two-fold-cross-validation predictions), a standard stacked model based on MBW (with two-fold-cross-validation predictions), and an online stacked graphical model based on MBW. We compared stacked graphical model to relational dependency networks [25].

Table 5.2 shows that on all of the four relational datasets, stacked graphical learning improves the performance of the base learner significantly. The two local models achieved performance at the same level, so did the stacked graphical models based on them. Our comparison to relational dependency networks shows that stacked models can achieve competitive results to the state-of-art model. However, the online stacked graphical model requires much less training time, which will be discussed in the next section.

One thing we want to point out is that, due to the lack of information on the year of publication, we can not implement online stacked model to Citeseer data. And the performance of online stacked model for Cora data is not as good as the standard stacked graphical models. The reason for the performance drop is that providing papers in the order of years of publication to the online learner can only provide the predictions of papers that were published before the current timestamp and were cited by the current paper, i.e., the predictions available so far can only provide information on the papers cited by the current paper, while in reality, there is also information contained in the paper that would be published and would cite the current paper.

Table 5.3 shows the performance of online stacked models on sequence partitioning. The state-of-art models we consider here are conditional random fields (CRFs). On all of the three datasets, stacked graphical learning improves the performance of the base learner significantly. The MaxEnt model did better than MBW on two of three tasks, yet the stacked graphical models based on them achieved performance of the same level.

Table 5.4 reported the F1-accuracy of online stacked graphical learning for Named Entity Extraction. In Section 3.3 we described two relational templates for named entity extraction. One relational template captures sequential dependency only(denoted as relational template 1 in Table 5.4), the other one can also capture the dependency among the adjacent and repeated tokens(denoted as relational template 2 in Table 5.4).

Table 5.4 shows that on all of the four named entity extraction tasks, stacked graphical learning improves the performance of the base learner. With relational template 1, the stacked graphical models can capture the sequential dependency and achieved comparable results to CRFs. With relational template 2, the stacked graphical models achieved better performance than CRFs. Moreover, the online stacked graphical model requires much less training time.

5.2.3 Training efficiency of online stacked graphical learning

One big success of online stacked graphical learning is the efficiency during training. We compared the training time of online stacked graphical models (with one iteration) to that of competitive relational models and the baseline standard stacked graphical model. The baseline algorithm we compare to is the best algorithm in previous work [32], the standard stacked graphical model based on MaxEnt, with 5-fold-cross-validation to obtain predictions during training. We compare the baseline algorithm

Table 5.5: Comparison on training time.

	Standard SGM vs Online SGM	Competitive relational model vs Online SGM
SLIF	38.1	7.9
WebKB	50.0	10.1
Cora	49.7	9.9
Signature	67.4	13.6
FAQ	69.0	14.0
Video	45.0	9.7
UT	68.7	20.3
Yapex	60.6	17.1
Genia	69.4	22.4
CSpace	52.0	15.3
Average speed-up	57.0	14.0

to the online stacked graphical learning with implementation shown in Figure 5.2.

Table 5.5 shows the speedup, i.e., in the table "38.1" means the training in standard stacked graphical learning is 38.1 times slower than that of online stacked graphical learning. Table 5.5 shows that compared to online stacked graphical learning, standard stacked graphical learning based on MaxEnt is approximately 57 times slower in training.

We also compared online stacked graphical learning with the competitive relational models. Table 5.5 shows that online stacked graphical learning is approximately 14 times faster in training. Moreover, in the previous work [32], it has been shown that during inference stacked graphical learning is 40 to 80 times faster than Gibbs sampling in relational dependency networks.

Therefore, online stacked graphical models can achieve high accuracy with efficient training and testing.

5.3 Conclusions

In this chapter, we combine an online training scheme with stacked graphical learning, so as to be able to handle large streaming data with linear training time and minimal memory overhead. Integrating single-pass online learning algorithm with stacked graphical learning can save the time and memory cost during training.

With high accuracy and efficiency and low memory cost, stacked graphical learning is very competitive in real world large-scale applications where an efficient algorithm is extremely important. Furthermore, with the online learning scheme, stacked graphical learning is also able to be applied to streaming data.

