Data Driven Resource Allocation for Distributed Machine Learning

Venkata Krishna Pillutla

CMU-CS-15-145

December 2015

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Maria-Florina Balcan, Chair
Alexander J Smola
Christos Faloutsos

Submitted in partial fulfillment of the requirements for the degree of Master of Science.

This research was sponsored in part by the National Science Foundation under grant numbers CCF-1451177 and CCF-1422910.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Machine Learning, Distributed ML, Fault-Tolerance, Clustering, Balanced Clustering, Sample Complexity, CTR

For my mother, father and sister.

Abstract

In distributed machine learning, data is dispatched to multiple machines for processing. Traditional distributed Machine Learning systems dealing with high volume of data randomly distribute the data among workers and either learn a linear model for each partition of data or in tandem where the workers contribute updates to one synchronized global model. Motivated by the fact that similar data points are often belonging to the same or similar classes, and more generally, classification rules of high accuracy tend to be "locally simple but globally complex" [68], we propose data dependent dispatching that takes advantage of such structure. In particular, we propose to use balanced clustering to this end.

In practice, it is crucial for such a system to have redundancy, for *fault tolerance* since the machines are not assumed to be reliable, and *load-balancing* for good utilization and throughput. Since the model served in each machine is not going to be the same, queries should be served to the appropriate server during deployment. Moreover, since queries are to be responded in real-time, reponse time shuold be fast.

Since clustering large datasets in their entirety is time consuming, we propse to cluster an initial sample in a balanced and fault-tolerant manner, and 'extend' the clustering to the rest of the dataset. Our main technical contribution is to provide algorithms with provable guarantees for data-dependent dispatching, that partition the data in a way that satisfies important conditions for effective distributed learning, including fault tolerance and load-balancing. For clustering, we present approximation algorithms with provable guarantees for the NP-hard problem of balanced clustering with fault tolerance. For dispatch, we use ideas from Nearest Neighbor Classification to show that our dispatcher respects locality and load balancing properties of the clustering.

In particular, we provide bicriteria approximation algorithms that use LP rounding techniques for balanced clustering with fault-tolerance that achieve constant factor approximations (5, 11, 95 respectively) to the k-center, k-median, and k-means objectives, and violate cluster size constraints by a small constant multiplicative factor. Moreover, we show a more complicated true approximation algorithm that achieves a 6-approximation for the balanced k-center problem with fault tolerance and this does not violate any of the constraints.

To extend the clustering to the rest of the training set, we propose *Nearest Neighbor Extension* (also knows as Nearest Neighbor Dispatch). It is an online, efficient and provably correct procedure that returns a balanced clustering of the entire dataset. During deployment, we use the same algorithm for dispatching points to appropriate machines, and we show that it is load-balancing.

After data-dependent partitioning, learning can be performed independently with no communication, or with complete communication, with each machine storing a local correction. We discuss how to efficiently implement our method approximately and show its effectiveness over the widely used random partitioning scheme with the same communication budget on several real world image and advertising datasets. For example, we obtain around 14% improvement over a single global model on the MNIST-8M dataset using the communication free model.

Acknowledgments

First and foremost, I'd like to thank my awesome advisor Nina Balcan, for the firm belief, constant encouragement, mentoring and guidance. I have truly learnt a whole lot in the short span of one year. She took me in, gave me an amazing research project to work on and a great office to work in (not common for Masters students). The discussions and technical insights have had a profound impact on me and only increased my interest in Machine Learning. I've had the freedom to explore but the guidance was always there when I needed it. I shall always strive to match Nina's energy, enthusiasm and dedication, but it is a daunting task. Nina also ensured that all her students worked on their soft skills - speaking, writing and presenting - and the improvement, in my case, is quite substantial. I'm looking forward to work closely with and continue to learn from Nina in the coming years.

I would like to whole-heartedly thank my committee members Alex Smola and Christos Faloutsos. Alex has always been a source of great technical insight and am thankful for his time and willingness to share some wisdom with me. It has been a tremendous pleasure to work with Christos, albeit not so much on this thesis. I am thankful to him for one of the most fun classes I have taken and for guiding me on an extremely interesting and related research topic. I am also very grateful to Dave and Tracy for making things easy so I could focus on the research, and for being extremely patient in answering my possibly inane queries.

It has been my greatest pleasure to work with wonderful collaborators and co-authors. Travis Dick and Colin White have been ever-present and have made many a summer afternoon fun and exciting. Our long and fruitful discussions have left me wanting for more. We do make a great team. Thanks are in order to Mu Li for his tremendous insights and constant guidance on the experimental side of things.

I would like to thank all my great friends, bandmates and music buddies, gym buddies and basketball mates. Life wouldn't even have been half as much fun without you guys.

Finally, to my two greatest role models ever - my mom and dad, and my favorite person in the whole world - my sister. You guys are the best. A shout out also to my awesome grandparents, cousins, aunts and uncles- I am truly lucky to have such marvellous people in my family.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advent of big data has, the last few decades have seen an explosion in distributed machine learning techniques, with work in designing new algorithms [2, 28, 53], proving better guarantees [70, 71], and desgning more efficient systems [41, 44, 48], all while pushing boundaries of scale and speed. The effect has been further enhanced by ubiquity of clusters of several low cost, commodity machines.

On the algorithms side, amount of communication has emerged as a key resource [6] with network bandwidth often being the bottleneck. Algorithms that minimize the delays caused by communication and synchronization and hence increasing CPU utilization have seen a rise in popularity and relevance, as have designing new algorithms with provable guarantees and analyzing statistical performance of the proposed techniques. On the systems side, work has gone into creating abstractions that hide low-level details from the user enabling faster development of big machine learning algorithms [25, 44, 48, 52].

Data collection in ML tends to take two high-level forms. The first is when the data itself is collected in a distributed manner, whether from geographically-distributed experiments, distributed sensors, distributed click data, etc., and the goal is to design an intelligent learning algorithm that can take advantage of all this data without incurring the substantial overhead of first communicating it all to some central location. The Yahoo! PNUTS database [23] is an example - this database is geographically distributed. The second high-level form, is where massive amounts of data are collected centrally, and for space and efficiency reasons, this data must be dispatched to distributed machines in order to perform the processing needed [48, 71]. It is this latter form that this thesis addresses, and the core objective is to provide algorithms with the best possible guarantees given the available data by most efficiently using the given resources available.

For instance, consider the multi-billion dollar online advertising industry. Predicting Click Through Rate (CTR) of advertisements [2, 16] is of paramount importance. Given petabytes of past data and some information about a new user, the problem is to predict whether the user will click on a particular advertisement. Each time a user opens a web-page, the query is passed to the central server, dispatched to the appropriate machine, and the based on the result, an appropriate advertisement is displayed. Speed is of essence in this application, and this requirement is a driving factor in design decisions.

When data is dispatched to distributed machines, the simplest approach and what past theo-

retical work has focused on is to perform the dispatching randomly [70, 71]. Random dispatching has the advantage that dispatching is easy, and also, because each machine receives data from the same distribution, it is rather clean to think about theoretically. However, when learning with no communication, this method is statistically sub-optimal, since the model on each machine needs to be essentially identical. As an example, one-hot encoding is a popular scheme for CTR data [42]. The dimensionality of the generated representation can be very large, in the order of many millions. Feature occurence approximately follows a power law [22, 56], and parameters corresponding to the rare keys (features in long tail) cannot be learnt reliably in this manner.

Motivated by the fact that in practice, similar data points tend to have the same or similar class, and more generally, classification rules of high accuracy tend to be "locally simple but globally complex" [68], this thesis investigates a new paradigm for doing *data dependent dispatching* that takes advantages of such structure. Indeed, Jacobs and Jordan have proposed mixtures of experts in the 90's [34, 36] to learn a (data-dependent) soft-partitioning along with the experts. We take an alternate route and explore clustering as a means of data dependent partitioning. We first cluster a small sample and an efficient dispatcher to cluster the rest of the dataset. The outline of our algorithm is given in algorithm 1.

Clustering: Cluster a small segment of the data with some balanced clustering algorithm
Assign clusters to machines to balance load
Dispatcher: Dispatch incoming points to clusters efficiently
Learning: Use your favorite learning algorithm on each machine independently or in tandem

Algorithm 1: High level overview of the proposed paradigm

In distributed systems, load-balancing and fault-tolerance are of prime importance, and are the central requirements of several web scale distributed systems such as Amazon Dynamo [27], Yahoo! PNUTS [23] and others [19]. For instance, Dynamo and PNUTS have high redundancy for fault-tolerance. For this reason, we require our clustering and dispatch to have these properties. Further, ensuring varying degrees of read and write consistency is of great importance in these systems, but for this thesis, we shall assume that our data is static. That is, we work under the simplifying assumption that no writes are made to existing data in our system. This is reasonable assumption in our setting. For example, in the CTR example, new data can be collected over a period of time (say, for a day) and all updates can be applied at the end of that period but existing data is not modified.

Incremental scalability is another requirement - given $\mathcal{O}(1)$ new machines, the system must be able to scale efficiently by moving the minimum amount of data possible. Dynamo, for instance, achieves this by using Consistent Hashing [37, 38].

Once the data is distributed to various machines, learning algorithms with or without any communication can be used. For instance, one could train an independent model "local" on each cluster, and this would be embarrassingly parallel, that is, free of communication and synchronization. This gives the freedom to use sophisticated training algorithms on each cluster, including ones that are difficult to perform in the distributed setting. As we shall see, this approach works best for low dimensional data such as the image data. On the other hand, one can

synchronize the local models on parameters corresponding to infrequent features in sparse high-dimensional data such as the CTR data. The communication cost of this model depends on how many parameters are being synchronized. At the other end of the spectrum of communication complexity is the model that maintains a local correction to a fully synchronized global model for each cluster. This approach works best when there is insufficient data in each cluster even for the head features, as is the case for the CTR data.

Intuitively, this can be explained by the bias-variance trade off. If we restrict ourselves to linear models for tractability, our scheme gives rise to a piecewise linear model, with each piece defined over a cluster. This gives us a more expressive class of functions, over having a single global linear model. In other words, the bias is reduced. For the no communication scheme, for low dimensional data, given enough datapoints per cluster (e.g. MNIST-8M), we expect the variance to be low as well, and our method can outperform a single global linear model. However, for high dimensional datasets, of datasets where clusters do not have enough data, the variance will be quite high, and so the performance will not match the performance of having a single global model, as in the CTR datasets. On the brighter side, the reduced bias means performance will be better than random partitioning. Moreover, in these scenarios, some amount of synchronization on features in the long power law tail will increase the bias but reduce the variance to a larger extent and performance is expected to improve.

Throughout the thesis, we use machines as an abstraction. By a machine, we refer to a virtual machine that can be physically be implemented by a CPU, a computer, a rack in a data-center or even an entire data-center.

Problem Statement and Objectives

Formally, we are given a big dataset of N points in $\mathbb{R}^q$ and k machines. The problem we solve is to efficiently divide the big dataset into k parts, one for each machine and then learn a linear model over each part either independently or jointly with the final goal of maximizing classification accuracy. Further, we must be able to efficiently dispatch test points to appropriate machines and answer queries in real time with the learnt models. The approach we use is balanced clustering coupled with an appropriate dispatch procedure.

As motivated earlier, we require load-balancing and fault-tolerance in addition to efficiency. For load-balancing in our application, we require clusters to be of roughly the same size. In other words, we should be able to enforce upper and lower bounds on the cluster sizes. This is called balanced clustering and is not very well understood. In past literature, clustering with upper bounds on cluster sizes, also known as capacitated clustering is well studied [3, 17, 40, 49]. On the other hand, clustering with lower bounds on cluster sizes [29] is much less studied. Recent work showed that having lower bounds on cluster sizes can make the problem much harder [7]. Further, in many real distributed systems, low cost machines are used. A given machine can fail at any time. For this reason, we would like some amount of replication in the clustering for fault-tolerance. Replication also has the benefit that boundary points are replicated across multiple clusters, thereby improving predictive properties. To sum up, the first concerete problem we tackle is of balanced clustering with fault-tolerance. Given parameters k, ℓ, L, p, the number of clusters, lower and upper bounds on cluster sizes and the replication factor respectively, we wish to find the best k-clustering according to some objective function (e.g. k-median) where each

cluster has between ℓ and L fraction of the data and each point is replicated across p different clusters.

In our setting, the dataset is very large and is stored in a distributed manner. Clustering the entire dataset is too expensive. Hence, the clustering is only performed on an initial sample, and the clustering has to be "extended" to the rest of the dataset by some dispatch procedure. Further, unseen query points during deployment should also dispatched. For these reasons, we should have an efficient dispatch procedure that maintains the favorable qualities of load balancing and fault tolerance of the clustering while still preserving locality. Moreover, because of the cluster size constraints, the cluster to which a point belongs to may no longer be the nearest cluster, as in center based clustering. This requires a more sophisticated dispatch rule, that maintains correctness and is also efficient during deployment. For instance, in the aforementioned online advertising application, the request has to be correctly dispatched and served, crucially within a tenth of a second. Specifically, given a balanced clustering on a sample, and given a new point, we wish to assign the new point to p different *nearby* clusters so that each cluster gets assigned the new point with probability approximately between ℓ and L. By Hoeffding's inequality, this gives load balancing during deployment.

Lastly and most importantly, we wish to obtain maximum performance. Data driven resource allocation is just a tool that helps achieve locality, but obtaining best possible performance is the ultimate objective.

Contributions and Outline

Chapter 2 surverys relevant literature.

Clustering is NP-hard. Adding balancing contraints only makes it harder. Given the erratic behavior of balanced clustering [7], it is not clear whether this can even be approximated. In chapter 3, we answer this question positively, showing provably correct polynomial time constant-factor and bicriteria approximation algorithms. In particular, we provide an algorithm that returns a constant factor approximation (11, 95, 5 respectively) for balanced k-median, k-means, k-center while violating the upper bound on cluster sizes and fault tolerance by a factor of at most 2 each. Moreover, we provite a true 6-approximation algorithm for balanced k-center with fault tolerance that does not violate any constraints. Both of these algorithms solve the LP relaxation of the natural LPs of these problems and round the fractional variables into integers in a controlled manner that leads to a provable approximation factor. While we do not explicity design our system for incremental scalability, one can achieve it by playing the doubling trick. If k machines are available, one can start off with $2k$ clusters, with each machine serving 2 clusters. Each time we get a new machine, it can pick up one cluster from machines that serves two. This requires re-clustering, an expensive operation, only when the number of machines have been doubled and the cost of re-clustering is amortized over the k machines added.

For the dispatcher, in Chapter 4 we propose the provably correct Nearest Neighbor Dispatch, an efficient online procedure based on ideas from Nearest Neighbor Classification. Since balanced clustering is not center based, we assign a new point to the same clusters as its nearest neighbor from the sample. Much work goes into proving correctness - that NN dispatch returns a load-balanced clustering with good locality, where the objective function is a proxy for locality. We analyze the size of the initial sample required for good dispatch, under mild regularity

conditions. This step is potentially much more expensive over existing distributed systems, especially in applications that require real time responses. To efficiently implement nearest neighbor search, one could use effective approximate NN search techniques such as Locality Sensitive Hashing (LSH) [4, 63] and Random Partition Trees [26]. Hashing, in particular, is amenable to efficient hardware implementations, and this can make our proposed system a viable alternative, especially in the light of improved performance obtained in our experiments.

For the learning part, we try out various settings in Chapter 5 - ranging from totally communication free to full communication and provide a recipe for when to use each setting. We compare favourably against the popular random partitioning with no communication and full communication (equivalent to running on a single machine) on large, real life image and CTR data. In particular, we observed a 14% increase in classification accuracy on the MNIST-8M dataset, and a very significant 1% increase on some CTR data. This section is more a proof of concept of the novel idea of data driven partitioning rather than a full blown implementation of a real system. Further performance boosts can be obtained by caching results of previous queries. Yahoo!'s Content Delivery System, for example, uses an intelligent distributed cachcing scheme [19] that is designed to balance load.

This theis is join work with Maria-Florina Balcan, Travis Dick, Mu Li, Alex Smola and Colin White [7].

Chapter 2

Literature Review

In this chapter, we shall first review various distributed computing paradigms, survey the literature on balanced clustering and dispatch methods.

2.1 Distributed Learning Paradigms.

The most commonly used paradigm is when the data is randomly partitioned among the worker machines [55, 70, 71]. Each independently learnt model can be used to answer queries or the learnt models can be averaged to give a single model [70]. Averaging is quite popular, and various weighted averaging schemes have emerged, e.g. [2]. Another approach is to to take a majority vote of all the learnt models for each query point [55]. While it has better predictive properties, majority vote is much slower than the previous approaches.

Massively Distributed Systems. Dynamo [27], Amazon's highly available distributed key-value store, has moved away from traditional distributed databases in order to improve availability. They trade-off consistency (and the typical ACID properties) for availability by settling for a much weaker guarantee of eventually consistency. Dynamo has redundancy and replication for fault-tolerance, partition-aware client library for load-balancing, object versioning for consistency and consistent hashing [37, 38] for incremental scalability. It is fully decentralized too.

Consistent Hashing [37, 38] is a key component of Dynamo and other distributed systems. It is a hashing scheme that is relatively robust to small changes in the number of buckets. Traditional hashing schemes requires re-hashing even for ± 1 buckets, and this means a lot of the data has to be moved around in the context of distributed systems. Consistent hashing is designed to minimize this movement.

Yahoo!'s PNUTS [23] is a centrally-managed, geographically distributed database. Each record has multiple replica for fault tolerance. PNUTS has a stronger consistency guarantee than Dynamo: per-record timeline consistency. Degree of read consistency can be specified too. Load-balancing happens is automated. Records are grouped in to blocks called tablets (size$\sim$ 1G), and assigment of tablets to servers is flexible - this allows for load-balancing.

Yahoo!'s Content Delivery system (e.g. for videos) uses some interesting ideas in caching. Requests for content are served by multiple servers, and these servers cache the most recent requests. The problem becomes difficult because most content is unpopular where some content is extremely popular (typically, power law behavior is exhibited). Static indexing is not incrementally scalable. The solution here is to use consistent hashing again. If there is no machine serving the hashed value, it hashed repeatedly until it can be served. Similarly, to balance load, if some content is extremely popular and multiple requests are received in quick succession for the same content, the request is passed on (by hashing the hased value again). This simple scheme achieves load balancing, while still retaining good properties of hashing.

The massively popular Hadoop Distributed File System [41] has replication and fault-tolerance, although the partitioning is not data aware. In its default setting, HDFS stores three copies of each file, two of which are on the same rack at the data center.

Distributed ML platforms With the rising popularity of distributed machine learning, several platforms have emerged. The Parameter Server [48] is an abstraction that uses servers to store parameters and workers to pull the values of the parameters, perform computations and push updates. Petuum [25], MLBase [44], GraphLab [52] are other such systems.

Distributed Optimization Orthogonal to our work, there has been a lot of work about optimization schemes in a distributed setting. Primal-dual distributed optimization schemes are particularly well studied: Alternating Direction Method of Multipliers [13], COCOA [35] and its follow up COCOA+ [53] are popular. It is common for these methods to work with the dual problem because this often gives a natural method of combining solutions from different workers. These methods and their convergence guarantees are agnostic of how the data is distributed across various machines because a single global model is learnt over the entire dataset. However, the rate of convergence may depend indirectly on the distribution, e.g. eigenvalues of the data matrix [35, 53]. Stochastic gradient methods, particular asynchronous methods [28, 59, 65, 72] are also immemsely popular in the distributed setting. Asynchronous methods can perform more computation without the overhead of synchronization but primal-dual methods enjoy faster convergence.

