Approximation Algorithms and New Models for Clustering and Learning

Pranjal Awasthi

August 2013
CMU-ML-13-107

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

Thesis Committee:
Avrim Blum, Chair
Anupam Gupta
Ryan O'Donnell
Ravindran Kannan, Microsoft Research India

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

This research was supported in part by the National Science Foundation under grants CCF-1116892 and IIS-1065251. 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: Clustering, PAC learning, Interactive learning,

To my parents.

Abstract

This thesis is divided into two parts. In part one, we study the k-median and the k-means clustering problems. We take a different approach than the traditional worst case analysis models. We show that by looking at certain well motivated stable instances, one can design much better approximation algorithms for these problems. Our algorithms achieve arbitrarily good approximation factors on stable instances, something which is provably hard on worst case instances. We also study a different model for clustering which introduces limited amount of interaction with the user. Such interactive models are very popular in the context of learning algorithms but their effectiveness for clustering is not well understood. We present promising theoretical and experimental results in this direction.

The second part of the thesis studies the design of provably good learning algorithms which work under adversarial noise. One of the fundamental problems in this area is to understand the learnability of the class of disjunctions of Boolean variables. We design a learning algorithm which improves on the guarantees of the previously best known result for this problem. In addition, the techniques used seem fairly general and promising to be applicable to a wider class of problems. We also propose a new model for learning with queries. This model restricts the algorithms ability to only ask certain "local" queries. We motivate the need for the model and show that one can design efficient local query algorithms for a wide class of problems.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Avrim Blum. Avrim has played a great role in shaping me up as a researcher. I truly admire his ability to work on a broad range of research directions and I am completely in awe of his approach towards solving problems: a seamless blend of intuition and rigor. He just makes it all look very easy. On top of that he is the nicest person I have ever met and has been extremely kind, patient and forgiving to me. Thanks Avrim for everything!

I would like to thank all my coauthors for being wonderful colleagues and teaching me a lot. Special mention goes to Or Sheffet. I was extremely fortunate to have him around as a collaborator since my early days in graduate school. Or has patiently listened to every bad idea of mine to help make it a good one. He has patiently listened to every bad talk of mine to help make it a good one. I have learnt a great deal from him and will always cherish our time together at CMU. Thanks Or, and I wish you all the best!

Many thanks to my thesis committee members. The presence of Anupam Gupta around CMU has always created a very relaxed and pleasant environment. He is one person I can always go to for advice. Ryan O'Donnell has always been very positive and excited about my work. I have truly enjoyed his courses and discussing research with him. Ravi Kannan is simply brilliant and it was an honor having him on my committee.

I would like to thank all the faculty and students of the CMU theory group. In particular, I would like to thank Manuel Blum and Steven Rudich, my two favorite teachers at CMU. I would also like to thank the administrative staff at CMU, especially Diane Stidle. She is awesome! My experience at CMU and in Pittsburgh has been a memorable one thanks to the many close friendships that I have formed. Special thanks to Sivaraman Balakrishnan, Vishal Dwivedi, Ravishankar Krishnaswamy, Vivek Seshadri, Aravindan Vijayaraghavan, Srivatsan Narayanan and Siddharth Gopal.

A very special thanks to Balaraman Ravindran and Venkatesan Chakaravarthy. As my undergraduate thesis advisor, Balaraman Ravindran introduced me to machine learning and gave me the first opportunity to do and experience research. I will always be grateful

to him. Venkatesan Chakaravarthy showed me how cool theory research can be. He is truly an inspiring person.

Finally, I would like to thank my parents without whom none of this would have been possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning is a prominent area of computer science with focus on analyzing data to identify patterns and make accurate predictions. Over the past decade, learning algorithms have found widespread applications in numerous areas such as computer vision, web search, natural language processing, computational biology etc. Traditionally, learning is classified as being either *unsupervised* or *supervised*. In unsupervised learning, the algorithm has access to data and the goal is to broadly classify the data into a small set of coherent groups, also known as clusters. For example, given a set of news articles, one might want to run an unsupervised learning algorithm to partition the set of articles into clusters corresponding to various topics such as sports, politics, science etc. This task is also popularly known as *clustering*.

In supervised learning, in addition to data, the algorithm also gets feedback on its performance on the data. This process usually involves a human in the loop. For example, consider the task of designing a spam filter for an e-mail system. In this case, a learning algorithm will have access to data consisting of various emails. In addition, each email will be labeled as *spam* or *not-spam* by a human who is typically a domain expert. The goal of the algorithm then is to come up with a prediction rule which can accurately classify future emails as spam or no spam.

In thesis we study important problems in both supervised and unsupervised learning. Below we briefly describe the main contributions of this thesis.

1.1 Clustering

Broadly, the goal of clustering is to partition n given data objects into k groups that share some commonality. This is often achieved by viewing the data as points in a metric space and then optimizing a given objective function over them. Two of the most popular such objectives are the k-median and the k-means objectives. An important direction for research is to design better approximation algorithms for these clustering objectives. Unfortunately, there are known hardness results which limit the possibility of achieving good approximations on worst case instances. In this thesis we attempt to bypass these hardness results by focusing on the kind of instances that might arise in practice, called stable instances. One might expect to find a distinguishing property of such instances that yields better approximation algorithms. There has been recent interest in exploring this direction of research(Ostrovsky et al. [2006], Balcan et al. [2009a], Kumar and Kannan [2010]). As discussed in Chapter 3 we define a new notion of stability which we call *weak-deletion* stability. We then design polynomial time approximation schemes for such instances. This improves upon previous work of Ostrovsky et al. [2006] and Balcan et al. [2009a] in two ways: our notion of stability is weaker and we achieve better approximation guarantees.

In the second part we take a different look at the problem of clustering. Clustering is traditionally defined as an unsupervised learning task and hence there is inherent uncertainty in the output of any traditional clustering algorithm. For example, there is no guarantee that the optimal solution to the k-means or the k-median objective is the desired target clustering which a particular user had in mind. In fact, there might be no principled way to reach the target clustering which a teacher has in mind without actually interacting with him/her. For example consider documents representing news articles. These documents could be clustered as $\{$politics, sports, entertainment, other$\}$. However, this is just one of the many possible clusterings. The clustering $\{$entertainment + sports, politics, other$\}$ is an equally likely apriori. Or perhaps the user would like these articles to be clustered into $\{$news articles$\}$ vs. $\{$opinion pieces$\}$. These scenarios motivate the need to consider the problem of clustering under feedback. Recently, there has been an interest in investigating such models and to come up with a more formal theoretical framework for analyzing clustering problems and algorithms. One such framework was proposed by Balcan and Blum [2008] who, motivated by different models for learning under queries, proposed a model for clustering under queries. As discussed in Chapters 4 and 5, we further explore the implications of their model and extend it in several important directions.

1.1.1 Results

- We consider k-median clustering in finite metric spaces and k-means clustering in Euclidean spaces, in the setting where k is part of the input (not a constant). We propose a notion of data stability which we call *weak deletion-stability*. An instance of k-median/k-means clustering satisfies weak deletion-stability if in the optimal solution, deleting any of the centers and assigning all points in cluster to one of the remaining $k-1$ centers results in an increase in the k-median/k-means cost by an (arbitrarily small) constant factor. We show that for such instances one can design polynomial time approximation schemes. This result also improves on earlier work of Ostrovsky et al. [2006] and Balcan et al. [2009a]

- We study a recently proposed framework(Balcan and Blum [2008]) for supervised clustering where there is access to a teacher. In this model any clustering algorithm works in stages. At each stage, the algorithm proposes a clustering to the teacher and gets limited feedback. We give an improved generic algorithm to cluster any concept class in this model. Our algorithm is query-efficient in the sense that it involves only a small amount of interaction with the teacher. We also propose and study various generalizations of the basic model which remove the assumption that the teacher responses to the algorithm are perfect. We also motivate and study the need to design local algorithms in this model. These are algorithms which are only allowed to make small local changes to the given clustering at each step. We show that under natural stability conditions on the data, one can design efficient local algorithms for supervised clustering.

1.2 Learning

The PAC model of learning introduced by Valiant [1984] is the most widely studied theoretical framework for studying learning problems. In this model, the algorithm gets access to samples from an unknown distribution which are labeled according to an unknown function in a class C of possible functions. The algorithm must produce a hypothesis which has good prediction performance for the labels of future samples drawn from the same distribution. There has been a lot of progress on various learning algorithms for important classes of functions in the PAC model of learning. However, an inherent assumption in the original PAC model is that there exists a function in the class C which perfectly predicts the labels of the samples. This assumption is unrealistic in practice and various variants of the PAC model have been studied which try to relax this assumption. In this thesis we

focus on the Agnostic PAC model of learning (Kearns and Valiant [1994]). In this model, it is assumed that no function in C is perfect and the goal is to output a hypothesis which approximates the error of best function in the class C to a small multiplicative factor. Learning under this model is notoriously difficult and very few positive results are known. For example, even the problem of agnostically learning the class of disjunctions is not well understood. We make progress (Chapter 7) on this problem by improving on the best known approximation factor. Our techniques also have the promise of being applicable to a wider class of functions.

In the second part we look at a different model of learning which is called the PAC + MQ model (MQ stands for membership queries). In this model, in addition to having access to random examples, the learner can also query for the label of any particular example x of its choice. This model is significantly more powerful than the PAC model and one can design efficient learning algorithms for complex classes of functions that seem out of reach in the PAC model using current techniques. Two celebrated results in this model are the algorithms of Bshouty [1993] and that of Kushilevitz and Mansour [1993] for learning decision trees and the algorithm of Jackson [1997] for learning DNF formulas under the uniform distribution. Despite being polynomial time algorithms, one unsatisfactory feature of these results is that the learner tends to query for labels of the points which seem very far away from typical points generated from the distribution. A more realistic query based algorithm would be one which takes samples from the distribution and additionally queries for points which are close to the samples. We call such queries as local Membership Queries (local MQs). In Chapter 8, we study the possibility of designing local MQ algorithms for decision trees and DNF formulas.

1.2.1 Results

- Given some arbitrary distribution D over $\{0,1\}^n$ and arbitrary target function f, the problem of agnostic learning of disjunctions is to achieve an error rate comparable to the error OPT_{disj} of the best disjunction with respect to (D, f). Achieving error $O(n \cdot \mathrm{OPT}_{disj}) + \epsilon$ is trivial, and the famous Winnow algorithm (Littlestone [1987]) achieves error $O(r \cdot \mathrm{OPT}_{disj}) + \epsilon$, where r is the number of relevant variables in the best disjunction. In recent work, Peleg [2007] shows how to achieve a bound of $\tilde{O}(\sqrt{n} \cdot \mathrm{OPT}_{disj}) + \epsilon$ in polynomial time. We improve on Peleg's bound, giving a polynomial-time algorithm achieving a bound of

  $$O(n^{1/3+\alpha} \cdot \mathrm{OPT}_{disj}) + \epsilon$$

  for any constant $\alpha > 0$.

- We propose a new model of learning under membership queries where the learning algorithm is restricted to queries which are local in nature. In other words the algorithm is only allowed to query points in the input space which are close to the distribution of the data. We argue that this is an important model to study and present a local algorithms for learning various classes of functions under a wide set of distributions. In particular, we show how to efficient learn sparse polynomials over $\{0,1\}^n$ using a local query algorithm under log-Lipschitz distributions. These distributions, in particualr, capture uniform, product and smooth distributions (Kalai et al. [2009b]) which are popularly studied in the context of learning problems.

Part I

Clustering

Chapter 2

Background

One of the most popular approaches to clustering is to define an objective function over the data points and find a partitioning which achieves the optimal solution, or an approximately optimal solution to the given objective function. Common objective functions include center based objective functions such as k-median and k-means where one selects k center points and the clustering is obtained by assigning each data point to its closest center point. In k-median clustering the objective is to find center points $c_1, c_2, \cdots c_k$, and a partitioning of the data so as to minimize $\Phi = \sum_x min_i\, d(x, c_i)$. This objective is historically very useful and well studied for facility location problems (Arya et al. [2004], Jain et al. [2002]). Similarly the objective in k-means is to minimize $\Phi = \sum_x min_i\, d(x, c_i)^2$. The k-means objective function is exactly the log-likelihood of data coming from a mixture of spherical Gaussians with identical variance. Hence, optimizing this objective is closely related to fitting the maximum likelihood mixture model for a given dataset. For a given set of centers, the optimal clustering for that set is obtained by assigning each data point to its closest center point. This is known as the Voronoi partitioning of the data. Unfortunately optimizing both these objectives turns out to be NP-hard. Hence a lot of the work in the theoretical community focuses on designing good approximation algorithms for these problems (Arya et al. [2004], Arora et al. [1998], Charikar et al. [1999a], de la Vega et al. [2003], Jain et al. [2002], Kanungo et al. [2002], Kumar et al. [2004], Ostrovsky et al. [2006], Balcan et al. [2009a]) with formal guarantees on worst case instances, as well as providing better guarantees for nicer, stable instances.

We will begin by describing a very popular heuristic for the k-means problem known as Lloyd's method. Lloyd's method (Lloyd [1982]) is an iterative procedure which starts out with a set of k seed centers and at each step computes a new set of centers with a lower k-means cost. This is achieved by computing the Voronoi partitioning of the current

set of centers and replacing each center with the center of the corresponding partition. We will describe the theoretical properties and limitations of Lloyd's method which will also motivate the need for good worst case approximation algorithms for k-means and k-median. We will see that the method is very sensitive to the choice of the seed centers.

2.1 Notation and Preliminaries

For a given a set S of n points, we will denote $\mathcal{C} = \{C_1, C_2, \ldots C_k\}$ to be a k-clustering instance. Here C_i's refer to the individual clusters. When studying k-median, we assume the n points reside in a finite metric space, and when discussing k-means, we assume they all reside in a finite dimensional Euclidean space. We denote $d : S \times S \to \mathbb{R}_{\geq 0}$ as the distance function which is an indication of how similar or dissimilar any two points in S are. A solution to the k-median objective partitions the n points into k disjoint subsets, $C_1, C_2, \ldots, C_k$ and assigns a center c_i for each subset. The k-median cost of this partition is then measured by $\sum_{i=1}^{k} \sum_{x \in C_i} d(x, c_i)$. A solution to the k-means objective again gives a k-partition of the n data points, but now we may assume uses the center of mass, $\mu_{C_i} = \frac{1}{|C_i|} \sum_{x \in C_i} x$, as the center of the cluster C_i. We then measure the k-means cost of this clustering by $\sum_{i=1}^{k} \sum_{x \in C_i} d^2(x, \mu_{C_i}) = \sum_{i=1}^{k} \sum_{x \in C_i} \|x - \mu_{C_i}\|^2$.

The optimal clustering (w.r.t. to either the k-median or the k-means objective) is denoted as $\mathcal{C}^* = \{C_1^*, C_2^*, \ldots, C_k^*\}$, and its cost is denoted as OPT. The centers used in the optimal clustering are denoted as $\{c_1^*, c_2^*, \ldots, c_k^*\}$. Clearly, given the optimal clustering, we can find the optimal centers (either by brute-force checking all possible points for k-median, or by $c_i^* = \mu_{C_i^*}$ for k-means). Alternatively, given the optimal centers, we can assign each x to its nearest center, thus obtaining the optimal clustering. Thus, we use $\mathcal{C}^*$ to denote both the optimal k-partition, and the optimal list of k centers. We use OPT_i to denote the contribution of the cluster i to OPT, that is $\mathsf{OPT}_i = \sum_{x \in C_i^*} d(x, c_i^*)$ in the k-median case, or $\mathsf{OPT}_i = \sum_{x \in C_i^*} d^2(x, c_i^*)$ in the k-means case.

2.2 Lloyd's method for k-means

Consider a set A of n points in the d-dimensional Euclidean space. We start by formally defining Voronoi partitions.

Definition 2.2.1 (Voronoi Partition)**.** *Given a clustering instance $\mathcal{C} \subset \mathbb{R}^d$ and k points $c_1, c_2, \cdots c_k$, a Voronoi partitioning using these centers consists of k disjoint clusters.*

Cluster i consists of all the points $x \in \mathcal{C}$ satisfying $d(x, c_i) \leq d(x, c_j)$ for all $j \neq i$.[1]

Lloyd's method is the most popular heuristic for k-means clustering in the Euclidean space which has been shown to be one of the most popular algorithms in data mining (Wu et al. [2008]). The method is an iterative procedure which is described below.

Lloyd's method

1. **Seeding:** Choose k seed points $c_1, c_2, \cdots c_k$. Compute the k-means cost using seed points as centers.
2. **Repeat:** Until no change in the k-means cost
   (a) **Voronoi partitioning:** Compute the Voronoi partitioning of the data based on the centers $c_1, c_2, \cdots c_k$. Let $C_1, C_2, \cdots, C_k$ be the corresponding clusters.
   (b) **Reseeding:** Compute new centers $\hat{c_1}, \hat{c_2}, \cdots, \hat{c_k}$, where $\hat{c_i} = \text{mean}(C_i)$. Here mean($C_i$) refers to the point obtained by taking coordinate-wise average of all the points in the set C_i. Compute the k-means cost using the new centers.

An attractive feature of Lloyd's method is that the k-means cost of the clustering obtained never increases. This follows from the fact that for any set of points, the 1-means cost is minimized by choosing the mean of the set as the center. Hence for any cluster C_i in the partitioning, choosing mean(C_i) will never lead to a solution of higher cost. Hence if we repeat this method until there is no change in the k-means cost, we will reach a local optimum of the k-means cost function in finite time. In particular the number of iterations will be at most $n^{O(kd)}$ which is the maximum number of Voronoi partitions of a set of n points in $\Re^d$ (Inaba et al. [1994]). The basic method mentioned above leads to a class of algorithms depending upon the choice of the seeding method. A simple way is to start with k randomly chosen data points. This choice however can lead to arbitrarily bad solution quality as shown in Figure 2.1. In addition it is also known that the Lloyd's method can take upto 2^n iterations to converge even in 2 dimensions (Arthur and Vassilvitskii [2006], Vattani [2009]).

In sum, from a theoretical standpoint, k-means with random/arbitrary seeds is not a good clustering algorithm in terms of efficiency or quality. Nevertheless, the speed and simplicity of k-means are quite appealing in practical applications. Therefore, recent work has focused on improving the initialization procedure: deciding on a better way to initialize the clustering dramatically changes the performance of the Lloyd's iteration, both in

[1] Ties can be broken arbitrarily.

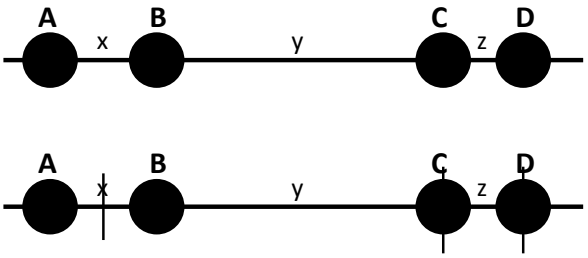


Figure 2.1: Consider 4 points $\{A, B, C, D\}$ on a line separated by distances x, y and z such that $z < x < y$. Let $k = 3$. The optimal solution has centers at A, B and the centroid of C, D with a total cost of $\frac{z^2}{2}$. When choosing random seeds, there is a constant probability that we choose $\{A, C, D\}$. In this case the final centers will be C, D and the centroid of A, B with a total cost of $\frac{x^2}{2}$. This ratio can be made arbitrarily bad.

terms of quality and convergence properties. For example, Arthur and Vassilvitskii [2007] showed that choosing a good set of seed points is crucial and if done carefully can itself be a good candidate solution without the need for further iterations. Their algorithm called k-means$++$ uses the following seeding procedure: it selects only the first center uniformly at random from the data and each subsequent center is selected with a probability proportional to its contribution to the overall error given the previous selections. See Algorithm KMEANS++for a formal description:

---

kmeans++

1. Initialize a set S by choosing a data point at random.
2. While $|S| < k$, choose a data point x with probability proportional to $min_{z \in S} d(x, z)^2$, and add it to S.
3. Output the clustering obtained by the Voronoi partitioning of the data using the centers in S.

Arthur and Vassilvitskii [2007] showed that Algorithm KMEANS++is an $\log k$ approximation algorithm for the k-means objective. We say that an algorithm is an α-approximation for a given objective function Φ if for every clustering instance the algorithm outputs a solution of expected cost at most α times the cost of the best solution. The design of approximation algorithms for NP-hard problems has been a fruitful research direction and has led to a wide array of tools and techniques. Formally, Arthur and Vassilvitskii [2007] show that:

Theorem 2.2.2 (Arthur and Vassilvitskii [2007])**.** *Let S be the set of centers output by the above algorithm and $cost(S)$ be the k-means cost of the clustering obtained using S as the centers. Then $E[cost(S)] \leq O(\log k)\mathsf{OPT}$, where OPT is the cost of the optimal k-means solution.*

We would like to point out that in general the output of k-means$++$ is not a local optimum. Hence it might be desirable in practice to run a few steps of the Lloyd's method starting from this solution. This could only lead to a better solution.

Subsequent work of Ailon et al. [2009] introduced a streaming algorithm inspired by the k-means++ algorithm that makes a single pass over the data. They show that if one is allowed to cluster using a little more than k centers, specifically $O(k \log k)$ centers, then one can achieve a constant-factor approximation in expectation to the k-means objective.

Such approximation algorithms which use more than k centers are also known as bi-criteria approximations.

As mentioned earlier, Lloyd's method can take up to exponential iterations in order to converge to a local optimum. However Arthur et al. [2011] showed that the method converges quickly on an "average" instance. In order to formalize this, they study the problem under the smoothed analysis framework of Spielman and Teng [2004]. In the smoothed analysis framework the input is generated by applying a small Gaussian perturbation to an adversarial input. Spielman and Teng [2004] showed that the simplex method takes polynomial number of iterations on such smoothed instances. In a similar spirit, Arthur et al. [2011] showed that for smoothed instances Lloyd's method runs in time polynomial in n, the number of points and $\frac{1}{\sigma}$, the standard deviation of the Gaussian perturbation. However, these works do not provide any guarantee on the quality of the final solution produced.

We would like to point out that in principle the Lloyd's method can be extended to the k-median objective. A natural extension would be to replace the mean computation in the Reseeding step with computing the median of a set of points X in the Euclidean space, i.e., a point $c \in \Re^d$ such that $\sum_{x \in X} d(x, c)$ is minimized. However this problem turns out to be NP-hard (Megiddo and Supowit [1984]). For this reason, the Lloyd's method is typically used only for the k-means objective.

2.3 Properties of the k-means objective

In this section we provide some useful facts about the k-means clustering objective. We will use $\mathcal{C}$ to denote the set of n points which represent a clustering instance. The first fact can be used to show that given a Voronoi partitioning of the data, replacing a given center with the mean of the corresponding partition can never increase the k-means cost.

Fact 2.3.1. *Consider a finite set $X \subset \mathbb{R}^d$ and c =mean(X). For any $y \in \mathbb{R}^d$, we have that, $\sum_{x \in X} d(x, y)^2 = \sum_{x \in X} d(x, c)^2 + |X| d(c, y)^2$.*

Proof. Representing each point in the coordinate notation as $x = (x_1, x_2, \cdots, x_d)$, we

have that

$$\begin{aligned}
\sum_{x\in X} d(x,y)^2 &= \sum_{x\in X}\sum_{i=1}^{d} |x_i - y_i|^2 \\
&= \sum_{x\in X}\sum_{i=1}^{d} (|x_i - c_i|^2 + |c_i - y_i|^2 + 2(x_i - c_i)(c_i - y_i)) \\
&= \sum_{x\in X} d(x,c)^2 + |X| d(c,y)^2 + \sum_{i=1}^{d} 2(c_i - y_i) \sum_{x\in X} (x_i - c_i) \\
&= \sum_{x\in X} d(x,c)^2 + |X| d(c,y)^2
\end{aligned}$$

Here the last equality follows from the fact that for any i, $c_i = \sum_{x\in X} x_i/n$. □

An easy corollary of the above fact is the following,

Corollary 2.3.2. *Consider a finite set $X \subset \mathbb{R}^d$ and let c = mean(X). We have $\sum_{x,y\subset X} d(x,y)^2 = |X| \sum_{x\in X} d(x,c)^2$.*

Below we prove another fact which will be useful later.