Chapter 6

Literature Review and Related Work

Traditionally, machine learning methods have assumed that instances are independent and identically distributed, which makes it possible to classify examples on an one-by-one basis. With relational data, the entities influence each other's category. Hence it is beneficial to classify all related instances simultaneously, i.e., do collective classification. Collective classification over relational data is closely related to collective inference over graphical models and statistical relational learning. Our stacked graphical learning model is an efficient statistical learning approach for collective classification, which simplifies a relational graphical model by using a base learner, a relational template, and a collective inference procedure.

In this section we first review some related approaches in the context of graphical models and their relational extensions, followed by reviewing related work on relational template design and efficient approximation for graphical models, especially several variations of conditional random fields for collective inference. We conclude the section by a comparison of stacked graphical learning to the related work.

6.1 Graphical Models and Their Relational Extensions

6.1.1 Bayes networks and probabilistic relational models

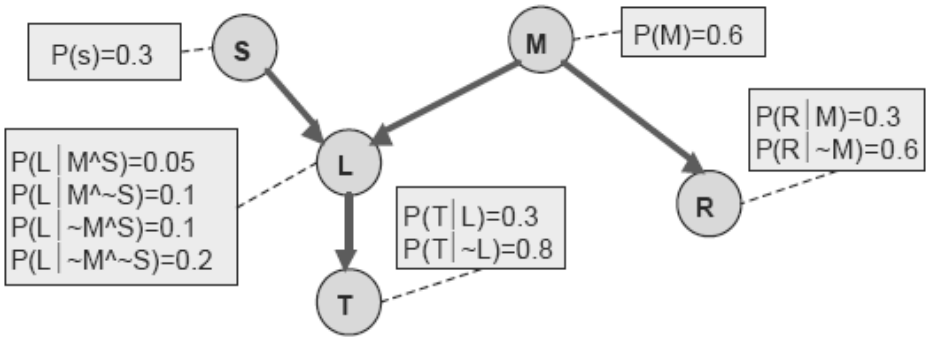

Figure 6.1: An example Bayes network

Bayes networks (BNs) [46] are directed graphical models to represent a joint probabilistic distribution among a set of random variables. A Bayes network consists of a directed acyclic graph in which nodes represent random variables, and the directed edges represent conditional independence assumptions, and a set of local conditional probability distributions (CPDs) . A Bayes Network can be parameterized by specifying all the local CPDs, i.e., the distributions $P(X_i|Pa_i)$, where X_i represents node i and Pa_i are its parents. In a Bayes network, the joint distribution of X_i's can be written as the product of the local distributions of each X_i and its parents, i.e.,

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i|Pa_i)$$

.

Figure 6.1 shows an example Bayes network. Bayes networks can be used to calculate the conditional distribution of a subset of random variables, given some know

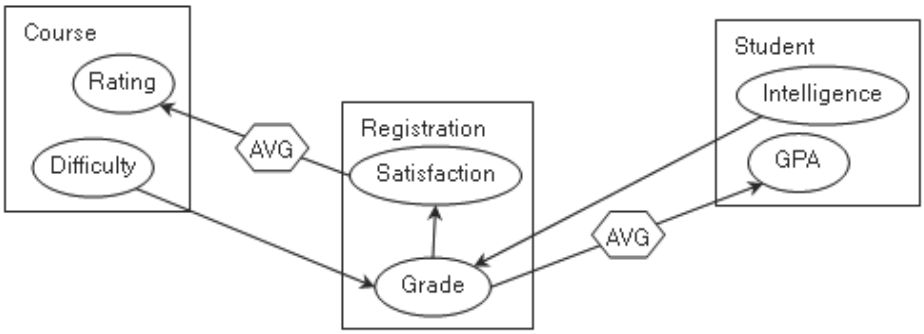

Figure 6.2: A dependency model for the university registration example

values for another subset of variables (i.e., the evidence). Typically this procedure is done by apply Bayes rule.

Though Bayes networks are useful for answering queries about any variable in the network given some evidence, BNs are not well suited for modelling relational datasets since BNs are attribute-based and lack the concept of an object. Probabilistic relational models (PRMs) [18, 31] are a relational extension to Bayesian networks which allows relational structure to be represented and exploited. PRMs specify a relational schema for the corresponding relational domain, a set of probabilistic dependencies between the attributes (i.e., local CPDs), and a joint probability distribution over a collection of related entities. A relational schema describes entities, their attributes and relations between them.