2.2 Locally Simple but Globally Complex Models

Jacobs and Jordan [34, 36], explored the mixtures of experts, where each expert would specialize in a particular region of the space, as decided by a gating network. The gating network and the expert would be trained together. Diving the data reduces the bias but increases the variance. The authors use soft assignments as a means to reduce variance. Further, the gating network was hierarchical with experts at the leaves. The generative model can be described as follows. Suppose $\mathbf{x}$ is the input and y is the output. For a simple two-level network (level i being the first layer, and $j|i$ being the second layer),

$$P(y|\mathbf{x}, \text{parameters}) = \sum_i g_i(\mathbf{x}, v_i) \sum_j g_{j|i}(\mathbf{x}, v_{ij}) P(y|\mathbf{x}, \theta_{ij})$$

Given $\mathbf{x}$, the posterior h_i at the first layer can be written as:

$$h_i(y|\mathbf{x}, \text{parameters}) \propto g_i(\mathbf{x}, v_i) \sum_j g_{j|i}(\mathbf{x}, v_{ij}) P(y|\mathbf{x}, \theta_{ij}).$$

Similar expressions can be derived for the intermediate and final posteriors. Training the experts and the gating network together is non-convex, and is performed by an Expectation-Maximization algorithm. Different parametric forms for $P(y|\mathbf{x}, \theta)$ gives various types of classification or regression.

There has been some past work on Local SVMs [20, 21, 60, 61] and some of these touch upon clustering befor learning on each cluster separately using different notions of clustering. However, most of these are heuristic in nature and there is no analysis of theory involved. Moreover, none of them are aimed at the distributed setting.

2.3 Clustering

Balanced Clustering. Clustering with upper bounds on cluster sizes is known in literature as *Capacitated Clustering*. There has been a whole line of work for approximation algorithms for this problem (since the problem is NP-hard) for various clustering objectives such as k-center [3, 9, 24, 40] and k-median [1, 15, 17, 49]. On the other hand, clustering with lower bounds has been shown to be much harder [7] and there is not much work on it [29]. Chapter 3 gives a more detailed review about balanced clustering.

Large-scale Clustering. Two approaches have been taken to cluster large-datasets. The first is use tailor-made approaches for Map-Reduce and make the problem tractable by sub-sampling the data. For instance, Bateni et al [10] use Mapping Coresets in a multi-stage approach to find a capacitated clustering of a large dataset on Map-Reduce. On the other hand, BOW [31] uses random sampling, takes into account the distribution of data across various machines to try and minimize IO costs and network traffic. The second is to treat the large dataset as a stream and use existing streaming approaches [18, 57].

2.4 Dispatch

Clustering the entire distribution on the basis of a sample was first considered by von Luxburg and Ben-David [69]. They study when clustering a finite sample can generalize well to a large population for center based clusterings. The dispatch rule in this case is straight forward. Clusters are Voronoi partitions about the centers in center based clustering schemes. Bubeck and von Luxburg [14] analyze the statistical consistency of clustering a small sample optimally and then extending the clustering to the rest of the dataset using *Nearest Neighbor Clustering*. The dispatch rule for a new point is to find the nearest neighbor of this point from the initial sample and assign it to the same cluster. Since they look at samples that are logarithmic in the size of the dataset, the initial clustering can be bruteforced.

An orthogonal approach is taken by Banerjee and Ghosh [8]. They cluster a small sample and try to find the best assignment respecting cluster size lower bounds using an extension of the bipartite marriage. While it is interesting in its own right, it does not apply in our setting. We require our dispatcher to be online and efficient for the applications studied here.

Chapter 3

Approximation Algorithms for Balanced Clustering

We have previously motivated the need for balanced clustering- that is, clustering with both lower and upper bounds on cluster size, and the need for fault tolerance. In this chapter, we shall present a bicriteria approximation algorithm for fault-tolerant balanced k-median, k-means and k-center on a given set of points. This algorithm violates the upper bound size constraint on a cluster by a fixed amout. Further, for k-center, we give a more involved algorithm that outputs a 6-approximation and respects all size constraints. Both algorithms are LP rounding algorithms.

Preliminaries

A clustering instance consists of a set V of n_0 points, and a distance metric d. Given two points i and j in V, denote the distance between i and j by $d(i, j)$. The task is to find a set of k centers $C = \{c_1, \ldots, c_k\}$ and assignments of each point to p of the centers $f : V \to \binom{C}{p}$, where $\binom{C}{p}$ represents the subset of C^p with no duplicates. Each objectives tries to the minimize following quantities.

- k-center: $\max_{i \in V} \max_{j \in f(i)} d(i, j)$
- k-median: $\sum_{i \in V} \sum_{j \in f(i)} d(i, j)$
- k-means: $\sum_{i \in V} \sum_{j \in f(i)} d(i, j)^2$

The k-center, k-median, and k-means objectives can be thought of as minimizing the L_∞, L_1, and L_2 norms, respectively. If we add size constraints l and L, then each cluster must have between ℓ and L fraction of points. For fault-tolerance, each point must be assigned to exactly p centers instead of just one. Further, assume that each point i has an integer weight w_i. This may represent, for instance the multiplicity of the point or its importance.

The parameters are summarized in table 3.1.

3.1 Prior Work

Clustering with upper bounds is known in literature as *Capacitated Clustering*.

Symbol	Description
k	Number of clusters
ℓ	Minimum fractional size of a cluster
L	Maximum fractional size of a cluster
p	Replication factor, for fault-tolerance
V	Set of points to cluster
w_j	Weight of point j
n	$\sum_{j \in V} w_j$, the total weight of all points

Table 3.1: Parameters of the problem

Capacitated k-center

The (uniform) capacitated k-center problem is to minimize the maximum distance between a cluster center and any point in its cluster subject to the constraint that the maximum size of a cluster is L. It is NP-Hard, so research has focused on finding approximation algorithms. Bar-Ilan et al [9] introduced the problem and presented the first constant factor polynomial time algorithm achieving a factor of 10, which was subsequently improved by Khuller et al [40]. It was a combinatorial algorithm that first guessed the optimal objective and a graph, G of all points with an edge between two points iff they are separated by less than the guess. They construct a set of monarchs: a maximal independent set in G^2, assign points close to the monarchs as their "empires", move points around empires and open new centers are required to satisfy the capacity constraints, while increasing the objective in a bounded manner. This essentially means that a point cannot be assigned to a cluster arbitrarily far away from it.

In [24], the capacitated k-center problem with non-uniform capacities is written as a feasible point of an integer linear program, and a procedure to round fractional solutions obtained from the LP relaxation is described. They construct a path-like tree structure with nearby vertices close in the original graph, and all vertices with fractional opening values at the leaves (which they call a "caterpillar" structure), transfer "openings" between distant vertices by transferring a limited number of clients to neighboring facilities through a chain (so as to increase the objective only in a bounded manner), and end up with all integral openings in polynomial time. Assignments are then made via a bipartite matching between points and enters as Hall's marriage theorem can now be applied. The appoximation factor is not explicitly computed, although it is mentioned to be "in the order of hundreds".

An et al [3] follows a similar procedure. They describe "tree instances" as generalizations of caterpillar structures of [24], and use a rounding procedure that is somewhat similar to the previous approach to get an approximation factor of 8. Further, for the special case of uniform capacities, they show a 6-approximation.

k-center with lower bounds

Ene et al [29] describe a 4-approximation to k-center with lower bounds by constructing r-nets. They describe an efficient algorithm to this end. The reduction is specific to this objective and does not work with upper bounds on cluster sizes.

Capacitated k-median

k-median with capacities is a notoriously difficult problem in clustering. It is much less understood than k-center with capacities, and uncapacitated k-median, both of which have constant factor approximations. Despite numerous attempts by various researchers, still there is no known constant factor approximation for capacitated k-median (even though there is no better lower bound for the problem than the one for uncapacitated k-median). As stated earlier, there is a well-known unbounded integrality gap for the standard LP even when violating the capacity or center constraints by a factor of $2 - \epsilon$ [1]. Recent work has emerged to break this integrality gap, but with a different LP [50].

Charikar et al. gave a 16-approximation when constraints are violated by a factor of 3 [17]. Byrka et al. improved this violation to $2 + \epsilon$, while maintaining an $\mathcal{O}(\frac{1}{\epsilon^2})$ approximation [15]. Recently, Li improved the latter to $\mathcal{O}(\frac{1}{\epsilon})$, specifically, when constraints are violated by $2 + \frac{2}{\alpha}$ for $\alpha \geq 4$, they give a $6 + 10\alpha$ approximation [49]. These results are all for the *hard* capacitated k-median problem. In the *soft* capacities variant, we can open a point more than once to achieve more capacity, although each extra opening counts toward the budget of k centers. In hard capacities, each center can only be opened once. The hard capacitated version is more general, as each center can be replicated enough times so that the soft capacitated case reduces to the hard capacitated case. Therefore, we will only discuss the hard capacitated case.

All of the algorithms for capacitated k-median mentioned above share the same high-level idea but with different refinements in the algorithm and analysis. They are all LP rounding algorithms. They work by first using a monarch procedure to aggregate fractional center openings, where each demand is only moved a constant factor away. Each cluster must have at least $\frac{1}{2}$ (or $\frac{\alpha-1}{\alpha}$) total opening after this step. Then we must partition the fractional openings into star structures, and round the openings within each star.

Universal and load balanced facility location

In the facility location problem, we are given a set of demands and a set of possible locations for facilities. We should open facilities at some of these locations, and connect each demand point to an open facility so as to minimize the total cost of opening facilities and connecting demands to facilities. Capacitated facility location is a variant where each facility can supply only a limited amount of the commodity. This and other special cases are captured by the Universal Facility Location problem where the facility costs are general concave functions. Local search techniques [54] have been proposed and applied successfully. Also, LP rounding techniques suffer from unbounded integrality gap for capacitated facility location [54].

Load-balanced facility location [39], [33], is yet another variant where every open facility must cater to a minimum amount of demand. An unconstrained facility location problem with modified costs is constructed and solved. Every open facility that does not satisfy the capacity constraint is closed and the demand is rerouted to nearby centers. The modified problem is constructed so as to keep this increase in cost bounded.

3.2 Bicriteria Algorithm

The bicriteria algorithm presented here is inspired by previous work on capacitated k-median [49]. However, we consider a different problem in that we have lower bounds on cluster sizes and fault tolerance. It turns out that fault tolerance makes the problem easier to solve, provided we are ready to lose a factor of 2 on fault-tolerance. Intuitively, it is because it is easier to find a nearby center with remaining capacity since with greater fault-tolerance, each center has more capacity.

3.2.1 Linear Program

Each problem can be framed as an Integer Linear Program. For each i, let y_i be an indicator for whether i is a center. For i, j, let x_{ij} be an indicator for whether point j is assigned to center i. In literature, y_i is called the *opening at point* i and x_{ij} is called the *assignment of* j *to* i.

The algorithm is an LP rounding algorithm where we consider the LP relaxation of the IP (which has the constraints that x, y are 0 or 1). Now, y_i represents the fraction to which a center is opened, and x_{ij} represents the fractional assignment of j to i. For a center i and point j, let their contribution to the objective be denoted by c_{ij}. That is, for k-median, $c_{ij} = d(i, j)$ and for k-means $c_{ij} = d(i, j)^2$. The notation for this section is summarized in table 3.2. Here is the LP

Symbol	Description	k-median	k-means	k-center
y_i	Fractional opening at center i	-		
x_{ij}	Fractional assignment of point j to center i	-		
c_{ij}	Cost of assigning j to center i	$d(i,j)$	$d(i,j)^2$	t
C_j	Avg cost of assignment of point j to all its centers	$\sum_i c_{ij}x_{ij}/p$		$= t$
C_{LP}	Cost of LP	$\sum_j pC_j$		
ρ	parameter for monarch procedure	2	4	1

Table 3.2: Notation for Bicriteria Algorithm

for the k-median and k-means.

$$\min \sum_{i,j \in V} w_j c_{ij} x_{ij} \tag{3.1a}$$

$$\text{subject to: } \sum_{i \in V} x_{ij} = p, \qquad \forall j \in V \tag{3.1b}$$

$$nly_i \leq \sum_{j \in V} w_j x_{ij} \leq nLy_i, \qquad \forall i \in V \tag{3.1c}$$

$$\sum_{i \in V} y_i \leq k; \tag{3.1d}$$

$$0 \leq x_{ij} \leq y_i, \qquad \forall i, j \in V. \tag{3.1e}$$

The k-center LP is a little different. As in prior work [40] [24] [3], we guess the optimal radius, t. Since there are a polynomial number of possibilities, we can try all of them to find the minimum possible t for which program 3.2 is feasible. Note that k-center can be reduced to k-median by setting all distances $\leq t$ to t, and all distances $> t$ to $+\infty$.

$$\sum_{i \in V} x_{ij} = p, \qquad \forall j \in V \tag{3.2a}$$

$$nly_i \leq \sum_{j \in V} w_j x_{ij} \leq nLy_i, \qquad \forall i \in V \tag{3.2b}$$

$$\sum_{i \in V} y_i \leq k; \tag{3.2c}$$

$$0 \leq x_{ij} \leq y_i, \qquad \forall i, j \in V \tag{3.2d}$$

$$x_{ij} = 0 \qquad \text{if } d(i,j) > t. \tag{3.2e}$$

In the size constraints, by $n\ell$ and nL, we refer to $\lceil n\ell \rceil$ and $\lfloor nL \rfloor$ repectively. This leaves the solution to the original integer program remains unchanged.

The very first step of the algorithm is to solve the LP. For k-center, solving the LP refers to guessing t by finding the smallest t for which the LP is feasible. This will give a fractional solution, and the rest of the algorithm seeks to round this solution such that the objective does not increase too much, while still trying to maintain feasibility.

Let (x, y) denote an optimal solution to the LP relaxation. Let C_{LP} denote the objective value for k-median and k-means. For k-center, we require C_{LP} to be the smallest threshold t at which the LP is feasible, up to constants. For consistency with the other objectives, we scale it as $C_{LP} = tnp$.

For all $j \in V$, define the connection cost C_j as the average contribution of a point to the objective. For k-median and k-means, it is $C_j = \frac{w_j}{p} \sum_{i \in V} c_{ij} x_{ij}$. That is, for k-median, it is the average distance of a point to its fractional centers while for k-means, it is the average squared distance of a point to its fractional centers. For k-center, it is simply the threshold times the weight $C_j = tw_j$. Therefore, $C_{LP} = \sum_{j \in V} pC_j$ in all cases.

Integrality gap It is well known that the LP's given above have unbounded integrality gap in general for all three objectives. For k-median, [1] and references therein show an example for this. The same example also applies for k-means. Hence, we have to resort to bicriteria algorithms. For k-center, Cygan et al [24] show that the LP has unbounded integrality gap. However, if we only look at connected components of the threshold graph, the problem vanishes, and we can have a more involved approximation algorithm, that does not violate capacities, as described in the section 3.3.

3.2.2 The Algorithm

First we provide an outline of the algorithm and then describe each step in detail. In

1. Sovle the LP to get a fractional solution.

2. Partition all points into coarse clusters (called "empires") around monarchs (heads of empires). The empires actually are the Voronoi partitions induced by the monarchs. The construction also ensures that any two monarchs are far apart. We show that each empire $\mathcal{E}_u$ has total opening (i.e. $\sum_{i \in \mathcal{E}_u} y_i$) at least $\frac{p}{2}$ for k-median, k-means and p for k-center. That is, for $p \geq 2$, we have at least a full center inside each empire.
3. For each empire $\mathcal{E}_u$, if the total opening is $Y_u \triangleq \sum_{i \in \mathcal{E}_u} y_i$, open $\lfloor Y_u \rfloor$ many centers in the empire. The monarch u and $\lfloor Y_u \rfloor - 1$ other points closest to the monarch are chosen towards this end.
4. Round the x_{ij}'s by constructing a bipartite graph and finding an integral min cost flow that corresponds to finding the best assignment for the chosen centers.

3.2.3 Monarch Procedure

Intuition. In this step, we partition all points into coarse clusters such that points with the coarse clusters are close and two coarse clusters are far apart. Following the convention of [40], we call these coarse clusters as "empires", by defining one point as the "monarch", the ruler and the empire as the Voronoi partition around the monarch. This procedure is similar in spirit to the monarch procedure described by Khuller et al [40] with a cutoff distance weighted by each point's contribution. In fact, for k-center, it turns out to be very similar, except that Khuller et al [40] can give stronger guarantees. Note that this step does not change any of the (x, y) variables. It only group points into coarse clusters to make the rounding easy.

In subsequent steps, monarchs are always opened as centers. Hence, it would be beneficial to make points with higher connection cost C_j as monarchs since they contribute more to the objective. This intuition gives rise to the following greedy procedure.

Formal description. Let $\mathcal{M}$ be the set of monarchs, and for each $u \in \mathcal{M}$, denote $\mathcal{E}_u$ as the empire of monarch u. We also define a parameter $\rho = 1$ for k-center, $\rho = 2$ for k-median, and $\rho = 4$ for k-means, for convenience.

For the coarse clustering, sort the points in non-decreasing order of the connection cost C_j and consider points in this sorted order. Pick the first monarch as the point j with the highest C_j and pick subsequent monarchs greedily if they are too far away from an existing monarch. Assign empires greedily as Voronoi partitions about the monarchs. See algorithm 2 for a complete description.

Properties. As formalized by lemma 1, we obtain the following desirable properties from the coarse clustering:

- The empires partition the point set V (Property 3.3a).
- Each point in the empire is close to the monarch (Property 3.3b).
- Any two monarchs are far apart (Property 3.3c).
- Each monarch has a minimum amount of opening (Property 3.3d). An important implication is that each empire has at least a full center's worth of opening for $p \geq 2$ for k-median and k-means and any p for k-center.

Input: V, set of points
Output: Set of monarchs, $\mathcal{M}$, and empire $\mathcal{E}_j$ for each monarch $j \in \mathcal{M}$

$\mathcal{M} \leftarrow \emptyset$
Order all points in non-decreasing order of C_i
// Identify Monarchs
foreach $i \in V$ **do**
    if $\nexists j \in \mathcal{M}$ such that $w_i c_{ij} \leq 2\rho C_i$ **then**
        $\mathcal{M} \leftarrow \mathcal{M} \cup \{i\}$
// Assign Empires as Voronoi partitions around monarchs
foreach $j \in V$ **do**
    Let $u \in \mathcal{M}$ be the closest monarch to j
    $\mathcal{E}_u \leftarrow \mathcal{E}_u \cup \{j\}$

Algorithm 2: Monarch procedure for coarse clustering: Greedy algorithm to create monarchs and assign empires

It is the last property above that makes it easier than the case for $p = 1$ (no replication), since we are ready to give up a factor of two on fault-tolerance.

Lemma 1. *The output of Algorithm 2 satisfies the following properties:*

$$\bullet \bigcup_{u \in \mathcal{M}} \mathcal{E}_u = V; \text{ and } \forall u, u' \in \mathcal{M} \text{ s.t. } u \neq u',\ \mathcal{E}_u \cap \mathcal{E}_{u'} = \emptyset; \tag{3.3a}$$

$$\bullet\ \forall j \in \mathcal{E}_u,\ u \in \mathcal{M},\ w_j c_{uj} \leq 2\rho C_j; \tag{3.3b}$$

$$\bullet\ \forall u, u' \in \mathcal{M} \text{ s.t. } u \neq u',\ c_{uu'} > 2\rho \max\{{}^{C_u}/_{w_u}, {}^{C_{u'}}/_{w_{u'}}\}; \tag{3.3c}$$

$$\bullet\ \forall u \in \mathcal{M},\ \sum_{j \in \mathcal{E}_u} y_j \geq \frac{p}{2} \text{ (or for the case of } k\text{-center, } \sum_{j \in \mathcal{E}_u} y_j \geq p). \tag{3.3d}$$

Proof. The first two properties follow easily from construction (for the third property, recall we ordered the points at the start of the monarch procedure). Here is the proof of the third property, depending on the objective function.

For k-center and k-median, it is clear that for some $u \in \mathcal{M}$, if $w_u d(i, u) \leq \rho C_u$, then $i \in \mathcal{E}_u$ (from the triangle inequality and Property 3.3c). For k-means, however: if $w_u d(i, u)^2 \leq 2C_u$, then $i \in \mathcal{E}_u$. Note that the factor is $\rho/2$ for k-means. This is because of the triangle inequality is a little different for squared distances.