Fact 2.3.3. *Let $X \subset \mathbb{R}^d$ be finite set of points. Let ${\Delta_1}^2(X)$ denote the 1-means cost of X. Given a partition of X into X_1 and X_2 such that c =mean(X), c_1 =mean(X_1) and c_2 = mean(X_2), we have that a) ${\Delta_1}^2(X) = {\Delta_1}^2(X_1) + {\Delta_1}^2(X_2) + \frac{|X_1||X_2|}{|X|} d(c_1,c_2)^2$. and b) $d(c,c_1)^2 \leq \frac{{\Delta_1}^2(X)|X_2|}{|X||X_1|}$.*

Proof. We can write ${\Delta_1}^2(X) = \sum_{x\in X_1} d(x,c)^2 + \sum_{x\in X_2} d(x,c)^2$. Using Fact 2.3.1 we can write

$$\sum_{x\in X_1} d(x,c)^2 = {\Delta_1}^2(X_1) + |X_1| d(c,c_1)^2.$$

Similarly, $\sum_{x\in X_2} d(x,c)^2 = {\Delta_1}^2(X_2) + |X_2| d(c,c_2)^2$. Hence we have

$$ {\Delta_1}^2(X) = {\Delta_1}^2(X_1) + {\Delta_1}^2(X_2) + |X_1| d(c,c_1)^2 + |X_2| d(c,c_2)^2.$$

Part (a) follows by substituting $c = \frac{|X_1|c_1 + |X_2|c_2}{|X_1| + |X_2|}$ in the above equation.

From Part (a) we have that

$$ {\Delta_1}^2(X) \geq \frac{|X_1||X_2|}{|X|} d(c_1,c_2)^2.$$

Part (b) follows by substituting $c_2 = \frac{(|X_1|+|X_2|)}{X_2}c - \frac{|X_1|}{|X_2|}c_1$ above. □

Fact 2.3.4. *Let S be a (finite) set of points in an Euclidean space, and let $c = mean(S)$ denote their center of mass ($c = \frac{1}{|S|}\sum_{x\in S} x$). Let A be a random subset of S of size m, and denote by c_A the center of mass of A. Then for any $\delta < 1/2$, we have both*

$$\Pr\left[d^2(c, c_A) > \frac{1}{\delta m} \cdot \Delta_1^2(S)\right] < \delta \tag{2.1}$$

$$\Pr\left[\sum_{x\in S} d^2(x, c_A) > |S|(1 + \frac{1}{\delta m}) \cdot \Delta_1^2(S)\right] < \delta \tag{2.2}$$

Proof. First, notice that $E[c_A] = c$. Also it is easy to see that $E[d^2(c, c_A)] = \frac{1}{m}\Delta_1^2(S)$. Equation 3.1 then follows from Markov's inequality. Equation 3.2 follows from Equation 3.1 and noting that $\sum_{x\in S} d^2(x, c_A) = |S|\Delta_1^2(S) + |S|d^2(c, c_A)$ (See Fact 2.3.1). □

2.4 Hierarchical clustering

So far we have discussed clustering algorithms which take as input the number of clusters k and optimize an objective function for the specific value of k. Another line of work in the clustering literature studies the design of clustering algorithms which are oblivious to the value of k. In other words, one should be able to get a k clustering for any desired value of k from the output of the algorithm. Such algorithms, often called hierarchical or agglomerative algorithms, induce a tree structure over the given set of data points. The root of the tree contains a single node with all the points in one cluster. On the other hand, the bottom level contains n clusters with each point belonging to its own cluster. Given such a tree structure, it is easy to output a k clustering: simply output the set of nodes at level k from the root node. We will briefly describe some popular hierarchical clustering algorithms. See Hartigan [1985] for a detailed discussion.

2.4.1 Single Linkage algorithm

The single linkage algorithm is one of the simplest hierarchical clustering algorithm. The algorithm starts with the given n points as leaf nodes. At each step, the algorithm merges the two closest nodes in the tree to create a new internal node. Given two sets of points A and B, the distance between them is defined as $d_{min}(A, B) = min_{x\in A, y\in B} d(x, y)$. The algorithm is shown below.

Single Linkage

1. Initialize the n leaf nodes with the n given data points in S. Mark all the nodes as *active*.
2. While # active nodes > 1
   - Let A and B be the active nodes with the minimum value of $d_{min}(A, B)$.
   - Create a new parent node $N = A \cup B$ connected to A and B. Mark A and B as inactive and N as active.
3. Output the tree constructed.

Although simple to describe, the single linkage algorithm is not very popular in practice and is very sensitive to the presence of outliers in the data (Hartigan [1985]). A more robust version of the single linkage algorithm is the *average linkage* algorithm. The overall structure is the same as single linkage except the choice of the distance function used to measure similarity between two sets of points. As the name suggests, the average linkage algorithm uses the average distance between two sets of points as a measure of the distance between the two sets. In other words, the distance between A and B is defined to be $d_{avg}(A, B) = \frac{1}{|A||B|} \sum_{x \in A, y \in B} d(x, y)$. The algorithm is shown below

Average Linkage

1. Initialize the n leaf nodes with the n given data points in S. Mark all the nodes as *active*.
2. While # active nodes > 1
   - Let A and B be the active nodes with the minimum value of $d_{avg}(A, B)$.
   - Create a new parent node $N = A \cup B$ connected to A and B. Mark A and B as inactive and N as active.
3. Output the tree constructed.

Chapter 3

Approximation algorithms for clustering

In this chapter, we consider two popular clustering objectives, k-median and k-means. Both measure a k-partition of the points by choosing a special point for each cluster, called the *center*, and define the *cost* of a clustering as a function of the distances between the data points and their respective centers. In the k-median case, the cost is the sum of the distances of the points to their centers, and in the k-means case, the cost is the sum of these distances squared. The k-median objective is typically studied for data in a finite metric (complete weighted graph satisfying triangle inequality) over the n data points; k-means clustering is typically studied for n points in a (finite dimensional) Euclidean space. Both objectives are known to be NP-hard (we view k as part of the input and not a constant, though even the 2-means problem in Euclidean space was recently shown to be NP-hard (Dasgupta [2008])). For k-median in a finite metric, there is a known $(1 + 1/e)$-hardness of approximation result (Jain et al. [2002]) and substantial work on approximation algorithms (Guha and Khuller [1998], Charikar et al. [1999b], Arya et al. [2001], Jain et al. [2002], de la Vega et al. [2003]), with the best guarantee a $3 + \epsilon$ approximation. For k-means in a Euclidean space, there is also a vast literature of approximation algorithms (Ostrovsky and Rabani [2000], Bādoiu et al. [2002], de la Vega et al. [2003], Effros and Schulman [2004], Har-Peled and Mazumdar [2004], Kanungo et al. [2002]) with the best guarantee a constant-factor approximation if polynomial dependence on k and the dimension d is desired.[1]

[1]If k is constant, then k-median in finite metrics can be trivially solved in polynomial time and there is a PTAS known for k-means in Euclidean space (Kumar et al. [2004]). There is also a PTAS known for low-dimensional Euclidean spaces (Arora et al. [1998], Har-Peled and Mazumdar [2004]).

3.1 Bypassing NP-hardness

The work of Ostrovsky et al. [2006] proposed a notion of stability under which one can achieve better k-means approximations in time polynomial in n and k. They consider k-means instances where the optimal k-clustering has cost noticeably smaller than the cost of any $(k-1)$-clustering, motivated by the idea that "if a near-optimal k-clustering can be achieved by a partition into fewer than k clusters, then that smaller value of k should be used to cluster the data" (Ostrovsky et al. [2006]). Under the assumption that the ratio of the cost of the optimal $(k-1)$-means clustering to the cost of the optimal k-means clustering is at least $\max\{100, 1/\epsilon^2\}$, Ostrovsky et al. show that one can obtain a $(1+f(\epsilon))$-approximation for k-means in time polynomial in n and k, by using a variant on Lloyd's algorithm.

Balcan et al. [2009a], motivated by the fact that objective functions are often just a proxy for the underlying goal of getting the data clustered correctly, study clustering instances that satisfy the condition that all $(1+\alpha)$ approximations to the given objective (e.g., k-median or k-means) are δ-close, in terms of how points are partitioned, to a target clustering (such as a correct clustering of proteins by function or a correct clustering of images by who is in them). Such instances are called $(1+\alpha, \delta)$ approximation-stable instances. Balcan et al. show that for any α and δ, given an instance satisfying this property for k-median or k-means objectives, one can in fact efficiently produce a clustering that is $O(\delta/\alpha)$-close to the target clustering (so, $O(\delta)$-close for any constant $\alpha > 0$), *even though obtaining a* $1+\alpha$ *approximation to the objective is* NP*-hard* for $\alpha < \frac{1}{e}$, and remains hard even under this assumption. Thus they show that one can approximate the target even though it is hard to approximate the objective.

3.2 Our results

In this thesis we study a new notion of stability which we call as *weak deletion-stability*. We show that it is implied by both the separation condition of Ostrovsky et al. [2006] as well as (when target clusters are large) the stability condition of Balcan et al. [2009a]. We design polynomial time approximation schemes for instances of k-median and k-means which satisfy weak deletion-stability .

As a byproduct we improve on the approximation guarantee provided in Ostrovsky et al. [2006]. We show that under the much weaker assumption that the ratio of these costs is just at least $(1+\alpha)$ for *some* constant $\alpha > 0$, we can achieve a PTAS: namely, $(1+\epsilon)$-approximate the k-means optimum, for any constant $\epsilon > 0$. Thus, we decouple the

strength of the assumption from the quality of the conclusion, and in the process allow the assumption to be substantially weaker.

Our result also resolves a question raised by the work of Balcan et al. [2009a] regarding the approximability of the k-median and the k-means objectives when all target clusters in the $(1+\alpha, \delta)$ stable instance are large compared to δn. Our result can be used to show that for both k-median and k-means objectives, if all clusters contain more than δn points, then for any constant $\alpha > 0$ we can in fact get a PTAS. Thus, we (nearly) resolve the approximability of these objectives under stability condition of Balcan et al. [2009a]. Note that this further implies finding a δ-close clustering (setting $\epsilon = \alpha$). Thus, we also extend the results of Balcan et al. [2009a] in the case of large clusters and constant α by getting exactly δ-close for both k-median and k-means objectives. (In Balcan et al. [2009a] this exact closeness was achieved for the k-median objective but needed a somewhat larger $O(\delta n(1+1/\alpha))$ minimum cluster size requirement). The results in this chapter are based on work in Awasthi et al. [2010a].

3.3 Stability Properties

As mentioned above, our results are achieved by exploiting implications of a stability condition we call weak deletion-stability. In this section we define weak deletion-stability and relate it to conditions of Ostrovsky et al. [2006] and Balcan et al. [2009a].

Definition 3.3.1. *For $\alpha > 0$, a k-median/k-means instance satisfies* $(1+\alpha)$ weak deletion-stability*, if it has the following property. Let $\{c_1^*, c_2^*, \ldots, c_k^*\}$ denote the centers in the optimal k-median/k-means solution. Let OPT denote the optimal k-median/k-means cost and let $\mathsf{OPT}^{(i \to j)}$ denote the cost of the clustering obtained by removing c_i^* as a center and assigning all its points instead to c_j^*. Then for any $i \neq j$, it holds that*

$$\mathsf{OPT}^{(i \to j)} > (1+\alpha)\mathsf{OPT}$$

3.3.1 ORSS-Separability

Ostrovsky et al. [2006] define a clustering instance to be ϵ-separated if the optimal k-means solution is cheaper than the optimal $(k-1)$-means solution by at least a factor ϵ^2. For a given objective (k-means or k-median) let us use $\mathsf{OPT}_{(k-1)}$ to denote the cost of the optimal $(k-1)$-clustering. Introducing a parameter $\alpha > 0$, say a clustering instance is

$(1+\alpha)$-*ORSS separable* if

$$\frac{\mathsf{OPT}_{(k-1)}}{\mathsf{OPT}} > 1+\alpha$$

If an instance satisfies $(1+\alpha)$-ORSS separability then all $(k-1)$ clusterings must have cost more than $(1+\alpha)\mathsf{OPT}$ and hence it is immediately evident that the instance will also satisfy $(1+\alpha)$-weak deletion-stability. Hence we have the following claim:

Claim 3.3.2. *Any $(1+\alpha)$-ORSS separable k-median/k-means instance is also $(1+\alpha)$-weakly deletion stable.*

Proof. If an instance satisfies $(1+\alpha)$-ORSS separability then all $(k-1)$ clusterings must have cost more than $(1+\alpha)\mathsf{OPT}$. For all i, j, $\mathsf{OPT}^{i\to j}$ denotes the cost of a particular $(k-1)$ clustering denotes as $C^{i\to j}$, and hence $\mathsf{OPT}^{i\to j} > (1+\alpha)\mathsf{OPT}$. It is now immediately evident that the instance will also satisfy $(1+\alpha)$-weak deletion-stability. □

3.3.2 BBG-Stability

Balcan et al. [2009a] (see also Balcan and Braverman [2009] and Balcan et al. [2009b]) consider a notion of stability to approximations motivated by settings in which there exists some (unknown) target clustering $\mathcal{C}^{target}$ we would like to produce. Balcan et al. [2009a] define a clustering instance to be $(1+\alpha, \delta)$ approximation-stable with respect to some objective Φ (such as k-median or k-means), if any k-partition whose cost under Φ is at most $(1+\alpha)\mathsf{OPT}$ agrees with the target clustering on all but at most δn data points. That is, for any $(1+\alpha)$ approximation $\mathcal{C}$ to objective Φ, we have $\min_{\sigma\in S_k}\sum_i |C_i^{target} - C_{\sigma(i)}| \leq \delta n$ (here, σ is simply a matching of the indices in the target clustering to those in $\mathcal{C}$). In general, δn may be larger than the smallest target cluster size, and in that case approximation-stability need not imply weak deletion-stability (not surprisingly since Balcan et al. [2009a] show that k-median and k-means remain hard to approximate). However, when all target clusters have size greater than δn (note that δ need not be a constant) then approximation-stability indeed also implies weak deletion-stability, allowing us to get a PTAS (and thereby δ-close to the target) when $\alpha > 0$ is a constant.

Claim 3.3.3. *A k-median/k-means clustering instance that satisfies $(1+\alpha, \delta)$ approximation-stability, and in which all clusters in the target clustering have size greater than δn, also satisfies $(1+\alpha)$ weak deletion-stability.*

Proof. Consider an instance of k-median/k-means clustering which satisfies $(1+\alpha, \delta)$ approximation-stability. As before, let $\{c_1^*, c_2^*, \ldots, c_k^*\}$ be the centers in the optimal solution and consider the clustering $C^{(i\to j)}$ obtained by no longer using c_i^* as a center and

instead assigning each point from cluster i to c_j^*, making the ith cluster empty. The distance of this clustering from the target is defined as $\frac{1}{n}\min_{\sigma\in S_k}\sum_{i'}|C_{i'}^{target}-C_{\sigma(i')}^{(i\to j)}|$. Since $C^{(i\to j)}$ has only $(k-1)$ nonempty clusters, one of the target clusters must map to an empty cluster under any permutation σ. Since by assumption, this target cluster has more than δn points, the distance between C^{target} and $C^{(i\to j)}$ will be greater than δ and hence by the BBG stability condition, the k-median/k-means cost of $C^{(i\to j)}$ must be greater than $(1+\alpha)\mathsf{OPT}$. □

We now state our main result:

Theorem 3.3.4. *Given $\epsilon,\alpha > 0$ and a $(1+\alpha)$ weakly deletion-stable instance of k-median clustering, one can find a $(1+\epsilon)$ approximation to the k-median objective in time $n^{O(1/\alpha\epsilon)}k^{O(1/\alpha)}$.*

A similar result holds true for k-means clustering in Euclidean space

Theorem 3.3.5. *Given $\epsilon,\alpha > 0$ and a $(1+\alpha)$ weakly deletion-stable instance of k-means clustering, one can find a $(1+\epsilon)$ approximation to the k-means objective in time $(k\log n)^{O(1/\alpha\epsilon)}O(n^3)$.*

Finally, we would like to point out that NP-hardness of the k-median problem is maintained even if we restrict ourselves only to weakly deletion-stable instances. This is proved in the following theorem

Theorem 3.3.6. *For any constant $\alpha > 0$, finding the optimal k-median clustering of $(1+\alpha)$-weakly deletion-stable instances is NP-hard.*

Also the reduction (See in Section 3.8.3) uses only integer poly-size distances, and hence rules out the existence of a FPTAS for the problem, unless P = NP. Thus, our algorithm, is optimal in the sense that the super-polynomial dependence on $1/\epsilon$ and $1/\alpha$ in the running time is unavoidable. In addition, the reduction can be modified to show that NP-hardness is maintained under the conditions studied in Ostrovsky et al. [2006] and Balcan et al. [2009a].

Our algorithms mentioned in the following sections use the following important consequence of weakly deletion stable instances.

Theorem 3.3.7. *Let C^* be the optimal k-median/k-means instance of a $(1+\alpha)$-weakly deletion-stable instance. Then we have for any cluster C_i^* and any point $p\notin C_i^*$,*

$$d(c_i^*, p) > \frac{\alpha}{2}\frac{OPT}{|C_i^*|} \qquad \textit{(for k-median)} \tag{3.1}$$

$$d^2(c_i^*, p) > \frac{\alpha}{4}\frac{OPT}{|C_i^*|} \qquad \textit{(for k-means)} \tag{3.2}$$

Proof. Fix any center in the optimal k-clustering, c_i^*, and fix any point p that does not belong to the C_i^* cluster. Denote by C_j^* the cluster that p is assigned to in the optimal k-clustering. Therefore it must hold that $d(p, c_j^*) \leq d(p, c_i^*)$. Consider the clustering obtained by deleting c_i^* from the list of centers, and assigning each point in C_i^* to C_j^*. Since the instance is $(1+\alpha)$-weakly deletion-stable this should increase the cost by at least $\alpha\mathsf{OPT}$.

Suppose we are dealing with a k-median instance. Each point $x \in C_i^*$ originally pays $d(x, c_i^*)$, and now, assigned to c_j^*, it pays $d(x, c_j^*) \leq d(x, c_i^*) + d(c_i^*, c_j^*)$. Thus, the new cost of the points in C_i^* is upper bounded by $\sum_{x \in C_i^*} d(x, c_j^*) \leq \mathsf{OPT}_i + |C_i^*| d(c_i^*, c_j^*)$. As the increase in cost is lower bounded by $\alpha\mathsf{OPT}$ and upper bounded by $|C_i^*| d(c_i^*, c_j^*)$, we deduce that $d(c_i^*, c_j^*) > \alpha\frac{\mathsf{OPT}}{|C_i^*|}$. Observe that triangle inequality gives that $d(c_i^*, c_j^*) \leq d(c_i^*, p) + d(p, c_j^*) \leq 2d(c_i^*, p)$, so we have that $d(c_i^*, p) > (\alpha/2)\frac{\mathsf{OPT}}{|C_i^*|}$.

Suppose we are dealing with a Euclidean k-means instance. Again, we have created a new clustering by assigning all points in C_i^* to the center c_j^*. Thus, the cost of transitioning from the optimal k-clustering to this new $(k-1)$-clustering, which is at least $\alpha\mathsf{OPT}$, is upper bounded by $\sum_{x \in C_i^*} \|x - c_j^*\|^2 - \|x - c_i^*\|^2$. As $c_i^* = \mu_{C_i^*}$, it follows that this bound is *exactly* $\sum_{x \in C_i^*} \|c_j^* - c_i^*\|^2 = |C_i^*| d^2(c_i^*, c_j^*)$, see Inaba et al. [1994] (§2, Theorem 2). It follows that $d^2(c_i^*, c_j^*) > \alpha\frac{\mathsf{OPT}}{|C_i^*|}$. As before, $d^2(c_i^*, c_j^*) \leq \big(d(c_i^*, p) + d(p, c_j^*)\big)^2 \leq 4d^2(c_i^*, p)$, so $d^2(c_i^*, p) > \frac{\alpha}{4}\frac{\mathsf{OPT}}{|C_i^*|}$. □

3.4 Algorithm Intuition and Techniques

We now informally describe the algorithm for finding a $(1+\epsilon)$-approximation of the k-median optimum for weakly deletion-stable instances. First, we comment that using a standard doubling technique, we can assume we approximately know the value of OPT.[2]

[2]Instead of doubling from 1, we can alternatively run an off-the-shelf 5-approximation of OPT, which will return a value $v \leq 5\mathsf{OPT}$.

Our algorithm works if instead of OPT we use a value v s.t. $\mathsf{OPT} \leq v \leq (1+\epsilon/2)\mathsf{OPT}$, but for ease of exposition, we assume that the exact value of OPT is known.

Furthermore we will assume that our instance does not have any cluster which dominates the overall cost of the optimal clustering. Specifically, we say a cluster C_i^* in the optimal k-median clustering $\mathcal{C}^*$ (hereafter also referred to as the target clustering) is *cheap* if $\mathsf{OPT}_i \leq \frac{\alpha\epsilon\mathsf{OPT}}{64}$, otherwise, we say C_i^* is *expensive*. Note that in any event, there can be at most a constant $(\frac{64}{\alpha\epsilon})$ number of expensive clusters.

The intuition for our algorithm and for introducing the notion of cheap clusters is the following. Pick some cluster C_i^* in the optimal k-median clustering. The first main observation is that if the instance is $(1+\alpha)$ weakly deletion-stable, then any $x \notin C_i^*$ is far from c_i^*, namely, $d(x, c_i^*) > \alpha\frac{\mathsf{OPT}}{2|C_i^*|}$. In contrast, the average distance of $x \in C_i^*$ from c_i^* is $\frac{\mathsf{OPT}_i}{|C_i^*|}$. Thus, if we focus on a cluster whose contribution, OPT_i, is no more than, say, $\frac{\alpha}{200}\mathsf{OPT}$, we have that c_i^* is 100 times closer, on *average*, to the points of C_i^* than to the points outside C_i^*. Furthermore, using the triangle inequality we have that any two "average" points of C_i^* are of distance at most $\frac{\alpha}{100}\frac{\mathsf{OPT}}{|C_i^*|}$, while the distance between any such "average" point and any point outside of C_i^* is at least $\frac{99\alpha}{200}\frac{\mathsf{OPT}}{|C_i^*|}$. So, if we manage to correctly guess the size s of a cheap cluster, we can set a radius $r = \Theta\left(\alpha\frac{\mathsf{OPT}}{s}\right)$ and collect data-points according to the size and intersection of the r-balls around them. We note that this use of balls with an inverse relation between size and radius is similar to that in the min-sum clustering algorithm of Balcan and Braverman [2009].

Note that in the general case we might have up to $\frac{32}{\alpha\epsilon}$ expensive clusters. We handle them by brute force guessing their centers.

The general algorithm populates a list $\mathcal{Q}$, where each element in this list is a subset of points. Ideally, each subset is contained in some target cluster, yet we might have a few subsets with points from two or more target clusters. The first stage of the algorithm is to add components into $\mathcal{Q}$, and the second stage is to find k good components in $\mathcal{Q}$, and use these k components to retrieve a clustering with low cost.

Since we do not have many expensive clusters, we can run the algorithm for all possible guesses for the centers of the expensive clusters and choose the solution which has the minimum cost. It can be shown that one such guess will lead to a solution of cost at most $(1+\epsilon)\mathsf{OPT}$. The complete algorithm for k-median is shown in Figure 3.1.

For the case of k-means in Euclidean space, we use sampling techniques, similar to those of Kumar et al. [2004] and Ostrovsky et al. [2006], to get good substitutes for the centers of the expensive clusters. Note however an important difference between the approach of Kumar et al. [2004], Ostrovsky et al. [2006] and ours. While they sample points

from all k clusters, we sample points only for the $O(1)$ expensive clusters. As a result, the runtime of the PTAS of Kumar et al. [2004], Ostrovsky et al. [2006] has exponential dependence in k, while ours has only a polynomial dependence in k.

1. **Input** $\mathcal{Q} \leftarrow \mathcal{Q}_{init}$.
2. **Population Stage:** For $s = n, n-1, n-2, \ldots, 1$ do:
   (a) Set $r = \frac{\alpha \mathsf{OPT}}{8s}$.
   (b) Remove any point x such that $d(x, \mathcal{Q}) < 2r$.
   (Here, $d(x, \mathcal{Q}) = \min_{T \in \mathcal{Q}; y \in T} d(x, y)$.)
   (c) For any remaining data point x, denote the set of data points whose distance from x is at most r, by $B(x, r)$. Connect any two remaining points a and b if: (i) $d(a, b) \leq r$, (ii) $|B(a, r)| > \frac{s}{2}$ and (iii) $|B(b, r)| > \frac{s}{2}$.
   (d) Let T be a connected component of size $> \frac{s}{2}$. Then:
      i. Add T to $\mathcal{Q}$. (That is, $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{T\}$.)
      ii. Define the set $B(T) = \{x : \ d(x, y) \leq 2r \text{ for some } y \in T\}$. Remove the points of $B(T)$ from the instance.
3. **Centers-Retrieving Stage:** For any choice of k components $T_1, T_2, \ldots, T_k$ out of $\mathcal{Q}$ (we later show that $|\mathcal{Q}| < k + O(1/\alpha)$)
   (a) Find the best center c_i for $T_i \cup B(T_i)$. That is $c_i = \arg\min_{p \in T_i \cup B(T_i)} \sum_{x \in T_i \cup B(T_i)} d(x, p)$.[a]
   (b) Partition all n points according to the nearest point among the k centers of the current k components.
   (c) If a clustering of cost at most $(1+\epsilon)\mathsf{OPT}$ is found – output these k centers and halt.