In Bayes networks, there is one graph that represents the structure and dependencies. While in PRMs, there are three components: skeleton, model, and ground graph.

A university registration problem is used to illustrate a PRM [19]. The task defines three classes of instances: Course, Student, and Registration. A *skeleton* is a set of instances for each class as well as the relationships among them.

One crucial difference between a PRM and a BN is that a PRM defines the de-

pendency model between attributes at the class level. Figure 6.2 demonstrates the dependency structure in the university registration problem. The local probability model, which specifies CPDs, can rely on a set of simple attributes and an aggregation of a set of attributes of related entities. For example, in the university registration problem shown in Figure 6.2, a Student's GPA will depend on an aggregate property of the set of all of his Registrations: the average Grade.

During inference, the skeleton and model are rolled-out to get an ground graph, i.e., a massive Bayes network for inference.

6.1.2 Dependency networks and relational dependency networks

Dependency networks (DNs) [23] approximate the joint distribution with a set of conditional probability distributions which are learned independently, i.e., dependency networks model the pseudo-likelihood. Thus DNs are an approximate representation and are not guaranteed to specify a consistent joint distribution.

Relational dependency networks (RDNs) are a relational extension to DNs [25]. In RDNs, there are three graphs: the data graph, the model graph, and the inference graph, corresponding to skeleton, model, and ground graph in PRMs. The model graph captures the dependency among the variables and the data graph represents the dataset. And the inference is carried out over a massive dependency network. Relational dependency networks learn local CPDs independently due to the use of pseudo-likelihood learning techniques. A relational learning approach, Relational Probability Trees, is used in standard relational dependency networks to model the local CPDs [25]. Gibbs sampling is applied to estimate the full joint distribution and extract probabilities of interest in RDNs.

6.1.3 Conditional random fields and relational Markov networks

Conditional Random Fields [36] are an undirected graphical model (Markov networks) widely used for labeling and segmenting structured data, such as sequences, trees and lattices. A CRF defines a global log-linear conditional probability of $\mathbf{Y}$ given $\mathbf{X}$ in a discriminative manner. Let $G = (V, E)$ be a graph such that so that $\mathbf{Y} = (\mathbf{Y}_v)_{v \in V}$ is indexed by the vertices of $\mathbf{Y}$. The joint distribution over given has the form

$$p(\mathbf{Y}|\mathbf{X}) \propto \exp\left(\sum_{e \in E,k} \lambda_k f_k(e, \mathbf{y}|_e, \mathbf{x}) + \sum_{v \in V,k} \mu_k g_k(v, \mathbf{y}|_v, \mathbf{x})\right)$$

, where $\mathbf{Y}|_s$ is the set of component of $\mathbf{Y}$ associated with the vertices in subgraph s, f_k and g_k are feature functions, and λ_k and μ_k are parameters to be estimated from training data. CRFs offer several advantages, such as the ability to relax strong independence assumptions and avoid the label bias.

Taskar et al introduced relational Markov networks [58], which are the relational version of Markov networks. A RMN specifies a set of cliques and potentials between attributes of related entities at a template level. Usually the cliques are instantiated by a certain relational template and the corresponding clique potentials are also in a relational setting up. Given a relational dataset, the RMN produces an unrolled Markov network over all the attributes in the data.

Unlike RDNs, there is no pseudo-likelihood technique in RMN to facilitate the calculation of the joint probability. Loopy belief propagation is usually applied for the inference over RMNs, and learning RMNs is in general an iterative optimization procedure requiring the inference procedure.

6.1.4 Markov logic networks

Domingos et al. proposed Markov logic networks [29, 38, 55]. A Markov logic network [55] (or MLN) is a uniform framework to describe and also relate the existing statistical relational learning approaches, such as relational Markov networks and relational dependency networks. An MLN is a first-order logic with a weight attached to each formula. The weights associated with the formulas in an MLN jointly determine the probabilities of those formulas in the form of a log-linear model. An MLN defines a probability distribution over possible worlds. Domingos and Richardson showed that many existing statistical relational learning approaches are special cases of Markov logic, for example, formulas and corresponding weights can be viewed as a template for constructing Markov networks. Inference in MLNs can be performed using standard Markov network inference techniques, such as Gibbs sampling, loopy belief propagation, or approximation via pseudo-likelihood, over the minimal subset of the relevant Markov network required for answering the query.