To see why this is true for k-means, assume towards contradiction that $\exists i \in V$, $u, u' \in \mathcal{M}$, $u \neq u'$ such that $u \in \mathcal{E}_{u'}$ and $d(i, u)^2 \leq 2C_u/w_u$. Then $d(i, u') \leq d(i, u)$ by construction. Therefore, $d(u, u')^2 \leq (d(u, i) + d(i, u'))^2 \leq 4d(i, u)^2 \leq 8C_u/w_u$, and we have reached a contradiction by Property 3.3c.

Now, to prove property 3.3d:

k-center. From the LP constraints, for every u, $\sum_{j \in V} x_{ju} = p$. But x_{ju} is non-zero only they are separated by at most t, the threshold. Combining this with the fact that if $d(j, u) \leq C_u/w_u =$

t, then $j \in \mathcal{E}_u$, we get, for each $u \in \mathcal{M}$:

$$\sum_{j \in \mathcal{E}_u} y_j \geq \sum_{j \in \mathcal{E}_u} x_{ju} = p$$

k-median and k-means. Note that $C'_u = C_u/w_u$ is a weighted average of costs c_{iu} with weights x_{iu}/p, i.e., $C'_u = {}^{C_u}/_{w_u} = \sum_i c_{iu}{}^{x_{iu}}/_p$. By Markov's inequality,

$$\sum_{j : c_{ju} > 2C'_u/w_u} \frac{x_{ju}}{p} < \frac{C'_u}{2C'_u} = \frac{1}{2}$$

Combining this with the fact that if $c_{ju} \leq 2C'_u$, then $j \in \mathcal{E}_u$ for both k-median and k-means , we get, for each $u \in \mathcal{M}$:

$$\sum_{j \in \mathcal{E}_u} y_j \geq \sum_{j : c_{ju} \leq 2C'_u} y_j \geq \sum_{j : c_{ju} \leq 2C'_u} x_{ju} \geq \frac{p}{2}.$$

□

3.2.4 Aggregation

Intuition. The coarse clusters that we get at the end of the previous step are self-contained (each empire has opening at least one) and are well separated (the monarchs are far apart). At the end of the aggregation step, we open the monarch and possibly more points for each empire, and end up with at most k open centers. While making these movements, we try to maintain the feasibility, possibly violating the upper bound on the size by a small factor. Note that at the end of this step, the assignments may still be fractional. These are rounded in the next step.

Preliminaries. In this step, we aggregate all openings to the monarchs and then round the y_i's, violating the size constraints by a factor of $\frac{p+2}{p}$.

The procedure relies on a suboperation called *Move*, which is the standard way to transfer openings between points (both in k-median and k-center arguments [24, 49]) to maintain all LP constraints. Performing a *Move* between i and j means increasing y_i by some δ, decreasing y_j by the same δ, and changing all x_{il} and x_{jl} so that the fractional demand switches from j to i.

Definition 1 (Operation "Move")**.** *The operation "Move" moves a certain opening δ from a to b. Let (x', y') be the updated (x, y) after a movement of $\delta \leq y_a$ from a to b. Define*

$$y'_a = y_a - \delta \tag{3.4a}$$
$$y'_b = y_b + \delta \tag{3.4b}$$
$$\forall u \in V,\ x'_{au} = x_{au}(1 - {}^{\delta}/_{y_a}) \tag{3.4c}$$
$$\forall u \in V,\ x'_{bu} = x_{bu} + x_{au} \cdot {}^{\delta}/_{y_a} \tag{3.4d}$$

Now, we show that this operation does not violate any of the LP constraints except the constraint that $y_i \leq 1$. Should we require $\delta \leq \min(y_a, 1 - y_b)$, the constraint $y_i \leq 1$ would not be violated. But to get a bicriteria approximation, we allow this violation. The amount by which the objective gets worse can then be bounded by the triangle inequality.

Lemma 2. *The operation Move does not violate any of the LP constraints except possibly the constraint $y_i \leq 1$ and the threshold constraint 3.2e of k-center.*

Proof. To show that Equation 3.4 satisfies all the LP constraints, first note that the only quantities that change are $y_a, y_b, x_{au}, x_{bu}, \forall u \in V$. Further, x, y satisfy all the constraints of the LP. Using this,

- Constraint 3.1a: For every u, $\sum_i x'_{iu} = \sum_i x_{iu} = p$.
- Constraint 3.1b (1):
$$\sum_u w_u x'_{au} = \sum_u w_u x_{au}(1 - \delta/y_a) \leq nLy_a(1 - \delta/y_a) = nLy'_a$$
$$\sum_u x'_{bu} = \sum_u w_u x_{bu} + \sum_u w_u x_{au} \cdot \delta/y_a \leq nLy_b + nLy_a \cdot \delta/y_a = nLy'_b$$
- Constraint 3.1b (2):
$$\sum_u w_u x'_{au} = \sum_u w_u x_{au}(1 - \delta/y_a) \geq n\ell y_a(1 - \delta/y_a) = n\ell y'_a$$
$$\sum_u w_u x'_{bu} = \sum_u w_u x_{bu} + \sum_u x_{au} \cdot \delta/y_a \geq n\ell y_b + n\ell y_a \cdot \delta/y_a = n\ell y'_b$$
- Constraint 3.1c: $\sum_i y'_i = \sum_i y_i \leq k$
- Constraint 3.1d (1):
$$x'_{au} = x_{au}(1 - \delta/y_a) \leq y_a(1 - \delta/y_a) = y'_a$$
$$x'_{bu} = x_{bu} + x_{au} \cdot \delta/y_a \leq y_b + y_a \cdot \delta/y_a = y'_b.$$
- Non-negative constraint: this is true since $\delta \leq y_a$.

□

More intuition. This procedure uses a sequence of *Move*'s to aggregate all openings near each monarch greedily. Suppose we have a total opening of 3.4 in an empire. We shall open 3 full cluster centers, and distribute the total opening among them. This may be interpreted as a violation of the upper limit on the size of the clusters instead, by a factor that turns out to be $(p+2)/p$. For this reason, each x_{ij} may be as high as $(p+2)/p$. As we shall see, in order to make all variables integral, any x_{ij} may have to be rounded up to 2 in the procedure that rounds the x's. As the constraint $\sum_j x_{ij} = p$ is not violated, this may lead to a possible loss in fault tolerance by a factor of 2.

Formal Description. For an empire $\mathcal{E}_u$, let $Y_u = \sum_{i \in \mathcal{E}_u} y_i$ and $z_u = \frac{Y_u}{\lfloor Y_u \rfloor}$. We will give opening z_u to the $\lfloor Y_u \rfloor$ closest points to the monarch. Note that by Property 3.3d, we have $Y_u \geq 1$ (always for k-center and whenever $p \geq 2$ for k-median and k-means). Then by construction, $z_u \geq 1$.

The procedure is simple. In each empire $\mathcal{E}_u$, start with the point i with nonzero y_i that is farthest away from the monarch u. Move its opening to the monarch u. Continue this process until u has opening exactly z_u, and then start moving the farthest openings to the point j nearest to the monarch u. Continue this until the $\lfloor Y_u \rfloor$ closest points to u all have opening z_u.

Call the new variables (x', y').

Properties. (x', y') have the following properties, stated formally in lemmas 3 and 4.

- All LP constraints are satisfied by (x', y') except $y_i \leq 1$, $x_{ij} \leq 1$ and the threshold constraint of k-center (Properties 3.5a and 3.5d of lemma 3).
- The cost of aggregation is bounded.

Lemma 3. *The aggregated solutions (x', y') satisfy the following constraints:*

$$\forall i \in V,\ 1 \leq y'_i < \frac{p+2}{p} \text{ or } y'_i = 0; \tag{3.5a}$$

$$\forall i \in V,\ n\ell y'_i \leq \sum_{j \in V} x'_{ij} \leq nLy'_i; \tag{3.5b}$$

$$\sum_{i \in V} y'_i = k \tag{3.5c}$$

$$\forall i, j \in V,\ x'_{ij} \leq y'_i; \tag{3.5d}$$

$$\forall i \in V,\ \sum_j x'_{ji} = p; \tag{3.5e}$$

$$|\{i \mid y'_i > 0\}| \leq k. \tag{3.5f}$$

Proof. For the first property, recall that each cluster $\mathcal{E}_u$ has total opening $\geq \frac{p}{2}$, so by construction, all i with nonzero y'_i has $y'_i \geq 1$. We also have $\frac{Y_u}{\lfloor Y_u \rfloor} \leq \frac{\lfloor Y_u \rfloor + 1}{\lfloor Y_u \rfloor} \leq \frac{\frac{p}{2}+1}{\frac{p}{2}} = \frac{p+2}{p}$, which gives the desired bound.

The next four properties are checking that the LP constraints are still satisfied (except for $y'_i \leq 1$). These follow from 2 because *Move* does not violate the constraints. The last property is a direct result of Properties 3.5a and 3.5c. □

Now we prove a lemma which bounds how far away a center moved from the points it serves by repeated applications of the triangle inequality.

Lemma 4. *$\forall j \in V$ whose opening moved from i' to i,*

- *k-center: $d(i, j) \leq 5t$,*
- *k-median: $d(i, j) \leq 3d(i', j) + 8 \cdot {}^{C_j}/_{w_j}$,*
- *k-means: $d(i, j)^2 \leq 15d(i', j)^2 + 80 \cdot {}^{C_j}/_{w_j}$.*

Proof. **k-center.** Use the fact that all $C_j = w_j t$, and $x_{ij} > 0 \implies d(i, j) \leq t$ with property 3.3b to get:

$$d(i, j) \leq d(i, u) + d(u, i') + d(i', j) \leq 2 \cdot {}^{C_i}/_{w_i} + 2 \cdot {}^{C_{i'}}/_{w_{i'}} + d(i', j) \leq 5t.$$

k-median. By construction, if the demand of point j moved from i' to i, then $\exists u \in \mathcal{M}$ s.t. $i, i' \in \mathcal{E}_u$ and $d(u, i) \leq d(u, i')$. Denote u' as the closest point in $\mathcal{M}$ to j. Then $d(u, i') \leq d(u', i')$

because $i' \in \mathcal{E}_u$. Then,

$$\begin{aligned} d(i,j) &\leq d(i,u) + d(u,i') + d(i',j) \\ &\leq 2d(u,i') + d(i',j) \\ &\leq 2d(u',i') + d(i',j) \\ &\leq 2(d(u',j) + d(j,i')) + d(i',j) \\ &\leq 8C_j + 3d(i',j). \end{aligned}$$

k-means. The argument is similar to k-median. We shall repeatedly use the following identity instead of the triangle inequality: $(\alpha x+y)^2 \leq (\alpha x+y)^2+(x-\alpha y)^2 = (\alpha+1)x^2+(\alpha+1)y^2$.

$$\begin{aligned} d(i,j)^2 &\leq (d(i,u) + d(u,i') + d(i',j))^2 \\ &\leq (2d(u,i') + d(i',j))^2 \leq 5d(u,i')^2 + 5d(i',j)^2 \\ &\leq 5d(u',i')^2 + 5d(i',j)^2 \\ &\leq 5(d(u',j) + d(j,i'))^2 + 5d(i',j)^2 \leq 10d(u',j)^2 + 15d(i',j)^2 \\ &\leq 80C_j/w_j + 15d(i',j)^2. \end{aligned}$$

□

Since we have $\leq k$ points with nonzero opening, we can set them all to 1 to round the y's. Now all that is left is to round the x's.

3.2.5 Rounding the Assignments

Intuition. We round the x's by setting up a min cost flow problem, where a solution corresponds to an assignment of points to centers. We set the problem up so that each unit of flow corresponds to one assignment, and cost of the flow equals the clustering objective for the assignment defined by the flow. The Integral Flow Theorem guarantees integral x_{ij}'s at the optimal. We can actually find these by using algorithms for min cost flow and this corresponds to the best possible assignments with the chosen centers without incurring any additional cost. Since each x_{ij} can now be up to $^{p+2}/_{p}$, these get rounded to 2. In other words, some points may have replication $p/2$ instead of p. Strictly speaking, the algorithm is actually a *tricriteria algorithm* (two constraints are violated to get a constant factor approximation).

Formal description. Set $\{i \mid y_i \neq 0\} = Y$. We show details of the min cost flow network in the proof of the following lemma.

Lemma 5. *There exists an integral assignment of the x'_{ij}'s such that $\forall i, j \in V$, $x'_{ij} \leq 2$ and it can be found in polynomial time.*

Proof. We construct a flow graph as in figure 3.1 so that a flow in this graph is equivalent to a clustering of V with centers from Y.

- Each $j \in V$ is a vertex with supply pw_j. This is to ensure replication.

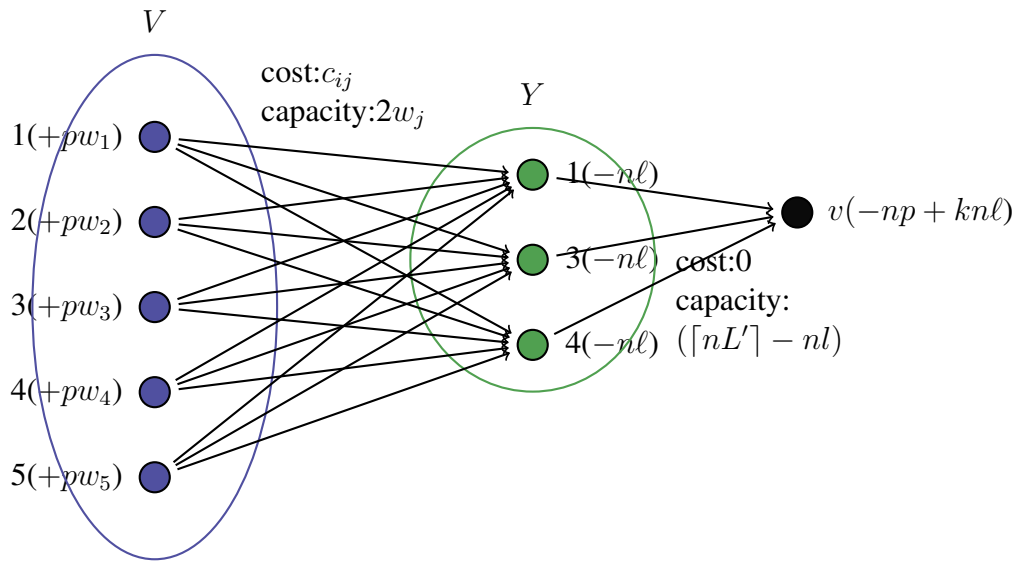


Figure 3.1: Flow network for rounding the x's: Term in brackets indicates the supply: the flow generated at a point (negative supply indicates demand). Recall that $L' = L.\frac{p+2}{p}$. The y-rounded solution gives a feasible flow in this network. By the Integral Flow Theorem, there exists a minimum cost flow which is integral and we can find it in polynomial time.

- For each $i \in Y$, add an additional vertex with demand $n\ell$ (or, by convention, supply $-n\ell$). This ensures that clusters are at least $n\ell$ in size.
- Given $j \in V$ and $i \in Y$, add a directed edge from j to i with capacity $2w_j$ and cost c_{ij} for k-median and k-means. For k-center, the cost is the new threshold $5t$ if $d(i,j) < 5t$ and $+\infty$ otherwise. The capacity ensures that we get a replication of at least $p/2$ for each point.
- We also add a sink vertex, v, with demand $np - kn\ell$ to complete the network.
- Given $i \in Y$, add a directed edge from i to v with cost 0 and capacity $\lceil \frac{p+2}{p} L \rceil - l$. This ensures that clusters are no bigger than $\lceil \frac{p+2}{p} L \rceil$.

In this graph, there exists a feasible flow: $\forall i, j \in V$, send x'_{ij} units of flow along the edge from i to j, and send $\sum_{j \in V} x_{ij}$ units of flow along the edge from i to v. Therefore, by the integral flow theorem, there exists a maximal integral flow which we can find in polynomial time. Also, by construction, this flow corresponds to an integral assignment of the x'_{ij}'s such that $x'_{ij} \leq 2$. □

To put all if these together we have results with all thee objectives.

Theorem 6. *There exists a polynomial time 5-approximation algorithm to solve the balanced k-center with p-replication problem for $p \geq 1$, where the upper size constraints are violated by at most a factor of $\frac{p+2}{p}$, and each point can be assigned to each center at most twice.*

Proof. Recall that we defined $C_{LP} = tnp$, where t is the threshold for the k-center LP. From Lemma 4, when we reassign the demand of point j from i' to i, $d(i,j) \leq 5t$. In other words, the

y-rounded solution is feasible at threshold $5t$. Then the k-center cost of the new y's is $np(5t) = 5C_{LP}$. From Lemma 5, we can also round the x's at no additional cost. □

Theorem 7. *There exists a polynomial time 11-approximation algorithm to solve the balanced k-median with p-replication problem for $p \geq 2$, where the upper size constraints are violated by at most a factor of $\frac{p+2}{p}$, and each point can be assigned to each center at most twice.*

Proof. From Property 4, when we reassign the demand of point j from i' to i, $d(i,j) \leq 3d(i',j) + 8C_j$. Then we can bound the cost of the new assignments with respect to the original LP solution as follows.

$$\begin{aligned}\sum_{i\in V}\sum_{j\in V} w_j d(i,j)x'_{ij} &\leq \sum_{i\in V}\sum_{j\in V}(8C_j + 3w_j d(i,j))x_{ij} \\ &\leq 8\sum_{j\in V} C_j \sum_{i\in V} x_{ij} + 3\sum_{i\in V}\sum_{j\in V} w_j d(i,j)x_{ij} \\ &= 8\sum_{j\in V} pC_j + 3C_{LP} = 11C_{LP}.\end{aligned}$$

Then from Lemma 5, we get a solution of cost at most $11C_{LP}$, which also has integral x's. □

Theorem 8. *There exists a polynomial time 95-approximation algorithm to solve the balanced k-means with p-replication problem for $p \geq 2$, where the upper size constraints are violated by at most a factor of $\frac{p+2}{p}$, and each point can be assigned to each center at most twice.*

Proof. The proof is similar to the k-median proof. From lemma 4, when we reassign the demand of point j from i' to i, $d(i,j)^2 \leq 15d(i',j)^2 + 80C_j/w_j$. Then we can bound the cost of the new assignments with respect to the original LP solution as follows.

$$\begin{aligned}\sum_{i\in V}\sum_{j\in V} w_j d(i,j)^2 x'_{ij} &\leq \sum_{i\in V}\sum_{j\in V}(80C_j + 15w_j d(i',j)^2)x_{ij} \\ &\leq 80\sum_{j\in V} C_j \sum_{i\in V} x_{ij} + 15\sum_{i\in V}\sum_{j\in V} w_j d(i,j)^2 x_{ij} \\ &= 80\sum_{j\in V} pC_j + 15C_{LP} = 95C_{LP}.\end{aligned}$$

Then from Lemma 5, we get a solution of cost at most $95C_{LP}$, which also has integral x's. □

3.3 Constant Factor Approximation Algorithm

In this section, we present a more complicated algorithm that is specific to k-center, which achieves a true approximation algorithm - the capacities are no longer violated. The algorithm is again an LP rounding algorithm.

3.3.1 Approach

As in the previous section and in prior work [40] [24] [3], we start off by guessing the optimal distance t. Since there are a polynomial number of possibilities, it is still only polynomially expensive. We then construct the threshold graph $G_t = (V, E_t)$, with j being the set of all points, and $(x, y) \in E_t$ iff $d(x, y) \leq t$.

A high-level overview of the rounding algorithm that follows is given in algorithm 3.

Connection to the previous section. The algorithm here is similar to the bicriteria algorithm presented previously. There are, however, two differences. Firstly, we work only with connected components of the threshold graph. This is necessary to circumvent the unbounded integrality gap of the LP [24]. Secondly, the rounding procedure of the y's can now move opening across different empires. Since the threshold graph is connected, the distance between any two adjacent monarchs is bounded and turns out to exactly be thrice the threshold. This enables us to get a constant factor approximation without violating any constraints.