[a] This can be done before fixing the choice of k components out of $\mathcal{Q}$.

Figure 3.1: The algorithm to obtain a PTAS for weakly deletion-stable instances of k-median.

3.5 Formal Analysis

We now analyze the algorithm is presented in Figure 3.1. We assume that at the beginning, the list $\mathcal{Q}$ is initialized with $\mathcal{Q}_{init}$ which contains the centers of all the expensive clusters. In general, the algorithm will be run several times with $\mathcal{Q}_{init}$ containing different guesses for the centers of the expensive clusters. Before going into the proof of correctness of the algorithm, we introduce another definition. We define the *inner ring* of C_i^* as the set $\left\{x;\ d(x, c_i^*) \leq \frac{\alpha \mathsf{OPT}}{16|C_i^*|}\right\}$. Note the following fact:

Fact 3.5.1. *If C_i^* is a cheap cluster, then no more than an $\epsilon/4$ fraction of its points reside outside the inner ring. In particular, at least half of a cheap cluster is contained within the inner ring.*

Proof. This follows from Markov's inequality. If more than $(\epsilon/4)|C_i^*|$ points are outside of the inner ring, then $\mathsf{OPT}_i > \frac{\epsilon|C_i^*|}{4} \cdot \frac{\alpha \mathsf{OPT}}{16|C_i^*|} = \alpha\epsilon\mathsf{OPT}/64$. This contradicts the fact that C_i^* is cheap. □

Our high level goal is to show that for any cheap cluster C_i^* in the target clustering, we insert a component T_i that is contained within C_i^*, and furthermore, contains only points that are close to c_i^*. It will follow from the next claims that the component T_i is the one that contains points from the inner ring of C_i^*. We start with the following Lemma which we will utilize a few times.

Lemma 3.5.2. *Let T be any component added to $\mathcal{Q}$. Let s be the stage in which we add T to $\mathcal{Q}$. Let C_i^* be any cheap cluster s.t. $s \geq |C_i^*|$. Then (a) T does not contain any point z s.t. the distance $d(c_i^*, z)$ lies within the range $\left[\frac{\alpha}{4}\frac{\mathsf{OPT}}{|C_i^*|}, \frac{3\alpha}{8}\frac{\mathsf{OPT}}{|C_i^*|}\right]$, and (b) T cannot contain both a point p_1 s.t. $d(c_i^*, p_1) < \frac{\alpha}{4}\frac{\mathsf{OPT}}{|C_i^*|}$ and a point p_2 s.t. $d(c_i^*, p_2) > \frac{3\alpha}{8}\frac{\mathsf{OPT}}{|C_i^*|}$.*

Proof. We prove (a) by contradiction. Assume T contains a point z s.t. $\frac{\alpha}{4}\frac{\mathsf{OPT}}{|C_i^*|} \leq d(c_i^*, z) \leq \frac{3\alpha}{8}\frac{\mathsf{OPT}}{|C_i^*|}$. Set $r = \frac{\alpha \mathsf{OPT}}{8s} \leq \frac{\alpha \mathsf{OPT}}{8|C_i^*|}$, just as in the stage when T was added to $\mathcal{Q}$, and let p be any point in the ball $B(z, r)$. Then by the triangle inequality we have that $d(c_i^*, p) \geq d(c_i^*, z) - d(z, p) \geq \frac{\alpha}{8}\frac{\mathsf{OPT}}{|C_i^*|}$, and similarly $d(c_i^*, p) \leq d(c_i^*, z) + d(z, p) \leq \frac{\alpha \mathsf{OPT}}{2|C_i^*|}$. Since our instance is weakly deletion-stable it holds that p belongs to C_i^*, and from the definition of the inner ring of C_i^*, it holds that p falls *outside* the inner ring. However, z is added to T because the ball $B(z, r)$ contains more than $s/2 \geq |C_i^*|/2$ many points. So more than half of the points in C_i^* fall outside the inner ring of C_i^*, which contradicts Fact 3.5.1.

Assume now (b) does not hold. Recall that T is a connected component, so exists some path $p_1 \to p_2$. Each two consecutive points along this path were connected because their distance is at most $\frac{\alpha\mathsf{OPT}}{8s} \leq \frac{\alpha\mathsf{OPT}}{8|C_i^*|}$. As $d(c_i^*, p_1) < \frac{\alpha}{4}\frac{\mathsf{OPT}}{|C_i^*|}$ and $d(c_i^*, p_2) > \frac{3\alpha}{8}\frac{\mathsf{OPT}}{|C_i^*|}$, there must exist a point z along the path whose distance from c_i^* falls in the range $\left[\frac{\alpha}{4}\frac{\mathsf{OPT}}{|C_i^*|}, \frac{3\alpha}{8}\frac{\mathsf{OPT}}{|C_i^*|}\right]$, contradicting (a). □

Claim 3.5.3. *Let C_i^* be any cheap cluster in the target clustering. By stage $s = |C_i^*|$, the algorithm adds to $\mathcal{Q}$ a component T that contains a point from the inner ring of C_i^*.*

Proof. Suppose that up to the stage $s = |C_i^*|$ the algorithm has not inserted such a component into $\mathcal{Q}$. Now, it is possible that by stage s, the algorithm has inserted some component T' to $\mathcal{Q}$, s.t. some x in the inner ring of C_i^* is too close to some $y \in T'$ (namely, $d(x, y) \leq 2r$), thus causing x to be removed from the instance. Assume for now this is not the case. This means that the inner ring of cluster C_i^* still contains more than $|C_i^*|/2$ points. Also observe that all inner ring points are of distance at most $\frac{\alpha\mathsf{OPT}}{16|C_i^*|}$ from the center, so every pair of inner ring points has a distance of at most $\frac{\alpha\mathsf{OPT}}{8|C_i^*|}$. Hence, when we reach stage $s = |C_i^*|$, any ball of radius $r = \frac{\alpha\mathsf{OPT}}{8s} = \frac{\alpha\mathsf{OPT}}{8|C_i^*|}$ centered at any inner-ring point, must contain all other inner-ring points. This means that at stage $s = |C_i^*|$ all inner ring points are connected among themselves, so they form a component (in fact, a clique) of size $> s/2$. Therefore, the algorithm inserts a new component, containing all inner ring points.

So, by stage $s = |C_i^*|$, one of two things can happen. Either the algorithm inserts a component that contains some inner ring point to Q, or the algorithm removes an inner ring point due to some component $T' \in \mathcal{Q}$. If the former happens, we are done. So let us prove by contradiction that we cannot have only the latter.

Let $s \geq |C_i^*|$ be the stage in which we throw away the first inner ring point of the cluster C_i^*. At stage s the algorithm removes this inner ring point x because there exists a point y in some component $T' \in Q$, s.t. $d(x, y) \leq 2r = \frac{\alpha\mathsf{OPT}}{4s}$, and so $d(c_i^*, y) \leq d(c_i^*, x) + d(x, y) \leq \frac{\alpha\mathsf{OPT}}{16|C_i^*|} + \frac{\alpha\mathsf{OPT}}{4s} \leq \frac{5}{16}\frac{\alpha\mathsf{OPT}}{|C_i^*|}$. This immediately implies that T' cannot be the center of an expensive cluster since any such point will be at a distance at least $\frac{\alpha\mathsf{OPT}}{2|C^*|}$ from c_i^*. Let $s' \geq s \geq |C_i^*|$ be the previous stage in which we added the component T' to Q. As Lemma 3.5.2 applies to T', we deduce that $d(c_i^*, y) < \frac{\alpha}{4}\frac{\mathsf{OPT}}{|C_i^*|}$. Recall that T' contains $> s'/2 \geq |C_i^*|/2$ many points, yet, by assumption, contains none of the $|C_i^*|/2$ points that reside in the inner ring of C_i^*. It follows from Fact 3.5.1 that some point $w \in T'$ must belong to a different cluster C_j^*. Since the instance is weakly deletion-stable , we have that $d(c_i^*, w) > \frac{\alpha\mathsf{OPT}}{2|C_i^*|}$. The existence of both y and w in T' contradicts part (b)

of Lemma 3.5.2. □

We call a component $T \in \mathcal{Q}$ *good* if it contains an inner ring point of some cheap cluster C_i^*. A component is called *bad* if it is not good and is not one of the initial centers present in $\mathcal{Q}_{init}$. We now discuss the properties of good components.

Claim 3.5.4. *Let T be a good component added to $\mathcal{Q}$, containing an inner ring point from a cheap cluster C_i^*. (By Claim 3.5.3 we know at least one such T exists.) Then: (a) all points in T are of distance at most $\frac{\alpha \mathsf{OPT}}{4|C_i^*|}$ from c_i^*, (b) $T \cup B(T)$ is fully contained in C_i^*, and (c) the entire inner ring of C_i^* is contained in $T \cup B(T)$, and (d) no other component $T' \in \mathcal{Q}$, $T' \neq T$, contains an inner ring point from C_i^*.*

Proof. As we do not know (d) in advance, it might be the case that $\mathcal{Q}$ contains many good components, all containing an inner-ring point from the same cluster, C_i^*. Out of these (potentially many) components, let T denote the first one inserted to $\mathcal{Q}$. Denote the stage in which T was inserted to $\mathcal{Q}$ as s. Due to the previous claim, we know $s \geq |C_i^*|$, and so Lemma 3.5.2 applies to T. We show (a), (b), (c) and (d) hold for T, and deduce that T is the only good component to contain an inner ring point from C_i^*.

Part (a) follows immediately from Lemma 3.5.2. We know T contains some inner ring point x from C_i^*, so $d(c_i^*, x) \leq \frac{\alpha}{16} \frac{\mathsf{OPT}}{|C_i^*|} < \frac{\alpha}{4} \frac{\mathsf{OPT}}{|C_i^*|}$, so we know that any $y \in T$ must satisfy that $d(c_i^*, y) < \frac{\alpha}{4} \frac{\mathsf{OPT}}{|C_i^*|}$. Since we now know (a) holds and the instance is weakly deletion-stable , we have that $T \subset C_i^*$, so we only need to show $B(T) \subset C_i^*$. Fix any $y \in B(T)$. The point y is assigned to $B(T)$ (thus removed from the instance) because there exists some point $x \in T$ s.t. $d(x, y) \leq 2r$. So again, we have that $d(c_i^*, y) \leq d(c_i^*, x) + d(x, y) \leq \frac{\alpha \mathsf{OPT}}{2|C_i^*|}$, which gives us that $y \in C_i^*$ (since the instance is weakly deletion-stable).

We now prove (c). Because of (b), we deduce that the number of points in T is at most $|C_i^*|$. However, in order for T to be added to $\mathcal{Q}$, it must also hold that $|T| > s/2$. It follows that $s < 2|C_i^*|$. Let x be an inner ring point of C_i^* that belongs to T. Then the distance of any other inner ring point of C_i^* and x is at most $\frac{\alpha \mathsf{OPT}}{8|C_i^*|} < \frac{\alpha \mathsf{OPT}}{4s} = 2r$. It follows that any inner ring point of C_i^* which isn't added to T is assigned to $B(T)$. Thus $T \cup B(T)$ contains all inner-ring points. Finally, observe that (d) follows immediately from the definition of a good component and from (c). □

We now show that in addition to having all k good components, we cannot have too many bad components.

Claim 3.5.5. *We have less than $32/(3\alpha)$ bad components.*

Proof. Let T be a bad component, and let s be the stage in which T was inserted to $\mathcal{Q}$. Let y be any point in T, and let C^* be the cluster to which y belongs in the optimal clustering with center c^*. We show $d(c^*, y) > \frac{3\alpha}{16}\frac{\mathsf{OPT}}{s}$. We divide into cases.

Case 1: C^* is an expensive cluster. Note that we are working under the assumption that $\mathcal{Q}_{init}$ contains the correct centers of the expensive clusters. In particular, $\mathcal{Q}_{init}$ contains c^*. Also, the fact that point y was not thrown out in stage s implies that $d(c^*, y) > 2r = \frac{\alpha\mathsf{OPT}}{4s} > \frac{3\alpha\mathsf{OPT}}{16s}$.

Case 2: C^* is a cheap cluster and $s \geq |C^*|$. We apply Lemma 3.5.2, and deduce that either $d(c^*, y) < \frac{\alpha}{4}\frac{\mathsf{OPT}}{|C^*|}$ or that $d(c^*, y) > \frac{3\alpha}{8}\frac{\mathsf{OPT}}{|C^*|} \geq \frac{3\alpha}{8}\frac{\mathsf{OPT}}{s}$. As the inner ring of C^* contains $> |C^*|/2$ and T contains $> s/2 \geq |C^*|/2$ many points, none of which is an inner ring point, some point $w \in T$ does not belong to C^* and hence $d(c^*, w) > \frac{\alpha\mathsf{OPT}}{2|C^*|} > \frac{3\alpha}{8}\frac{\mathsf{OPT}}{|C^*|}$. Part (b) of Lemma 3.5.2 assures us that all points in T are also far from c^*.

Case 3: C^* is a cheap cluster and $s < |C^*|$. Using Claim 3.5.3 we have that some good component containing a point x from the inner ring of C^* was already added to $\mathcal{Q}$. So it must hold that $d(x, y) > 2r$, for otherwise we removed y from the instance and it cannot be added to any T. We deduce that $d(c^*, y) \geq d(x, y) - d(c^*, x) \geq \frac{\alpha\mathsf{OPT}}{4s} - \frac{\alpha\mathsf{OPT}}{16|C^*|} > \frac{3\alpha}{16}\frac{\mathsf{OPT}}{s}$.

All points in T have distance $> \frac{3\alpha\mathsf{OPT}}{16s}$ from their respective centers in the optimal clustering, and recall that T is added to $\mathcal{Q}$ because T contains at least $s/2$ many points. Therefore, the contribution of all elements in T to OPT is at least $\frac{3\alpha\mathsf{OPT}}{32}$. It follows that we can have no more than $32/3\alpha$ such bad components. □

We can now prove the correctness of our algorithm.

Theorem 3.5.6. *The algorithm outputs a k-clustering whose cost is no more than $(1 + \epsilon)\mathsf{OPT}$.*

Proof. Using Claim 3.5.4, it follows that there exists some choice of k components, $T_1, \ldots, T_k$, such that we have the center of every expensive cluster and the good component corresponding to every cheap cluster $\mathcal{C}^*$. Fix that choice. We show that for the optimal clustering, replacing the true centers $\{c_1^*, c_2^*, ..., c_k^*\}$ with the centers $\{c_1, c_2, ..., c_k\}$ that the algorithm outputs, increases the cost by at most a $(1 + \epsilon)$ factor. This implies that using the $\{c_1, c_2, ..., c_k\}$ as centers must result in a clustering with cost at most $(1+\epsilon)\mathsf{OPT}$.

Fix any C_i^* in the optimal clustering. Let OPT_i be the cost of this cluster. If C_i^* is an expensive cluster then we know that its center c_i^* is present in the list of centers chosen. Hence, the cost paid by points in C_i^* will be at most OPT_i. If C_i^* is a cheap cluster then denote by T the good component corresponding to it. We break the cost of C_i^* into two parts: $\mathsf{OPT}_i = \sum_{x \in C_i^*} d(x, c_i^*) = \sum_{x \in T \cup B(T)} d(x, c_i^*) + \sum_{x \in C_i^*, \text{ yet } x \notin T \cup B(T)} d(x, c_i^*)$ and

compare it to the cost C_i^* using c_i, the point picked by the algorithm to serve as center: $\sum_{x\in C_i^*} d(x,c_i) = \sum_{x\in T\cup B(T)} d(x,c_i) + \sum_{x\in C_i^*, \text{ yet } x\notin T\cup B(T)} d(x,c_i)$. Now, the first term is exactly the function that is minimized by c_i, as $c_i = \arg\min_p \sum_{x\in T\cup B(T)} d(x,p)$. We also know c_i^*, the actual center of C_i^*, resides in the inner ring, and therefore, by Claim 3.5.4 must belong to $T \cup B(T)$. It follows that $\sum_{x\in T\cup B(T)} d(x,c_i) \leq \sum_{x\in T\cup B(T)} d(x,c_i^*)$. We now upper bound the 2nd term, and show that $\sum_{x\in C_i^*, \text{ yet } x\notin T\cup B(T)} d(x,c_i) \leq (1+\epsilon)\sum_{x\in C_i^*, \text{ yet } x\notin T\cup B(T)} d(x,c_i^*)$

Any point $x \in C_i^*$, s.t. $x \notin T \cup B(T)$, must reside outside the inner ring of C_i^*. Therefore, $d(x,c_i^*) > \frac{\alpha\mathsf{OPT}}{16|C_i^*|}$. We show that $d(c_i,c_i^*) \leq \epsilon\frac{\alpha\mathsf{OPT}}{16|C_i^*|}$, and thus we have that $d(x,c_i) \leq d(x,c_i^*) + d(c_i^*,c_i) \leq (1+\epsilon)d(x,c_i^*)$, which gives the required result.

Note that thus far, we have only used the fact that the cost of any cheap cluster is proportional to $\alpha\mathsf{OPT}/|C_i^*|$. Here is the first (and the only) time we use the fact that the cost is actually at most $(\epsilon/64)\cdot\alpha\mathsf{OPT}/|C_i^*|$. Using the Markov inequality, we have that the set of points satisfying $\{x;\ d(x,c_i^*) \leq \epsilon\cdot\alpha\mathsf{OPT}/(32|C_i^*|)\}$ contains at least half of the points in C_i^*, and they all reside in the inner ring, thus belong to $T\cup B(T)$. Assume for the sake of contradiction that $d(c_i,c_i^*) \geq \epsilon\frac{\alpha\mathsf{OPT}}{16|C_i^*|}$. Then at least half of the points in C_i^* contribute more than $\epsilon\frac{\alpha\mathsf{OPT}}{32|C_i^*|}$ to the sum $\sum_{x\in T\cup B(T)} d(x,c_i)$. It follows that this sum is more than $\epsilon\frac{\alpha\mathsf{OPT}}{64|C_i^*|} \geq \mathsf{OPT}_i$. However, c_i is the point that minimizes the sum $\sum_{x\in T\cup B(T)} d(x,p)$, and by using $p = c_i^*$ we have $\sum_{x\in T\cup B(T)} d(x,p) \leq \mathsf{OPT}_i$. Contradiction.

□

3.5.1 Runtime analysis

A naive implementation of the 2nd step of algorithm in Section 3.4 takes $O(n^3)$ time (for every s and every point x, find how many of the remaining points fall within the ball of radius r around it). Finding c_i for all components takes $O(n^2)$ time, and measuring the cost of the solution using a particular set of k data points as centers takes $O(nk)$ time. Guessing the right k components takes $k^{O(1/\alpha)}$ time. Overall, the running time of the algorithm in Figure 3.1 is $O(n^3k^{O(1/\alpha)})$. The general algorithm that brute-force guesses the centers of all expensive clusters, makes $n^{O(1/\alpha\epsilon)}$ iterations of the given algorithm, so its overall running time is $n^{O(1/\alpha\epsilon)}k^{O(1/\alpha)}$.

3.6 A PTAS for any $(1+\alpha)$weakly deletion-stable Euclidean k-Means Instance

Analogous to the k-median algorithm, we present an essentially identical algorithm for k-means in Euclidean space. Indeed, the fact that k-means considers distances squared, makes upper (or lower) bounding distances a bit more complicated, and requires that we fiddle with the parameters of the algorithm. In addition, the centers c_i^* may not be data points. However, the overall approach remains the same. Roughly speaking, converting the k-median algorithm to the k-means case, we use the same constants, only squared. As before we handle expensive clusters by guessing good substitutes for their centers and obtain *good* components for cheap clusters.

Often, when considering the Euclidean space k-means problem, the dimension of the space plays an important factor. In contrast, here we make no assumptions about the dimension, and our results hold for any $\mathrm{poly}(n)$ dimension. In fact, for ease of exposition, we assume all distances between any two points were computed in advance and are given to our algorithm. Clearly, this only adds $O(n^2 \cdot \mathsf{dim})$ to our runtime. In addition to the change in parameters, we utilize the following facts that hold for the center of mass in Euclidean space.

Fact 2.3.3, proven in Ostrovsky et al. [2006] (Lemma 2.2), allows us to upper bound the distance between the real center of a cluster and the empirical center we get by averaging all points in $T \cup B(T)$ for a good component T. Fact 2.3.4 allows us to handle expensive clusters. Since we cannot brute force guess a center (as the center of the clusters aren't necessarily data points), we guess a sample of $O(\beta^{-1} + \epsilon^{-1})$ points from every expensive cluster, and use their average as a center. Both properties of Fact 2.3.4, proven in Inaba et al. [1994] (§3, Lemma 1 and 2), assure us that the center is an adequate substitute for the real center and is also close to it. This motivates the approach behind our first algorithm, in which we brute-force traverse all choices of $O(\epsilon^{-1} + \beta^{-1})$ points for any of the expensive clusters.

The second algorithm, whose runtime is $(k \log n)^{\mathrm{poly}(1/\epsilon, 1/\beta)} O(n^3)$, replaces brute-force guessing with random sampling. Indeed, if a cluster contains $\mathrm{poly}(1/k)$ fraction of the points, then by randomly sampling $O(\epsilon^{-1} + \beta^{-1})$ points, the probability that all points belong to the same expensive cluster, and furthermore, their average can serve as a good empirical center, is at least $1/k^{\mathrm{poly}(1/\epsilon, 1/\beta)}$. In contrast, if we have expensive clusters that contain few points (e.g. an expensive cluster of size $\sqrt{n}$, while $k = \mathrm{poly}(\log(n))$), then random sampling is unlikely to find good empirical centers for them. However, recall that our algorithm collects points and deletes them from our instance. So, it is possible that

in the middle of the run, we are left with so few points, so that expensive clusters whose size is small in comparison to the original number of points, contain a $\mathrm{poly}(1/k)$ fraction of the *remaining* points.

Indeed, this is the motivation behind our second algorithm. We run the algorithm while interleaving the Population Stage of the algorithm with random sampling. Instead of running s from n to 1, we use $\left\{n, \frac{n}{k^2}, \frac{n}{k^4}, \frac{n}{k^6}, \ldots, 1\right\}$ as break points. Correspondingly, we define l_i to be the number of expensive clusters whose size is in the range $[n \cdot k^{-2i-2}, n \cdot k^{-2i})$. Whenever s reaches such a $n \cdot k^{-2i}$ break point, we randomly sample points in order to guess the l_{i+3} centers of the clusters that lie 3 intervals "ahead" (and so, initially, we guess all centers in the first 3 intervals). We prove that in every interval we are likely to sample good empirical centers. This is a simple corollary of Fact 2.3.4 along with the following two claims. First, we claim that at the end of each interval, the number of points remaining is at most $n \cdot k^{-2i+1}$. Secondly, we also claim that in each interval we do not remove even a single point from a cluster whose size is smaller than $n \cdot k^{-2i-6}$. We refer the reader to Section 3.8.1 for the algorithms and their analysis.

3.7 Other notions of stability and relations between them

The notion of ORSS-separability, is related to the notion of approximation-stability discussed in section 3.3.2. Indeed, from Theorem 5.1 in, Ostrovsky et al. [2006] it follows that $(1+\alpha)$-separability implies that any near-optimal solution to k-means is $O((1-\alpha)^2)$-close to the k-means optimal clustering. However, the converse is not necessarily the case: an instance could satisfy approximation-stability without being ORSS-separable.[3] Balcan et al. [2013] presents a specific example of points in Euclidean space with $c = 2$. In fact, for the case that k is much larger than $1/\delta$, the difference between the two properties can be more substantial. See Figure 3.2 for an example.

Kumar and Kannan [2010] consider the problem of recovering a target clustering under deterministic separation conditions that are motivated by the k-means objective and by Gaussian and related mixture models. They consider the setting of points in Euclidean space, and show that if the projection of any data point onto the line joining the mean of its cluster in the target clustering to the mean of any other cluster of the target is $\Omega(k)$ standard deviations closer to its own mean than the other mean, then they can recover the target

[3]Ostrovsky et al. [2006] shows an implication in this direction (Theorem 5.2); however, this implication requires a substantially stronger condition, namely that data satisfy $((1+\alpha), \epsilon)$-BBG-stability for $c = 1/\epsilon^2 - 1$ (and that target clusters be large). In contrast, the primary interest of Balcan et al. [2013] is in the case where c is below the threshold for existence of worst-case approximation algorithms.