6.2 Other Related Work

6.2.1 First-Order learner (FOIL) for hypertext classification

Sean Slattery studied webpage classification in his thesis [57] and explored the application of a First-order learner (FOIL) for hypertext classification. FOIL learns a set of rules based on the hyperlinks and the key words in a document to classify webpages. Slattery proposed an extension of FOIL, FOIL-PILFs, which uses both the hyperlinks and document content effectively.

6.2.2 Conditional graphical models

Conditional graphical models [51] are a modification of CRF-like algorithms to solve multi-class problems. In conditional graphical models, the CRF loss-function is upper bounded in order to obtain an optimization functional, which is easier to optimize via decomposing the training into independent optimization problems per clique. The decomposition in this model also allows for using large-scale training datasets and application over more complex graphs. It is also shown that training per clique in conditional graphical models is more precise than training of the sequences as a whole.

6.2.3 Associative Markov networks

Associative Markov networks (AMN) [59] aim at a subclass of Markov networks where the clique potential favors the same class labels for all variables in the clique. A linear programming relaxation of the MAP problem is proposed for max-margin training of associative Markov networks. Hence a polynomial time algorithm can be reached. For binary classification problem, AMNs are guaranteed to find the optimal solution.

6.2.4 Languages for statistical relational learning

There have also been studies on the languages for statistical relational learning includes "PRL: A probabilistic relational language" [20] and "Blog: Probabilistic Models with Unknown Objects" [44]. Their work describe a set of languages to define probability models over attribute uncertainty, structural uncertainty, and identity uncertainty. This work aims to develop language and standard for the representation of relational domains.

6.2.5 Two-stage CRFs

Krishnan and Manning [35] independently developed a "two stage" learning method for Named Entity Recognition, in which predictions from one CRF are used to generate predictions for another. This method is like Cohen and Carvalho's stacked CRFs [5], but in Krishnan and Manning's experiments, they used different functions to aggregate the predictions of the base classifier. Stacked graphical models are a generalization of Krishnan and Manning's method.

6.2.6 Piecewise CRFs

McCallum and Sutton introduced parameter independence diagrams for introducing additional independence assumptions into parameter estimation for efficient training of undirected graphical models[43]. Their method is a pseudo-likelihood measure for Markov fields and obtained a gain in accuracy via training in less than one-fifth the time.

Our work is focusing on an efficient approach for relational data and the gain is primarily in inference time, though an online version has been studied for efficient training.

6.2.7 Sub-sampling techniques for mining massive relational databases

Stacked graphical learning provides an efficient approach for collective classification, as well as a solution for large-scale relational datasets.

Hulten et al. address the problem of applying statistical relational learning to massive datasets [24]. Their work is based on applying sampling techniques. The first way they used is to minimize the number of instances and relationships that

need to be read, while ensuring that the sufficient statistics (and consequently the model) obtained from them is essentially the same that would be obtained from the full dataset. The second is to minimize the number of instances that are used in computing an aggregate (e.g., sum, average, count), again ensuring that the result is not significantly different from what we would obtain using all the relevant instances.

6.2.8 Aggregation and relational template design

The choice of relational template and proper aggregation are important in modeling relational datasets. Perlich and Provost studied aggregation for relational learning [52]. They present a hierarchy of relational concepts, with increasing complexity, and derive several classes of aggregation operators that are needed to express and learn these concepts. An analysis of desirable properties of aggregation operators is also discussed in Perlich's work. They also explored target-dependant aggregation and demonstrated empirically that in a noisy domain a complex aggregation can improve performance.

6.3 Comparison of stacked graphical learning to other relational models

The biggest difference between stacked graphical learning and other relational graphical models is the learning and inference strategy. Stacked graphical learning is a meta-learning schema which is used to augment a base learner, which does not require the iterative optimization of a graphical model. Instead, the learning of stacked graphical models is the learning of a base learner. The inference in stacked graphical does not require many iterations, compared to Gibbs sampling in RDNs.

Chapter 7

Conclusions and Future Directions

In this chapter, we summarize the major research results presented in this thesis, and discuss some future directions.