Intuition

The approach is to guess the optimal threshold, construct the threshold graph at this threshold, write and round several LPs for each connected component of this graph for different values of k. The intuition behind why this works is that at the optimal threshold, each cluster is fully contained within a connected component (by definition of the threshold graph).

We the round the opening variables, but this time, open exactly k centers. Most of the work goes into rounding the openings, and showing that it is correct. Then, we simply round the assignments using a minimum cost flow again.

3.3.2 Linear Program

As earlier, let y_i be an indicator variable to denote whether vertex i is a center, and x_{ij} be indicators for whether j belongs to the cluster centered at i. By convention, i is called a facility and j is called a client.

Consider the following LP relaxation for the IP for each connected component of G. Note that it is exactly the same as the one from the previous section, except it is described in terms of the threshold graph G. Let us call it $\texttt{LP}\text{-}k\text{-}\texttt{center}(G)$:

Input: V: the set of points, k: the number of clusters, (ℓ, L): min and max allowed cluster size

Output: A k-clustering of V respecting cluster size constraints, p: replication factor

Procedure `balanced-k-center` (V, k, p, ℓ, L)

- **foreach** threshold t **do**
  - Construct the threshold graph G_t
  - **foreach** connected component $G^{(c)}$ of G_t **do**
    - **foreach** k' in $1, ...k$ **do**
      - *// Solve balanced k'-clustering on $G^{(c)}$*
      - Solve `LPRound` ($G^{(c)}, k', p, \ell, L$)
  - Find a solution for each $G^{(c)}$ with k_c centers such that $\sum_c k_c = k$ by linear search; call is s
  - **if** no such a solution exists **then** **return** "No Solution Found"
  - **else** **return** solution s

Procedure `LPRound` (G, k, p, ℓ, L)

- $(x, y) \leftarrow$ relaxed solution of LP in equation 3.6
- $(x', y') \leftarrow$ `yRound`(G, x, y)
- Round x' to get x'' from theorem 15
- **return** (x'', y')

Procedure `yRound` (G, x, y)

- Construct coarse clustering to get a tree of clusters from algorithm 4
- Round clusters in a bottom up manner in the tree, moving mass around to nodes within a distance of 5 away (algorithm 5)
- **return** rounded solution with integral y

Algorithm 3: Overview of approximation algorithm for balanced k-center.

$$\sum_{i \in V} y_i = k \tag{3.6a}$$

$$x_{ij} \leq y_i \qquad \forall i, j \in V \tag{3.6b}$$

$$\sum_{j:ij \in E} w_j x_{ij} \leq nLy_i \qquad \forall i \in V \tag{3.6c}$$

$$\sum_{j:ij \in E} w_j x_{ij} \geq n\ell y_i \qquad \forall i \in V \tag{3.6d}$$

$$\sum_{i:ij \in E} x_{ij} = p \qquad \forall j \in V \tag{3.6e}$$

$$x_{ij} = 0 \qquad \forall ij \notin E \tag{3.6f}$$

$$0 \leq x, y \leq 1 \tag{3.6g}$$

Once we have the threshold graph, for the purpose of k-center, all distances can now be measured in terms of the length of the shortest path in the threshold graph. Let $d_G(i, j)$ represent the distance between i and j measured by the length of the shortest path between i and j in G.

Connected Components

It is well known [24] that even without lower bounds and replication, the LP has unbounded integrality gap for general graphs. However, for connected components of the threshold graph, this is not the case.

To begin with, we show that it suffices to be able to do the LP rounding procedure for only connected threshold graphs, even in our generalization. The following theorem reduces the general balanced k-center problem at a given threshold to the balanced k-center at the same threshold on a connected threshold graph.

Theorem 9. *If there exists an algorithm that takes as input a connected graph G, sizes ℓ, L, replication p, and k for which* `LP-`k`-center`(G_t) *is feasible, and computes a set of k centers to open and an assignment of every vertex j to p centers i such that $d_G(i, j) \leq r$ satisfying the size constraints, then we can obtain a r-approximation algorithm to the balanced k-centers problem with p-replication.*

Proof. Let connected component i have k_i clusters. For each connected component, do a linear search on the range $[1, ...k]$ to find values of k_i for which the problem is feasible. These feasible values will form a range, if size constraints are to be satisfied. To see why this is the case, note that if (x_1, y_1) and (x_2, y_2) are fractional solutions for $k = k_1$ and $k = k_2$ respectively, then $((x_1 + x_2)/2, (y_1 + y_2)/2)$ is a valid fractional soluion for $k = (k_1 + k_2)/2$.

Suppose the feasible values of k_i are $m_i \leq k_i \leq M_i$. If $\sum_i m_i > k$ or $\sum_i M_i < k$, return NO (at this threshold t). Otherwise, start with each k_i equal to m_i. Increase them one by one up to M_i until $\sum_i k_i = k$. This process takes polynomial time. □

From now on, the focus is entirely on a single connected component.

3.3.3 Rounding Openings

Given an integer feasible point to the IP for each connected component, we can obtain the desired clustering. Hence, we must find a way to obtain an integer feasible point from any feasible point of LP-k-center.

To round the opening y, we follow the approach of An et al[3]. The basic idea is to create a coarse clustering of vertices, and have the cluster centers form a tree. The radius of each cluster will be at most 2, and the length of any edge in the tree will exactly be three, by construction.

Now, to round the y, we first start from the leaves of the tree, moving opening around in each coarse cluster such that at most one node (which we pick to be the center, also called the monarch). In subsequent steps, this fractional opening is passed to the parent cluster, where the same process happens. The key to getting a constant factor approximation is to ensure that fractional openings that transferred from a child cluster to a parent cluster are not propagated further. Note that the bicriteria algorithm did not move opening from one coarse cluster (empire) to another because we didn't have an upper bound of the cost incurred by making this shift.

Preliminaries. We start with some definitions.

Definition 2 (δ-feasible solution [24])**.** *A solution (x, y) feasible on G^δ, the graph obtained by connecting all nodes within δ hops away from each other.*

Next, we introduce the notion of a distance-r shift. Intuitively, a distance-r shift is a series of movements of openings, none of which traverses a distance more than r in the threshold graph. Note that the definition is similar to what is used in An et al[3].

Definition 3 (Distance-r shift)**.** *Given a graph $G = (V, E)$ and $y, y' \in \mathbb{R}_*^{|V|}$, y' is a distance-r shift of y if y' can be obtained from y via a series of disjoint movements of the form* **"Move δ from i to i'" where** *$\delta \leq \min(y_i, 1 - y_{i'})$ and every i and i' are at most a distance r apart in the threshold graph G. Further, if all y' are zero or one, it is called an* **integral distance-r shift**.

Note that, by the definition of a distance-r shift, each unit of y moves only once and if it moves more than once, all the movements are put together as a single, big movement, and this distance still does not exceed r.

Lemma 10 (Realizing distance-r shift)**.** *For every distance-r shift y' of y such that $0 \leq y'_i \leq 1 \forall i \in V$, we can find x' in polynomial time such that (x', y') is $(r + 1)$-feasible.*

Proof. We can use the Move operation described earlier and in Cygan et al [24] to change the corresponding x for each such a movement to ensure that the resulting (x', y') are $(r+1)$-feasible.

Let (x', y') be the updated (x, y) after a movement of $\delta \leq \min(y_a, 1 - y_b)$ from a to b. To show that the Move operation (equation 3.4) satisfies all the LP constraints, first note that the only quantities that can change are $y_a, y_b, x_{au}, x_{bu} \forall u \in V$. Constraints 3.6a to 3.6e can be seen to hold from the proof of Lemma 2. The additional restriction that $\delta \leq 1 - y_b$ ensures the following:

- Constraint 3.6f: If $x_{au} > 0$, we have $d_G(a, u) \leq 1$. Now, we transfer from a to b such that $d_G(a, b) \leq r$. By the triangle inequality, $d_G(b, u) \leq r + 1$, and the edge bu exists in G^{r+1}.
- Constraint 3.6g: this is true since $\delta \leq \min(y_a, 1 - y_b)$.

Since each unit of y moves only once, all the movements put together will also lead a solution feasible in G^{r+1}, i.e. we get a $(r + 1)$-feasible solution.

□

From here on, we **assume that $x_{ij}, x_{i'j}$ are adjusted as described above for every movement between i and i'.**

The algorithm to round y [3] proceeds in two phases. In the first phase, we cluster points into a tree of coarse clusters (monarchs) such that nearby clusters are connected using the monarch procedure of Khuller et al [40]. In the second phase, fractional opening are aggregated to get an integral distance-5 shift.

Monarch Procedure. The monarch procedure presented a little differently but is very similar to the monarch procedure presented earlier. Since the threshold graph is connected, we can get guarantees on how big the distance between two monarchs is.

Algorithm 4 describes the first phase where we construct a tree of monarchs and assign empires to each monarch. Let $\mathcal{M}$ be the set of all monarchs. For some monarch, $u \in \mathcal{M}$, let $\mathcal{E}_u$ denote its empire. For each vertex i, let $m(i)$ denote the the monarch u to whose empire $\mathcal{E}_u$, i belongs.

Input: $G = (V, E)$
Output: Tree of monarchs, $T = (\mathcal{M}, E')$, and empires for each monarch
$\texttt{Marked} \leftarrow \emptyset$
foreach $j \in V$ **do**
  initialize $\texttt{ChildMonarchs}(j)$ and $\texttt{Dependents}(j)$ to $\emptyset$
Pick any vertex u and make it a monarch
$\mathcal{E}_u \leftarrow N^+(u)$; Initialize T to be a singleton node u
$\texttt{Marked} \leftarrow \texttt{Marked} \cup \mathcal{E}_u$
while $\exists w \in (V \setminus \texttt{Marked})$ such that $d_G(w, \texttt{Marked}) \geq 2$ **do**
  Let $u \in (V \setminus \texttt{Marked})$ and $v \in \texttt{Marked}$ such that $d_G(u, v) = 2$
  Make u a monarch and assign its empire to be $\mathcal{E}_u \leftarrow N^+(u)$
  $\texttt{Marked} \leftarrow \texttt{Marked} \cup \mathcal{E}_u$
  Make u a child of $m(v)$ in T
  $\texttt{ChildMonarchs}(v) \leftarrow \texttt{ChildMonarchs}(v) \cup \{u\}$
foreach $v \in (V \setminus \texttt{Marked})$ **do**
  Let $u \in \texttt{Marked}$ be such that $d_G(u, v) = 1$
  $\texttt{Dependents}(u) \leftarrow \texttt{Dependents}(u) \cup \{v\}$
  $\mathcal{E}_{m(u)} \leftarrow \mathcal{E}_{m(u)} \cup \{v\}$

Algorithm 4: Monarch Procedure: Algorithm to construct tree of monarchs and assign empires

The guarantees now translate to the following (Lemma 11):

- Empires partition the point set.
- The empire includes **all** immediate neighbors of a monarch and additionally, some other nodes of distance two (called dependents).

- Adjacent monarchs are exactly distance 3 from each other.

Lemma 11. *Algorithm 4, the monarch procedure is well-defined and its output satisfies the following:*

- $\mathcal{E}_u \cap \mathcal{E}_{u'} = \emptyset$.
- $\forall u \in \mathcal{M}: \ \mathcal{E}_u = N^+(u) \cup (\bigcup_{j \in N^+(u)} \texttt{Dependents}(j))$.
- *The distance between a monarch and any node in its empire is at most 2.*
- *Distance between any two monarchs adjacent in T is exactly 3.*
- *If* $\texttt{ChildMonarchs}(j) \neq \emptyset$ *or* $\texttt{Dependents}(j) \neq \emptyset$*, then j is at distance one from some monarch.*

Proof. Note that the whole graph is connected and $V \neq \emptyset$. For the while loop, if there exists w such that $d_G(w, \texttt{Marked}) \geq 2$, there exists u such that $d_G(u, \texttt{Marked}) = 2$ because the graph is connected. By the end of the while loop, there are no vertices at a distance 2 or more from `Marked`. Hence, vertices not in `Marked`, if any, should be at a distance 1 from `Marked`. Thus, the algorithm is well defined.

Each time a new monarch u is created, $N^+(u)$ is added to its empire. This shows the first statement. The only other vertices added to any empire are the dependents in the foreach loop. Each dependent j is directly connected to i, a marked vertex. Hence, i has to be a neighbor of a monarch. If i were a monarch, j would have been marked in the while loop. Thus, $d_G(j, m(i)) = 2$.

If the first statement of the while loop, v is a marked vertex, and has to be a neighbor of some monarch $m(v)$. New monarch u is chosen such that $d_G(u, v) = 2$. The parent monarch of u is $m(v)$ and $d_G(u, m(v)) = d_G(u, v) + d_G(v, m(v)) = 3$.

□

Initial Aggregation. Now, we shall turn to the rounding algorithm of An et al [3]. The algorithm begins with changing y_u of every monarch $u \in \mathcal{M}$ to 1. Call this the initial aggregation. It requires transfer of at most distance one because the neighbors of the monarchs has enough opening.

Lemma 12. *The initial aggregation can be implemented by a distance-1 shift.*

Proof. For every vertex $u \in V$, we have $\sum_{j \in N(u)} y_j \geq \sum_{j \in N(u)} x_{uj} = p \geq 1$. Hence, there is enough y-mass within a distance of one from u. The actual transfer can happen by letting $\delta = \min(1 - y_u, y_j)$ for some neighbor j of u and then transferring δ from j to u. That is, $y_j = y_j - \delta$ and $y_u = y_u + \delta$.

□

Rounding. The rounding procedure now proceeds in a bottom-up manner on the tree of monarchs, rounding all y using movements of distance 5 or smaller. After rounding the leaf empires, all fractional opening, if any is at the monarch. For internal empires, the centers of child monarch (remnants of previous rounding steps) and dependents are first rounded. Then the neighbors of the monarch are rounded to leave the entire cluster integral except the monarch. The two step procedure is adopted so that the opening propagated from this monarch to its parent originates entirely from the 1-neighborhood of the monarch.

Formally, at the end of each run of round on $u \in \mathcal{M}$, all the vertices of the set I_u are integral, where $I_u := (\mathcal{E}_u \setminus u) \cup (\bigcup_{j \in N(u)} \texttt{ChildMonarchs}(j))$.

Input: Tree of monarchs, T, and empires for each monarch after the initial aggregation
Output: y', an integral distance-5 shift of y

Procedure `Round` (Monarch u)
  //Recursive call
  foreach child w of u in T **do** `Round`(w)
  //Phase 1
  foreach $j \in N(u)$ **do**
    $X_j \leftarrow \{j\} \cup \texttt{ChildMonarchs}(j) \cup \texttt{Dependents}(j)$
    $W_j \leftarrow \{\lfloor y(X_j) \rfloor$ nodes from $X_j\}$; (Avoid picking j if possible)
    `LocalRound` $(W_j, X_j, \emptyset)$
    `LocalRound` $(\{j\}, X_j \setminus W_j, \emptyset)$
  //Phase 2
  $F = \{j | j \in N(u)$ and $0 < y_j < 1\}$
  $W_F \leftarrow \{$ any $\lfloor y(F) \rfloor$ nodes from $F\}$
  `LocalRound` $(W_F, F, \emptyset)$
  //Residual
  if $y(F \setminus W_F) > 0$ **then**
    Choose $w^* \in F \setminus W_F$
    `LocalRound` $(\{w^*\}, F \setminus W_F, u)$
Procedure `LocalRound` (V_1, V_2, V_3)
  while $\exists i \in V_1$ such that $y_i < 1$ **do**
    Choose a vertex w with non-zero opening from $V_2 \setminus V_1$
    if there exists none, choose j from $V_3 \setminus V_1$
    $\delta \leftarrow \min(1 - y_i, y_j)$
    `Move` δ from j to i

Algorithm 5: Algorithm to round y

The rounding procedure is described in detail in Algorithm 5. The following lemma states and proves that algorithm 5 rounds all points and doesn't move opening very far.

Lemma 13 (Adaptation of Lemma 19 of An et al [3])**.** *Let* $I_u := (\mathcal{E}_u \setminus u) \cup (\bigcup_{j \in N(u)} \texttt{ChildMonarchs}(j))$.

- `Round`(u) *makes the vertices of* I_u *integral with a set of opening movements within* $I_u \cup \{u\}$.
- *This happens with no incoming movements to the monarch* u *after the initial aggregation.*
- *The maximum distance of these movements is five, taking the initial aggregation into account.*

Proof. **Integrality.** From lemma 11, it can be seen that $X_j, j \in N(u)$ above form a partition of I_u. Hence, it suffices to verify that each node of every X_j is integral.

At the end of line 8, the total non-integral opening in X_j is $y(X_j) - \lfloor y(X_j) \rfloor$, and is hence smaller than one. Line 9 moves all these fractional openings to j. By now, all openings of $X_j \setminus \{j\}$ are integral.

Now, F is the set of all non-integral $j \in N(u)$. So, by the end of line 13, the total non-integral opening in $N(u)$ (and hence in all of I_u) is $y(F \setminus W_F) = y(F) - \lfloor y(F) \rfloor$, and is again smaller than one. If this is zero, we are done.

Otherwise, we choose a node w^*, shift this amount to w^* in line 17. To make this integral, this operations also **transfers the remaining amount, i.e.** $1 - y(F \setminus W_F)$ **from the monarch** u. If this happens, the monarch u's opening is no longer integral, but I_u's is.

This shows the first bullet. For the second one, notice that after the initial aggregation, this last operation is the only one involving the monarch u and hence, there are no other incoming movements into u.

Distance. In the first set of transfers in line 8 the distance of the transfer is at most 4. This is because dependents are a distance one away from j and child monarchs are at a distance two away. The maximum distance is when the transfer happens from one child monarch to another, and this distance is 4 (recall that there are no incoming movements into monarchs).

The transfers in line 9 moves openings from a child monarch or a dependent to j. The distances are 2 and 1 respectively. Accounting for the initial aggregation, this is at most 3.

The rounding on line 13 moves openings between neighbors of the monarch, i.e. from some j to j' where $j, j' \in N(u)$. So, the distance between j and j' is at most 2. From the preceding transfers, the openings at j moved a distance of at most three to get there, and thus, we conclude that openings have moved at most a distance of 5 so far.

The first step of rounding on line 17 moves openings from some j to w^*, where $j, w^* \in N(u)$. As above, the maximum distance in this case is 5. The second step of rounding on line 17 moves opening from the monarch u to its neighbor w^*. This distance is one, and after accounting for the initial aggregation, is 2.

From this, we see that the maximum distance any opening has to move is 5. □

The algorithms, their properties in conjunction with lemma 10 leads to the following theorem, which also summarizes this subsection.

Theorem 14. *There exists a polynomial time algorithm to find a 6-feasible solution with all y integral.*

3.3.4 Rounding Assignments

Once we have integral y, rounding the x is fairly straight-forward, without making the approximation factor any worse. Exactly the same procedure used in bicriteria algorithms works here too. But, we can have an easier construction since for k-center since we can use distances in the threshold graph instead.

If there were no lower bounds on cluster sizes and no replication, this phase can be done by computing the a matching between points and centers to fix the cluster assignments (x) [3]. Once we introduce replication, the assignments can be fixed by using maximum flows, which are generalization of matchings. Further generalizations are needed when we introduce lower bounds. Minimum cost flows are general enough to handle this extension.