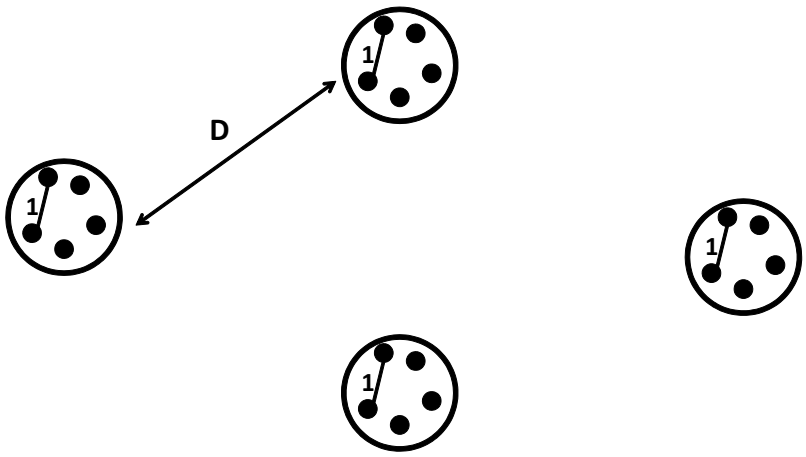


Figure 3.2: Suppose δ is a small constant, and consider a clustering instance in which the target consists of $k = \sqrt{n}$ clusters with $\sqrt{n}$ points each, such that all points in the same cluster have distance 1 and all points in different clusters have distance $D+1$ where D is a large constant. Then, merging two clusters increases the cost additively by $\Theta(\sqrt{n})$, since D is a constant. Consequently, the optimal $(k-1)$-means/median solution is just a factor $1 + O(1/\sqrt{n})$ more expensive than the optimal k-means/median clustering. However, for D sufficiently large compared to $1/\delta$, this example satisfies $(2, \delta)$-BBG-stability or even $(1/\delta, \delta)$-BBG-stability – see Balcan et al. [2013] for formal details.

clusters in polynomial time. This condition was further analyzed and reduced by work of Awasthi et al. [2012]. This separation condition is formally incomparable to weakly deletion-stable (even restricting to the case of k-means with points in Euclidean space). In particular, if the dimension is low and k is large compared to $1/\alpha$, then this condition can require more separation than weakly deletion-stable (e.g., with k well-spaced clusters of unit radius BBG-stability would require separation only $O(1/\alpha)$ and independent of k). On the other hand if the clusters are high-dimensional, then this condition can require less separation than weakly deletion-stable since the ratio of projected distances will be more pronounced than the ratios of distances in the original space.

Bilu and Linial [2010] consider inputs satisfying the condition that the optimal solution to the objective remains optimal even after bounded perturbations to the input weight matrix. They give an algorithm for maxcut (which can be viewed as a 2-clustering problem) under the assumption that the optimal solution is stable to (roughly) $O(n^{2/3})$-factor multiplicative perturbations to the edge weights. Awasthi et al. [2012] consider this condition for center-based clustering objectives such as k-median and k-means, and give an algorithm that finds the optimal solution when the input is stable to only factor-3 perturbations. This factor is improved to $1 + \sqrt{2}$ by Balcan and Liang [2012], who also design algorithms under a relaxed (c, ϵ)-stability to perturbations condition in which the optimal solution need not be identical on the c-perturbed instance, but may change on an ϵ fraction of the points (in this case, the algorithms require $c = 2 + \sqrt{7}$). Note that for the k-median objective, $(1+\alpha, \delta)$-BBG-stability with respect to $\mathcal{C}^*$ implies $(1+\alpha, \delta)$-stability to perturbations because an optimal solution in a $(1 + \alpha)$-perturbed instance is guaranteed to be a $(1 + \alpha)$-approximation on the original instance;[4] so, $((1 + \alpha), \delta)$-stability to perturbations is a weaker condition. Similarly, for k-means, $((1 + \alpha), \delta)$-stability to perturbations is implied by $((1 + \alpha)^2, \delta)$-BBG-stability. However, as noted above, the values of α known to lead to efficient clustering in the case of stability to perturbations are larger than for BBG-stability, where any constant $\alpha > 0$ suffices.

[4]In particular, a $(1 + \alpha)$-perturbed instance $\tilde{d}$ satisfies $d(x, y) \leq \tilde{d}(x, y) \leq (1 + \alpha)d(x, y)$ for all points x, y. So, using Φ to denote cost in the original instance, $\tilde{\Phi}$ to denote cost in the perturbed instance and using $\tilde{\mathcal{C}}$ to denote the optimal clustering under $\tilde{\Phi}$, we have $\Phi(\tilde{C}) \leq \tilde{\Phi}(\tilde{\mathcal{C}}) \leq \tilde{\Phi}(\mathcal{C}^*) \leq (1 + \alpha)\Phi(\mathcal{C}^*)$.

3.8 Additional Proofs

3.8.1 Algorithm for weakly deletion-stable k-Means Instances

We present the algorithm for $(1+\epsilon)$-approximation to the k-means optimum of a weakly deletion-stable instance. Much like in Section 3.4, we call a cluster in the optimal k-means solution cheap if $\mathsf{OPT}_i = \sum_{x \in C_i^*} d^2(x, c_i^*) \leq \frac{\alpha\epsilon\mathsf{OPT}}{4^7}$. The algorithm is presented in Figure 3.3. The correctness is proved in a similar fashion to the proof of correctness presented in Section 3.4. First, observe that by the Markov inequality, for any cheap cluster C_i^*, we have that the set $\left\{x;\ d^2(x, c_i^*) > t\frac{\alpha\mathsf{OPT}}{4|C_i^*|}\right\}$ cannot contain more than $\epsilon/(4^7 t)$ fraction of the points in $|C_i^*|$. It follows that the inner ring of C_i^*, the set $\left\{x;\ d^2(x, c_i^*) \leq \frac{\alpha\mathsf{OPT}}{1024|C_i^*|}\right\}$, contains at least half of the points of C_i^*. As mentioned Section 3.6 the algorithm populates the list $\mathcal{Q}$ with *good* components corresponding to cheap clusters. Also from Section 3.6, we know that for every expensive cluster, there exists a sample of $O(\frac{1}{\alpha} + \frac{1}{\epsilon})$ data points whose center is a good substitute for the center of the cluster. In the analysis below, we assume that $\mathcal{Q}$ has been initialized correctly with $\mathcal{Q}_{init}$ containing these good substitutes. In general, the algorithm will be run multiple times for all possible guesses of samples from expensive clusters. We start with the following lemma which is similar to Lemma 3.5.2.

Lemma 3.8.1. *Let $T \in \mathcal{Q}$ be any component and let s be the stage in which we insert T to $\mathcal{Q}$. Let C_i^* be any cheap cluster s.t. $s \geq |C_i^*|$. Then (a) T does not contain any point z s.t. the distance $d^2(c_i^*, z)$ lies within the range $\left[\frac{\alpha}{64}\frac{\mathsf{OPT}}{|C_i^*|}, \frac{\alpha}{1}6\frac{\mathsf{OPT}}{|C_i^*|}\right]$, and (b) T cannot contain both a point p_1 s.t. $d^2(c_i^*, p_1) \leq \frac{\alpha}{64}\frac{\mathsf{OPT}}{|C_i^*|}$ and a point p_2 s.t. $d^2(c_i^*, p_2) > \frac{\alpha}{16}\frac{\mathsf{OPT}}{|C_i^*|}$.*

Proof. Assume (a) does not hold. Let z be such point, and let $B(z, r)$ be the set of all points p s.t. $d^2(z, p) \leq r = \frac{\alpha\mathsf{OPT}}{256s} \leq \frac{\alpha\mathsf{OPT}}{256|C_i^*|}$. As $d^2(z, c_i^*) \geq \frac{\alpha\mathsf{OPT}}{64|C_i^*|}$, we have that $d(z, p) \leq \frac{1}{2}d(z, c_i^*)$. It follows that $d^2(c_i^*, p) \geq \left(d(c_i^*, z) - d(z, p)\right)^2 \geq \left(d(c_i^*, z)/2\right)^2 = \frac{\alpha\mathsf{OPT}}{256|C_i^*|}$. Similarly, $d^2(c_i^*, p) \leq \left(d(c_i^*, z) + d(z, p)\right)^2 \leq \left(3d(c_i^*, z)/2\right)^2 \leq \frac{9\alpha}{64}\frac{\mathsf{OPT}}{|C_i^*|}$. Thus $B(z, r)$ is contained in C_i^*, but falls outside the inner-ring of C_i^*, yet contains $s/2 \geq |C_i^*|/2$ many points. Contradiction.

Assume (b) does not hold. Let p_1 and p_2 the above mentioned points. As T is a connected components, it follows that along the path $p_1 \to p_2$, exists a pairs of neighboring nodes, x, y, s.t. $d^2(x, y) \leq r \leq \frac{\alpha\mathsf{OPT}}{256|C_i^*|}$ yet $d^2(c_i^*, x) \leq \frac{\alpha}{64}\frac{\mathsf{OPT}}{|C_i^*|}$ while $d^2(c_i^*, y) \geq \frac{\alpha}{16}\frac{\mathsf{OPT}}{|C_i^*|}$. However, a simple computation gives that $d^2(c_i^*, y) \leq \left(3d(c_i^*, x)/2\right)^2 \leq \frac{9\alpha}{256}\frac{\mathsf{OPT}}{|C_i^*|}$. Contradiction. □

1. **Initialization Stage:** Set $\mathcal{Q} \leftarrow \mathcal{Q}_{init}$.

2. **Population Stage:** For $s = n, n-1, n-2, \ldots, 1$ do:

   (a) Set $r = \frac{\alpha \mathsf{OPT}}{256s}$.

   (b) Remove any point x such that $d^2(x, \mathcal{Q}) < 4r$.
   (Here, $d(x, \mathcal{Q}) = \min_{T \in \mathcal{Q}; y \in T} d(x, y)$.)

   (c) For any remaining data point x, denote the set of data points whose distance squared from x is at most r, by $B(x, r)$. Connect any two remaining points a and b if:
   (i) $d^2(a, b) \leq r$, (ii) $|B(a, r)| > \frac{s}{2}$ and (iii) $|B(b, r)| > \frac{s}{2}$.

   (d) Let T be a connected component of size $> \frac{s}{2}$. Then:

      i. Add T to $\mathcal{Q}$. (That is, $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{T\}$.)

      ii. Define the set $B(T) = \{x : \ d^2(x, y) \leq 4r \text{ for some } y \in T\}$. Remove the points of $B(T)$ from the instance.

3. **Centers-Retrieving Stage:** For any choice of k components $T_1, T_2, \ldots, T_k$ out of $\mathcal{Q}$

   (a) Find the best center c_i for $T_i \cup B(T_i)$.
   That is $c_i = \mu(T_i \cup B(T_i)) = \frac{1}{|T_i \cup B(T_i)|} \sum_{x \in T_i \cup B(T_i)} x$.

   (b) Partition all n points according to the nearest point among the k centers of the current k components.

   (c) If a clustering of cost at most $(1+\epsilon)\mathsf{OPT}$ is found – output these k centers and halt.

Figure 3.3: A PTAS for weakly deletion-stable instances of Euclidean k-means.

Lemma 3.8.1 allows us to give the analogous claims to Claims 3.5.3 and 3.5.4. As before, call a component T *good* if it is contained within some target cluster C_i^* *and* $T \cup B(T)$ contains all of the inner ring points of C_i^*. Otherwise, the component is called *bad* provided it is not one of the initial centers present in $\mathcal{Q}_{init}$. We now show that each cheap target cluster will have a single, unique, good component.

Claim 3.8.2. *Let C_i^* be any cheap cluster in the target clustering. By stage $s = |C_i^*|$, the algorithm adds to $\mathcal{Q}$ a component T that contains a point from the inner ring of C_i^*.*

Claim 3.8.3. *Let T be a good connected component added to $\mathcal{Q}$, containing an inner ring point from cluster C_i^*. Then: (a) all points in T are of distance squared at most $\frac{\alpha \mathsf{OPT}}{64|C_i^*|}$ from c_i^*, (b) $T \cup B(T)$ is fully contained in C_i^*, and (c) the entire inner ring of C_i^* is contained in $T \cup B(T)$, and (d) no other component $T' \neq T$ in $\mathcal{Q}$ contains an inner ring point from C_i^*.*

As the proofs of Claims 3.8.2 and 3.8.3 are identical to the Claims 3.5.3 and 3.5.4, we omit them.

Lemma 3.8.4. *We do not add to $\mathcal{Q}$ more than $4000/\alpha$ bad components.*

Proof. Consider any bad component T that we add to $\mathcal{Q}$ and denote that stage in which we insert T to $\mathcal{Q}$ as s. So the size of this component is $> \frac{s}{2}$. Let y be an arbitrary point from T which belongs to cluster C^* in the optimal clustering. Let c^* be the center of C^*. We show that $d^2(c^*, y) > \frac{\alpha \mathsf{OPT}}{2000s}$.

We divide into cases.

Case 1: C^* is a cheap cluster and $s \geq |C^*|$. Recall that T must contain $s/2 \geq |C^*|/2$ points, so it follows that T contains some point x that does not belong to C^*. β-stability gives that this point has distance $d^2(c^*, x) > \alpha \frac{\mathsf{OPT}}{4|C^*|}$, and we apply Lemma 3.8.1 to deduce that all points in T are of distance squared of at least $\frac{\alpha}{16} \frac{\mathsf{OPT}}{|C^*|}$.

Case 2: C^* is a cheap cluster and $s < |C^*|$. In this case we have that the entire inner ring of C^* already belongs to some $T' \in Q$. Let $x \in T'$ be any inner ring point from C^*, and we have that $d(c^*, x)^2 \leq \frac{\alpha \mathsf{OPT}}{1024|C^*|} \leq \frac{\alpha \mathsf{OPT}}{1024s}$, while $d^2(x, y) > \frac{\alpha \mathsf{OPT}}{64s}$. It follows that $d^2(c^*, y) \geq (3d(x, y)/4)^2 > \frac{\alpha \mathsf{OPT}}{2000s}$.

Case 3: C^* is an expensive cluster and $s > 2|C^*|$. We claim that $d^2(c^*, y) > \frac{\alpha \mathsf{OPT}}{128|C^*|}$. If, by contradiction, we have that $d^2(c^*, y) \leq \frac{\alpha \mathsf{OPT}}{128|C^*|}$, then we show that the ball $B(y, r)$ contains only points from C_i^*, yet it must contains $s/2 > |C_i^*|$ points. This is because each

$p \in B(y,r)$ satisfies that $d^2(c^*,p) \leq (d(c^*,y)+d(y,p))^2 \leq \left(\sqrt{\frac{\alpha\mathsf{OPT}}{128|C^*|}}+\sqrt{\frac{\alpha\mathsf{OPT}}{64s}}\right)^2 < \frac{\alpha\mathsf{OPT}}{4|C^*|}$.

Case 4: C^* is an expensive cluster and $s \leq 2|C^*|$. In this case, from Fact 2.3.4 we know that $\mathcal{Q}_{init}$ contains a a good empirical center c for the expensive cluster C^*, in the sense that $\|c-c^*\|^2 \leq \frac{\alpha\mathsf{OPT}}{2048|C^*|} \leq \frac{\alpha\mathsf{OPT}}{1024s}$. Then, similarly case 2 above we have $d^2(y,c^*) \geq (d(y,c)-d(c,c^*))^2 > \frac{\alpha\mathsf{OPT}}{2000s}$. It follows that every point in T has a large distance from its center. Therefore, the $s/2$ points in this component contribute at least $\alpha\mathsf{OPT}/4000$ to the k-means cost. Hence, we can have no more than $4000/\beta$ such bad components. □

We now prove the main theorem.

Theorem 3.8.5. *The algorithm outputs a k-clustering whose cost is at most $(1+\epsilon)\mathsf{OPT}$.*

Proof. Using Claim 3.8.3, it follows that there exists some choice of k components which has good components for all the cheap clusters and good substitutes for the centers of the expensive clusters. Fix that choice and consider a cluster C_i^* with center c_i^*. If C_i^* is an expensive cluster then from Section 3.6 we know that $\mathcal{Q}_{init}$ contains a point c_i such that $d^2(c_i,c_i^*) \leq \frac{\alpha\epsilon}{\alpha+4\epsilon}\frac{\mathsf{OPT}_i}{|C_i^*|}$. Hence, the cost paid by the points in C_i^* will be atmost $(1+\epsilon)\mathsf{OPT}_i$. If C_i^* is a cheap cluster then denote by T the good component that resides within C_i^*. Denote $T \cup B(T)$ by A, and $C_i^* \setminus A$ by B. Let c_i be the center of A. We know that the entire inner-ring of C_i^* is contained in A, therefore, B cannot contain more than $\epsilon/16$ fraction of the points of C_i^*. Fact **??** dictates that in this case, $\|c_i^*-c_i\|^2 \leq \epsilon^2\frac{\alpha\mathsf{OPT}}{4^7|C_i^*|}$. We know every $x \in B$ contributes at least $\frac{\alpha\mathsf{OPT}}{1024|C_i^*|}$ to the cost of C_i^*, so $\|c_i^*-c_i\|^2 \leq \frac{\epsilon}{16}\|x-c_i^*\|^2$. Thus, for every $x \in B$, we have that $\|x-c_i\|^2 \leq (1+\epsilon)\|x-c_i^*\|^2$. It follows that $\sum_{x\in B}\|x-c_i\|^2 \leq (1+\epsilon)\sum_{x\in B}\|x-c_i^*\|^2$, and obviously $\sum_{x\in A}\|x-c_i\|^2 \leq \sum_{x\in A}\|x-c_i^*\|^2$ as c_i is the center of mass of A. Therefore, when choosing the good k components out of $\mathcal{Q}$, we can assign them to the centers in such a way that costs no more than $(1+\epsilon)\mathsf{OPT}$. Obviously the assignment of each point to the nearest of the k-centers only yields a less costly clustering, and thus its cost is also at most $(1+\epsilon)\mathsf{OPT}$. □

3.8.2 A Randomized Algorithm for weakly deletion-stable k-Means Instances

We now present a randomized algorithm which achieves a $(1+\epsilon)$ approximation to the k-means optimum of a weakly deletion-stable instance and runs in time $(k\log_k n)^{\mathrm{poly}(1/\epsilon,1/\alpha)}O(n^3)$.

1. Guess $l \leq \frac{4^7}{\alpha\epsilon}$, the number of expensive clusters. Set $t = \frac{1}{2}(\log_k n)$. Guess non-negative integers $g_1, g_2, \ldots g_t$, such that $\sum_i g_i = l$.
2. Sample $g_1 + g_2 + g_3$ sets, by sampling independently and u.a.r $O(\frac{1}{\alpha} + \frac{1}{\epsilon})$ points for each set. For each such set $\tilde{T}_j$, add the singleton $\{\mu(\tilde{T}_j)\}$ to $\mathcal{Q}$.
3. Modify the Population Stage from the previous algorithm, so that whenever $s = \frac{n}{k^{2i}}$ for some $i \geq 1$ (We call this the *interval* i)
   - Sample g_{i+3} sets, by sampling independently and u.a.r $O(\frac{1}{\alpha} + \frac{1}{\epsilon})$ points for each set. For each such set $\tilde{T}_j$, add the singleton $\{\mu(\tilde{T}_j)\}$ to $\mathcal{Q}$.

The algorithm is similar in nature to the one presented in the previous section, except that for expensive clusters we replace brute force guessing of samples with random sampling. Note that the straightforward approach of sampling the points right at the start of the algorithm might fail, if there exist expensive clusters which contain very few points. A better approach is to interleave the sampling step with the rest of the algorithm. In this way we sample points from an expensive cluster only when it contains a reasonable fraction of the total points remaining, hence our probability of success is noticeable (namely, $\mathrm{poly}(1/k)$).

The high-level approach of the algorithm is to partition the main loop of the Population Stage, in which we try all possible values of s (starting from n and ending at 1), into *intervals*. In interval i we run s on all values starting with $\frac{n}{k^{2i}}$ and ending with $\frac{n}{k^{2i+2}}$. So overall, we have no more than $t = \frac{1}{2}\log_k(n)$ intervals. Our algorithm begins by guessing l, the number of expensive clusters, then guessing $g_1, g_2, \ldots, g_t$ s.t. $\sum_i g_i = l$. Each g_i is a guess for the number of expensive clusters whose size lies in the range $\left[\frac{n}{k^{2i}}, \frac{n}{k^{2(i-1)}}\right)$. Note that $\sum_i g_i =$ # expensive clusters $\leq \frac{4^7}{\alpha\epsilon}$. Hence, there are at most $(\log_k n)^{\frac{4^7}{\alpha\epsilon}}$ number of possible assignments to g_i's and we run the algorithm for every such possible guess.

Fixing $g_1, g_2, \ldots, g_t$, we run the Population Stage of the previous algorithm. However, whenever s reaches a new interval, we apply random sampling to obtain good empirical centers for the expensive clusters whose size lies *three intervals "ahead"*. That is, in the beginning of interval i, the algorithm tries to collect centers for the clusters whose size $\geq \frac{n}{k^{6+2i}} = \frac{s}{k^6}$, yet $\leq \frac{n}{k^{4+2i}} = \frac{s}{k^4}$. We assume for this algorithm that k is significantly greater than $\frac{4}{\alpha}$. Obviously, if k is a constant, then we can use the existing algorithm of Kumar et al. [2004].

In order to prove the correctness of the new algorithm, we need to show that the sam-

pling step in the initialization stage succeeds with noticeable probability. Let l_i be the actual number of expensive clusters whose size belongs to the range $\left[\frac{n}{k^{2i}}, \frac{n}{k^{2(i-1)}}\right)$. In the proof which follows, we assume that the correct guess for l_i's has been made, i.e. $g_i = l_i$, for every i. We say that the algorithm *succeeds at the end of interval* i if the following conditions hold:

1. In the beginning of the interval, our guess for all clusters that belong to interval $(i+3)$ produces good empirical centers. That is, for every expensive cluster C^* of size in the range $\left[\frac{n}{k^{6+2i}}, \frac{n}{k^{4+2i}}\right)$, the algorithm picks a sample $\tilde{T}$ such that the mean $\mu(\tilde{T})$ satisfies:
   (a) $d^2(\mu(\tilde{T}), c^*) \leq \frac{\alpha \mathsf{OPT}}{1024|C^*|}$.
   (b) $\sum_{x \in C^*} d^2(x, \mu(\tilde{T})) \leq (1+\epsilon) \sum_{x \in C^*} d^2(x, c^*)$.
2. During the interval, we do not delete any point p that belongs to some target cluster C^* of size $\leq \frac{n}{k^{4+2(i+1)}}$ points.
3. At the end of the interval, the total number of remaining points (points that were not added to some $T \in \mathcal{Q}$ or deleted from the instance because they are too close to some $T' \in \mathcal{Q}$) is at most $\frac{n}{k^{2i-1}}$.

Lemma 3.8.6. *For every $i \geq 1$, let S_i denote the event that the algorithm succeeds at the end of interval i. Then* $\Pr[S_i | S_1, S_2, \ldots, S_{i-1}] \geq k^{-l_{(i+3)} \cdot O(\frac{1}{\alpha} + \frac{1}{\epsilon})}$

Before going into the proof we show that Lemma 3.8.6 implies that with noticeable probability, our algorithm returns a $(1+\epsilon)$-approximation of the k-means optimal clustering. First, observe the technical fact that for the first three intervals l_1, l_2, l_3, we need to guess the centers of clusters of size $\geq \frac{n}{k^6}$ before we start our Population Stage. However, as these clusters contain k^{-6} fraction of the points, then using Fact 2.3.4, our sampling finds good empirical centers for all of these $l_1 + l_2 + l_3$ expensive clusters w.p. $\geq k^{-(l_1+l_2+l_3)O(\frac{1}{\beta}+\frac{1}{\epsilon})}$. Applying Lemma 3.8.6 we get that the probability our algorithm succeeds after all intervals is $\geq 1/k^{O(\frac{\alpha+\epsilon}{\alpha^2\epsilon^2})}$. Now, a similar analysis as in the previous section gives us that for the correct guess of the good components in $\mathcal{Q}$, we find a clustering of cost at most $(1+\epsilon)\mathsf{OPT}$.

Proof of Lemma 3.8.6. Recall that α is a constant, whereas k is not. Specifically, we assume throughout the proof that $k^2 > \frac{200}{\alpha}$, and so we allow ourselves to use asymptotic notation.

We first prove that condition 2 holds during interval i. Assume for the sake of contradiction that for some cluster C^* whose size is less than $\frac{n}{k^{6+2i}}$, there exists some point $y \in C^*$, which was added to some component T during interval i, at some stage $s \in \left[\frac{n}{k^{2i+2}}, \frac{n}{k^{2i}}\right)$. This means that by setting the radius $r = \frac{\alpha\mathsf{OPT}}{256s}$, the ball $B(y,r)$ contains $> s/2 \geq \frac{n}{2k^{2i+2}}$ points. Since C^* contains at most $\frac{n}{k^{6+2i}}$ many point, we have $|C^*| \ll s/2$, so at least $s/4$ points in $B(y,r)$ belong to other clusters. Our goal is to show that these $s/4$ points contribute more than OPT to the target clustering, thereby achieving a contradiction.