7.1 Contributions

The contributions of this thesis can be summarized as follows,

1. This thesis proposes a framework for a statistical relational learning model, *stacked graphical learning*, which allows fast inference for collective inference. We formally analyze an idealized version of the algorithm and provide theoretical proof for the convergency, in order to better understand its performance.

2. This thesis evaluates the proposed approach with many real problems including collective classification, sequential partitioning, information extraction and demonstrates the accuracy and efficiency of stacked graphical learning. Also this thesis surveys a number of state-of-art models and compares their performance to stacked graphical learning.

3. This thesis provides four in-depth case studies. In the case study, we not only provide the performance of stacked graphical learning to solve collective classification problem, but also analyze how to tune the setup to further improve the accuracy and explore the application in multi-task systems to multi-task learning.

4. This thesis extends the standard stacked graphical learning to an online version, to save training time and to handle large streaming datasets with minimal memory overhead. Analysis of the time and memory cost of online stacked graphical learning is also provided.

7.2 Related Publications

Part of the thesis work have been published in major conferences of computational biology and data mining. Below is an incomplete list. The following publications are related to Chapter 2, Chapter 3, and Chapter 4:

- Zhenzhen Kou, William W. Cohen, and Robert F. Murphy (2007). *A Stacked Graphical Model for Associating Information from Text And Images In Figures*, Pacific Symposium on Biocomputing, 2007.

- Zhenzhen Kou and William W. Cohen (2007). *Stacked Graphical Models for Effcient Inference in Markov Random Fields*, SIAM Conference on Data Mining (SDM07), 2007.

The following publication is related to Chapter 5:

- Zhenzhen Kou, Vitor R. Carvalho and William W. Cohen (2007). *Online Stacked Graphical Learning*, to be presented at NIPS 2007 Workshop on Efficient Machine Learning.

7.3 Future Directions

There are many interesting future research directions that can be explored, including:

Application to large-scale datasets The goal of stacked graphical learning is to do efficient learning and inference for classifying relational data. We have applied the algorithm to several real problems and provided several in-depth case studies. However, the largest dataset we have explored is the job title classification dataset with about 20,000 instances. We plan to evaluate stacked graphical models with much larger datasets to demonstrate its efficiency.

Explore the "stacking" procedure more We have proved the convergence of the idealized version of stacked graphical learning. The idealized stacked graphical learning is identical to the stacked graphical learning algorithm introduced in Chapter 2, only if (1) we assume that the relational template $C(x_i, \mathbf{y})$ returns $\mathrm{MB}_i(\mathbf{y})$, (2) the predictions from cross-validation are comparable to sampling from the previous stage. Therefore exploring how the cross-validated predictions simulate the sampling procedure will provide more theoretic support to our approach.

Relational template design The choice of relational template and proper aggregation are important in modeling relational datasets, especially in stacked graphical models we use relational template to generate new features to capture the dependencies among instances. So far the choice of relational template in our work is very empirical. Studies on relational concepts and relational autocorrelations will provide guidelines to the relational template design.

Appendix A

Detailed t-test results

Here are the detailed t-test results for each task.

The following tables show the results of t-test on eight datasets: SLIF Task1, WebKB, Cora, and Citeseer datasets for collective classification; and UT, Yapex, Genia, CSpace corpus for named entity extractions. Each element in the table records the significance score of the corresponding t-test, i.e., model i in the row does not achieve an accuracy different than model j in the column. If the significance score is lower than 0.05, the null hypothesis is rejected, i.e., the difference between the performance of model i in the row and the performance of model j in the column is considered statistically significant. Since most of the significance score is very small, such as "5.73e(-9)", we denote all the small numbers with "$<$ .01" and only list the exact significant score if it is larger than .01.

Table A.1: Evaluation with t-test on collective classification: SLIF data.

	Local	RDN	RMN	Naive	Stack (k=1)	Stack (k=2)
Local		$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
RDNs			.039	$< .01$	$< .01$	$< .01$
RMNs				$< .01$	$< .01$	$< .01$
Naive model					$< .01$	$< .01$
Stacked (k=1)						0.245
Stacked (k=2)						

Table A.2: Evaluation with t-test on collective classification: WebKB data.