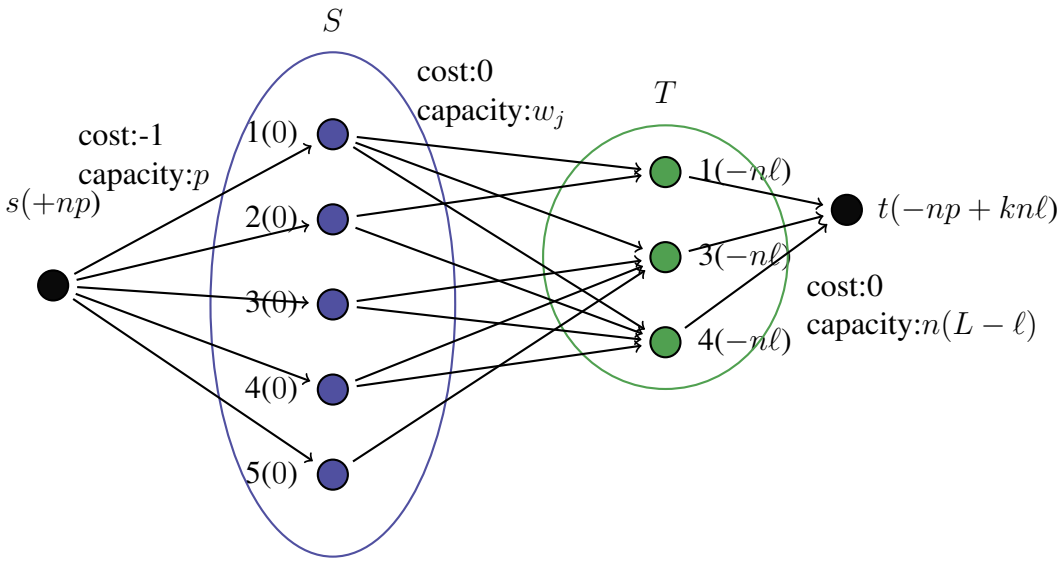


Figure 3.2: Minimum cost flow network to round x's. Quantities in parentheses indicate supply at a node, and negative supply indicates demand.

Theorem 15. *There exists a polynomial time algorithm that given a δ-feasible solution (x, y) with all y integral, finds a δ-feasible solution (x', y) with all x' integral.*

Proof. We shall use a minimum cost flow network to this. Consider a directed bipartite graph (S, T, E'), where $S = V$ and $T = \{i : y_i = 1\}$ and $j \to i \in E'$ iff $x_{ij} > 0$. Add a dummy vertex s, with edges to every vertex in S, and t with edges from every vertex in T. In this network, let edge $j \to i$ of the bipartite graph have capacity w_j. Further, all the $s \to S$ edges have capacity p. s supplies a flow of np units, while each $u \in T$ has a demand of $n\ell$ units. To ensure no excess demand or supply, t has a demand of $np - kn\ell$. All the $t \to T$ edges have a capacity of $n(L-\ell)$.

All the $s \to S$ edges have a cost of -1 and every other edge has a cost of zero. See figure 3.2.

Clearly, a feasible assignment (x, y) to $\texttt{LP-}k\texttt{-center}(G^\delta)$ with integral y is a feasible flow in this network. In fact, it is a minimum-cost flow in this network. This can be verified by the absence of negative cost cycles in the residual graph (because all negative cost edges are at full capacities).

Since, the edge capacities are all integers, there exists a minimum cost integral flow by the Integral Flow Theorem. This flow can be used to fix the cluster assignments.

□

Piecing together theorems 14 and 15, we have the following theorem:

Theorem 16. *Given an instance of the k-centers problem with p-replication and for a connected graph G, and a fractional feasible solution to $\texttt{LP-}k\texttt{-center}(G)$, there exists a polynomial time algorithm to obtain a 6-feasible integral solution. That is, for every i, j such that $x_{ij} \neq 0$, we have $d_G(i, j) \leq 6$.*

Chapter 4

Nearest Neighbor Dispatch

In the previous chapter, we developed provably good algorithms for the balanced clustering on a sample S drawn iid from some unknown probability distribution μ over some metric space $(\mathcal{X}, d())$. The next step is to cluster the rest of the large dataset for training, and on previously unseen query points during deployment- all of which are assumed to be iid draws from the same unknown distribution μ.

During deployment, each query point should be dispatched to a nearby cluster. Load balancing requires that each cluster sees approximately the same number of query points. That is, no machine is over-utilized or under-utilized.

The goal of this chapter is to output a clustering of the entire space $\mathcal{X}$ that does well according to the k-median objective defined over the distribution μ. For load balancing, we are interested in finding clusterings of $\mathcal{X}$ that satisfy cluster size constraints, also known as capacity constraints, in that every cluster has total probability mass at least ℓ and at most L for two numbers $0 \leq \ell \leq L \leq 1$.

Our basic strategy will be to extend clustering of S to the entire space $\mathcal{X}$ using a simple rule. Each point $x \in \mathcal{X}$ will be assigned to the same p clusters as the nearest data point in S.

Previous work by von Luxburg and Ben-David [69] was for center based clustering. The introduction of size constraints in our case means that these techniques cannot be used. Further, we require the dispatch procedure to be fast and online, ruling out [8]. We use, instead, ideas similar to Bubeck et al [14] to extend the clustering to the population. Further, the goal in our case, and hence the analysis is very different.

For the remainder of this chapter, assume that we have a set $S = \{x_1, \ldots, x_n\}$ drawn iid from μ and that we know how to obtain a balanced clustering of S. While this chapter focuses primarily on k-median, similar techniques can be applied to k-means as well. Symbols used are informally described in table 4.1.

Preliminaries

Before stating the algorithm, we introduce some notation. A k-clustering of $\mathcal{X}$ with p replication is an assignment of each point $x \in \mathcal{X}$ to exactly p cluster indices in $[k]$ and a collection of k centers. We encode the cluster assignments as a function $f : \mathcal{X} \to \binom{k}{p}$ and the centers as a

Symbol	Description
k	Number of clusters
ℓ	Minimum size of a cluster
L	Maximum size of a cluster
p	Replication factor, for fault-tolerance
f, g, g_n	Cluster assignments. Adopted convention is f for distribution and g for sample
c, c_n	Centers
f, c	Clustering of the distribution
g_n, c_n	Clustering of a sample of size n
$Q(f, c)$	Cost of a clustering f, c of the distribution
$Q_n(g_n, c_n)$	Cost of a clustering g_n, c_n of a sample of size n with true weights
$\hat{Q}_n(g_n, c_n)$	Cost of a clustering g_n, c_n of a sample of size n with estimated weights
$\bar{g}_n$	Nearest neighbor extension of clustering g_n
$F(\ell, L)$	Set of all clustering of the distribution that satisfy size constraints ℓ, L
$G_n(\ell, L)$	Set of all clusterings of a sample of size n that satisfy true size constraints ℓ, L
$\hat{G}_n(\ell, L)$	Set of all clusterings of a sample of size n that satisfy estimated size constraints ℓ, L

Table 4.1: Parameters of the problem

vector $c \in \mathcal{X}^k$ where the i^{th} entry, denote by $c(i)$, is the center for cluster i. The population-level k-median objective [69] is the expected total distance from a point x sampled from μ to its p assigned centers:

$$Q(f, c) = \mathop{\mathbb{E}}_{x \sim \mu} \left[\sum_{i \in f(x)} d(x, c(i)) \right].$$

For any lower bound ℓ and upper bound L on the cluster capacities, we denote the set of cluster assignments that satisfy the capacity constraints by

$$F(\ell, L) = \left\{ f : \mathcal{X} \to \binom{k}{p} \ : \ \mathop{\mathbb{P}}_{x \sim \mu} (i \in f(x)) \in [\ell, L] \text{ for all } i = 1, \ldots, k \right\}.$$

Similarly, a k-clustering of the data set S with p replication is a function $g : S \to \binom{k}{p}$ and a vector of centers $c \in \mathcal{X}^k$. The weighted k-median objective on S is

$$Q_n(g, c) = \sum_{j=1}^{n} w_j \sum_{i \in g(x_j)} d(x_j, c(i))$$

where $\mathrm{NN}_S(x) = \operatorname{argmin}_{x' \in S} d(x, x')$ is the nearest neighbor in the set S to the point x and

$$w_i = \mathop{\mathbb{P}}_{x \sim \mu} (\mathrm{NN}_S(x) = x_i).$$

Intuitively, each $x_i \in S$ acts as a representative for all points of $\mathcal{X}$ that it is closest to.

For any lower bound ℓ and upper bound L on the cluster capacities, we denote the set of cluster assignments on S that satisfy the weighted capacity constraints by

$$G_n(\ell, L) = \left\{ g : S \to \binom{k}{p} \; : \; \sum_{j : i \in g(x)} w_j \in [\ell, L] \text{ for all } i = 1, \ldots, k \right\}.$$

Finally, Given estimates $\hat{w}_i$ of the true weights w_i, we also define

$$\hat{Q}_n(g, c) = \sum_{j=1}^{n} \hat{w}_j \sum_{i \in g(x_j)} d(x_j, c(i))$$

and

$$\hat{G}_n(\ell, L) = \left\{ g : S \to \binom{k}{p} \; : \; \sum_{j : i \in g(x)} \hat{w}_j \in [\ell, L] \text{ for all } i = 1, \ldots, k \right\}$$

to be the corresponding estimates of the weighted objective and set of assignments that satisfy the weighted capacity constraints.

4.1 Nearest Neighbor Extension

Independently of how we cluster the set S, when we extend the clustering to the entire space $\mathcal{X}$ using the nearest neighbor extension, each data point $x_i \in S$ acts as a representative for some subset of $\mathcal{X}$. In particular, the point x_i represents the set of points closer to it than any other data point, denoted by $V_i = \{x \in \mathcal{X} \; : \; \mathrm{NN}_S(x) = x_i\}$. For instance, when $\mathcal{X}$ is a subset of the Euclidean space $\mathbb{R}^q$, the set V_i is the tile corresponding to x_i in a Voronoi partition of the space induced by the set S.

When we assign the point x_i to some cluster, we are implicitly also assigning all the points in V_i to that cluster. Since the probability mass of the V_i sets might not be equal, some of the data points will have a larger influence on the objective value of the extended clustering and on the probability masses of the clusters. Therefore, when we cluster the set S, we consider a weighted version of the clustering objective function and capacity constraints where each point x_i is weighted by the probability mass of V_i. Since the probability distribution μ is unknown, we estimate these weights using a second sample S' drawn randomly from μ.

Every cluster assignment $g \in G_n(\ell, L)$ has a nearest neighbor extension defined by $\bar{g}(x) = g(\mathrm{NN}_S(x))$. Define

$$\bar{G}_n(\ell, L) = \{\bar{g} \; : \; g \in G_n(\ell, L)\}$$

to be the set of cluster assignment functions on $\mathcal{X}$ that are nearest neighbor extensions of some clustering $g \in G_n(\ell, L)$. Notice that every assignment in $\bar{G}_n(\ell, L)$ satisfies the population-level capacity constraints, i.e. $\bar{G}_n(\ell, L) \subset F(\ell, L)$.

Algorithm 6 gives pseudocode for our algorithm. As described above, it first draws a second sample S' from the data distribution μ which is used to estimate each of the weights $w_1, \ldots, w_n$. Then we apply any approximation algorithm to the estimated weighted k-median objective over S subject to the estimated weighted capacity constraints. Finally, we output the nearest neighbor extension of the resulting clustering.

Parameters: Clustering parameters k, p, ℓ, L, second sample size n'.

procedure NNEXTENSION($S = \{x_1, \ldots, x_n\}$)

Let S' be an iid sample drawn from μ of size n'.

Let $\hat{w}_i = |S' \cap V_i|/n'$ be the estimated mass of V_i for each $i = 1, \ldots, n$.

Let (g_n, c_n) be an approximation to $\min_{g,c} \hat{Q}_n(g, c)$ subject to $g \in \hat{G}_n(\ell, L)$.

Define $\bar{g}_n(x) = g_n(\mathrm{NN}_S(x))$ and output clustering $(\bar{g}_n, c_n)$.

end procedure

Algorithm 6: Nearest Neighbor Clustering Extension

4.2 Sample Complexity

4.2.1 Size of the Second Sample

We bound the sub-optimality of any clustering $(\bar{g}_n, c_n)$ returned by Algorithm 6 with respect to any clustering (f^*, c^*) of the population and the violation of the size constraints. The bound will depend on

1. the quality of the finite-data algorithm,
2. the extension cost or the "average radius" of the Voronoi cells

$$\alpha(S) := \mathop{\mathbb{E}}_{x \sim \mu} \left[d(x, \mathrm{NN}_S(x)) \right] \tag{4.1}$$

and

3. the unavoidable bias from returning clusterings that are constant over V_i,

$$\beta(S, \ell', L') := \min_{h \in G_n(\ell', L'), c} \left(Q(\bar{h}, c) - Q(f^*, c^*) \right) \tag{4.2}$$

where (f^*, c^*) is the optimal clustering that satisfies size constraints ℓ, L.

When ℓ', L' are clear from the context, we just write it as $\beta(S)$. All proofs from this section are presented in the next section.

Theorem 17. *For any $\epsilon > 0, \delta > 0$, let $(\bar{g}_n, c_n)$ be the output of Algorithm 6 with parameters k, p, ℓ, L and second sample size $n' = O\big((n + \log 1/\delta)/\epsilon^2\big)$. Let (f^*, c^*) be any clustering of $\mathcal{X}$ with size constraints (ℓ', L') and (g_n^*, c_n^*) be an optimal clustering of S under $\hat{Q}_n$ satisfying the estimated weighted capacity constraints (ℓ, L). Suppose the finite data algorithm used satisfies $\hat{Q}(g_n, c_n) \leq r \cdot Q(g_n^*, c_n^*) + s$. Then w.p $\geq 1 - \delta$ over the second sample the output $(\bar{g}_n, c_n)$ belongs to $\bar{G}_n(\ell', L')$ where $\ell' = \ell - \epsilon$ and $L' = L + \epsilon$ and we have*

$$Q(\bar{g}_n, c_n) \leq r \cdot Q(f^*, c^*) + s + 2(r+1)pD\epsilon + p(r+1)\alpha(S) + r\beta(S, \ell + \epsilon, L - \epsilon),$$

where D is the diameter of S.

When S is drawn randomly from μ, we can bound the terms $\alpha(S), \beta(S)$.

4.2.2 Bounding the Extension Cost

We bound the sample size required to make $\alpha(S)$ small when $\mathcal{X} \subseteq \mathbb{R}^q$ and S is drawn randomly from an arbitrary μ. However, in the worst case, the curse of dimensionality results in an exponential dependence on dimension (eg. [11]).

Additionally, when the distribution has a lower intrinsic dimension, we can do better. The doubling condition is one such a condition. Let $B(x, r)$ be a ball of radius r around x with respect to the metric $d()$.

Definition 4 (Doubling measure)**.** *A measure μ with support $\mathcal{X}$ is said to be a doubling measure of dimension d_0 if for all points $x \in \mathcal{X}$ and all radiuses $r > 0$ we have*

$$\mu(B(x, 2r)) \leq 2^{d_0}\mu(B(x, r)).$$

We have the following lemma bounding $\alpha(S)$ for randomly drawn S.

Lemma 18. *For any $\epsilon, \delta > 0$, and $\mathcal{X} \subseteq \mathbb{R}^q$, if a randomly drawn S from μ is of size*

- $\mathcal{O}(q^{q/2}\epsilon^{-(q+1)}(q \log \frac{\sqrt{q}}{\epsilon} + \log \frac{1}{\delta}))$ *in the general case, or*
- $\mathcal{O}(\epsilon^{-d_0}(d_0 \log \frac{1}{\epsilon} + \log \frac{1}{\delta}))$ *if μ is doubling with dimension d_0,*

then w.p $\geq 1 - \delta$, $\alpha(S) \leq \epsilon D$

4.2.3 Bounding the bias

Nearest neighbor dispatch may be interpretted as applying nearest neighbor classification with the initial sample serving the role of the training set, and the clustering assignment function serving the role of the label function. With this in mind, we shall make use of the rich literature on nearest neighbor classification to bound $\beta(S)$. At the outset, from the work of Bubeck and von Luxburg [14], inspired by Fritz [32] before them, it can be seen that the probability (over S) that $\beta(S)$ is large goes exponentially quickly to zero as S gets bigger. However, their analysis has some unknown constants, and hence cannot be used to get a sample complexity statement.

A first attempt: Lipschitz Continuity. Lipschitz continuity is a common assumption in nearest neighbor classification. But if the label function f is deterministic (i.e. the Bayes error is zero), as it is in our case, L-Lipschitzness implies a margin of $1/L$ between regions of different labels.

In particular, if we assume that the optimal population clustering is L-Lipschitz, it means that the optimal clusters have a $1/L$ margin of probability mass zero between them. In lemma 25, if we choose $r < 1/L$, nearest neighbor dispatch will not misclassify any region of the space (with high probability). Further, the capacity constraints will be satisfied for the optimal clustering projected on the sample for any sample because the nearest neighbor is always in the same optimal cluster. In this case, $\beta(S)$ is zero w.h.p, given that we set the size of the first sample to be large enough to guarantee that $r < 1/L$. However, Lipschitz continuity is a very strong assumption which is seldom realistic. This motivates the need for a weaker condition that allows the probability mass to taper off near the boundaries, rather than have a hard margin.

Probabilistic Lipschitzness. The Probabilistic Lipschitzness (PL) condition was introduced by Urner et al [66, 67] as a relaxation of the Lipschitz condition that is both realistic and mathematically neat. A deterministic labeling function f is ϕ-PL if the probability mass of points that have non-zero mass of differently labeled points in a λ-ball around them is atmost $\phi(\lambda)$.

Definition 5 (Probabilistic Lipschitzness)**.** *Let $(\mathcal{X}, d())$ be some metric space of diameter D and let $\phi : [0,1] \to [0,1]$. $f : \mathcal{X} \to \binom{k}{p}$ is ϕ-Lipschitz with respect to some distribution μ over $\mathcal{X}$, if $\forall \lambda \in [0,1]$:*

$$\mathop{\mathbb{P}}_{x\sim\mu}\left[\exists y : \mathbb{I}(f(x) \neq f(y)) \text{ and } d(x,y) \leq \lambda D\right] \leq \phi(\lambda)$$

Note that for the clustering, we have made two modifications to the original definition. Since f maps each point to a set in $\binom{k}{p}$, we overload notation to let $f(x) = f(y)$ iff the sets $f(x)$ and $f(y)$ are equal i.e. $i \in f(x) \iff i \in f(y)$. Secondly, to achieve scale invariance and since we consider bounded sets here, we have introduced the diameter of the set $\mathcal{X}$ in the definition. Further, since ϕ is non-decreasing, define $\phi^{-1}(x) = \inf\{\lambda : \phi(\lambda) \geq x\}$.

The main assumption we make in this section is that the optimal clustering f^* is ϕ-PL. This is, in some sense, a roundness assumption on the clusters- that the probability mass "close to" the boundaries of the clusters is small.

Bound on $\beta(S)$. The main sample complexity result is that for any confidence parameter δ and accuracy parameter ϵ, what should be the size of S to ensure that $\beta(S)$ is small with high probability?

If a clustering function f is PL, it means the clusters are, in some sense, "round"- that the probability mass "close to" the boundaries of the clusters is small, akin to the roundness in [12]. Under this condition, we have the following sample complexity result for β. We can compare to a clustering with slightly tighter capacity constraints:

Lemma 19. *Let μ be a measure on $\mathbb{R}^q$ with support $\mathcal{X}$ of diameter D. Let f^*, some clustering of μ that satisfies capacities $(\ell+\epsilon, L-\epsilon)$, be ϕ-PL. If we see a sample S drawn iid from μ of size at least $O\left(\frac{1}{\epsilon}\left(\frac{1}{\phi^{-1}(\epsilon/2)}\right)^q\left(q\log\frac{\sqrt{q}}{\phi^{-1}(\epsilon/2)}+\log\frac{1}{\delta}\right)\right)$ in the general case or $O\left(\left(\frac{1}{\phi^{-1}(\epsilon)}\right)^{d_0}\left(d_0\log\frac{4}{\phi^{-1}(\epsilon)}+\log\frac{1}{\delta}\right)\right)$ when μ is a doubling measure of dimension d_0 then, w.p. at least $1-\delta$ over the draw of S, we have that $\beta(S,\ell,L) \leq pD\epsilon$.*

Putting everything together

We have the following theorem that captures the sample size required for Algorithm 6 to output a good enough clustering that satisfies the capacity constraints approximately.