Let x be such point, and denote the cluster that x is assigned to in the target clustering by $C_j^* \neq C^*$. Since the instance is weakly deletion-stable we have that $d^2(c^*,x) > \frac{\alpha\mathsf{OPT}}{4|C^*|} \geq \alpha\mathsf{OPT}\frac{4k^{6+2i}}{n}$. On the other hand, $d^2(x,y) \leq r = \alpha\mathsf{OPT}\frac{1}{256s} \leq \alpha\mathsf{OPT}\frac{k^{2+2i}}{256n}$. Therefore, $d^2(c^*,x) = \Omega(k^4)\cdot r$, so $d^2(y,c^*) = \left(d(c^*,x) - d(x,y)\right)^2 = \Omega(k^4)\cdot r$. Recall that in the target clustering each point is assigned to its nearest center, so $d^2(c_j^*,y) \geq d^2(c^*,y) = \Omega(k^4)\cdot r$. So we have that $d^2(c_j^*,x) \geq \left(d(c_j^*,y) - d(x,y)\right)^2 = \Omega(k^4)\cdot r = \Omega(k^4)\cdot\frac{\alpha\mathsf{OPT}}{256}\frac{k^{2i}}{n}$.

So, at least $s/4 = \Omega(\frac{n}{k^{2i+2}})$ points contribute $\Omega(k^4)\frac{\alpha\mathsf{OPT}}{256}\frac{k^{2i}}{n}$ to the cost of the optimal clustering. Their total contribution is therefore $\Omega(k^2)\cdot\frac{\alpha}{256}\mathsf{OPT} > \mathsf{OPT}$. Contradiction.

A similar proof gives that no point $y \in C^*$ is deleted from the instance because for some $x \in T$, where T is some component in $\mathcal{Q}$, we have that $d^2(y,x) < 4r$. Again, assume for the sake of contradiction that such y,x and T exist. Denote by $s \in \left[\frac{n}{k^{2i+2}}, \frac{n}{k^{2i}}\right)$ the stage in which we remove y, and denote by $s' \geq s$ the stage in which we insert T into $\mathcal{Q}$. By setting the radius $r' = \frac{\alpha\mathsf{OPT}}{256s'} \leq r$, we have that the ball $B(x,r')$ contains at least $s'/2 \geq s/2$ points, and therefore, the ball $B(y,5r)$ contains at least $s/2$ points. We now continue as in the previous case.

We now prove condition 1. We assume the algorithm succeeded in all previous intervals. Therefore, at the beginning of interval i, all points that belong to clusters of size $\leq \frac{n}{k^{2i+4}}$ remain in the instance, and in particular, the clusters we wish to sample from at interval i remain intact. Furthermore, by the assumption that the algorithm succeeded up to interval $(i-1)$, we have that each expensive cluster that should be sampled at the beginning of interval i, contains a $1/k^7$ fraction of the remaining points. We deduce that the probability that we pick a random sample of $O(\frac{1}{\alpha}+\frac{1}{\epsilon})$ points from such expensive cluster is at least $k^{-O(\frac{1}{\alpha}+\frac{1}{\epsilon})}$. Using Fact 2.3.4 we have that with probability $\geq k^{-O(\frac{1}{\alpha}+\frac{1}{\epsilon})}$ this sample yields a good empirical center.

We now prove condition 3, under the assumption that 1 is satisfied. We need to bound the number of points left in the instance at the end of interval i. There are two types of remaining points: points that in the target clustering belong to clusters of size $> \frac{n}{2k^{2i}}$, and

points that belong to clusters of size $\leq \frac{n}{2k^{2i}}$. To bound the number of points of the second type is simple – we have k clusters, so the overall number of points of the second type is at most $\frac{n}{2k^{2i-1}}$. We now bound the number of remaining points of the first type.

At the end of the interval $s = \frac{n}{k^{2i+2}}$, so we remove from the instance any point p whose distance (squared) from some point in $\mathcal{Q}$ is at most $4r = \frac{\mathsf{OPT}}{16}\frac{k^{2i+2}}{n}$. We already know that by the end of interval i, either by successfully sampling an empirical center or by adding an inner-ring point to a component in $\mathcal{Q}$, for every cluster C^* of size $> \frac{n}{2k^{2i}}$, exists some $T \in \mathcal{Q}$ with a point $c' \in T$, s.t. $d^2(c^*, c') \leq \frac{\alpha\mathsf{OPT}}{1024|C^*|} \leq \frac{\alpha\mathsf{OPT}}{512}\frac{k^{2i}}{n}$. Thus, if $x \in C^*$ is a point that wasn't removed from the instance by the end of interval i, it must hold that $d^2(c^*, x) \geq (d(c', x) - d(c^*, c'))^2 = \Omega(k^{2i+2})\frac{\mathsf{OPT}}{n}$. Clearly, at most $n \cdot O(k^{-2i-2})$ points can contribute that much to the cost of the optimal k-means clustering, and so the number of points of the first type is at most $\frac{n}{2k^{2i-1}}$ as well. □

As we need to traverse all guesses g_is, the runtime of this algorithm takes $O(n^3(\log_k n)^{O(\frac{1}{\alpha\epsilon})})$. Repeating this algorithm $k^{O(l(\frac{1}{\alpha}+\frac{1}{\epsilon}))}$ many times, we increase the probability of success to be $\geq 1/2$, and incur runtime of $O(n^3(\log_k n)^{O(\frac{1}{\alpha\epsilon})}k^{O(\frac{\alpha+\epsilon}{\alpha^2\epsilon^2})})$.

3.8.3 NP-hardness under weak-deletion satbility

Proof of Theorem 3.3.6. Fix any constant $\alpha > 0$. We give a poly-time reduction from Set-Cover to $(1+\alpha)$-weakly deletion-stable k-median instances. Under standard notation, we assume our input consists of n subsets of a given universe of size m, for which we seek a k-cover. We reduce such an instance to a k-median instance over $m+k(n+4\alpha km)$ points. We start with the usual reduction of Set-Cover to an instance with m points representing the items of the universe and n points representing all possible sets. Fix integer $D \gg 1$ to be chosen later. If j belongs to the ith set, fix the distance $d(i,j) = D$, otherwise we fix the distance $d(i,j) = D+1$, and between any two set-points we fix the distance to be 1. (The distance between any two item points is shortest-path distance.) However, we augment the n set-points with additional $2mD$ points, setting the distance between all of the $(n+2mD)$ points as 1. Furthermore, we replicate k copies of these $(n+2mD)$ augmented set-points, all connected only via the m-item points.

Observe that each of the k copies of our augmented set-points components contains many points, and all points outside this copy are of distance $\geq D$ from it. Therefore, in the optimal k-median solution, each center resides in one unique copy of the augmented set-points. Now, if our Set-Cover instance has a k-cover, then we can pick the respective centers and have an optimal solution with cost exactly $k(n+2mD-1)+mD$. Otherwise,

no k sets cover all m items, so for any k centers, some item-point must have distance $D+1$ from its center, and so the cost of any k-partition is $\geq k(n+2mD-1)+mD+1$. Furthermore, the resulting instance is $(1+\alpha)$ weakly deletion-stable, in fact, even $(1+\alpha)$ ORSS-separable. In particular, using one center from each augmented set-point results in a k-median solution of cost $\leq m(D+1)+k(n+2mD-1) < (k+1)(n+2mD)$; hence, OPT is at most this quantity. However, in any $k-1$ clustering, one of the copies of the augmented set-points must not contain a center and therefore $\mathsf{OPT}_{(k-1)} \geq \mathsf{OPT} + (n+2mD)(D-1)$. Choosing $D = \alpha(k+1)+1$ ensures that this cost is at least $(1+\alpha)\mathsf{OPT}$. $\square$

Chapter 4

Supervised Clustering

Clustering is typically studied in an unsupervised learning scenario where the goal is to partition the data given pairwise similarity information. Designing provably good clustering algorithms is challenging since given a similarity function, there could be multiple plausible clusterings of the data. Traditional approaches get around this ambiguity by making assumptions on the data generation process. For example, there is a large body of work which focuses on clustering data which is generated by a mixture of Gaussians (Achlioptas and McSherry [2005], Kannan et al. [2005], Dasgupta [1999], Arora and Kannan [2001], Brubaker and Vempala [2008], Kalai et al. [2010], Moitra and Valiant [2010], Belkin and Sinha [2010]). Although this helps define the "right" clustering one should be looking for, real world data rarely satisfies such strong assumptions. An alternate approach is to use limited user supervision to help the algorithm reach the desired answer. This has been facilitated by the availability of cheap crowd-sourcing tools in recent years. It has become clear that in certain applications such as search or document classification, where users are willing to help a clustering algorithm arrive at their own desired answer with a small amount of additional prodding, interactive algorithms are very useful.

In this thesis we study a model (Balcan and Blum [2008]) proposed for clustering which incorporates limited amount of user interaction to deal with the inherent ambiguity of the task at hand. The model is similar to the Equivalence Query(EQ) model of learning (Angluin [1998]) but with a different kind of feedback. We assume that the given set S of n points belongs to a target clustering $\{C_1^*, C_2^*, \ldots, C_k^*\}$, where each cluster is defined by a Boolean function f belonging to a class of functions H. For example, the points belonging to the cluster C_1^* might be the set $\{x \in S | f_1(x) = 1\}$. We also assume that each point belongs to exactly one of the k clusters. As in the EQ model of learning, the algorithm presents a hypothesis clustering $\{C_1, C_2, \ldots, C_{k'}\}$ to the teacher. If the clustering

is incorrect the algorithm gets feedback from the teacher. However, the feedback in this case is different from the one in the EQ model. In the learning model, the algorithm gets a specific point x as a counter-example to its proposed hypothesis. For clustering problems this may not a very natural form of feedback. In a realistic scenario, the teacher can look at the clustering proposed and give limited feedback, for example whether he thinks a given cluster is "pure" or not. Such limited interaction was modeled in Balcan and Blum [2008] using *split* and *merge* requests and is the starting point for our work.

4.1 The Model

The clustering algorithm is given a set $\mathcal{S}$ of n points. Each point belongs to one of the k target clusters. Each cluster is defined by a function $f \in H$, where H is a class of Boolean functions. The goal of the algorithm is to figure out the target clustering by interacting with the teacher as follows:

1. The algorithm proposes a hypothesis clustering $\{C_1, C_2, \ldots, C_{k'}\}$ to the teacher.

2. The teacher can request $split(C_i)$ if C_i contains points from two or more target clusters. The teacher can request $merge(C_i, C_j)$ if $C_i \cup C_j$ is a subset of one of the target clusters.

The assumption is that there is no noise in the teacher response. The goal is to use as few queries to the teacher as possible. Ideally, we would like the number of queries to be $poly(k, \log m, \log |C|)$. Notice that if we allow the algorithm to use the number of queries linear in n, then there is a trivial algorithm, which starts with all the points in separate clusters and then merges clusters as requested by the teacher.

4.2 Our Results

In their paper, Balcan and Blum [2008] gave efficient clustering algorithms for the class of intervals and the class of disjunctions over $\{0,1\}^d$. They also gave a generic algorithm which clusters any class of functions using $O(k^3 \log |H|)$ queries. The algorithm however is computationally inefficient. In this thesis we extend these results in several directions as discussed below.

4.2.1 A generic algorithm

We reduce the query complexity of the generic algorithm for clustering any class of functions (Balcan and Blum [2008]), from $O(k^3 \log |H|)$ to $O(k \log |H|)$. In addition our algorithm is simpler than the original one. We would like to point out that as in Balcan and Blum [2008] the generic algorithm is also computationally inefficient.

Theorem 4.2.1. *There is a generic algorithm that can cluster any finite concept class using at most $k \log |D|$ queries.*

4.2.2 Clustering geometric concepts

We extend the result of Balcan and Blum [2008] on clustering the class of intervals on a line to more general geometric function classes.

Theorem 4.2.2. *There is an algorithm which can cluster the class of k rectangles in d dimensions using at most $O((kd \log m)^d)$ queries.*

Corollary 4.2.3. *There is an algorithm which can cluster the class of k hyperplanes in d dimensions having a known set of slopes of size at most s, using at most $O((kds \log m)^d)$ queries.*

The results in this chapter are based on work in Awasthi and Zadeh [2010].

4.2.3 A generic algorithm for learning any finite concept class

We reduce the query complexity of the generic algorithm for learning any concept class (Balcan and Blum [2008]), from $O(k^3 \log |H|)$ to $O(k \log |H|)$. In addition our algorithm is simpler than the original one. The new algorithm is described below.

Given n points let $VS = \{$ the set of all possible k clusterings of the given points using concepts in $H\}$. This is also known as the version space. Notice that $|VS| \leq |H|^k$. Given a set $h \subseteq \mathcal{S}$ of points we say that a given clustering R is consistent with h if h appears as a subset of one of the clusters in R. Define, $VS(h) = \{R \in VS | R$ is consistent with $h.\}$. At each step the algorithm outputs clusters as follows:

1. Initialize $i = 1$.
2. Find the largest set of points h_i, s.t. $|VS(h_i)| \geq \frac{1}{2}|VS|$.

3. Output h_i as a cluster.

4. Set $i = i + 1$ and repeat steps 1-3 on the remaining points until every point has been assigned to some cluster.

5. Present the clustering $\{h_1, h_2, \ldots, h_J\}$ to the teacher.

If the teacher says $split(h_i)$, remove all the clusterings in VS which are consistent with h_i If the teacher says $merge(h_i, h_j)$, remove all the clusterings in VS which are inconsistent with $h_i \cup h_j$.

Theorem 4.2.4. *The generic algorithm can cluster any finite concept class using at most* $k \log |H|$ *queries.*

Proof. At each request, if the teacher says $split(h_i)$, then all the clusterings consistent with h_i are removed, which by the construction followed by the algorithm will be at least half of $|VS|$. If the teacher says $merge(h_i, h_j), i < j$, then all the clusterings inconsistent with $h_i \cup h_j$ are removed. This set will be at least half of $|VS|$, since otherwise the number of clusterings consistent with $h_i \cup h_j$ will be more than half of $|VS|$ which contradicts the maximality of h_i. Hence, after each query at least half of the version space is removed. From the above claim we notice that the total number of queries will be at most $\log |VS| \leq log|C|^k \leq k \log |H|$. □

The analysis can be improved if the VC-dimension d of the concept class H is much smaller than $\log |H|$. In this case the size of VS can be bounded from above by $C[m]^k$, where $C[m]$ is the number of ways to split m points using concepts in H. Also from Sauer's lemma (Vapnik [1998]) we know that $C[m] \leq m^d$. Hence, we get $|VS| \leq m^{kd}$. This gives a query complexity of $O(kd \log m)$.

4.3 Clustering geometric concepts

We now present an algorithm for clustering the class of rectangles in 2 dimensions. We first present a simple but less efficient algorithm for the problem. The algorithm uses $O((k \log m)^3)$ queries and runs in time $poly(k, m)$. In the appendix, we show that the query complexity of the algorithm can be improved to $O((k \log m)^2)$. Our algorithm generalizes in a natural way to rectangles in d dimensional space, and to hyperplanes in d dimensions with known slopes.

4.3.1 An algorithm for clustering rectangles

Each rectangle c in the target clustering can be described by four points $(a_i, a_j), (b_i, b_j)$ such that $(x, y) \in c_k$ iff $a_i < x < a_j$ and $b_i < y < b_j$. Hence, corresponding to any k-clustering there are at most $2k$ points $a_1, a_2, \ldots, a_{2k}$ on the x-axis and at most $2k$ points $b_1, b_2, \ldots, b_{2k}$ on the y-axis. We call these points the *target points*. The algorithm works by finding these points. During its course the algorithm maintains a set of points on the x-axis and a set of points on the y-axis. These points divide the entire space into rectangular regions. The algorithm uses these regions as its hypothesis clusters. The algorithm is sketched below:

1. Start with points $({a_{start}}', {a_{end}}')$ on the x-axis and points $({b_{start}}', {b_{end}}')$, such that all the points are contained in the rectangle defined by these points.

2. At each step, cluster the m points according to the region in which they belong. Present this clustering to the teacher.

3. On a merge request, replace the two clusters by the minimum enclosing rectangle containing all the points from the two clusters.

4. On a split of $({a_i}', {a_j}'), ({b_i}', {b_j}')$, create a new point ${a_r}'$ such that ${a_i}' < {a_r}' < {a_j}'$, and the projection of all the points onto $({a_i}', {a_j}')$ is divided into half by ${a_r}'$. Similarly, create a new point ${b_r}'$ such that ${b_i}' < {b_r}' < {b_j}'$, and the projection of all the points onto $({b_i}', {b_j}')$ is divided into half by ${b_r}'$. Abandon all the merges done so far.

Theorem 4.3.1. *The algorithm can cluster the class of rectangles in 2 dimensions using at most $O((k \log m)^3)$ queries.*

Proof. Lets first bound the total number of split requests. If the teacher says split on $(x_i, x_j), (y_i, y_j)$, then we know that either (x_i, x_j) contains a target point a or (y_i, y_j) contains a target point b or both. By creating two splits we are ensuring that the size of at least one of the regions containing a target point is reduced by half. There are at most $2k$ intervals on the x-axis and at most $2k$ intervals on the y-axis. Hence, the total number of split requests is $\leq 4k \log m$. Now lets bound the merge requests. Between any two split requests the total number of merge requests will be at most the total number of regions which is $\leq O((k \log m)^2)$. Since, t points on the x and the y axis can create at most t^2 regions, we get that the total number of merge requests is at most $\leq O(k \log m)^3$. Alos, notice that we will never get a split request made to the result of doing a merge. Hence, the total number of queries made by the algorithm is $O((k \log m)^3)$. □

If we are a bit more careful, we can avoid redoing the merges after every split and reduce the query complexity to $O((k \log m)^2)$. So, for rectangles we have the following result[1].

Theorem 4.3.2. *There is an algorithm which can cluster the class of rectangles in* 2 *dimensions using at most* $O((k \log m)^2)$ *queries.*

We can also generalize this algorithm to work for rectangles in a d-dimensional space. Hence, we get the following result

Corollary 4.3.3. *There is an algorithm which can cluster the class of rectangles in* d *dimensions using at most* $O((kd \log m)^d)$ *queries.*

Corollary 4.3.4. *There is an algorithm which can cluster the class of hyperplanes in* d *dimensions having a known set of slopes of size at most* s*, using at most* $O((kds \log m)^d)$ *queries.*

4.3.2 Dynamic model

Next we study a natural generalization of the original model. In the original model we assume that the teacher has access to the entire set of points. In practice, this will rarely be the case. For example, in the case of clustering news articles, each day the teacher sees a small fresh set of articles and provides feedback. Based on this the algorithm must be able to figure out the target clustering for the entire space of articles. More formally, let X be the space of all the points. There is a target k clustering for these points where each cluster corresponds to a function in a class H. At each step, the world picks n points and the algorithm clusters these n points and presents the clustering to the teacher. If the teacher is unhappy with the clustering he may provide feedback. Note that the teacher need not provide feedback every time the algorithm proposes an incorrect clustering. The goal of the algorithm is to minimize the amount of feedback necessary to figure out the target clustering. Notice that at each step the algorithm may get a fresh set of n points. We assume that the requests have no noise and the algorithm has access to all the points in X. We can show that one can efficiently cluster the class of intervals in this model.

Theorem 4.3.5. *There is an efficient algorithm that can cluster the class of* k *intervals on a line using at most* $O(k \log n)$ *queries.*

[1] See Section 4.5

An algorithm for clustering intervals

We assume that the space X is discretized into n points. Let us assume that there exist points $\{a_1, a_2, \ldots, a_{k+1}\}$, on the x-axis such that the target clustering is the intervals $\{[a_1, a_2], [a_2, a_3], \ldots, [a_k, a_{k+1}]\}$. The algorithm maintains a set of points on the x-axis and uses the intervals induced by them as its hypothesis. Also each interval is associated with a state of $marked/unmarked$. When a new interval is created, it is always $unmarked$. An interval is marked if we know that none of the points(a_i's) in the target clustering can be present in that interval. The algorithm is sketched below:

1. Start with one unmarked interval containing all the points in the space.
2. Given a set of m points, first form preliminary clusters $h_1, \ldots, h_J$ such that each cluster corresponds to an interval. Next output the final clusters as follows:
   - set i=1
   - If h_i and h_{i+1} correspond to adjacent intervals at least one of them is unmarked, then output $h_i \cup h_{i+1}$ and set $i = i + 2$. Else output h_i and set $i = i + 1$.
3. On a split request, split every unmarked interval in the cluster in half.
4. On a merge request, mark every unmarked contained in the cluster.

Theorem 4.3.6. *The algorithm can cluster the class of intervals using at most $O(k \log n)$ mistakes.*

Proof. Notice that by our construction, every cluster will contain at most 2 unmarked intervals. Lets first bound the total number of split requests. For every point a_i in the target clustering we define two variables $left\_size(a_i)$ and $right\_size(a_i)$. If a_i is inside a hypothesis interval $[x, y]$ then $left\_size(a_i) =$ number of points in $[x, a_i]$ and $right\_size(a_i) =$ number of points in $[a_i, y]$. If a_i is also a boundary point in the hypothesis clustering $([x, a_i], [a_i, y])$ then again $left\_size(a_i) =$ number of points in $[x, a_i]$ and $right\_size(a_i) =$ number of points in $[a_i, y]$. Notice, that every split request reduces either the $left\_size$ or the $right\_size$ of some boundary point by half. Since there are at most k boundary points in the target clustering, the total number of split requests is $\leq O(k \log n)$ times. Also note that the number of unmarked intervals is at most $O(k \log n)$ since, unmarked intervals increase only via split requests. On every merge request either an unmarked interval is marked or two marked intervals are merged. Hence, the total number of merge requests is atmost twice the number of unmarked intervals $\leq O(k \log n)$. Hence, the total number of mistakes is $\leq O(k \log n)$. □

Its easy to notice that the generic algorithm for learning any finite concept class in the original model also works in this model. Hence, we can learn any finite concept class in this model using at most $k \log |C|$ queries.

4.3.3 η noise model

The previous two models assume that there is no noise in the teacher requests. This is again an unrealistic assumption since we cannot expect the teacher responses to be perfect. For example, if the algorithm proposes a clustering in which there are two clusters which are almost pure, i.e., a large fraction of the points in both the clusters belong to the same target clusters, then there is a good chance that the teacher will ask the algorithm to merge these two clusters, especially if the teacher has access to the clusters through a random subset of the points. Here we study a model which removes this assumption. For simplicity, we consider the noisy version of the original model of Balcan and Blum [2008]. As in the original model, the algorithm has n points. At each step, the algorithm proposes a clustering $\{C_1, C_2, \ldots, C_{k'}\}$ to the teacher and the teacher provides feedback. But now, the feedback is noisy in the following sense

1. **Split:** As before the teacher can say $split(C_i)$, if C_i contains points from more than one target clusters.

2. **Merge:** The teacher can say $merge(C_i, C_j)$, if C_i and C_j each have at least one point from the same target cluster.

It turns out that such arbitrary levels of noise might be difficult for any query efficient clustering algorithm.

Theorem 4.3.7. *Consider n points on a line and $k = 2$. Any clustering algorithm must use $\Omega(n)$ queries in the worst case to figure out the target clustering in the noisy model.*

Proof. Given m points an adversary can force any algorithm to make at least m queries. The adversary strategy is as follows:

1. If the algorithm proposes a single cluster ask to split.

2. If the algorithm proposes two or more clusters, choose any two clusters and ask the algorithm to merge them.

There are m choices for the adversary to choose the split point for the target intervals. After every merge request at most one position is ruled out. This position corresponds to any point in between (a_i, a_j) where $a_i \in h_i$ and $a_j \in h_j$ and a_i and a_j are the closest pair of points across the two clusters. □

Hence, we consider a relaxed notion of noise. If the teacher says $merge(C_i, C_j)$ then we assume that at least a constant η fraction of the points in both the clusters, belong to a single target cluster. Under this model of noise we give an algorithm for clustering the class of k-intervals on a line.

Theorem 4.3.8. *There is an efficient algorithm that clusters the class of k intervals on a line using at most $O(k(\log_{\frac{1}{1-\eta}} m)^2)$ split and merge requests.*

An algorithm for clustering intervals

The algorithm is a generalization of the interval learning algorithm in the original model. The main idea is that when the teacher asks to merge two intervals (a_i, a_j) and (a_j, a_k), then we know than at least η fraction of the portion to the left and the right of a_j is pure. Hence, the algorithm can still make progress. As the algorithm proceeds it is going to mark certain intervals as "pure" which means that all the points in that interval belong to the same cluster. More formally the algorithm is as follows

1. Start with one interval $[a_{start}', a_{end}']$ containing all the points.
2. At each step, cluster the points using the current set of intervals and present that clustering to the teacher.
3. On split request : Divide the interval in half.
4. On a merge request
   - If both the intervals are marked "pure", merge them.
   - If both the intervals are unmarked, then create 3 intervals where the middle interval contains η fraction of the two intervals. Also make the middle interval as "pure".
   - If one interval is marked and one is unmarked, then shift the boundary between the two intervals towards the unmarked interval by a fraction of η.