	Local	RDN	RMN	Naive	Stack (k=1)	Stack (k=2)
Local		$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
RDNs			0.190	$< .01$	0.431	0.405
RMNs				$< .01$	0.478	0.452
Naive model					$< .01$	$< .01$
Stacked (k=1)						0.597
Stacked (k=2)						

Table A.3: Evaluation with t-test on collective classification: Cora data.

	Local	RDN	RMN	Naive	Stack (k=1)	Stack (k=2)
Local		$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
RDNs			0.371	$< .01$	0.533	0.534
RMNs				$< .01$	0.278	0.391
Naive model					$< .01$	$< .01$
Stacked (k=1)						0.849
Stacked (k=2)						

Table A.4: Evaluation with t-test on collective classification: CiteSeer data.

	Local	RDN	RMN	Naive	Stack (k=1)	Stack (k=2)
Local		$< .01$	$< .01$	$< .01$	$< .01$	$< .01$
RDNs			0.295	0.242	0.035	0.035
RMNs				0.547	0.021	0.021
Naive model					0.031	0.031
Stacked (k=1)						0.903
Stacked (k=2)						

Table A.5: Evaluation with t-test on named entity extraction: UT data. "Stack (seq.)" denotes the stacked model with *sequential* relational template, "Stack (rel.)" denotes the stacked model with *relational* relational template.

	CRFs	Stacked CRFs	Naive	Stack (seq.)	Stack (rel.)
CRFs		$< .01$	$< .01$	$< .01$	$< .01$
Stacked CRFs			$< .01$	0.903	$< .01$
Naive model				$< .01$	$< .01$
Stacked (seq.)					$< .01$
Stacked (rel.)					

Table A.6: Evaluation with t-test on named entity extraction: Yapex data."Stack (seq.)" denotes the stacked model with *sequential* relational template, "Stack (rel.)" denotes the stacked model with *relational* relational template.

	CRFs	Stacked CRFs	Naive	Stack (seq.)	Stack (rel.)
CRFs		$< .01$	$< .01$	$< .01$	$< .01$
Stacked CRFs			0.041	0.943	$< .01$
Naive model				0.048	$< .01$
Stacked (seq.)					$< .01$
Stacked (rel.)					

Table A.7: Evaluation with t-test on named entity extraction: Genia data."Stack (seq.)" denotes the stacked model with *sequential* relational template, "Stack (rel.)" denotes the stacked model with *relational* relational template.

	CRFs	Stacked CRFs	Naive	Stack (seq.)	Stack (rel.)
CRFs		$< .01$	$< .01$	$< .01$	$< .01$
Stacked CRFs			0.957	0.998	0.831
Naive model				0.956	0.828
Stacked (seq.)					0.830
Stacked (rel.)					

Table A.8: Evaluation with t-test on named entity extraction: CSpace data. "Stack (seq.)" denotes the stacked model with *sequential* relational template, "Stack (rel.)" denotes the stacked model with *relational* relational template.

	CRFs	Stacked CRFs	Naive	Stack (seq.)	Stack (rel.)
CRFs		$< .01$	$< .01$	$< .01$	$< .01$
Stacked CRFs			$< .01$	0.759	$< .01$
Naive model				$< .01$	0.908
Stacked (seq.)					$< .01$
Stacked (rel.)					

Bibliography

[1] J. Besag. Efficiency of pseudolikelihood estimation for simple gaussian fields. *Biometrika*, 64:616–618, 1977.

[2] R. Bunescu and et al. Comparative experiments on learning information extractors for proteins and their interactions. *Artificial Intelligence in Medicine*, 33:139–155, 2005.

[3] V. R. Carvalho and W. W. Cohen. Learning to extract signature and reply lines from email. In *Proceedings of CEAS 2004 - First Conference on Email and Anti-Spam*, Mountain View, CA, 2004.

[4] V. R. Carvalho and W. W. Cohen. Single-pass online learning: Performance, voting scheme and online feature selection. In *Proceedings of KDD-2006*, Philadelphia, PA, 2006.

[5] W. W. Cohen and V. R. Carvalho. Stacked sequential learning. In *Proceedings of Nineteenth International Joint Conferences on Artificial Intelligence*, Edinburgh, Scotland, 2005.