Theorem 20. *For any $\epsilon > 0$, any confidence parameter $\delta > 0$ and clustering parameters k, p, ℓ, L, define:*

$$(f^*, c^*) = \mathop{\text{argmin}}_{f\in F(\ell-2\epsilon, L+2\epsilon), c} Q(f,c)$$
$$(g_n^*, c_n^*) = \mathop{\text{argmin}}_{g\in\hat{G}_n(\ell,L), c} \hat{Q}_n(g,c)$$

to be the best clustering of μ with constraints relaxed by 2ϵ, and the best clustering of S using weights estimated from S' respectively. Further, assume that μ has a diameter D and that f^* is ϕ-PL. Suppose the finite-data algorithm used within Algorithm 6 outputs an (g_n, c_n) such that $\hat{Q}_n(g_n, c_n) \leq r\hat{Q}_n(g_n^*, c_n^*) + s$ when run with parameters k, p, ℓ, L on sample S. Let $\epsilon' = \min\{\epsilon, \phi^{-1}(\epsilon)\}$. If we have samples of sizes:

- $n = O\left(\frac{1}{\epsilon}\left(\frac{\sqrt{q}}{\epsilon'}\right)^q (q \log \frac{\sqrt{q}}{\epsilon'} + \log \frac{1}{\delta})\right)$ *in the general case*
- $n = O\left(\frac{d_0 \log(1/\epsilon') + \log(1/\delta)}{(\epsilon')^{d_0}}\right)$ *when μ is a doubling measure of dimension d_0*
- *and* $n' = O\left(\frac{n + \log(1/\delta)}{\epsilon^2}\right)$

then, with probability $1 - \delta$ over the draw of S, S', the output $(\bar{g}_n, c_n)$ of Algorithm 6 has the following desirable properties:

- *$\bar{g}_n$ satisfies capacity constraints $(\ell - \epsilon, L + \epsilon)$. In other words, $\bar{g}_n \in F(\ell - \epsilon, L + \epsilon)$.*
- *$\bar{g}_n, c_n$ is a good approximation to (f^*, c^*):*

$$Q(\bar{g}_n, c_n) - rQ(f^*, c^*) \leq s + (4r + 3)pD\epsilon$$

4.3 Proofs

Size of Second Sample

Before proving the Theorem 17, we shall need the following lemmas.

Our first results are regarding the size of the second sample, n'. As a function of the data set size $n = |S|$, how large does n' need to be so that $\hat{Q}_n$ is a faithful estimate of the true weighted objective Q_n and the set $\hat{G}_n(\ell, L)$ closely approximates $G_n(\ell, L)$?

Lemma 21. *For any $\epsilon > 0$ and any $\delta > 0$, if we set $n' = O\big(\frac{1}{\epsilon^2}(n + \log \frac{1}{\delta})\big)$ in Algorithm 6 then with probability at least $1 - \delta$ we have $\left|\sum_{i \in I}(w_i - \hat{w}_i)\right| \leq \epsilon$ uniformly for all index sets $I \subset [n]$. Whenever this happens:*

- *For any clustering (g, c) of the set S, we have $|\hat{Q}_n(g, c) - Q_n(g, c)| \leq 2pD\epsilon$ where $D = \max_{x,x' \in S} d(x, x')$ is the diameter of the set S, and*
- *$\hat{G}_n(\ell, L)$ is close to $G_n(\ell, L)$ in the sense that $G_n(\ell + \epsilon, L - \epsilon) \subseteq \hat{G}_n(\ell, L) \subseteq G_n(\ell - \epsilon, L + \epsilon)$.*

Proof. For any index set $I \subset [n]$, let V_I denote the union $\bigcup_{i \in I} V_i$. The key insight is that, since the sets $V_1, \ldots, V_n$ are disjoint, for any index set I, we have that $\sum_{i \in I} w_i = \mu(V_I)$ and $\sum_{i \in I} \hat{w}_i = \hat{\mu}(V_I)$, where $\hat{\mu}$ is the empirical measure induced by the set S'. Therefore, it is sufficient to show that $\hat{\mu}$ accurately estimates the probability mass of all 2^n unions of the sets $V_1, \ldots, V_n$. Applying the union bound over the 2^n unions and Hoeffding's inequality, we have

$\mathbb{P}\left(\sup_{I \subset [n]} \left|\sum_{i \in I} w_i - \hat{w}_i\right| > \epsilon\right) = \mathbb{P}\big(\sup_{I \subset [n]} |\mu(V_I) - \hat{\mu}(V_I)| > \epsilon\big) \leq \sum_{I \subset [n]} \mathbb{P}(|\mu(V_I) - \hat{\mu}(V_I)| > \epsilon) \leq 2^n \cdot 2e^{-2n'\epsilon^2}$. Setting $n' = \frac{1}{2\epsilon^2}((n+1)\log 2 + \log 1/\delta) = O\big(\frac{1}{\epsilon^2}(n + \log \frac{1}{\delta})\big)$ results in the upper bound being equal to δ.

The second implication is obvious. For the first implication, let $I \subset [n]$ be the indices of the estimated weights for which $\hat{w}_i \geq w_i$ and let $I^c = [n] - I$. Since the error in any sum of weights is at most ϵ, we have

$$\begin{aligned}|\hat{Q}_n(g,c) - Q_n(g,c)| &\leq \left|\sum_{j\in I}(\hat{w}_j - w_j)\sum_{i\in g(x_j)} d(x_j, c(i))\right| \\ &\quad + \left|\sum_{j\in I^c}(w_j - \hat{w}_j)\sum_{i\in g(x_j)} d(x_j, c(i))\right| \\ &\leq pD\left(\left|\sum_{j\in I}(\hat{w}_j - w_j)\right| + \left|\sum_{j\in I^c}(w_j - \hat{w}_j)\right|\right) \leq 2pD\epsilon.\end{aligned}$$

□

While this bound is still loose when we sum up almost all of the weights (or very few of them) it achieves the optimal worst-case dependence on n. If n' is sub-linear in n, some of the weights are estimated as zero.

Next we relate the objective value of any clustering (g, c) over the sample S to the population-level objective of the extension $(\bar{g}, c)$ over the entire space $\mathcal{X}$. In particular, the cost increases by at most p times the expected distance from a point sampled from μ to the nearest point in S. Intuitively, this quantity bounds the difference in the distance from a data point x_j and a point $x \in V_j$ to any center $c(i)$.

Lemma 22. *For any clustering (g, c) of S with extension $(\bar{g}, c)$, the cost of the extension is bounded as $|Q_n(g,c) - Q(\bar{g},c)| \leq p\,\mathbb{E}_{x\sim\mu}[d(x, \mathrm{NN}_S(x))]$.*

Proof. Rewrite the objective as $Q_n(g,c) = \mathbb{E}_{x\sim\mu}\left[\sum_{i\in\bar{g}(x)} d(\mathrm{NN}_S(x), c(i))\right]$. The triangle inequality gives

$$\begin{aligned}Q(\bar{g},c) &\leq \mathop{\mathbb{E}}_{x\sim\mu}\left[\sum_{i\in\bar{g}(x)} d(x, \mathrm{NN}_S(x))\right] + \mathop{\mathbb{E}}_{x\sim\mu}\left[\sum_{i\in\bar{g}(x)} d(\mathrm{NN}_S(x), c(i))\right] \\ &= p\mathop{\mathbb{E}}_{x\sim\mu}[d(x, \mathrm{NN}_S(x))] + Q_n(g,c).\end{aligned}$$

Similarly, we have that $Q_n(g,c) \leq p\,\mathbb{E}_{x\sim\mu}[d(x, \mathrm{NN}_S(x))] + Q(\bar{g},c)$, which concludes the proof. □

We are now ready to prove Theorem 17.

Proof of Theorem 17. For our choice of n', the good event from Lemma 21 holds with probability at least $1-\delta$ over the second sample S'. Since, g_n satisfies capacity constraints (ℓ, L) with estimated weights, whenever the good event above holds, $\bar{g}_n$ satisfies capacity constraints $(\ell - \epsilon, L + \epsilon)$. Let $(h_n, c_n) = \mathrm{argmin}_{h\in G_n(\ell+\epsilon, L-\epsilon), c} Q(\bar{h}, c)$ be the clustering of S, subject to the

tighter capacity constraints $(\ell+\epsilon, L-\epsilon)$, whose extension has the best objective value. Then we have

$$\begin{aligned}
Q(\bar{g}_n, c_n) &\leq Q_n(g_n, c_n) + p\alpha(S) \\
&\leq \hat{Q}_n(g_n, c_n) + 2pD\epsilon + p\alpha(S) \\
&= \hat{Q}_n(g_n, c_n) - r \cdot \hat{Q}_n(g_n^*, c_n^*) + \hat{r} \cdot Q_n(g_n^*, c_n^*) + 2pD\epsilon + p\alpha(S) \\
&\leq s + 2(r+1)pD\epsilon + p\alpha(S) + r \cdot Q_n(g_n^*, c_n^*) \\
&\leq s + 2(r+1)pD\epsilon + p\alpha(S) + r \cdot Q_n(h_n, c_n) \\
&\leq s + 2(r+1)pD\epsilon + p(r+1)\alpha(S) + r \cdot Q(\bar{h}_n, c_n) \\
&\leq s + 2(r+1)pD\epsilon + p(r+1)\alpha(S) + r\beta(S) + r \cdot Q(f^*, c^*)
\end{aligned}$$

□

Extension Cost

We shall first prove the statement for $\mathbb{R}^q$.

Lemma 23. *For any $r > 0$ and any $\epsilon > 0$, there exists a subset $\mathcal{Y}$ of $\mathcal{X}$ containing at least $1-\epsilon$ of the probability mass of μ such that, for any $\delta > 0$, if we see an iid sample S of size $n = \mathcal{O}(\frac{1}{\epsilon}(\frac{D\sqrt{q}}{r})^q(q\log\frac{D\sqrt{q}}{r} + \log\frac{1}{\delta}))$ drawn from μ, then with probability at least $1-\delta$ we have $\sup_{x\in\mathcal{Y}} d(x, \mathrm{NN}_S(x)) \leq r$.*

Proof. Let C be the smallest cube containing the support $\mathcal{X}$. Since the diameter of $\mathcal{X}$ is D, the side length of C is at most D. Let $s = r/\sqrt{q}$ be the side-length of a cube in $\mathbb{R}^q$ that has diameter r. Then it takes at most $m = \lceil D/s \rceil^q$ cubes of side-length s to cover the set C. Let $C_1, \ldots, C_m$ be such a covering of C, where each C_i has side length s.

Let C_i be any cube in the cover that has probability mass at least ϵ/m under the distribution μ. The probability that a sample of size S drawn from μ does not contain a sample in C_i is at most $(1-\epsilon/m)^n$. Let I denote the index set of all those cubes with probability mass at least ϵ/m under μ. Applying the union bound over the cubes indexed by I, the probability that there exists a cube C_i with $i \in I$ that does not contain any sample from S is at most $m(1-\epsilon/m)^n \leq me^{-n\epsilon/m}$. Setting $n = \frac{m}{\epsilon}(\ln m + \log\frac{1}{\delta}) = \mathcal{O}(\frac{1}{\epsilon}(\frac{D\sqrt{q}}{r})^q(q\log\frac{D\sqrt{q}}{r} + \log\frac{1}{\delta}))$ results in this upper bound being δ. For the remainder of the proof, suppose that this high probability event occurs.

Define $\mathcal{Y} = \bigcup_{i\in I} C_i$. Each cube from our cover not used in the construction of $\mathcal{Y}$ has probability mass at most ϵ/m and, since there are at most m such cubes, their total mass is at most ϵ. It follows that $\mathbb{P}_{x\sim\mu}(x \in \mathcal{Y}) \geq 1-\epsilon$. Moreover, every point x in $\mathcal{Y}$ belongs to one of the cubes, and every cube C_i with $i \in I$ contains at least one sample point. Since the diameter of the cubes is r, it follows that the nearest sample to x is at most r away. □

Setting $r = D\epsilon$, we obtain the bound on $\alpha(S)$.

For the remainder of this subsection, suppose that μ is a doubling measure of dimension d_0 with support $\mathcal{X}$ and that the diameter of $\mathcal{X}$ is $D > 0$. First, the following lemma is quite standard. See, for example, [43, 45]

Lemma 24. *For any radius r, $\mathcal{X}$ can be covered using no more than $(2D/r)^{d_0}$ balls of radius r.*

The next lemma tells us that we need a sample of size $O\big((\frac{D}{r})^{d_0}(d_0 \log \frac{D}{r} + \log \frac{1}{\delta})\big)$ in order to ensure that there is a neighbor from the sample no more than r away from any point in the support with high probability.

Lemma 25. *For any $r > 0$ and any $\delta > 0$, if we draw an iid sample S of size $n = (\frac{2D}{r})^{d_0}(d_0 \log(\frac{4D}{r}) + \log(\frac{1}{\delta}))$, then with probability at least $1 - \delta$ we have $\sup_{x \in \mathcal{X}} d(x, \mathrm{NN}_S(x)) \leq r$*

Proof. By Lemma 24 there is a covering of $\mathcal{X}$ with balls of radius $r/2$ of size $(4D/r)^{d_0}$. For each ball B in the cover, the probability that no sample point lands in B is $(1 - \mu(B))^n \leq (1 - (r/2D)^{d_0})^n \leq \exp(-n(r/2D)^{d_0})$. Let E be the event that there exists at least one ball B in our cover that does not contain one of the n sample points. Applying the union bound over the balls in the cover, we have that $\mathbb{P}(E) \leq (4D/r)^{d_0} \exp(-n(r/2D)^{d_0})$. Setting $n = (2D/r)^{d_0}(d_0 \log(4D/r) + \log(1/\delta)) = O\big((\frac{D}{r})^{d_0}(d_0 \log \frac{D}{r} + \log \frac{1}{\delta})\big)$, we have that $\mathbb{P}(E) < \delta$. When the bad event E does not occur, every ball in our covering contains at least one sample point. Since every point $x \in \mathcal{X}$ belongs to at least one ball in our covering and each ball has diameter r, we have $\sup_{x \in \mathcal{X}} d(x, \mathrm{NN}_S(x)) \leq r$. □

Lemmas 25 and 23 complete the proof of Lemma 18 since the expectation is no larger than the supremum.

Bounding the bias

We shall now prove Lemma 19.

Proof of Lemma 19. Suppose $\mathbb{P}_{x \sim \mu}(f^*(\mathrm{NN}_S(x)) \neq f^*(x)) \leq \epsilon$.

Define the restriction $f_S : S \to \binom{k}{p}$ of $f \in F(\ell, L)$ to be $f_S(x) = f(x)$ for $x \in S$. Firstly, we shall show that the cluster sizes of f_S^* can be bounded. Recall that the sizes of cluster i in a clustering f of $\mathcal{X}$ and a clustering g of the sample S are respectively is $\mathbb{P}_{x \sim \mu}(i \in f(x))$ and $\mathbb{P}_{x \sim \mu}(i \in \bar{g}(x))$. By the triangle inequality, $|\,\mathbb{P}_{x \sim \mu}(i \in \bar{f}_S^*(x)) - \mathbb{P}_{x \sim \mu}(i \in f^*(x))| \leq \mathbb{P}_{x \sim \mu}(\bar{f}_S^*(x) \neq f^*(x)) = \mathbb{P}_{x \sim \mu}(f^*(\mathrm{NN}_S(x)) \neq f^*(x))$ and this is at most ϵ, by our assumption.

Consider $\beta(S)$. Since $f^* \in F(\ell + 2\epsilon, L - 2\epsilon)$, we have that $f_S^* \in G_n(\ell - \epsilon, L + \epsilon)$, we have

$$\beta(S) \leq Q(\bar{f}_S^*, c^*) - Q(f^*, c^*) \quad = \mathop{\mathbb{E}}_{x \sim \mu}\Big[\sum_{i \in f^*(\mathrm{NN}_S(x))} \|x - c(i)\| - \sum_{i' \in f^*(x)} \|x - c(i')\|\Big]$$

By the triangle inequality, $\|x - c(i)\| - \|x - c(i')\| \leq \|c(i) - c(i')\|$. Since $f^*(x)$ and $f_S^*(\mathrm{NN}_S(x))$ can differ on at most p assignments, and since any two centers are most a distance D apart, we have that $\beta(S) \leq \mathbb{E}_{x \sim \mu}(pD \cdot \mathbb{I}\, f^*(\mathrm{NN}_S(x) \neq f^*(x))) = pD \cdot \mathbb{P}_{x \sim \mu}(f^*(\mathrm{NN}_S(x)) \neq f^*(x)) \leq pD\epsilon$.

All that remains is to show that $\mathbb{P}_{x \sim \mu}(f^*(\mathrm{NN}_S(x)) \neq f^*(x)) \leq \epsilon$ for big enough n. Lemma 26 lists the conditions when this is true. □

We require the following lemma for nearest neighbor classification, similar in spirit to that of Urner and Ben-David. Note that since f is a set of p elements, this lemma holds for multi-label nearest neighbor classification.

Lemma 26. *Let μ be a measure on $\mathbb{R}^q$ with support $\mathcal{X}$ of diameter D. Let the labeling function, f be ϕ-PL. For any accuracy parameter ϵ and confidence parameter δ, if we see a sample S of size at least*

- $\frac{2}{\epsilon}\left\lceil \frac{\sqrt{q}}{\phi^{-1}(\epsilon/2))} \right\rceil^q \left(q \log \lceil \frac{\sqrt{q}}{\phi^{-1}(\epsilon/2)} \rceil + \log \frac{1}{\delta} \right)$ *in the general case*
- $\left(\frac{2}{\phi^{-1}(\epsilon)}\right)^{d_0} \left(d_0 \log \frac{4}{\phi^{-1}(\epsilon)} + \log \frac{1}{\delta} \right)$ *when μ is a doubling measure of dimension d_0*

then nearest neighbor classification generalizes well. That is, with probability at least $1-\delta$ over the draw of S, the error on a randomly drawn test point, $\mathbb{P}_{x\sim\mu}(f(x) \neq f(\mathrm{NN}_S(x))) \leq \epsilon$.

Proof. Let $\lambda = \phi^{-1}(\epsilon)$. We know that most of $\mathcal{X}$ can be covered using hypercubes in the general case, as in Lemma 23 or entirely covered using balls in the case when μ is a doubling measure, as in Lemma 24, both of diameter λD. In case we have cubes in the cover, we shall use a ball of the same diameter instead. This does not change the sample complexity, since a cube is completely contained in a ball of the same diameter.

Formally, let $\mathcal{C}$ be the covering obtained from Lemma 23 or Lemma 24, depending on whether or not the measure is a doubling measure. Define $\mathcal{B}(x)$ to be the set of all the balls from $\mathcal{C}$ that contain the point x. A point will only be labeled wrongly if it falls into a ball with no point from S, or a ball that contains points of other labels. Hence,

$$\mathop{\mathbb{P}}_{x\sim\mu}(f(\mathrm{NN}_S(x)) \neq f(x)) \leq \mathop{\mathbb{P}}_{x\sim\mu}(\forall C \in \mathcal{B}(x) : S \cap C = \emptyset) + \mathop{\mathbb{P}}_{x\sim\mu}(\exists y \in \bigcup_{C\in\mathcal{B}(x)} C : f(y) \neq f(x))$$

Since each ball is of diameter λD, the second term is at most $\mathbb{P}_{x\sim\mu}(\exists y \in B(x, \lambda D) : f(y) \neq f(x))$. By the PL assumption, this is at most $\phi(\lambda) = \epsilon$, independent of the covering used.