Theorem 4.3.9. *The algorithm clusters the class of intervals using at most $O(k(\log_{\frac{1}{1-\eta}} m)^2)$ split and merge requests.*

Proof. We will call a merge request, as "impure" if it involves at least one impure interval,i.e., an interval which contains points from two or more clusters. Else we will call it as "pure". Notice that every split and impure merge request makes progress, i.e. the size of some target interval is reduced by at least η. Hence, the total number of split + impure merge requests $\leq k \log_{\frac{1}{1-\eta}} m$. We also know that the total number of unmarked intervals $\leq k \log_{\frac{1}{1-\eta}} m$, since only split requests increase the unmarked intervals. Also, total number of marked intervals $\leq$ total number of unmarked intervals, since every marked interval can be charged to a split request. Hence, the total number of intervals $\leq 2k \log_{\frac{1}{1-\eta}} m$.

To bound the total number of pure merges, notice that every time a pure merge is made, the size of some interval decreases by at least an η fraction. The size of an interval can decrease at most $\log_{\frac{1}{1-\eta}} m$ times. Hence, the total number of pure merges $\leq k(\log_{\frac{1}{1-\eta}} m)^2$.

Hence, the algorithm makes at most $O(k(\log_{\frac{1}{1-\eta}} m)^2)$ queries. □

4.4 Properties of the Data

We now adapt the query framework of Balcan and Blum [2008] to cluster datasets which satisfy certain natural separation conditions with respect to the target partitioning. For this section, sometimes we write $d = \langle e_1, e_2, \ldots, e_{\binom{n}{2}} \rangle$ to mean the set of distances that exist between all pairs of n points. This list is *always ordered* by increasing distance.

4.4.1 Threshold Separation

We introduce a (strong) property that may be satisfied by $d = \langle e_1, e_2, \ldots, e_{\binom{n}{2}} \rangle$ with respect to C^*, the target clustering. It is important to note that this property is imposing restrictions on d, defined by the data. An inner edge of C^* is a distance between two points inside a cluster, while an outer edge is a distance between two points in differing clusters.

STRICT THRESHOLD SEPARATION. There exists a threshold $t > 0$ such that all inner edges of C^* have distance less than or equal t, and all outer edges have distance greater than t.

In other words, the pairwise distances between the data are such that all inner edges of d (w.r.t. C^*) have distance smaller than all outer edges (again, w.r.t. C^*). This property gives away a lot of information about C^*, in that it allows Single-Linkage to fully recover C^* as we will see in theorem 4.4.1.

Kleinberg [2003], Jardine and Sibson [1971] introduce the following 3 properties which a clustering algorithm can satisfy. We will denote a clustering algorithm as a function $F(d, k)$ which takes as input a distance metric and the number of clusters k and outputs a k partition of the data. We will say that d' is a consistent transformation of d if it is obtained by changing d such that inner-cluster distances in d are decreased, and outer-cluster distances are increased.

1. CONSISTENCY. *Fix k. Let d be a distance function, and d' be a consistent transformation of d. Then $F(d, k) = F(d', k)$*

2. ORDER-CONSISTENCY. *For any two distance functions d and d', number of clusters k, if the order of edges in d is the same as the order of edges in d', then $F(d, k) = F(d', k)$*

3. k-RICHNESS. *For any number of clusters k, Range($F(\bullet, k)$) is equal to the set of all k-partitions of S*

We would like to emphasize that the above are desired properties of a clustering function and not the properties of a dataset. Before we present the algorithm to interact with the teacher, Theorem 4.4.1 will be useful (See Section 4.5 for the proof).

Theorem 4.4.1. *Fix k and a target k-partitioning C^*, and let d be a distance function satisfying Strict Threshold Separation w.r.t. C^*. Then for any Consistent, k-Rich, Order-Consistent partitioning function F, we have $F(d, k) = C^*$.*

Note that since Single-linkage is Consistent, k-Rich, and Order-Consistent (Zadeh and Ben-David [2009]), it immediately follows that SL$(d, k) = C^*$ - in other words, SL is guaranteed to find the target k-partitioning, but we still have to interact with the teacher to find out k. It is a recently resolved problem that Single-Linkage is not the only function satisfying the above properties (Zadeh and Ben-David [2011]), so the the class of Consistent, k-Rich, and Order-Consistent functions has many members. We now present the algorithm to interact with the teacher.

Theorem 4.4.2. *Given a dataset satisfying Strict Threshold Separation, there exists an algorithm which can find the target partitioning for any hypothesis class in $O(\log(n))$ queries*

Proof. Note that the threshold t and the number of clusters k are not known to the algorithm, else the target could be found immediately. By theorem 4.4.1, we know that the

target must be exactly what Single-Linkage returns for some k, and it remains to find the number of clusters. This can be done using a binary search on the number of clusters which can vary from 1 to n. We start with some candidate k. If our current guess of k is too large, then all clusters will be pure (so the only feedback one would get is a merge). If our guess of k is too small, then no two clusters produced by the algorithm will be subsets of the same target cluster (so the only feedback given will be a split). Thus we can find the correct number of clusters in $O(\log(n))$ queries. □

Note that since strict threshold separation implies strict separation, then the $O(k)$ algorithm presented in the next section can also be used, giving $O(\min(\log(n), k))$ queries.

Strict Separation: Now we relax strict threshold separation

STRICT SEPARATION. All points in the same cluster are more similar to one another than to points outside the cluster.

With this property, it is no longer true that all inner distances are smaller than outer distances, and therefore Theorem 4.4.1 does not apply. However, Balcan et al. [2008a] prove the following lemma

Lemma 4.4.3. *Balcan et al. [2008a] For a dataset satisfying strict separation, let SL(d) be the tree returned by Single-Linkage. Then any partitioning respecting the strict separation of d will be a pruning of SL(d).*

Theorem 4.4.4. *Given a dataset satisfying Strict Separation, there exists an algorithm which can find the target partitioning for any hypothesis class in $O(k)$ queries*

Proof. Let the distances between points be represented by the distance function d. By lemma 4.4.3 we know that the target partitioning must be a pruning of SL(d). Our algorithm will start by presenting the teacher with all points in a single cluster. Upon a split request, we split according to the relevant node in SL(d). There can be no merge requests since we always split perfectly. Each split will create a new cluster, so there will be at most $k - 1$ of these splits, after which the correct partitioning is found. □

γ-margin Separation: Margins show up in many learning models, and this is no exception. A natural assumption is that there may be a separation of at least γ between points in differing clusters, where the points all lie inside the unit ball.

γ-MARGIN SEPARATION. Points in different clusters of the target partitioning are at least γ away from one another.

With this property, we can prove the following for all hypothesis classes

Theorem 4.4.5. *Given a dataset satisfying γ-margin Separation, there exists an algorithm which can find the target partitioning for any hypothesis class in $O((\frac{\sqrt{d}}{\gamma})^d - k)$ queries*

Proof. We split the unit ball (inside which all points live) into hypercubes with edge length $\frac{\gamma}{\sqrt{d}}$. We are interested in the diameter of such a hypercube. The diameter of a d-dimensional hypercube with side $\frac{\gamma}{\sqrt{d}}$ is $\sqrt{d} \times \frac{\gamma}{\sqrt{d}} = \gamma$, so no two points inside a hypercube of side $\frac{\gamma}{\sqrt{d}}$ can be more than γ apart. It follows that if split the unit ball up using a grid of hypercubes, all points inside a hypercube must be from the same cluster. We say such a hypercube is "pure".

There are at most $O((\frac{\sqrt{d}}{\gamma})^d)$ hypercubes in a unit ball. We show each hypercube as a single cluster to the teacher. Since all hypercubes are pure, we can only get merge requests, of which there can be at most $O((\frac{\sqrt{d}}{\gamma})^d - k)$ until the target partitioning is found. □

4.5 Additional Results

4.5.1 A better algorithm for learning rectangles

In the original algorithm for learning rectangles described in 4.3.1, we reset the merges after every split. We can avoid it as follows: The algorithm, in addition to the hypothesis clusters, also maintains a graph G over the m points. Initially, the graph G has no edges. The algorithm proceeds as described below

1. Start with points $({a_{start}}', {a_{end}}')$ on the x-axis and points $({b_{start}}', {b_{end}}')$, such that all the points are contained in the rectangle defined by these points.

2. At each step, cluster the m points according to the region in which they belong. If the points in two regions form a clique in G merge the two regions. Repeat until no more regions can be merged. Present this clustering to the teacher.

3. On a merge request, create a clique in G corresponding to the points in the two clusters.

4. On a split of $({a_i}', {a_j}'), ({b_i}', {b_j}')$, create a new point ${a_r}'$ such that ${a_i}' < {a_r}' < {a_j}'$, and the projection of all the points onto $({a_i}', {a_j}')$ is divided into half by ${a_r}'$. Similarly, create a new point ${b_r}'$ such that ${b_i}' < {b_r}' < {b_j}'$, and the projection of all the points onto $({b_i}', {b_j}')$ is divided into half by ${b_r}'$.

Theorem 4.5.1. *The algorithm can cluster the class of rectangles in* 2 *dimensions using at most* $O((k \log m)^2)$ *queries.*

Proof. Lets first bound the total number of split requests. If the teacher says split on $(x_i, x_j), (y_i, y_j)$, then we know that either (x_i, x_j) contains a target point a or (y_i, y_j) contains a target point b or both. By creating two splits we are ensuring that the size of at least one of the regions containing a target point is reduced by half. There are at most $2k$ intervals on the x-axis and at most $2k$ intervals on the y-axis. Hence, the total number of split requests is $\leq 4k \log m$. Now, the total number of merge requests is at most the total number of regions created by the algorithm since, after every merge request we reduce the number of regions by 1. Since, t points on the x and the y axis can create at most t^2 regions, we get that the total number of merge requests is at most $\leq O(k \log m)^2$. Hence, the total number of queries made by the algorithm is $O((k \log m)^2)$. □

4.5.2 Proof of theorem 4.4.1

Proof. Let F be any Consistent, k-Rich, Order-Consistent partitioning function, and let d be any distance function on n points satisfying strict threshold separation w.r.t. C^* with threshold t. We want to show that for all $k > 0$, $F(d, k) = C^*$, the target. Whenever we say "inner" or "outer" edge for this proof, we mean with respect to C^*.

By k-Richness of F, there exists a d_1 such that $F(d_1, k) = \text{SL}(d, k) = C^*$. Now, through a series of transformations that preserve the output of F, we transform d_1 into d_2, then d_2 into d_3, ..., until we arrive at d. Let d_i be represented by an ordered list of its distances in ascending order $d_i = \langle e_1, e_2, \ldots, e_{\binom{n}{2}} \rangle$.

We begin the C^*-preserving transformations on d_1 to eventually transform d_1 into d while at each step i maintaining $F(d_i, k) = C^*$.

1. By k-Richness, we know there exists a d_1 such that $F(d_1, k) = C^*$.
2. Since all edges of p are inner edges, we can shrink them in d_1 until they are less than t. Call this newly created dataset d_2. This step maintains $F(d_2, k) = C^*$ by Consistency (we only shrank inner edges).
3. Now we reorder the inner edges of d_2 to be in the exactly the same order as they appear in d. Call the new dataset d_3. This step maintains $F(d_3, k) = C^*$ by Consistency (all these edges are of the same type - namely inner edges, so we may reorder them freely by shrinking an edge till it falls into place). Now we deal with the remaining outer edges.

4. We expand all outer edges until they are larger than all inner edges and call the result d_4. This step maintains $F(d_4, k) = C^*$ by Consistency.

5. Now we reorder all outer edges until their order in relation to each other is as they appear in d, and call the result d_5. This step maintains $F(d_5, k) = C^*$ by Consistency. Now, d_5 has all the inner edges in the same position as they appear in d, and has all the outer edges in the same order relative to one another as they appear in d. Since all inner edges are smaller than all outer edges (by the strict threshold property), all the edges are in the same order as they appear in d.

6. At this point in the edge ordering of d_5, all the edges lie in the same position as they do in d. However, their weights might be different than what appears in d. By using Order-Consistency, we can turn the weights of d_5 into exactly those of d, and call the result d_6. Since we didn't change the order of edges from d_5, by Order-Consistency we have that $F(d_6, k) = C^*$. It should be clear that $d_6 = d$.

7. Thus we have $F(d_6, k) = F(d, k) = C^*$.

We started with any d and k, and showed that

$$F(d, k) = C^*$$

□

Chapter 5

Local algorithms for supervised clustering

In this chapter, we modify the basic split-merge framework which was studied in Chapter 4 and study a new model with the goal of designing practically applicable supervised clustering algorithms. In the split-merge framework, the algorithm is given the freedom to propose arbitrary clusterings to the user and receive feedback. In most real-world clustering problems, however, we already start with a fairly good clustering computed with semi-automated techniques. For example, consider an online news portal that maintains a large collection of news articles. The news articles are clustered on the "back-end," and are used to serve several "front-end" applications such as recommendations and article profiles. For such a system, we do not have the freedom to compute arbitrary clusterings and present them to the user. But we can still *locally* edit the clustering and get limited feedback. In particular, one might only want to change the "bad" portion revealed by the feedback, otherwise respecting the clustering given. This is the problem that we address in this chapter.

We study an extension of the split-merge framework and provide strong experimental evidence supporting the practical applicability of our algorithms. In the new model the algorithm is given as input an initial clustering of the data. The algorithm then interacts with the user in stages. At each stage the algorithm makes a *local* change to the current clustering at hand and proposes it to the user. The user provides limited feedback in the form of *split* and *merge* requests. We say that a change made by the algorithm to the current clustering is local if in response to the user feedback the algorithm changes only the cluster assignments of the points in the corresponding clusters. If the user requests to split a cluster C_i, we may change only the cluster assignments of points in C_i, and if the

user requests to merge C_i and C_j , we may only reassign the points in C_i and C_j.

We study the query complexity of algorithms in the above model with respect to the error of the initial input clustering. The initial error can be naturally decomposed into *underclustering* error δ_u and *overclustering* error δ_o (See Section 5.1). Because the initial error may be fairly small,[1] we would like to develop algorithms whose query complexity depends polynomially on δ_u, δ_o and only logarithmically on n, the number of data points. Theoretically, we will view the initial input clustering as being adversarially given subject to bounded δ_u and δ_o. We show this is indeed possible assuming that the input similarity function satisfies a natural *stability* property with respect to the ground-truth clustering (see Section 8.1). The specific stability property we consider is a natural generalization of the "stable marriage" property (see Definition 5.1.2) that has been studied in a variety of previous works (Balcan et al. [2008b], Bryant and Berry [2001]). This is a realistic assumption and is strictly weaker than other often-studied stability notions such as strict separation and strict threshold separation (Balcan et al. [2008b], Krishnamurthy et al. [2012]).

Our Results

In Section 5.2 we study the η-merge model which was described in Chapter 4. For this model we show the following query bound

Theorem 5.0.2. *Given n data points, suppose the target clustering satisfies stability, and the initial clustering has overclustering error δ_o and underclustering error δ_u. In the η-merge model, for any $\eta > 0.5$, there is an efficient algorithm that requires at most δ_o split requests and $2(\delta_u + k)\log_{\frac{1}{1-\eta}} n$ merge requests to find the target clustering.*

In Section 5.3 we relax the condition on the merges and allow the user to issue a merge request even if C_i and C_j only have a single point belonging to the same target cluster. We call this the *unrestricted-merge* model. Here the requirement on accuracy of user response is much weaker and hence one needs to make further assumptions on the nature of requests. More specifically, we assume that each merge request is chosen uniformly at random from the set of feasible merges. Under this condition we show the following

Theorem 5.0.3. *Suppose the target clustering satisfies stability, and the initial clustering has overclustering error δ_o and underclustering error δ_u. In the unrestricted-merge model there exists an efficient algorithm which with probability at least $1 - \epsilon$, requires δ_o split requests and $O(\log \frac{k}{\epsilon}\delta_u^2)$ merge requests to find the target clustering.*

In Section 5.4 we also demonstrate the effectiveness of our algorithms on real data. We show that for the purposes of splitting known over-clusters, the splitting procedure pro-

[1] Given 2 different k clusterings, δ_u and δ_o is atmost k^2.

posed here computes the best splits, when compared to other often-used techniques. We also test the entire proposed framework on newsgroup documents data, which is quite challenging for traditional unsupervised clustering methods (Telgarsky and Dasgupta [2012], Heller and Ghahramani [2005], Dasgupta [2008], Dai et al. [2010], Boulis and Ostendorf [2004], Zhong [2005]). Several studies report that for this data set it is difficult to compute hierarchical-clustering trees that are consistent with the ground-truth (Telgarsky and Dasgupta [2012], Heller and Ghahramani [2005]). Still, using average-linkage trees that are only somewhat consistent with the ground-truth, our local algorithms are able to find the target clustering after a reasonable number of edit requests. In addition, the performance improves significantly when we first slightly prune the data. The results in thie chapter are based on work in Awasthi et al. [2013a].

5.1 Notation and Preliminaries

Given a cluster C_i and a clustering C', define

$$\mathrm{dist}(C_i, C') = |\{C'_j \in C' : C'_j \cap C_i \neq \emptyset\}| - 1.$$

This distance is the number of *additional* clusters in C' that contain points from C_i; it evaluates to 0 when all points in C_i are contained in a single cluster in C'. Naturally, we can then define the distance between C and C' as: $\mathrm{dist}(C, C') = \sum_{C_i \in C} \mathrm{dist}(C_i, C')$. Notice that this notion of clustering distance is asymmetric: $\mathrm{dist}(C, C') \neq \mathrm{dist}(C', C)$. Also note that $\mathrm{dist}(C, C') = 0$ if and only if C refines C'. If C is the ground-truth clustering, and C' is a proposed clustering, then $\mathrm{dist}(C, C')$ is define as the *underclustering error*, and $\mathrm{dist}(C', C)$ an *overclustering error*.

An underclustering error is an instance of several clusters in a proposed clustering containing points from the same ground-truth cluster; this ground-truth cluster is said to be *underclustered*. Conversely, an overclustering error is an instance of points from several ground-truth clusters contained in the same cluster in a proposed clustering; this proposed cluster is said to be *overclustered*. E.g., if running a linkage-style algorithm that starts with many tiny clusters and merges clusters together until its all one big ball, then initially data will be underclustered and at the end data will be overclustered. In the following sections we use $C^* = \{C^*_1, C^*_2, \ldots C^*_k\}$ to refer to the ground-truth clustering, and use C to refer to the initial clustering. We use δ_u to refer to the underclustering error of the initial clustering, and δ_o to refer to the overclustering error. In other words, we have $\delta_u = \mathrm{dist}(C^*, C)$ and $\delta_o = \mathrm{dist}(C, C^*)$.

Definition 5.1.1 (Local algorithm)**.** *We say that an interactive clustering algorithm is* local *if in each iteration only the cluster assignments of points involved in the oracle request may be changed. If the oracle proposes $split(C_i)$, only the points in C_i may be reassigned. If the oracle proposes $merge(C_i, C_j)$, only the points in $C_i \cup C_j$ may be reassigned.*

We next formally define the separation property of a clustering instance that we study in this work.

Definition 5.1.2 (Average Stability)**.** *Given a clustering instance $C = \{C_1, C_2, \cdots C_k\}$ over a domain S and a distance function $d : S \times S \mapsto \Re$, we say that C satisfies average stability if for all $i \neq j$, and for all $A \subset C_i$ and $A' \subseteq C_j$, $d_{avg}(A, C_i \setminus A) < d_{avg}(A, A')$. Here for any two sets A, A', $d_{avg}(A, A') = E_{x \in A, y \in A'} d(x, y)$.*

In the following sections we will assume that the ground-truth clustering of the data set satisfies this *stability* property. We study the following natural assumptions on the oracle requests, which require that the requests are consistent with the ground-truth.

Definition 5.1.3 (η-merge model)**.** *In the η-merge model the following guarantees hold for the oracle requests*

$split(C_i)$: C_i contains points from two or more target clusters.

$merge(C_i, C_j)$: At least an η-fraction of the points in each C_i and C_j belong to the same target cluster.

Definition 5.1.4 (Unrestricted-merge model)**.** *In the unrestricted-merge model the following guarantees hold for the oracle requests*

$split(C_i)$: C_i contains points from two or more target clusters.

$merge(C_i, C_j)$: At least 1 point in each C_i and C_j belongs to the same target cluster.

5.2 The η-merge model

In this section we describe and analyze the algorithms in the η-merge model. As a pre-processing step, we first run the average-linkage algorithm on all the points in the data set to compute the global average-linkage tree, which we denote by T_{avg}. The leaf nodes in this tree contain the individual points, and the root node contains all the points. The tree is computed in a bottom-up fashion: starting with the leafs in each iteration the two most similar nodes are merged, where the similarity between two nodes N_1 and N_2 is the average similarity between points in N_1 and points in N_2.

Our algorithms start by assigning a label "impure" to each cluster in the initial clustering. In each step, a local clustering edit is computed from T_{avg} as described in Figure 5.1 and Figure 5.2. To implement Step 1 in Figure 5.1, we start at the root of T_{avg} and "follow"

Figure 5.1: Split procedure

Algorithm: SPLIT PROCEDURE

Input: Cluster C_i, global average-linkage tree T_{avg}.

1. Search T_{avg} to find the node N at which the set of points in C_i are first split in two.
2. Let N_1 and N_2 be the children of N. Set $C_{i,1} = N_1 \cap C_i$, $C_{i,2} = N_2 \cap C_i$.
3. Delete C_i and replace it with $C_{i,1}$ and $C_{i,2}$. Mark the two new clusters as "impure".

the points in C_i down one of the branches until we find a node that splits them. In order to implement Step 2 in Figure 5.2, it suffices to start at the root of T_{avg} and perform a post-order traversal, only considering nodes that have "enough" points from both clusters, and return the first output node. We now state the performance guarantee for these split

Figure 5.2: Merge procedure for the η-merge model

Algorithm: MERGE PROCEDURE

Input: Clusters C_i and C_j, global average-linkage tree T_{avg}.

1. If C_i is marked as "pure" set $\eta_1 = 1$ else set $\eta_1 = \eta$. Similarly set η_2 for C_j.
2. Search T_{avg} for a node of maximal depth N that contains *enough* points from C_i and C_j: $|N \cap C_i| \geq \eta_1 |C_i|$ and $|N \cap C_j| \geq \eta_2 |C_j|$.
3. Replace C_i by $C_i \setminus N$, replace C_j by $C_j \setminus N$.
4. Add a new cluster containing $N \cap (C_i \cup C_j)$, mark it as "pure".

and merge algorithms.

Theorem 5.2.1. *Suppose the target clustering satisfies average stability, and the initial clustering has overclustering error δ_o and underclustering error δ_u. In the η-merge model, for any $\eta > 0.5$, the algorithms in Figure 5.1 and Figure 5.2 require at most δ_o split requests and $2(\delta_u + k)\log_{\frac{1}{1-\eta}} n$ merge requests to find the target clustering.*

In order to prove this theorem, we first establish key properties of the split and the merge procedures.

Definition 5.2.2 (Clean split)**.** *A partition (split) of a cluster C_i into clusters $C_{i,1}$ and $C_{i,2}$ is said to be* clean *if $C_{i,1}$ and $C_{i,2}$ are non-empty, and for each ground-truth cluster C_j^* such that $C_j^* \cap C_i \neq \emptyset$, either $C_j^* \cap C_i = C_j^* \cap C_{i,1}$ or $C_j^* \cap C_i = C_j^* \cap C_{i,2}$.*

To prove that the procedure in Figure 5.1 computes a clean split we use the property that each node of the global average-linkage tree is *laminar* (consistent) with respect to the ground-truth clustering. This property is formalized below.

Definition 5.2.3 (Laminarity)**.** *A set C_i is laminar with respect to the ground-truth clustering C^* if for each cluster $C_j^* \in C^*$ we have either $C_i \cap C_j^* = \emptyset$, $C_i \subseteq C_j^*$, or $C_j^* \subseteq C_i$.*

Lemma 5.2.4. *Suppose the ground-truth clustering C^* satisfies average stability. Let T_{avg} be the average-linkage tree for this data set. Then every node in T_{avg} is laminar w.r.t. C^*.*

Proof. The proof of this statement can be found in Balcan et al. [2008b]. The intuition comes from the fact that if there is a node in T_{avg} that is not laminar w.r.t. C^*, then the average-linkage algorithm, at some step, must have merged $A \subset C_i^*$, with $B \subset C_j^*$ for some $i \neq j$. However, this will contradict the stability property for the sets A and B. □

Lemma 5.2.5. *If the ground-truth clustering satisfies average stability and $\eta > 0.5$ then,*

- ***a.*** *The split procedure in Figure 5.1 always produces a clean split.*
- ***b.*** *The new cluster added in Step 4 in Figure 5.2 must be "pure", i.e., it must contain points from a single ground-truth cluster.*

Proof. **a.** For purposes of contradiction, suppose the returned split is not clean: $C_{i,1}$ and $C_{i,2}$ contain points from the same ground-truth cluster C_j^*. It must be the case that C_i contains points from several ground-truth clusters, which implies that w.l.o.g. $C_{i,1}$ contains points from some other ground-truth cluster $C_{l \neq j}^*$. This implies that N_1 is not laminar w.r.t C^*, which contradicts Lemma 5.2.4.

b. By our assumption, at least $\frac{1}{2}|C_i|$ points from C_i and $\frac{1}{2}|C_j|$ points from C_j are from the same ground-truth cluster C_l^*. Clearly, the node N' in T_{avg} that is equivalent to C_l^* (which contains all the points in C_l^* and no other points) must contain *enough* points from C_i and C_j, and only ascendants and descendants of N' may contain enough points from both clusters. Therefore, the node N that we find with a depth-first search must be N' or one of its descendants, and will only contain points from C_l^*. □

Using the above lemma, we can prove the bounds on the split and merge requests stated in Theorem 5.2.1.