[6] W. W. Cohen, E. Minkov, and A. Tomasic. Learning to understand web site update requests. In *Proceedings of Third International Joint Conference on Artificial Intelligence*, Edinburgh, Scotland, 2005.

[7] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: Combining semi-markov extraction processes and data integration methods. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[8] W. W. Cohen, R. Wang, and R. F. Murphy. Understanding captions in biomedical publications. In *Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2003)*, Washington, DC, 2003.

[9] N. Collier and et al. The genia project: Corpus-based knowledge acquisition and information extraction from genome research papers. In *Proceedings of EACL-99*, 1999.

[10] K. Crammer, O. Dekel, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. In *Proceedings of NIPS 03*, Bonn, Germany, 2003.

[11] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the world wide web. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, 1998.

[12] I. Dagan, Y. Karov, and D. Roth. Mistake-driven learning in text categorization. In *Proceedings of EMNLP*, 1997.

[13] T. G. Dietterich, A. Ashenfelter, and Y. Bulatov. Training conditional random fields via gradient tree boosting. In *Proceedings of the Twenty-first International Conference (ICML-04)*, Banff, Alberta, Canada, 2004.

[14] K. Franzé and et al. Protein names and how to find them. *International Journal of Medical Informatics*, 67:1–3, 2002.

[15] Y. Freund, Y. Mansour, and R. E. Schapire. Why averaging classifiers can protect against overfitting. In *Proceedings of the Eighth International Workshop on Artificial Intelligence and Statistics*, 2001.

[16] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. In *Computational Learing Theory*, pages 209–217, 1998.

[17] Y. Freund and R. E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–1296, 1999.

[18] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of Nineteenth International Joint Conference on Artificial Intelligence*, Stockholm, Sweden, 1999.

[19] L. Getoor. *Learning Statistical Models from Relational Data*. PhD thesis, Stanford University, June 2002.

[20] L. Getoor and J. Grant. Prl: A probabilistic relational language. *Machine Learning*, 62:7–31, 2006.

[21] Z. Ghahramani. Learning dynamic Bayesian networks. *Lecture Notes in Computer Science*, 1387:168–197, 1998.

[22] C. L. Giles, K. Bollacker, and S. Lawrence. Citesee: An automatic citation indexing system. In *Digital Libraries 98 - The Third ACM Conference on Digital Libraries*, Pittsburgh, PA, 1998.

[23] D. Heckerman, D. M. Chickering, C. Meek, R. Rounthwaite, and C. M. Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

[24] G. Hulten, P. Domingos, and Y. Abe. Mining massive relational databases. In *the IJCAI-2003 Workshop on Learning Statistical Models from Relational Data*, Acapulco, Mexico, 2003.

[25] D. Jensen and J. Neville. Dependency networks for relational data. In *Proceedings of 4th IEEE International Conference on Data Mining (ICDM-04)*, Brighton, UK, 2004.

[26] D. Jensen, J. Neville, and B. Gallagher. Why collective classification inference improves relational classification. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004.

[27] M. I. Jordan. Graphical models. *Statistical Science (Special Issue on Bayesian Statistics)*, 19:140–155, 2004.

[28] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An introduction to variational methods for graphical models. *Learning in Graphical Models*, Cambridge: MIT Press, 1999.

[29] S. Kok and P. Domingos. Learning the structure of markov logic networks. In *Proceedings of the Twenty-Second International Conference on Machine Learning (ICML05)*, Bonn, Germany, 2005.

[30] J. E. Kolassa. Convergence and accuracy of gibbs sampling for conditional distributions in generalized linear models. *Annals of Statistics*, 27:129–142, 1999.

[31] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of AAAI98*, Madison, Wisconsin, 1998.

[32] Z. Kou and W. W. Cohen. Stacked graphical learning for efficient inference in markov random fields. In *Proceedings of SDM 07*, 2007.

[33] Z. Kou, W. W. Cohen, and R. F. Murphy. Extracting information from text and images for location proteomics. In *Proceedings of the BIOKDD 2003*, Washington D.C., 2003.

[34] Z. Kou, W. W. Cohen, and R. F. Murphy. High-recall protein entity recognition using a dictionary. In *Proceedings of ISMB 2005*, 2005.