For the first term, our analysis will depend on which covering we use:

- From Lemma 23, we know that all but $1-\epsilon$ fraction of the space is covered by the covering $\mathcal{C}$. When the sample is of size $\mathcal{O}(\frac{1}{\epsilon}(\frac{\sqrt{q}}{\lambda})^q(q\log(\frac{\sqrt{q}}{\lambda}) + \log\frac{1}{\delta}))$, each $C \in \mathcal{C}$ sees a sample point. For a sample this large, the first term is $\leq \epsilon$. Substituting ϵ with $\epsilon/2$ completes this part of the proof.
- When μ is a doubling measure, we can do better.If every ball of the cover sees a sample point, the first term is necessarily zero. From the proof of Lemma 25, we know that if we draw a sample of size $n = (2/\lambda)^{d_0}(d_0\log(4/\lambda) + \log(1/\delta))$ samples, then every ball of the cover sees a sample point with probability at least $1-\delta$ over the draw of S. This completes the proof.

□

Chapter 5

Experiments

In this chapter, we discuss various approaches to learning depending on the communication budget, compare the performance against common alternative schemes.

First, we discuss various learning schemes depending on the communication budget. Next, we describe the datasets used in the comparison. Finally, we present the experimental results and discuss when each method would be best used.

Preliminaries

Suppose we have training data $\mathcal{D} = \{(y_i, \mathbf{x}_i)\}_{i=1}^{N}$ where each $\mathbf{x}_i \in \mathbb{R}^q$ is the datapoint and y_i is the label. Further, let $l(.)$ denote a loss function such as the logistic loss or the squared hinge loss.

Suppose we cluster an initial sample and partition the rest of the dataset using techniques from the previous chapters. Refer to the partitions as $\mathcal{D}_1, ..., \mathcal{D}_k$. During deployment, suppose we receive a point $\mathbf{x}$. Denote by $i(\mathbf{x})$ the partitions to which the dispatcher assigns point $\mathbf{x}$. If the model is $\mathbf{w}$ for binary classification, the prediction for $\mathbf{x}$ would be $\text{sign}(\mathbf{w} \cdot \mathbf{x})$. For multi-class classification, we use the popular one-vs-all technique.

Thoughout this chapter, we use linear models for reasons of scalability. For learning in the distributed setting, we use the setting of the Parameter Server [48]- asynchronous stochastic gradient descent (SGD) is used. That is, all the parameters are held by servers. Each worker reads a mini-batch of data, pulls the values of the necessary parameters from the servers, computes gradients and pushes updates to the servers. Further, all workers work asynchronously.

5.1 Learning Schemes

This section discusses alternatives for for the learning stage of Algorithm 1, having performed the data dependent partitioning. These schemes differ on the amount of communication required. The appropriate scheme for a situtation depends on the communication budget, and the nature of the data- whether it is dense and low-dimensional like image data, or sparse and very high dimensional like the CTR data that uses one-hot encoding.

Communication Free Scheme

The simplest learning scheme is to train an independent model on each partition of the data. This method has the advantage that learning is local and communication free. In other words, it is *embarrassingly parallel*. Further, for this reason, complicated learning algorithms that are not amenable to the distributed setting can be used. To be concrete, partition i solves:

$$\mathbf{w}_i = \operatorname*{argmin}_{\mathbf{w}} \sum_{(y,\mathbf{x})\in\mathcal{D}_i} l(y\mathbf{w}\cdot\mathbf{x})$$

and the linear model used for prediction for a new point $\mathbf{x}$ is

$$f(\mathbf{x}) = \mathbf{w}_{i(\mathbf{x})}\cdot\mathbf{x}.$$

As we shall see, this method gives tremendous improvements for image data, even over the global model with full synchronization.

Partial Communication Scheme

In sparse high-dimensional data, the communication free scheme has the disadvantage that there isn't enough data to learn parameters corresponding to the rare, tail features. For this reason, it is beneficial to to synchronize the tail features.

Let $\mathbf{x} = (\mathbf{x}_h; \mathbf{x}_t)$ where $x_h \in \mathbb{R}^{q'}$ are the frequently occuring features in the head whereas $\mathbf{x}_t \in \mathbb{R}^{q-q'}$ is the data from the power-law tail. Similarly, let $\mathbf{w} = (\mathbf{w}_h; \mathbf{w}_t)$. The learning procedure tries to learn $\mathbf{w}_1, ...\mathbf{w}_k \in \mathbb{R}^{q'}$ and $\mathbf{w}_t \in \mathbb{R}^{q-q'}$ where

$$\mathbf{w}_t, \mathbf{w}_1, ..., \mathbf{w}_k = \operatorname*{argmin}_{\mathbf{w}_t,\mathbf{w}_1,...,\mathbf{w}_k} \sum_{i=1}^{k} \sum_{(y,\mathbf{x})\in\mathcal{D}_i} l(y(\mathbf{w}_i\cdot\mathbf{x}_h + \mathbf{w}_t\cdot\mathbf{x}_t))$$

and the linear model used for a new point $\mathbf{x} = (\mathbf{x}_h; \mathbf{x}_t)$ is

$$f(\mathbf{x}) = \mathbf{w}_{i(\mathbf{x})}\cdot\mathbf{x}_h + \mathbf{w}_t\cdot\mathbf{x}_t.$$

The communication complexity of this scheme depends on the size of the head q'. If $q' = q$, this reduces to the previous case of communication free learning. If $q' = 0$, it reduces to having a single global synchronized model. Since the feature occurences approximately follow a power law, the communication required will be very small for moderate q'.

Complete Communication Scheme

As $q' \to 0$, the importance given to locality decreases. One can do better for frequently occuring features in the head. The idea here is to have global weights for all features and store a local "correction" to the global weights for features in the head. The intuition behind this is that different clusters can also share information about the head.

In particular, each cluster has a local model $\mathbf{w}_i \in \mathbb{R}^{q'}$ corresponding to the frequently occuring features in the head and the system also maintains a synchronized model $\mathbf{w}_g \in \mathbb{R}^q$ where

$$\mathbf{w}_g, \mathbf{w}_1, ..., \mathbf{w}_k = \underset{\mathbf{w}_g, \mathbf{w}_1, ..., \mathbf{w}_k}{\operatorname{argmin}} \sum_{i=1}^{k} \sum_{(y,\mathbf{x}) \in \mathcal{D}_i} l(y(\mathbf{w}_i \cdot \mathbf{x}_h + \mathbf{w}_g \cdot \mathbf{x}))$$

and the linear model used for a new point $\mathbf{x} = (\mathbf{x}_h; \mathbf{x}_t)$ is

$$f(\mathbf{x}) = \mathbf{w}_{i(\mathbf{x})} \cdot \mathbf{x}_h + \mathbf{w}_g \cdot \mathbf{x}.$$

Note that the features in the head are determined by both the global weights and the local corrections, whereas the local weights are only determined the global weights.

In this case, the communication complexity is exactly equal to the communication complexity of training a single synchronized global model, but as we shall see, the performance turns out to be better.

5.2 Experimental Evaluation

Algorithm

This section reports empirical performance of Algorithm 1. We used an approximate implementation of our algorithm. An intial sample was drawn uniformly at random from the dataset. The sample was then clustered using k-means++ [5], with heuristics for balancing. The dispatcher was implemented efficiently by approximate nearest neighbor search. One of the three aforementioned methods approaches were used for learning.

For clustering, we use k-means++ because it is efficient and scalable, as an approximation to LP rounding algorithms that are not as scalable, despite being theoretically well supported. We expect that using better clustering will lead to better classification accuarcy. For fault tolerance, we modified Lloyd's iterations to maintain p assignments for each data point. To satisfy the balance constraints, we use two simple heuristics: while the smallest cluster violates the lower size constraint, the smallest cluster is merged with the cluster whose center is closest to its own, and the center is updated as in Lloyd's algorithm. After this step, there are possibly fewer than k clusters, but they all satisfy the lower size constraint. Any cluster that violates the upper size constraints is randomly partitioned into evenly-sized clusters that satisfy the upper size constraint. This clustering approach may produce fewer or more than k clusters. In all experiments, we use the balance constraints $\ell = p/(2k)$ and $L = 2p/k$. It can be seen that for these values of ℓ, L, after one round of merging and splitting, clusters now satisfy the balance constraints.

Finally, rather than implementing exact nearest neighbor dispatch as in Algorithm 6, we use the random partition tree (RPT) algorithm of Dasgupta and Sinha [26] to dispatch based on approximate nearest neighbors to achieve huge speed-ups. This algorithm is similar to a randomized KD-Tree algorithm, that uses random split directions, instead of choosing a coordinate direction. Further, we approximate w_i's, the weights of Voronoi partitions, by their expected values, $1/n$.

For classification of image data, we used the linear one-vs-all multi-class SVM provided by Liblinear [30]. Liblinear is known to scale linearly with both the number of points and dimensionality. The regularization parameter is chosen by 5-fold cross validation. On the other hand, for sparse high-dimensional data, we use Asynchronous Stochastic Gradient Descent, as implemented in the Parameter Server [48]. Our implementation was of asynchronous SGD was only meant for proof of concept and was not suitable for measuring running time.

The clustering and dispatcher procedures scale as $\mathcal{O}(q)$ where q is the dimensionality of the data. Specifically, centroids computed by k-means++ and split directions generated by RPTs are dense. To make this step feasible in the high dimensional datasets, we use the popular trick of feature hashing without random signs [62]using the hash function $h(i) = i \mod q'$, for some $q' < q$. It is common to use this trick for CTR data [16].

The julia code is available online.

Setup

We run two versions of the experiment, both of which produce the same output. The first version is a simulation that runs sequentially on a single machine with simulated parallelism. The dispatcher writes each cluster to disk and the learner reads the clusters from disk. The second version is a truly distributed implementation. We start one worker process on each of the available processing cores. First, a single worker subsamples the data, clusters the subsample into k clusters, and then builds a random partition tree for fast nearest neighbor lookup. The subsample, clustering, and random partition tree describe a dispatching rule, which is then copied to every worker. Training the system has two steps: first, the training data is dispatched to the appropriate workers, and then each worker learns a model for the clusters they are responsible for. During the deployment phase, the workers load the training data in parallel and send each example to the appropriate workers (as dictated by the dispatch rule). To minimize network overhead examples are only sent over the network in batches of 5000. During the training phase, each worker runs the appropriate algorithm in tandem or independently, for each cluster they were responsible for. If multiple clusters were allotted to a worker, the worker would go over them sequentially, much like the previous simulation. For testing, each worker is allotted a portion of the testing data. As it reads through the data, it dispatches the point to and queries the appropriate cluster and receives the prediction. This process is performed in mini-batches to amortize the network overhead to sending data across the network. The experiments were performed on cluster of 15 machines, each with 8 Intel(R) Xeon(R) cores of clock rate 2.40 GHz and 32GB shared memory per machine. This implementation used Remote Procedure Call and Remote Reference primitives provided by Julia to communicate between machines.

Competing Algorithms

We compare the performance of our algorithm against the random partitioning approach with no communication or full communication.

Random Data is randomly partitioned to workers, where each worker learns independently from the given subset of the data. This is the random partitioning scheme with no communication.

Global Data is randomly partitioned to workers, and the workers update a shared model with complete communication. Equivalently, the same performance may be acheived by having all the data on a single machine.

Datasets

We used the following public datasets ranging from images to CTR to evaluate these methods. All the CTR datasets used contain ad impressions from commercial search engines or advertising companies and the label is whether or not the ad was clicked clicked by a user.

MNIST-8M: We used the raw pixels of this handwritten image dataset [51]. It has 8 million examples and 784 features.

CIFAR-10: The CIFAR-10 dataset [46] is an image classification task with 10 classes. Following Krizhevsky et al [47] we include 50 copies of each training, example each randomly rotated and cropped to get a training set of 2.5 million examples. We extracted the features from the Google Inception[1] [64] by using the output of an early layer (layer in3c) and a later layer (layer in4d).

CTRc: This CTR dataset with 860K examples with 232 continuous-valued features. Negative examples were subsampled to have 2.5 negative examples for every positive example.

CTRa: This CTR dataset is very sparse and high-dimensional with 0.3 million examples and 13 million binary features. Negative examples were subsampled to have a nearly balanced dataset.

Criteo-Kaggle: This CTR dataset, released by online advertising company Criteo has 45 million examples and 34 million features. This dataset is extrememly sparse, with just 40 non-zero entries per example, and has about 4 negative examples for every positive example. The size of the dataset is about 15GB in libsvm format.

Results

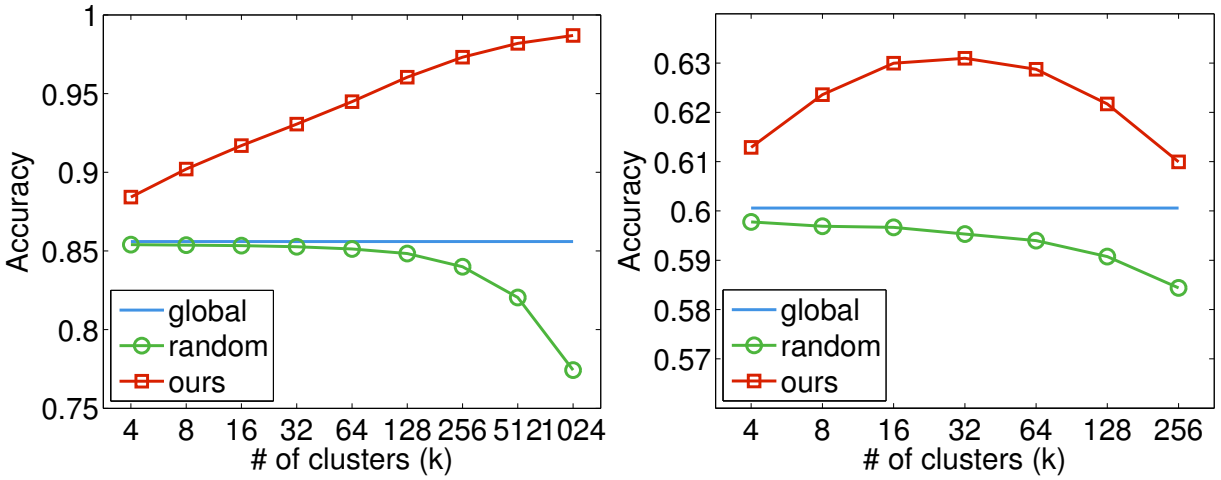
The standard deviation over multiple runs of our experiments was very small (of the order of $10^{-3} - 10^{-4}$), and hence we omit error bars from all plots. This also implies that the results are statistically significant.

Dense data

For dense data, we use the communicate free learning procedure. The results are shown in Figure 5.1. As can be seen, our method always performs better than random partitioning. The performance gap varies over datasets. For the simplest MNIST-8M, our method achieves tremendous improvements in performance (about 14%) compared to the random partitioning scheme. But the improvements are not as pronounced when the clustering structure is less obvious.

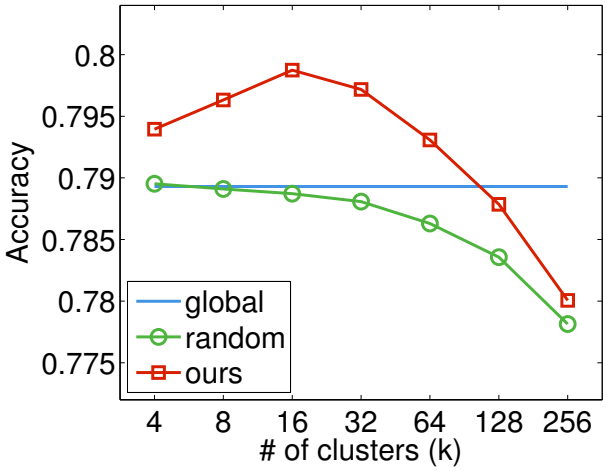
Also note that there is an optimal number of clusters, k^* for each dataset, where the classification accuracy obtained is the highest. It is large for simple datasets ($k^* \geq 1024$ for MNIST-8M

[1]For the specific network structure, refer to `https://github.com/dmlc/mxnet/blob/master/example/notebooks/cifar-recipe.ipynb`.

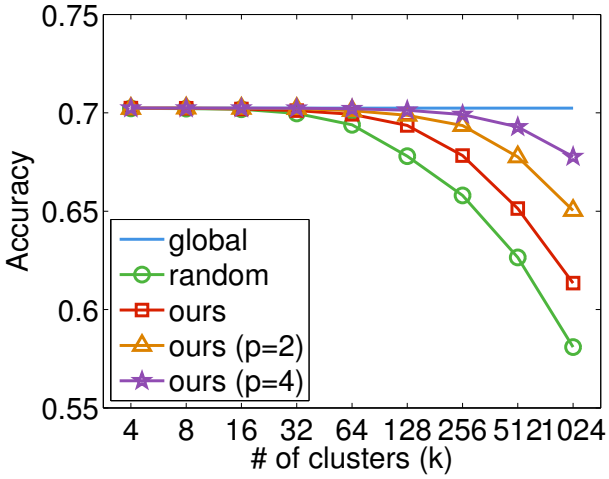


(a) MNIST-8M

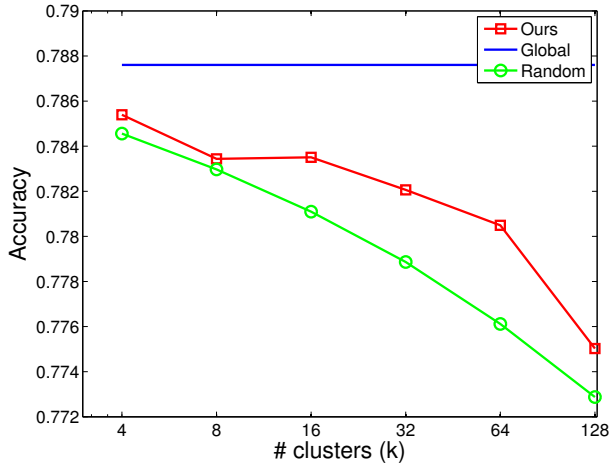
(b) CIFAR-10 Early Features



(c) CIFAR-10 Late Features



(d) CTRc Dataset



(e) Criteo-Kaggle Dataset

Figure 5.1: Parameter studies on the number k of clusters for MNIST-8M, CIFAR-10 (with two different feature representations), CTRc and Criteo-Kaggle dataset. The s.d. over different runs is $\sim 10^{-3}$ and therefore omitted.

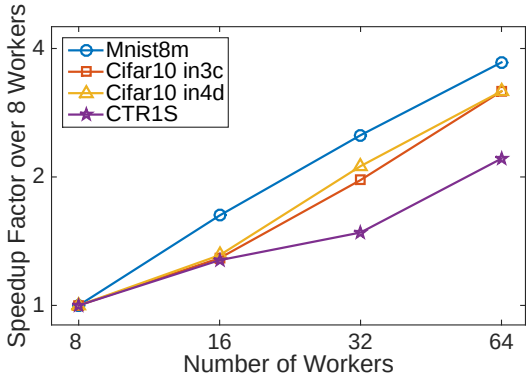


Figure 5.2: Linear speedup for communication-free learning: When the number of workers is doubled, the time for dispatch, learning and testing (averaged over 5 runs) drops by a constant factor. k was set to 128 for CIFAR-10 (both), CTR and 512 for MNIST-8M

but $k^* \leq 16$ for CTR). Further, there seems to be a "safe" number of clusters where the performance of our paradigm is at least as good as the global linear model. This value is larger than 16 for all dense datasets. That is, the proposed method can enjoy the communication-free, embarassingly parallel training without any loss of accuracy.

It is also interesting to note that the performance improves with increasing p. This suggests that we ought to replicate points not just for fault-tolerance, but also for better performance.

Our system exhibits the desirable property of *Strong Scaling*, as demonstrated by Figure 5.2. That is, for a fixed workload, if we have twice as much processing power, the time for dispatch, training and deployment roughly drops by a constant factor, until the number of worker processes equals the number of clusters, k. We get no further benefit after this point because each cluster is only served by a single worker. In principle, one could have multiple workers catering to a single cluster. Note that we do not include the clustering time because the clustering is a preprocessing step on a small sample.

Sparse Data

The results for sparse data are given in Figure 5.3. In the realm of sparse data, communication free learning cannot compete with the global model. Learning with partial communication does better than learning without communication but still cannot match global model in classification accuracy. It is interesting to note that learning with complete communication performs better than the global model. This shows that data dependent resource allocation is beneficial even for sparse CTR data.