Proof of Theorem 5.2.1. We first give a bound on the number of splits. Observe that each split reduces the overclustering error by exactly 1. To see this, suppose we execute Split(C_1), and call the resulting clusters C_2 and C_3. Call $\delta_o^{(1)}$ the overclustering error before the split, and $\delta_o^{(2)}$ the overclustering error after the split. Let's use k_1 to refer to the number of ground-truth clusters that intersect C_1, and define k_2 and k_3 similarly. Due to the *clean split* property, no ground-truth cluster can intersect both C_2 and C_3, therefore it must be the case that $k_2 + k_3 = k_1$. Also, clearly $k_2, k_3 > 0$. Therefore we have:

$$\begin{aligned}\delta_o^{(2)} &= \delta_o^{(1)} - (k_1 - 1) + (k_2 - 1) + (k_3 - 1) \\ &= \delta_o^{(1)} - k_1 + (k_2 + k_3) - 1 \\ &= \delta_o^{(1)} - 1.\end{aligned}$$

Merges cannot increase overclustering error. Therefore the total number of splits may be at most δ_o. We next give the arguments about the number of impure and pure merges.

We first argue that we cannot have too many "impure" merges before each cluster in C is marked "pure." Consider the clustering $P = \{C_i \cap C_j^* \mid C_i \text{ is not "pure" and } C_i \cap C_j^* \neq \emptyset\}$. Clearly, at the start $|P| = \delta_u + k$. A merge does not increase the number of clusters in P, and the splits do not change P at all (because of the *clean split* property). Moreover, each impure merge (a merge of two impure clusters or a merge of a pure and an impure cluster) *depletes* some $P_i \in P$ by moving $\eta|P_i|$ of its points to a pure cluster. Clearly, we can then have at most $\log_{1/(1-\eta)} n$ merges depleting each P_i. Since each impure merge must deplete some P_i, it must be the case that we can have at most $(\delta_u + k) \log_{1/(1-\eta)} n$ impure merges in total.

Notice that a pure cluster can only be created by an impure merge, and there can be at most one pure cluster created by each impure merge. Clearly, a pure merge removes exactly one pure cluster. Therefore the number of pure merges may be at most the total number of pure clusters that are created, which is at most the total number of impure merges. Therefore the total number of merges must be less than $2(\delta_u + k) \log_{1/(1-\eta)} n$. □

5.3 The unrestricted-merge model

In this section we further relax the assumptions about the nature of the oracle requests. As before, the oracle may request to split a cluster if it contains points from two or more target clusters. For merges, now the oracle may request to merge C_i and C_j if both clusters contain only a single point from the same ground-truth cluster. We note that this is a minimal set of assumptions for a local algorithm to make progress, otherwise the oracle may always propose irrelevant splits or merges that cannot reduce clustering error. For this model we propose the merge algorithm described in Figure 5.3. The split algorithm remains the same as in Figure 5.1.

Figure 5.3: Merge procedure for the unrestricted-merge model

Algorithm: MERGE PROCEDURE

Input: Clusters C_i and C_j, global average-linkage tree T_{avg}.

1. Let C_i', C_j' = Split($C_i \cup C_j$), where the split is performed as in Figure 5.1.
2. Delete C_i and C_j.
3. If the sets C_i' and C_j' are the same as C_i and C_j, then add $C_i \cup C_j$, otherwise add C_i' and C_j'.

To provably find the ground-truth clustering in this setting we require that each merge request must be chosen uniformly at random from the set of feasible merges. This assumption is consistent with the observation in Awasthi and Zadeh [2010] which implies that in the unrestricted-merge model with arbitrary request sequences, even very simple cases (ex. union of intervals on a line) require a prohibitively large number of requests. We do not make additional assumptions about the nature of the split requests; in each iteration any feasible split may be proposed by the oracle. In this setting our algorithms have the following performance guarantee.

Theorem 5.3.1. *Suppose the target clustering satisfies average stability, and the initial clustering has overclustering error δ_o and underclustering error δ_u. In the unrestricted-merge model, with probability at least $1 - \epsilon$, the algorithms in Figure 5.1 and Figure 5.3 require δ_o split requests and $O(\log \frac{k}{\epsilon} \delta_u^2)$ merge requests to find the target clustering.*

The above theorem is proved in a series of lemmas. We first state a lemma regarding the correctness of the Algorithm in Figure 5.3. We argue that if the algorithm merges

C_i and C_j, it must be the case that both C_i and C_j only contain points from the same ground-truth cluster.

Lemma 5.3.2. *If the algorithm in Figure 5.3 merges C_i and C_j in Step 3, it must be the case that $C_i \subset C_l^*$ and $C_j \subset C_l^*$ for some ground-truth cluster C_l^*.*

Proof. We prove the contrapositive. Suppose C_i and C_j both contain points from C_l^*, and in addition $C_i \cup C_j$ contains points from some other ground-truth cluster. Let us define $S_1 = C_l^* \cap C_i$ and $S_2 = C_l^* \cap C_j$. Because the clusters C_i', C_j' result from a "clean" split, it follows that $S_1, S_2 \subseteq C_i'$ or $S_1, S_2 \subseteq C_j'$. Without loss of generality, assume $S_1, S_2 \subseteq C_i'$. Then clearly $C_i' \neq C_i$ and $C_i' \neq C_j$, so C_i and C_j are not merged. □

The δ_o bound on the number of split requests follows from the observation that each split reduces the overclustering error by exactly 1 (as before), and the fact that the merge procedure does not increase overclustering error, which follows from the lemma below.

Lemma 5.3.3. *The merge algorithm in Figure 5.3 does not increase overclustering error.*

Proof. Suppose C_i and C_j are not both "pure," and hence we obtain two new clusters C_i', C_j'. Let us call $\delta_o^{(1)}$ the overclustering error before the merge, and $\delta_o^{(2)}$ the overclustering error after the merge. Let's use k_1 to refer to the number of ground-truth clusters that intersect C_i, k_2 to refer to the number of ground-truth clusters that intersect C_j, and define k_1' and k_2' similarly. The new clusters C_i' and C_j' result from a "clean" split, therefore no ground-truth cluster may intersect both of them. It follows that $k_1' + k_2' \leq k_1 + k_2$. Therefore we now have:

$$\begin{aligned} \delta_o^{(2)} &= \delta_o^{(1)} - (k_1 - 1) - (k_2 - 1) + (k_1' - 1) + (k_2' - 1) \\ &= \delta_o^{(1)} - (k_1 + k_2) + (k_1' + k_2') \leq \delta_o^{(1)}. \end{aligned}$$

If C_i and C_j are both "pure," then clearly the merge operation has no effect on the overclustering error. □

The following lemmas bound the number of impure and pure merges. To clarify, we call a proposed merge *pure* if both clusters are subsets of the same ground-truth cluster, and *impure* otherwise.

Lemma 5.3.4. *The merge algorithm in Figure 5.3 requires at most δ_u impure merge requests.*

Proof of Lemma 5.3.4. We argue that each impure merge reduces the underclustering error of the current clustering by at least 1. To make our argument, we use $\delta_u(C_i^*)$ to refer to the underclustering error with respect to the ground-truth cluster C_i^*. In other words, $\delta_u(C_i^*) = \mathrm{dist}(C_i^*, C')$, where C' is the current clustering.

Suppose we execute Merge(C_1, C_2), and C_1 and C_2 are not both "pure." Let us use C_1' and C_2' to refer to the resulting clusters. We divide the ground-truth clusters into three groups: clusters that intersect neither C_1 nor C_2 (group-1), clusters that intersect exactly one of C_1, C_2 (group-2), and clusters that intersect both C_1 and C_2 (group 3).

Let us use $\delta_u^{(1)}$ to refer to the underclustering error before the merge, and $\delta_u^{(2)}$ to refer to the underclustering error after the merge. Clearly, for ground-truth clusters C_i^* in group-1 we have $\delta_u^{(2)}(C_i^*) = \delta_u^{(1)}(C_i^*)$. The clusters C_1' and C_2' result from a "clean" split, therefore no ground-truth cluster may intersect both of them. It follows that for ground-truth clusters C_i^* in group-2 we also have $\delta_u^{(2)}(C_i^*) = \delta_u^{(1)}(C_i^*)$. It also follows that for ground-truth clusters C_i^* in group-3 we must have $\delta_u^{(2)}(C_i^*) = \delta_u^{(1)}(C_i^*) - 1$.

Our argument immediately follows from these observations. We have

$$\begin{aligned}
\delta_u^{(2)} &= \sum_{C_i^* \in \text{ group-1}} \delta_u^{(2)}(C_i^*) + \sum_{C_i^* \in \text{ group-2}} \delta_u^{(2)}(C_i^*) \\
&+ \sum_{C_i^* \in \text{ group-3}} \delta_u^{(2)}(C_i^*) \\
&= \sum_{C_i^* \in \text{ group-1}} \delta_u^{(1)}(C_i^*) + \sum_{C_i^* \in \text{ group-2}} \delta_u^{(1)}(C_i^*) \\
&+ \sum_{C_i^* \in \text{ group-3}} (\delta_u^{(1)}(C_i^*) - 1) \\
&= \delta_u^{(1)} - |\{C_i^* : C_i^* \in \text{ group-3}\}|.
\end{aligned}$$

Because there must be at least one ground-truth cluster in group-3, it follows that $\delta_o^{(2)} \leq \delta_o^{(1)} - 1$. □

Lemma 5.3.5. *The probability that the algorithm in Figure 5.3 requires more than $O(\log \frac{k}{\epsilon}\delta_u^2)$ pure merge requests is less than ϵ.*

Proof of Lemma 5.3.5. We first consider the pure merge requests involving points from some ground-truth cluster C_i^*, the total number of pure merge requests (involving any ground-truth cluster) can then be bounded with a union-bound.

Suppose we assign an identifier to each cluster containing points from C_i^* in the following manner:

1. Maintain a CLUSTER-ID variable, which is initialized to 1.

2. To assign a "new" identifier to a cluster, set its identifier to CLUSTER-ID, and increment CLUSTER-ID.

3. In the initial clustering, assign a *new* identifier to each cluster containing points from C_i^*.

4. When we split a cluster containing points from C_i^*, assign its identifier to the newly-formed cluster containing points from C_i^*.

5. When we merge two clusters and one or both of them are impure, if one of the clusters contains points from C_i^*, assign its identifier to the newly-formed cluster containing points from C_i^*. If both clusters contain points from C_i^*, assign a *new* identifier to the newly-formed cluster containing points from C_i^*.

6. When we merge two clusters C_1 and C_2, and both contain only points from C_i^*, if the outcome is one new cluster, assign it a *new* identifier. If the outcome is two new clusters, assign them the identifiers of C_1 and C_2.

Clearly, when clusters containing points from C_i^* are assigned identifiers in this manner, the maximum value of CLUSTER-ID is bounded by $O(\delta_i)$, where δ_i denotes the underclustering error of the initial clustering with respect to C_i^*: $\delta_i = \mathrm{dist}(C_i^*, C)$. To see this, consider that we assign exactly $\delta_i + 1$ new identifiers in Step-3, and each time we assign a new identifier in Steps 5 and 6, the underclustering error of the edited clustering with respect to C_i^* decreases by one.

We say that a *pure* merge request involving points from C_i^* is *original* if the user has never asked us to merge clusters with the given identifiers, otherwise we say that this merge request is *repeated*. Given that the maximum value of CLUSTER-ID is bounded by $O(\delta_i)$, the total number of *original* merge requests must be $O(\delta_i^2)$. We now argue that if a merge request is not original, we can lower bound the probability that it will result in the merging of the two clusters.

For repeated merge request $M_i = Merge(C_1, C_2)$, let X_i be a random variable defined as follows:

$$X_i = \begin{cases} 1 & \text{if neither } C_1 \text{ nor } C_2 \text{ have been involved in} \\ & \text{a merge request since the last time a merge of} \\ & \text{clusters with these identifiers was proposed.} \\ 0 & \text{otherwise.} \end{cases}$$

Clearly, when $X_i = 1$ it must be the case that C_1 and C_2 are merged. We can show that $\Pr[X_i = 1] \geq \frac{1}{4(\delta_i+1)}$. The intuition for this argument is quite simple: in each step the probability that the user requests to merge C_1 and C_2 is $\frac{1}{n'}$, and the probability that the user requests to merge C_1 or C_2 with some other cluster is $O(\frac{\delta_i}{n'})$, where n' is the total number of possible merges, so we can bound the probability that the former happens before the latter.

We can then use a Chernoff bound to argue that after $t = O(\log \frac{k}{\epsilon}\delta_i^2)$ *repeated* merge requests, the probability that $\sum_{i=1}^{t} X_i < \delta_i$ (which must be true if we need more *repeated* merge requests) is less than ϵ/k. Therefore, the probability that we need more than $O(\log \frac{k}{\epsilon}\delta_i^2)$ *repeated* merge requests is less than ϵ/k.

By the union-bound, the probability that we need more than $O(\log \frac{k}{\epsilon}\delta_i^2)$ *repeated* merge requests for *any* ground-truth cluster C_i^* is less than $k \cdot \epsilon/k = \epsilon$. Therefore with probability at least $1 - \epsilon$ for all ground-truth clusters we need $\sum_i O(\log \frac{k}{\epsilon}\delta_i^2) = O(\log \frac{k}{\epsilon}\sum_i \delta_i^2) = O(\log \frac{k}{\epsilon}\delta_u^2)$ *repeated* merge requests, where δ_u is the underclustering error of the original clustering. Similarly, for all ground-truth clusters we need $\sum_i O(\delta_i^2) = O(\delta_u^2)$ *original* merge requests. Adding the two terms together, it follows that with probability at least $1 - \epsilon$ we need a total of $O(\log \frac{k}{\epsilon}\delta_u^2)$ pure merge requests. □

5.4 Experimental Results

We perform two sets of experiments: we first test the proposed split procedure on the clustering of business listings maintained by Google, and also test the proposed framework in its entirety on the much smaller newsgroup documents data set.

5.4.1 Clustering business listings

Google maintains a large collection of data records representing businesses. These records are clustered using a similarity function; each cluster should contain records about the same distinct business; each cluster is summarized and served to users online via various front-end applications. Users report bugs such as "you are displaying the name of one business, but the address of another" (caused by over-clustering), or "a particular business is shown multiple times" (caused by under-clustering). These bugs are routed to operators who examine the contents of the corresponding clusters, and request splits/merges accordingly. However, the clusters involved in these requests are often quite "messy" (contain records about several businesses), and automated tools that can perform the requested edits

are needed.

In particular, here we evaluate the effectiveness of our proposed split procedure in computing desirable cluster splits. We consider a binary split desirable iff the two resulting sub-clusters are "clean" using Definition 5.2.2. For this application, automated splits are very relevant and "clean" splits are desirable because they must reduce the over-clustering error, and should correct some of the corresponding errors on the front-ends. To compute the splits, we use a "local" variation of the algorithm in Figure 5.1, where we use the average-linkage tree built only from the points in the cluster (referred to as *Clean-Split*).

For comparison purposes, we use two well-known techniques for computing binary splits: the optimal 2-median clustering (*2-Median*), and a "sweep" of the second-smallest eigenvector of the corresponding Laplacian matrix. Let $\{v_1, \ldots, v_n\}$ be the order of the vertices when sorted by their eigenvector entries, we compute the partition $\{v_1, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_n\}$ such that its conductance is smallest (*Spectral-Balanced*), and a partition such that the similarity between v_i and v_{i+1} is smallest (*Spectral-Gap*).

We compare the split procedures on 25 over-clusters that were discovered during a clustering-quality evaluation[2]. The results are presented in Table 5.1. We observe that the *Clean-Split* algorithm works best, giving a desirable split in 22 out of the 25 cases. The well-known *Spectral-Balanced* technique usually does not give desirable splits for this application: the balance constraint usually causes it to put records about the same business on both sides of the partition, especially when all the "clean" splits are not well-balanced. The result is a split that in fact usually increases the over-clustering error. As expected, the *Spectral-Gap* technique improves on this limitation, but the result is often still not desirable. The *2-Median* algorithm performs fairly well, but we believe that it is still not the right technique for this problem: the optimal centers may be records about the same business, and even if they are not, the resulting partition is still sometimes not desirable.

Table 5.1: Number of desirable splits

Clean-Split	2-Median	Spectral-Gap	Spectral-Balanced
22	18	16	4

[2]I would like to thank Konstantin Voevodski for allowing me to include the experimental results in this thesis

5.4.2 Clustering newsgroup documents

In order to test our entire framework (the iterative application of our algorithms), we perform computational experiments on newsgroup documents data.[3] The objects in these data sets are posts to twenty different online forums (newsgroups). We sample these data to get data sets of manageable size (labeled A through E in the figures).

We compute an initial clustering by perturbing the ground-truth. In each iteration, we compute the set of all feasible splits and merges: a split of a cluster is feasible if it contains points from 2 or more ground-truth clusters, and a merge is feasible if at least an η- fraction of points in each cluster are from the same ground-truth cluster. Then, we choose one of the feasible edits uniformly at random, and ask the algorithm to compute the corresponding edit. We continue this process until we find the ground-truth clustering or we reach 20000 iterations.

It is relevant to note how many iterations we expect to require in such an experiment. Our initial clusterings have over-clustering error of about 100, and under-clustering error of about 100. Our theoretical analysis indicates that in the worst case we would then require on the order of several thousand iterations in the first model, and several tens of thousands of iterations in the second model.

We notice that for newsgroup documents we cannot compute average-linkage trees that are very consistent with the ground-truth. This observation was also made in other clustering studies that report that the hierarchical trees constructed from these data have low "purity" (Telgarsky and Dasgupta [2012], Heller and Ghahramani [2005]). To test how well our algorithms can perform with better data, we prune the data sets by repeatedly finding the outlier in each target cluster and removing it, where the outlier is the point with minimum sum-similarity to the other points in the target cluster. For each data set, we perform experiments with the original (unpruned) data set, a pruned data set with 2 points removed per target cluster, and a pruned data set with 4 points removed per target cluster, which prunes 40 and 80 points, respectively (given that we have 20 target clusters).

Experiments in the η-merge model

We first experiment with local clustering algorithms in the η-restricted merge setting. Here we use the algorithm in Figure 5.1 to perform the splits, and the algorithm in Figure 5.2 to perform the merges. We show the results of running our algorithm Figure 5.4. We find that for the pruned data sets, the number of edit requests (necessary to find the target

[3] http://people.csail.mit.edu/jrennie/20Newsgroups/

clustering) is very favorable and is better than our worst-case theoretical analysis.

For the unpruned data sets, we struggle to find the ground-truth clustering for $\eta = 0.5$. Our investigations show that because of inconsistencies in the average-linkage tree we sometimes get loops of incorrect splits and merges and are unable to edit the clustering any further. Still, when we limit what merges may be proposed (by increasing η) we avoid loops of incorrect edits and quickly find the ground-truth clustering.

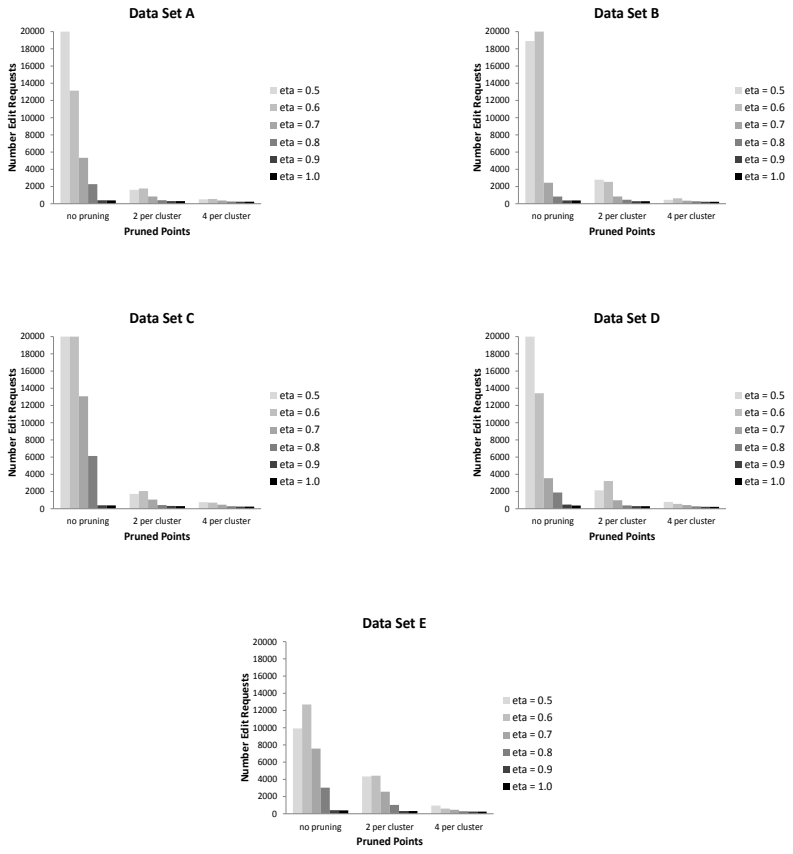


Figure 5.4: Performance of interactive clustering algorithms in the η-merge model.

Experiments in the unrestricted-merge model

We also experiment with algorithms in the unrestricted merge model. Here we use the same algorithm to perform the splits, but use the algorithm in Figure 5.3 to perform the

merges. We show the results in Figure 5.5. As before, we find that for the pruned data sets we require few edit requests to find the ground-truth clustering. For the unpruned data sets, we again struggle to find the ground-truth clustering for smaller values of η (because of inconsistencies in the average-linkage tree). For larger settings of η (we only show results for $\eta \geq 0.5$) the number of edit requests is once again better than our worst-case theoretical analysis.

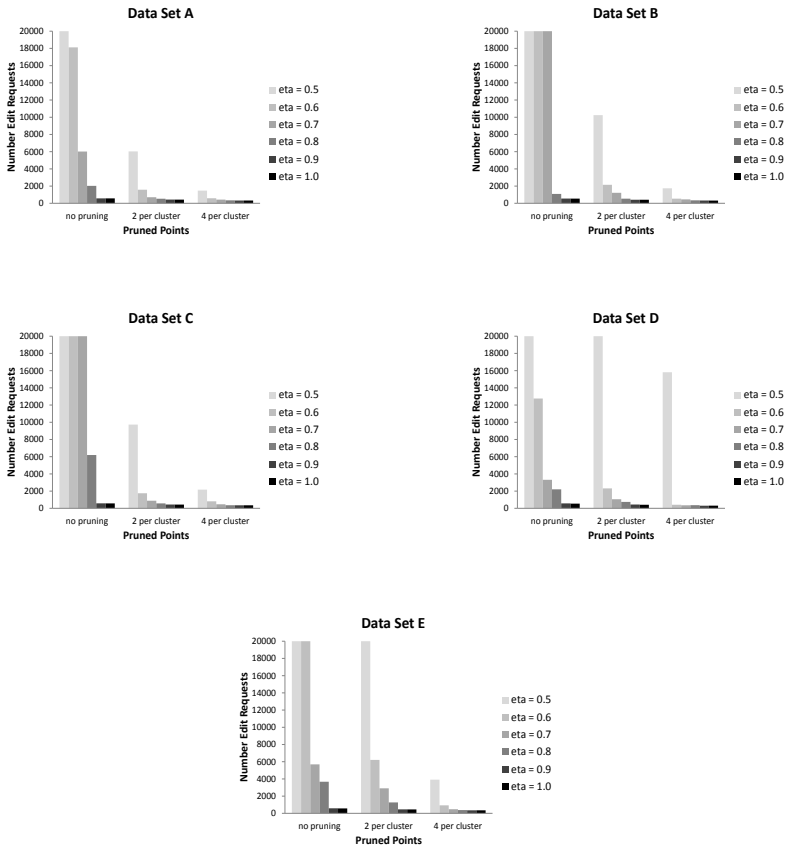


Figure 5.5: Performance of interactive clustering algorithms in the unrestricted-merge model.

We can address some of the inconsistencies in the average-linkage tree by constructing it in a more robust way, which indeed gives improved performance for unpruned data sets.