[35] V. Krishnan and C. D. Manning. An effective two-stage model for exploiting non-local dependencies in named entity recognition. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, Sydney, Australia, 2006.

[36] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning (ICML-2001)*, Williams, MA, 2001.

[37] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1988.

[38] D. Lowd and P. Domingos. Efficient weight learning for markov logic networks. In *Proceedings of the Eleventh European Conference on Principles and Practice of Knowledge Discovery in Databases*, Warsaw, Poland, 2007.

[39] Q. Lu and L. Getoor. Link-based classification. In *Proceedings of International Conference on Machine Learning*, Washington, DC, 2003.

[40] A. McCallum, D. Freitag, and F. Pereira. Maximum entropy Markov models for information extraction and segmentation. In *Proceedings of the International Conference on Machine Learning (ICML-2000)*, pages 591–598, Palo Alto, CA, 2000.

[41] A. McCallum and K. Nigam. A comparison of event models for naive bayes text classification. In *Proceedings of AAAI-98 Workshop on Learning for Text Categorization*, 1998.

[42] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3:127–163, 2000.

[43] A. McCallum and C. Sutton. Piecewise pseudolikelihood for efficient crf training. In *Proceedings of the International Conference on Machine Learning (ICML)*, Corvalis, Oregon, 2007.

[44] B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov. Blog: Probabilistic models with unknown objects. In *Proceedings of 19th International Joint Conference on Artificial Intelligence (IJCAI 05)*, 2005.

[45] E. Minkov, R. C. Wang, , and W. W. Cohen. Extracting personal names from emails: Applying named entity recognition to informal text. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing (HLT/EMNLP 2005)*, Vancouver, B.C., Canada, 2005.

[46] K. P. Murphy. Bayes net toolbox for matlab. *Computing Science and Statistics*, 33, 2001.

[47] R. F. Murphy, Z. Kou, J. Hua, M. Joffe, and W. W. Cohen. Extracting and structuring subcellular location information from on-line journal articles: the subcellular location image finder. In *Proceedings of the IASTED International Conference on Knowledge Sharing and Collaborative Engineering*, St. Thomas, US Virgin Islands, 2004.

[48] R. F. Murphy, M. Velliste, J. Yao, and G. Porreca. Searching online journals for fluorescence microscope images depicting protein subcellular location patterns. In *Proceedings of the 2nd IEEE International Symposium on Bio-informatics and Biomedical Engineering (BIBE-2001)*, pages 119–128, December 2001.

[49] J. Neville and D. Jensen. Iterative classification in relational data. In *Proceedings of the AAAI-2000 Workshop on Learning Statistical Models from Relational Data*, 2000.

[50] J. Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of AAAI-82*, 1982.

[51] F. Perez-Cruz, Z. Ghahramani, and M. Pontil. Conditional graphical models. In *Predicting Structured Data*, chapter 12, pages 265–282. MIT Press, 2006.

[52] C. Perlich and F. Provost. Aggregation-based feature invention and relational concept classes. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, Washington, D.C., 2003.

[53] A. Ratnaparkhi. A simple introduction to maximum entropy models for natural language processing. In *Technical Report 97-08, Institute for Research in Cognitive Science, University of Pennsylvania*, 1997.

[54] A. Ratnaparkhi. A simple introduction to maximum entropy models for natural language processing. Technical Report Technical Report 97-08, Institute for Research in Cognitive Science, University of Pennsylvania, 1997.

[55] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

[56] R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3), 1987.

[57] S. Slattery. Hypertext classification. Technical Report CMU-CS-02-142, Carnegie Mellon University, 2002. (PhD thesis).

[58] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proceedings of Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI02)*, Edmonton, Canada, 2002.

[59] B. Taskar, V. Chatalbashev, and D. Koller. Learning associative markov networks. In *Proceedings of the twenty-first international conference on Machine learning (ICML04)*, Banff, Alberta, Canada, 2004.

[60] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[61] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[62] J. Yedidia, W. Freeman, and Y. Weiss. Generalized belief propagation. In *Proceedings of Neural Information Processing Systems*, 2000.

[63] A. Zellner and C.-K. Min. Gibbs sampler convergence criteria. *Journal of the American Statistical Association*, 90:921–927, 1995.

Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213

Carnegie Mellon