Figure 5.4 shows the effect of varying the number of local features for the partial communication scheme. Moderate gains in performance can be obtained by sychronizing only on relatively rare features.

A possible explanation for the drop in classification accuracy as k increases is the dearth of data. Because of the high degree of sparsity, each cluster does not have enough data to learn its model reliably. Increasing the number of clusters reduces the amount of data in each cluster, making it harder to learn a reliable model. Partial communication overcomes this problem to

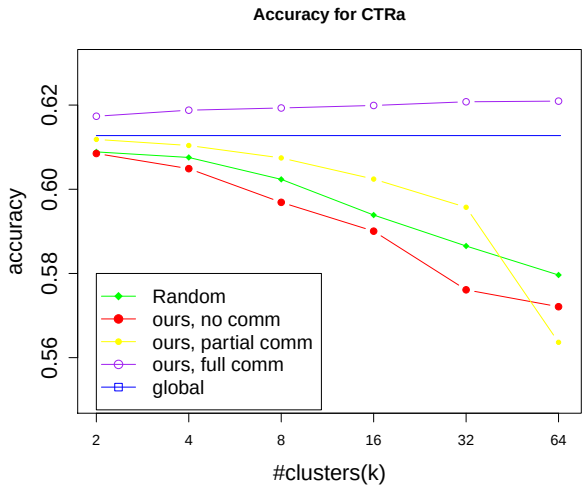
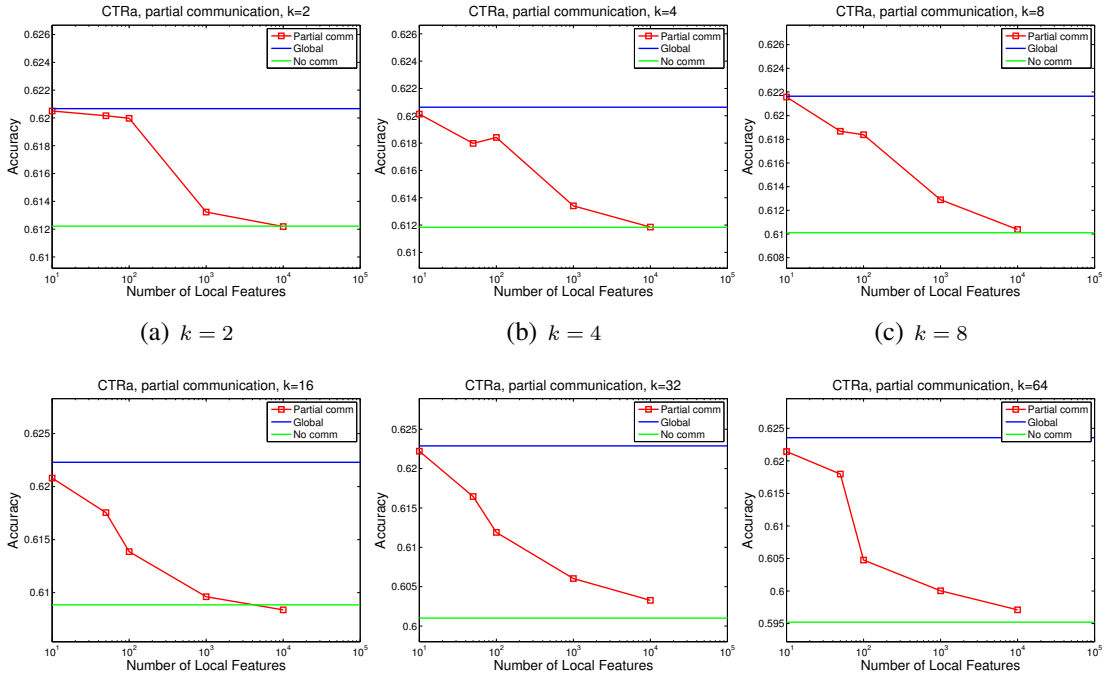


Figure 5.3: Classification Accuracy for different values of k for the CTRa dataset. For partial and complete communication model, the $10K$ most frequently occurring features were chosen as local.



(a) $k = 2$ (b) $k = 4$ (c) $k = 8$

(d) $k = 16$ (e) $k = 32$ (f) $k = 64$

Figure 5.4: Plot for Accuracy vs Number of local features for the partial communication scheme on CTRa. Note that having zero local features is the same as learning a single global model over the entire dataset whereas having all features as global ($\geq 10^5$ in the plots) is the same as learning without communication.

some extent by facilitating sharing of information about rarer tail features between clusters. The complete communication model also shares information about the frequently occurring features in the head. It outperforms the global model because it can also take advantage of the locality induced by clustering.

Further, the performace is directly correlated to the quality of clustering obtained. In the image datasets, k-means++ gives a good clustering, and the piece-wise linear model induced by the communication-free learning can better the global model, possibly by a large extent as in MNIST. On the other hand, for the sparse CTR data, Euclidean distance is probably not an appropriate metric because the data is generated by one-hot encoding. However, k-means seems to be able to capture enough locality for the complete communication model.

Overall, the results support our claim that our data dependent resource allocation scheme is good in both theory and practice.

Chapter 6

Conclusion

This thesis explores methods of data dependent resource allocation for distributed ML. In particular, we explore balanced clustering as a means to this end.

Clustering is NP-Hard. Adding balancing contraints only makes it harder. Given the erratic behavior of balanced clustering [7], it is not clear whether this can even be approximated. In chapter 3, we answered this question positively, showing provably correct polynomial time constant-factor and bicriteria approximation algorithms. In particular, we provided an LP Rounding algorithm that returns a constant factor approximation for balanced k-median, k-means, k-center while violating the upper bound on cluster sizes and fault tolerance by a factor of at most 2 each. Moreover, we provite a true 6-approximation algorithm for balanced k-center with fault tolerance that does not violate any constraints.

In Chapter 4, we proposed the provably correct Nearest Neighbor Dispatch, an efficient online procedure based on ideas from Nearest Neighbor Classification. We analyzed the size of the initial sample required for good dispatch.

In Chapter 5, we describe three learning schemes: with no communication, partial communication and full communication. The first approach is embarassingly parallel and works well on dense data such as images. For sparse data, the second and third approaches improve over the first, and show that in a variety of scenarios, one benefits from using data dependent resource allocation for distributed ML. In particular, we observed a 14% increase in classification accuracy on the MNIST-8M dataset, and a very significant 1% increase on some CTR data.

Open Questions

There are several concrete, open questions related to this thesis.

- In distributed ML, commodity machines can fail at any time. If $\mathcal{O}(1)$ machines fail, our method has built in fault tolerance. One can start off with two clusters on a single physical machine, and migrate some clusters in the event of obtaining $\mathcal{O}(1)$ new machines. However, if there is a major change planned such as gaining or losing $\mathcal{O}(k)$ machines, how does one minimize the transfer of data across clusters? In particular, one can explore heirarchical clustering approaches to this end.
- The sparse CTR datasets were generated by one-hot encoding. Evidently, Euclidean dis-

tance is not an appropriate distance metric on this data as the Euclidean distance does not accurately measure the distance between two real examples. In fact, one-hot encoding is good for making categorical variables amenable to linear boundaries. Can one come up with a clustering scheme that separates clusters via linear boundaries?

- To obtain better performance, data-dependent partitioning should to be performed to maximize learning performance. One way to do it would be by optimizing over the paritioning and learning jointly [36, 58], but the problem would be non-convex. It would be interesting to come up with similar methods that also scale well.
- The LP Rounding algorithms are not very scalable. In this thesis, k-means++ was used in practice. Can one devise more practicable approximation algorithms with provable guarantees?
- Provably correct dispatch for k-center.
- Proving guarantees about the final classification accuracy.

Bibliography

[1] Karen Aardal, Pieter L van den Berg, Dion Gijswijt, and Shanfei Li. Approximation algorithms for hard capacitated k-facility location problems. *European Journal of Operational Research*, (2):358–368, 2015. 2.3, 3.1, 3.2.1

[2] Alekh Agarwal, Olivier Chapelle, Miroslav Dudík, and John Langford. A reliable effective terascale linear learning system. *The Journal of Machine Learning Research*, 15(1):1111–1133, 2014. 1, 2.1

[3] Hyung-Chan An, Aditya Bhaskara, Chandra Chekuri, Shalmoli Gupta, Vivek Madan, and Ola Svensson. Centrality of trees for capacitated k-center. In *Integer Programming and Combinatorial Optimization*, pages 52–63. Springer, 2014. 1, 2.3, 3.1, 3.2.1, 3.3.1, 3.3.3, 3.3.3, 3.3.3, 16, 13, 3.3.4

[4] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006. 1

[5] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007. 5.2

[6] Maria-Florina Balcan, Avrim Blum, Shai Fine, and Yishay Mansour. Distributed learning, communication complexity, and privacy. In *Conference on Learning Theory*, 2012. 1

[7] Maria-Florina Balcan, Travis Dick, Mu Li, Venkata Krishna Pillutla, Alexander Smola, and Colin White. Data Driven Partitioning for Machine Learning. 1, 2.3, 6

[8] Arindam Banerjee and Joydeep Ghosh. On scaling up balanced clustering algorithms. In *SDM*, pages 333–349, 2002. 2.4, 4

[9] J. Barilan, G. Kortsarz, and D. Peleg. How to allocate network centers. *Journal of Algorithms*, 15(3):385 – 415, 1993. 2.3, 3.1

[10] Mohammadhossein Bateni, Aditya Bhaskara, Silvio Lattanzi, and Vahab Mirrokni. Distributed balanced clustering via mapping coresets. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2591–2599. Curran Associates, Inc., 2014. 2.3

[11] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is nearest neighbor meaningful? In *Database TheoryICDT99*, pages 217–235. Springer, 1999. 4.2.2

[12] Avrim Blum and Shuchi Chawla. Learning from labeled and unlabeled data using graph

mincuts. 2001. 4.2.3

[13] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1):1–122, 2011. 2.1

[14] Sébastien Bubeck and Ulrike von Luxburg. Nearest neighbor clustering: A baseline method for consistent clustering with arbitrary objective functions. *The Journal of Machine Learning Research*, 10:657–698, 2009. 2.4, 4, 4.2.3

[15] Jarosław Byrka, Krzysztof Fleszar, Bartosz Rybicki, and Joachim Spoerhase. Bi-factor approximation algorithms for hard capacitated k-median problems. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 722–736. SIAM, 2015. 2.3, 3.1

[16] Olivier Chapelle, Eren Manavoglu, and Romer Rosales. Simple and scalable response prediction for display advertising. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 5(4):61, 2014. 1, 5.2

[17] Moses Charikar, Sudipto Guha, Éva Tardos, and David B Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1999. 1, 2.3, 3.1

[18] Moses Charikar, Liadan O'Callaghan, and Rina Panigrahy. Better straming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium of Theory of computing*, pages 30–39. ACM, 2003. 2.3

[19] Ashish Chawla, Benjamin Reed, Karl Juhnke, and Ghousuddin Syed. Semantics of caching with spoca: a stateless, proportional, optimally-consistent addressing algorithm. In *Proceedings of the 2011 USENIX conference on USENIX annual technical conference*, pages 33–33. USENIX Association, 2011. 1

[20] Benhui Chen, Feiran Sun, and Jinglu Hu. Local linear multi-svm method for gene function classification. In *Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on*, pages 183–188. IEEE, 2010. 2.2

[21] Haibin Cheng, Pang-Ning Tan, and Rong Jin. Localized support vector machine and its efficient algorithm. In *SDM*, pages 461–466. SIAM, 2007. 2.2

[22] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009. 1

[23] Brian F Cooper, Raghu Ramakrishnan, Utkarsh Srivastava, Adam Silberstein, Philip Bohannon, Hans-Arno Jacobsen, Nick Puz, Daniel Weaver, and Ramana Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *Proceedings of the VLDB Endowment*, 1(2):1277–1288, 2008. 1, 1, 2.1

[24] Marek Cygan, MohammadTaghi Hajiaghayi, and Samir Khuller. Lp rounding for k-centers with non-uniform hard capacities. In *Foundations of Computer Science (FOCS), 2012 IEEE 53rd Annual Symposium on*, pages 273–282. IEEE, 2012. 2.3, 3.1, 3.2.1, 3.2.1, 3.2.4, 3.3.1, 3.3.2, 2, 3.3.3

[25] Wei Dai, Jinliang Wei, Xun Zheng, Jin Kyu Kim, Seunghak Lee, Junming Yin, Qirong

Ho, and Eric P Xing. Petuum: A framework for iterative-convergent distributed ml. *arXiv preprint arXiv:1312.7651*, 2013. 1, 2.1

[26] Sanjoy Dasgupta and Kaushik Sinha. Randomized partition trees for exact nearest neighbor search. *Algorithmica*, 72(1):237–263, 2015. 1, 5.2

[27] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 205–220. ACM, 2007. 1, 2.1

[28] John C Duchi, Alekh Agarwal, and Martin J Wainwright. Dual averaging for distributed optimization: convergence analysis and network scaling. *Automatic control, IEEE Transactions on*, 57(3):592–606, 2012. 1, 2.1

[29] Alina Ene, Sariel Har-Peled, and Benjamin Raichel. Fast clustering with lower bounds: No customer too far, no shop too small. *CoRR*, abs/1304.7318, 2013. 1, 2.3, 3.1

[30] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *The Journal of Machine Learning Research*, 9:1871–1874, 2008. 5.2

[31] Robson Leonardo Ferreira Cordeiro, Caitano Traina Junior, Agma Juci Machodo Traina, Julio López, U Kang, and Christos Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge Discovery and data mining*, pages 690–698, 2011. 2.3

[32] Jozsef Fritz. Distribution-free exponential error bound for nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 21(5):552–557, 1975. 4.2.3

[33] Sudipto Guha, Adam Meyerson, and Kamesh Munagala. Hierarchical placement and network design problems. In *FOCS*, pages 603–612. IEEE Computer Society, 2000. 3.1

[34] Robert A Jacobs, Michael I Jordan, Steven J Nowlan, and Geoffrey E Hinton. Adaptive mixtures of local experts. *Neural computation*, 3(1):79–87, 1991. 1, 2.2

[35] Martin Jaggi, Virginia Smith, Martin Takác, Jonathan Terhorst, Sanjay Krishnan, Thomas Hofmann, and Michael I Jordan. Communication-efficient distributed dual coordinate ascent. In *Advances in Neural Information Processing Systems*, pages 3068–3076, 2014. 2.1

[36] Michael I Jordan and Robert A Jacobs. Hierarchical mixtures of experts and the em algorithm. *Neural computation*, 6(2):181–214, 1994. 1, 2.2, 6

[37] David Karger, Eric Lehman, Tom Leighton, Rina Panigrahy, Matthew Levine, and Daniel Lewin. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 654–663. ACM, 1997. 1, 2.1

[38] David Karger, Alex Sherman, Andy Berkheimer, Bill Bogstad, Rizwan Dhanidina, Ken Iwamoto, Brian Kim, Luke Matkins, and Yoav Yerushalmi. Web caching with consistent hashing. *Computer Networks*, 31(11):1203–1213, 1999. 1, 2.1

[39] David R. Karger and Maria Minkoff. Building steiner trees with incomplete global knowledge. In *FOCS*, pages 613–623. IEEE Computer Society, 2000. 3.1

[40] Samir Khuller and Yoram J. Sussmann. The capacitated k-center problem. In *In Proceedings of the 4th Annual European Symposium on Algorithms, Lecture Notes in Computer Science 1136*, pages 152–166. Springer, 1996. 1, 2.3, 3.1, 3.2.1, 3.2.3, 3.3.1, 3.3.3

[41] Thomas Kiencke. Hadoop distributed file system (hdfs), 2013. 1, 2.1

[42] Gouthami Kondakindi, Satakshi Rana, Aswin Rajkumar, Sai Kaushik Ponnekanti, and Vinit Parakh. A logistic regression approach to ad click prediction. 1

[43] Samory Kpotufe. The curse of dimension in nonparametric regression. 2010. 4.3

[44] Tim Kraska, Ameet Talwalkar, John C Duchi, Rean Griffith, Michael J Franklin, and Michael I Jordan. Mlbase: A distributed machine-learning system. In *CIDR*, 2013. 1, 2.1

[45] Robert Krauthgamer and James R Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004. 4.3

[46] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. 5.2

[47] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 5.2

[48] Mu Li, David G Andersen, Alex J Smola, and Kai Yu. Communication efficient distributed machine learning with the parameter server. In *Advances in Neural Information Processing Systems*, pages 19–27, 2014. 1, 2.1, 5, 5.2

[49] Shanfei Li. An improved approximation algorithm for the hard uniform capacitated k-median problem. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, APPROX/RANDOM 2014, September 4-6, 2014, Barcelona, Spain*, pages 325–338, 2014. 1, 2.3, 3.1, 3.2, 3.2.4

[50] Shi Li. Approximating capacitated k-median with $(1 + \epsilon)k$ open facilities. *arXiv preprint arXiv:1411.5630*, 2014. 3.1

[51] Gaëlle Loosli, Stéphane Canu, and Léon Bottou. Training invariant support vector machines using selective sampling. *Large scale kernel machines*, pages 301–320, 2007. 5.2

[52] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012. 1, 2.1

[53] Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. *arXiv preprint arXiv:1502.03508*, 2015. 1, 2.1

[54] Mohammad Mahdian and Martin Pál. Universal facility location. In *Algorithms-ESA 2003*, pages 409–421. Springer, 2003. 3.1

[55] Ryan Mcdonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon S Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *Ad-*

vances in Neural Information Processing Systems, pages 1231–1239, 2009. 2.1

[56] Mark EJ Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005. 1

[57] Claire Monteleoni Nir Ailon, Ragesh Jaiswal. Streaming k-means approximation. In *NIPS*, 2009. 2.3

[58] Yashoteja Prabhu and Manik Varma. Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–272. ACM, 2014. 6

[59] Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In J. Shawe-Taylor, R.S. Zemel, P.L. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 693–701. Curran Associates, Inc., 2011. 2.1

[60] Christian Schüldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local svm approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36. IEEE, 2004. 2.2

[61] Nicola Segata and Enrico Blanzieri. Fast local support vector machines for large datasets. In *Machine Learning and Data Mining in Pattern Recognition*, pages 295–310. Springer, 2009. 2.2

[62] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alexander L Strehl, Alex J Smola, and SVN Vishwanathan. Hash kernels. In *International Conference on Artificial Intelligence and Statistics*, pages 496–503, 2009. 5.2

[63] Anshumali Shrivastava and Ping Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search. In *Advances in Neural Information Processing Systems*, 2014. 1

[64] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 5.2

[65] John N Tsitsiklis, Dimitri P Bertsekas, Michael Athans, et al. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986. 2.1

[66] Ruth Urner, Shai Shalev-Shwartz, and Shai Ben-David. Access to unlabeled data can speed up prediction time. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 641–648, 2011. 4.2.3

[67] Ruth Urner, Sharon Wulff, and Shai Ben-David. Plal: Cluster-based active learning. In *Conference on Learning Theory*, pages 376–397, 2013. 4.2.3

[68] Vladimir N. Vapnik and Lon Bottou. Local algorithms for pattern recognition and dependencies estimation. *Neural Computation*, 1993. (document), 1

[69] Ulrike Von Luxburg and Shai Ben-David. Towards a statistical theory of clustering. In *Pascal workshop on statistics and optimization of clustering*, pages 20–26, 2005. 2.4, 4, 4

[70] Yuchen Zhang, John Duchi, Michael Jordan, and Martin Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *Neural Information Processing Systems*, 2013. 1, 2.1

[71] Yuchen Zhang, John C. Duchi, and Martin Wainwright. Communication-efficient algorithms for statistical optimization. In *Neural Information Processing Systems*, 2012. 1, 2.1

[72] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. Parallelized stochastic gradient descent. In J.D. Lafferty, C.K.I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 2595–2603. Curran Associates, Inc., 2010. 2.1