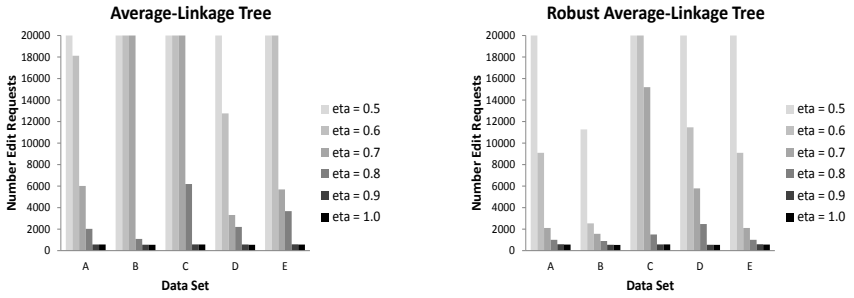


Figure 5.6: Performance of interactive clustering algorithms in the unrestricted-merge model, given different ways of constructing the average-linkage tree. Results presented for unpruned data sets.

5.4.3 Improved performance by using a robust average-linkage tree

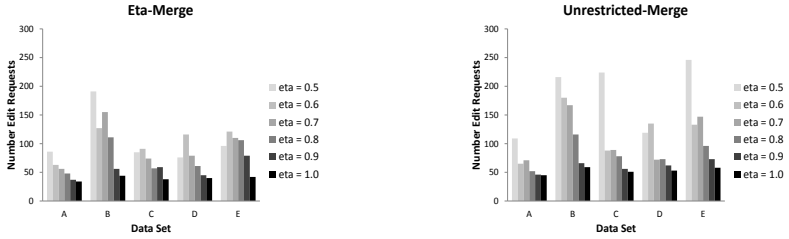
In certain cases our algorithms don't perform well, we find that certain inconsistencies in the average-linkage tree are the problem. There are several "outlier" points that are attached near the root of the tree, which are incorrectly split off and re-merged by the algorithm without making any progress towards finding the target clustering.

We can address these outliers by constructing the average-linkage tree in a more robust way: first find groups of similar points of some minimum size, compute an average-linkage tree for each group, and then merge these trees using average-linkage. The tree constructed in such fashion may then be used by our algorithms.

We tried this approach, using Algorithm 2 from Balcan and Gupta [2010] to compute the groups of points. We find that using the robust average-linkage tree gives better performance for the unpruned data sets, but gives no gains for the pruned data sets. Figure 5.6 displays the comparison for the five unpruned data sets. For the pruned data sets, it's likely that the robust tree and the standard tree are very similar, which explains why there is little difference in performance (results not shown).

5.4.4 Experiments with small initial error

We also consider a setting where the initial clustering is already very accurate. In order to simulate this scenario, when we compute the initial clustering, for each document we keep



(a) Small initial error, results presented for pruned data sets (4 points per cluster).

(b) Small initial error, results presented for pruned data sets (4 points per cluster).

Figure 5.7: Results in the η-merge and the unrestricted merge model.

its ground-truth cluster assignment with probability 0.95, and otherwise reassign it to one of the other clusters, which is chosen uniformly at random. This procedure gives us initial clusterings with over-clustering and under-clustering error between 5 and 20. As expected, in this setting our interactive algorithms perform much better, especially on pruned data sets. Figures 5.7a and 5.7b display the results; we can see that in these cases it often takes less than one hundred edit requests to find the target clustering in both models.

Part II

Learning

Chapter 6

Background

The most popular theoretical model for designing and analyzing learning algorithms is the Probably Approximately Correct (PAC) model of learning introduced by Valiant [1984]. In the PAC model, the goal is to design algorithms which can "learn" an unknown target function, f, from a concept class, C (for example, C may be polynomial-size decision trees or linear separators)[1], where f is a function over some instance space, X (typically $X = \{-1, 1\}^n$ or $X \subseteq \mathbb{R}^n$). The learning algorithm has access to random *labeled* examples, $(x, f(x))$, through an oracle, $\mathsf{EX}(f, D)$, where f is the unknown target concept and D is the target distribution. The goal of the learning algorithm is to output a hypothesis, h, with low error with respect to the target concept, f, under distribution, D. More formally, we have the following definition,

Definition 6.0.1 (PAC **Learning** Valiant [1984])**.** *Let $\mathcal{D}$ be a distribution over X and C be a concept class over X, and $f \in C$. An example oracle, $\mathsf{EX}(f, D)$, when queried, returns $(x, f(x))$, where x is drawn randomly from distribution D. The learning algorithm in the* PAC *model has access to an example oracle, $\mathsf{EX}(f, D)$, where $f \in C$ is the unknown target concept and D is the target distribution. The goal of the learning algorithm is to output a hypothesis, h, that has low error with respect to the target concept under the target distribution, i.e. $\mathbf{err}_D(h, f) = \Pr_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$. We say that the algorithm* PAC*-learns C, if for all $\epsilon, \delta > 0$, for all $f \in C$ and distributions D over X, it can, with probability at least $(1 - \delta)$, produce a hypothesis of error at most ϵ.*

In this thesis we will be interested in efficient PAC learning. Hence, we would require the learning algorithm to run in time $p(1/\epsilon, 1/\delta, length(C))$, for some polynomial p. Here

[1]Formally a concept class is a set of functions along with a representation for those functions.

$length(C)$ refers to the description length needed to specify a concept in the class C [2]. From now on, we would say that a class is "learnable" to mean that it is efficiently learnable in the above mentioned sense. Several interesting concept classes have been shown to be *learnable* in the PAC framework (e.g. boolean conjunctions and disjunctions, k-CNF and k-DNF formulas (for constant k), decision lists and the class of linear separators). On the other hand, it is known that very rich concept classes such as polynomial-sized circuits are not PAC-learnable under cryptographic assumptions (Valiant [1984], Goldreich et al. [1986]). The most interesting classes for which both efficient PAC learning algorithms and cryptographic lower bounds have remained elusive are polynomial-size decision trees (even $\log$-depth decision trees) and polynomial-size DNF formulas.

6.0.5 Membership Query Model

This learning setting is an extension of the PAC model and allows the learning algorithm to query the label of any point x of its choice in the domain. These queries are called *membership queries* and the learning model is popularly known as the PAC + MQ model. With this additional power it has been shown that the classes of finite automata (Angluin [1987]), monotone DNF formulas (Angluin and Laird [1988]), polynomial-size decision trees (Bshouty [1993]), and sparse polynomials over GF(2) (Schapire and Sellie [1996]) are learnable in polynomial time. In a celebrated result, Jackson [1997] showed that the class of DNF formulas is learnable in the PAC+MQ model under the uniform distribution. Jackson [1997] used Fourier analytic techniques to prove this result building upon previous work of Kushilevitz and Mansour [1993] on learning decision trees using membership queries under the uniform distribution. Formally, we have

Definition 6.0.2 (**Membership Queries).** *Let $f \in C$ be a concept defined over instance space X. Then a* membership query *is a point $x \in X$. A membership query oracle* $\mathsf{MQ}(f)$*, on receiving query $x \in X$, responds with value $f(x)$. In the* PAC+MQ *model of learning, along with the example oracle* $\mathsf{EX}(f, D)$*, the learning algorithm also has access to a membership oracle,* $\mathsf{MQ}(f)$.

6.0.6 Weak Learning

The definition of learning mentioned in 6.0.1 is also often referred to as *strong learning*. This is because we want the learning algorithm to be able to produce hypotheses which are

[2]For example if C is the class of disjunctions over $\{-1, 1\}^n$, we would need $O(n)$ bits to represent a concept in C.

arbitrarily good (in terms of error ϵ), provided enough training samples. In contrast, one could also consider designing algorithms which are able to produce hypothese only upto a certain error rate (say 1/4). This kind of learning is known as *weak learning*.

Definition 6.0.3 (γ **Weak Learning** Kearns and Valiant [1994])**.** *Let $\mathcal{D}$ be a distribution over X and C be a concept class over X, and $f \in C$. An example oracle, $\mathsf{EX}(f, D)$, when queried, returns $(x, f(x))$, where x is drawn randomly from distribution D. The learning algorithm has access to an example oracle, $\mathsf{EX}(f, D)$, where $f \in C$ is the unknown target concept and D is the target distribution. The goal of the learning algorithm is to output a hypothesis, h, that has an advantage γ over randomly guessing the labels of the examples. In other words, $\mathbf{err}_D(h, f) = \Pr_{x\sim D}[h(x) \neq f(x)] \leq \frac{1}{2} - \gamma$. We say that the algorithm is γ weak learner for class C if it can, with high probability, produce a hypothesis of advantage γ for all $f \in C$ and all distributions D over X.*

In a seminal paper, Schapire [1990] showed that in the PAC model, weak learning is equivalent to strong learning in a formal sense. In particular, Schapire [1990] showed the existence of an algorithm[3] which given black box access to a weak learner for a class C, produces a new hypothesis which strongly PAC learns C.

Definition 6.0.4 (**Boosting** Schapire [1990], Freund and Schapire [1995])**.** *There exists a procedure which given the example oracle $\mathsf{EX}(f, D)$ for a concept class C and a γ weak learner for a class C, for any $f \in C$ and distribution D over X, makes $O(\frac{1}{\gamma^2}\log(1/\epsilon))$ calls to the weak learner and outputs, with high probability, a hypothesis h such that, $\mathbf{err}_D(h, f) \leq \epsilon$.*

A more practical version of the boosting algorithm of Schapire [1990] called AdaBoost was later proposed in Freund and Schapire [1995].

6.0.7 Learning in the presence of noise

The PAC model of learning assumes that one has access to a perfect example oracle $\mathsf{EX}(f, D)$ so that each example x the algorithm receives is labeled correctly according to $f(x)$. A more realistic model would allow for noise in the response of the example oracle. We briefly describe two such noise models which are popularly studied.

Learning with random noise
This is a simple extension of the PAC model of learning. In this model, we assume that the example oracle has a noise rate η and is denoted by $\mathsf{EX}^{\eta}(f, D)$. At each step, an example

[3]Popularly known as a Boosting procedure

x is generated according to D and is labeled according to $f(x)$. Then the label is flipped independently with probability η. The learning algorithm gets to see the noisy labeled examples generated from $\mathsf{EX}^{\eta}(f, D)$.

Definition 6.0.5 ($\mathsf{PAC}+$ **random noise learning** Angluin and Laird [1988])**.** *Let $\mathcal{D}$ be a distribution over X and C be a concept class over X, and $f \in C$. An example oracle, $\mathsf{EX}^{\eta}(f, D)$, when queried, returns $(x, f(x))$ with probability $(1-\eta)$ and returns $(x, -f(x))$ with probability η. Here x is drawn randomly from distribution D. The learning algorithm in this model has access to an example oracle, $\mathsf{EX}(f, D)$, where $f \in C$ is the unknown target concept and D is the target distribution. The goal of the learning algorithm is to output a hypothesis, h, that has low error with respect to the target concept under the target distribution, i.e. $\mathbf{err}_D(h, f) = \Pr_{x \sim D}[h(x) \neq f(x)] \leq \epsilon$. We say that the algorithm learns C in the random noise model, if it can, with high probability, produce a low error hypothesis for all $f \in C$ and all distributions D over X.*

The random noise model is well understood and we know a lot of efficient algorithms in this model for learning classes such as disjunctions, linear separators etc. (Angluin and Laird [1988], Blum et al. [1996]). Most algorithms for the random noise model are known to be statistical in nature. Such algorithms only interact with the data through statistical queries (SQ's). These are queries of the form $E[f(x)h(x)]$ for an arbitrary function h. See Feldman [2007] for a detailed discussion SQ learnability.

Agnostic learning

In this model of learning one makes no assumptions about the target function f. In particular, one has access to an example oracle $\mathsf{EX}(f, D)$ where f is an arbitrary function not necessarily in the class C. The goal is to compete with the error of the best function in the class C.

Definition 6.0.6 (**Agnostic Learning** Haussler [1992], Kearns et al. [1992])**.** *Let $\mathcal{D}$ be a distribution over X and C be a concept class over X, and f be an arbitrary function over X. An example oracle, $\mathsf{EX}(f, D)$, when queried, returns $(x, f(x))$ where x is drawn randomly from distribution D. The learning algorithm in this model has access to an example oracle, $\mathsf{EX}(f, D)$. Let f^* be the function in class C which has the least error w.r.t. f, i.e., $f^* = argmin_{h \in C} \mathbf{err}_D(h, f)$. The goal of the learning algorithm is to output a hypothesis, h, that has error as close to that of f^*. In particular, we say that the algorithm β agnostically learns C, if for all f and all distributions D over X, it produces a hypothesis h such that $\mathbf{err}_D(h, f) \leq \beta \mathbf{err}_D(f^*, f) + \epsilon$.*

Ideally, we want β to be equal to 1. For this case, distribution independent agnostic learning is a hard problem (Feldman [2009], Diakonikolas et al. [2011], Guruswami and

Raghavendra [2006], Feldman et al. [2012]) and very few positive results are known (Peleg [2007], Kalai et al. [2005], Klivans et al. [2009]). In fact, even for the simple class of disjunctions, one can show that distribution independent agnostic learning with $\beta = 1$ will lead to an efficient PAC learning algorithm for the class of DNF formulas (Kearns et al. [1992]).

Chapter 7

Agnostic learning of disjunctions

Learning disjunctions (or conjunctions) over $\{0,1\}^n$ in the PAC model is a well-studied and easy problem. The simple "list-and-cross-off" algorithm runs in linear time per example and requires only $O(n/\epsilon)$ examples to achieve error ϵ (ignoring the logarithmic dependence on the confidence term δ). The similarly efficient Winnow algorithm of Littlestone [1987] requires only $O((r \log n)/\epsilon)$ examples to learn well when the target function is a disjunction of size r.

However, when the data is only "mostly" consistent with a disjunction, the problem becomes substantially harder. In particular, we study the agnostic noise model which was described in Chapter 6. In this agnostic setting, our goal is to produce a hypothesis h whose error rate $\mathbf{err}_{\mathcal{D}}(h, f) = \Pr_{\mathcal{D}}(h(x) \neq f(x))$ satisfies $\mathbf{err}_{\mathcal{D}}(h) \leq \beta \cdot \mathrm{OPT}_{disj} + \epsilon$, where OPT_{disj} is the error rate of the *best* disjunction and β is as small as possible and f is the target function. For example, while the Winnow algorithm performs well as a function of the number of *attribute errors* of the best disjunction[1] (Littlestone [1991], Auer and Warmuth [1995]) , this can be a factor $O(r)$ worse than the number of *mistakes* of the best disjunction. Feldman [2009] showed that for any constant $\epsilon > 0$, determining whether the best disjunction for a given dataset S has error $\leq \epsilon$ or error $\geq \frac{1}{2} - \epsilon$ is NP-hard. Furthermore, Feldman et al. [2012] extended this hardness result to the problem of agnostic learning disjunctions by the hypothesis class of halfspaces. Thus, these results show that the problem of finding a disjunction (or linear separator) of error at most $\frac{1}{2} - \epsilon$ given that the error OPT_{disj} of the best disjunction is at most ϵ is computationally hard for any constant $\epsilon > 0$.

[1]The minimum number of variables that would need to be flipped in order to make the data perfectly consistent with a disjunction. This is essentially the same as its hinge loss.

Given these hardness results, it is natural to consider what kinds of learning guarantees *can* be achieved. If the error OPT_{disj} of the best disjunction is $O(1/n)$ then learning is essentially equivalent to the noise-free case. Peleg [2007] showed how to improve this to a bound of $\tilde{O}(1/\sqrt{n})$. In particular, on any given dataset S, his algorithm produces a disjunction of error rate on S at most $\tilde{O}(\sqrt{n} \cdot \mathrm{OPT}_{disj}(S))$.[2]

In this thesis, we improve on the result of Peleg [2007], achieving a bound of $O(n^{1/3+\alpha} \cdot \mathrm{OPT}_{disj}) + \epsilon$ for any constant $\alpha > 0$, though our algorithm is not a "proper" learner (does not produce a disjunction as its output) (Awasthi et al. [2010b]).[3] Note that our guarantee holds for any distribution over $\{0,1\}^n$.

7.1 Our Results

We design a learning algorithm whose error rate is an $O(n^{1/3+\alpha})$ approximation to that of the best disjunction, for any $\alpha > 0$. Formally, we prove the following theorem.

Theorem 7.1.1. *There exists an algorithm that for an arbitrary distribution D over $\{0,1\}^n$ and arbitrary target function $c^* : \{0,1\}^n \mapsto \{1,-1\}$, for every constant $\alpha > 0$ and every $\epsilon, \delta > 0$, runs in time polynomial in $1/\epsilon$, $\log(1/\delta)$, and n, uses $\mathrm{poly}(1/\epsilon, \log(1/\delta), n)$ random examples from D, and outputs a hypothesis h, such that with probability $> 1 - \delta$,*

$$\mathbf{err}_{\mathcal{D}}(h, f) \leq O(n^{\frac{1}{3}+\alpha}\mathrm{OPT}_{disj}) + \epsilon$$

where $\mathrm{OPT}_{disj} = \min_{g \in \mathrm{DISJUNCTIONS}} \mathbf{err}_{\mathcal{D}}(g, f)$.

The proof of Theorem 7.1.1 is based on finding a weak-learner under the assumption that $\mathsf{OPT} \equiv \mathrm{OPT}_{disj} = O(n^{-(1/3+\alpha)})$. In particular, we show:

Theorem 7.1.2. *There exists an algorithm with the following property. For every distribution D over $\{0,1\}^n$ and every target function f such that $\mathsf{OPT} < n^{-\frac{1}{3}-\alpha}$, for some constant $\alpha > 0$, for every $\delta > 0$, the algorithm runs in time $t(\delta, n)$, uses $m(\delta, n)$ random samples drawn from D and outputs a hypothesis h, such that with probability $> 1 - \delta$,*

$$\mathbf{err}_{\mathcal{D}}(h, f) \leq \frac{1}{2} - \gamma$$

[2]His results are for the "Red-Blue Set-Cover Problem" (Carr et al. [2000]) which is equivalent to the problem of approximating the best disjunction, except that positive examples must be classified correctly (i.e., the goal is to approximate the minimum number of mistakes on negatives subject to correctly classifying the positives). The extension to allowing for two-sided error, however, is immediate.

[3]This bound hides a low-order term of $(\log n)^{1/\alpha}$. Solving for equality yields $\alpha = \sqrt{\frac{\log\log n}{\log n}}$ and a bound of $O(n^{1/3+o(1)})$.

where t and m are polynomials in $n, 1/\delta$, and $\gamma = \Omega(n^{-2})$.

Our weak-learner can then feed into known boosting procedures which work with agnostic noise (Gavinsky [2003], Kalai et al. [2008], Feldman [2010]), to achieve the claimed guarantee in Theorem 7.1.1. The results in this chapter are based on work in Awasthi et al. [2010b].

7.2 Algorithm Intuition and Techniques

The high-level idea of the algorithm and proof for Theorem 7.1.2 is as follows. First, we can assume the target function is balanced (nearly equal probability mass on positive and negative examples) and that similarly no individual variable is noticeably correlated with the target, else weak-learning is immediate. So, for each variable i, the probability mass of positive examples with $x_i = 1$ is approximately equal to the probability mass of negative examples with $x_i = 1$. Let c^{opt} denote the (unknown) optimal disjunction, which we may assume is monotone by including negated variables as additional features. Let r denote the number of relevant variables; i.e., the number of variables in c^{opt}. Also, assume for this discussion that we know the value of $\mathsf{OPT} = \mathbf{err}_D(c^{opt}, f)$. Call an example x "good" if $f(x) = c^{opt}(x)$ and "bad" otherwise. Now, since the only negative examples than can have a relevant x_i set to 1 are the bad negatives, this means that for relevant variables i, $\Pr_{x\sim D}(x_i = 1|f(x) = -1) = O(\mathsf{OPT})$. Therefore, $\Pr_{x\sim D}(x_i = 1|f(x) = +1) = O(\mathsf{OPT})$ and so $\Pr_{x\sim D}(x_i = 1) = O(\mathsf{OPT})$ as well. This means that by estimating $\Pr_{x\sim D}(x_i = 1)$ for each variable i, we can remove all variables of density $\omega(\mathsf{OPT})$ from the system, knowing they are irrelevant.

At this point, we have nearly all the ingredients for the $\tilde{O}(1/\sqrt{n})$ bound of Peleg [2007]. In particular, since all variables have density $O(\mathsf{OPT})$, this means the average number of variables set to 1 per example is $O(\mathsf{OPT} \cdot n)$. Let S' be the set of examples whose density is at most twice the average (so $\Pr(S') \geq 1/2$); we now claim that if $\mathsf{OPT} = o(1/\sqrt{n})$, then either S' is unbalanced or else some variable x_i must have noticeable correlation with the target over examples in S'. In particular, since positive examples must have on average at least $1 - O(\mathsf{OPT})$ relevant variables set to 1, and the good negative examples have zero relevant variables set to 1, the only way for S' to be balanced and have no relevant variable with noticeable correlation is for the bad negative examples to on average have $\Omega(1/\mathsf{OPT})$ relevant variables set to 1. But this is not possible since all examples in S' have only $O(\mathsf{OPT} \cdot n)$ variables set to 1, and $1/\mathsf{OPT} \gg \mathsf{OPT} \cdot n$ for $\mathsf{OPT} = o(1/\sqrt{n})$. So, some hypothesis of the form: "if $x \notin S'$ then flip a fair coin, else predict x_i" must be a weak-learner.

In order to improve over the $\tilde{O}(1/\sqrt{n})$ bound of Peleg [2007], we do the following. Assume all variables have nearly the same density and all examples have nearly the same density as well. This is *not* without loss of generality (and the general case adds additional complications that must be addressed), but simplifies the picture for this high-level sketch. Now, if no individual variable or its complement is a weak predictor, by the above analysis it must be the case that the bad negative examples on average have a substantial number of variables set to 1 in the relevant region (essentially so that the total hinge-loss (attribute-errors) is $\Omega(m)$). Suppose now that one "guesses" such a bad negative example e and focuses on only those n' variables set to 1 by e. The disjunction c^{opt} restricted to this set may now make many mistakes on positive examples (the "substantial number of variables set to 1 in the relevant region" in e may still be a small fraction of the relevant region). On the other hand, because we have restricted to a relatively small number of variables n', the *average density* of examples as a function of n' has dropped significantly.[4] As a result, suppose we again discard all examples with a number of 1's in these n' variables substantially larger than the average. Then, on the remainder, the *hinge-loss* (attribute-errors) caused by the bad negative examples is now substantially reduced. This more than makes up for the additional error on positive examples. In particular, we show one can argue that for *some* bad negative example e, if one performs the above procedure, then with respect to the remaining subset of examples, some variable must be a weak predictor. In the end, the final hypothesis is defined by an example e, a threshold θ, and a variable i, and will be of the form "if $x \cdot e \notin [1, \theta]$ then flip a coin, else predict x_i." The algorithm then simply searches over all such triples. In the general case (when the variables and the examples do not all have the same density), this is preceded by a pre-processing step that groups variables and examples into a number of buckets and then runs the above algorithm on each bucket.

We now formally prove Theorem 7.1.2. We achieve this in two steps: first we show how to get a weak learner for the special case that the examples and variables are fairly homogeneous (all variables set to 1 roughly the same number of times, and all examples with roughly the same number of variables set to 1 (actually a somewhat weaker condition than this)). We then show how to reduce a general instance to this special case. In Section 7.2.3 we use existing boosting algorithms combined with this weak-learner to prove Theorem 7.1.1.

Our complete weak learning algorithm has two stages: a preprocessing step (which we present later in Section 7.2.2) that ensures that all variables are set to 1 roughly the

[4]E.g., given two *random* vectors with $n' = n^{2/3}$ 1's, their intersection would have expected size $(n')^{1/2}$. Of course, our dataset need not be uniform random examples of the given density, but the fact that all variables have the same density allows one to make a similar argument.

same number of times and that the bad and good examples have roughly the same number 1s, and a core algorithm (which we present first in Section 7.2.1) that operates on data of this form. One aspect of the preprocessing step is that in addition to partitioning examples into buckets, it may involve discarding some relevant variables, yielding a dataset in which only some $\tilde{m} \geq m/\mathrm{polylog}(n)$ positive examples satisfy c^{opt} over the variables remaining. Thus, our assumption in Section 7.2.1 is that while the dataset has the "homogeneity" properties desired and the fraction of bad negative examples is $\mathsf{OPT}(1+o(1))$, the fraction of bad *positive* examples may be as large as $1 - 1/\mathrm{polylog}(n)$. Nonetheless, this will still allow for weak learning.

7.2.1 $(B, \alpha, \tilde{m})$-Sparse Instances

As mentioned above, in this section we give a weak learning algorithm for a dataset that has certain "nice" homogeneity properties. We call such a dataset a $(B, \alpha, \tilde{m})$-*sparse instance*. Our weak learning algorithm will output a short hypothesis which will perform better than random guessing on the given datatset. By Occam's razor bound Kearns and Vazirani [1994], if the dataset is sufficiently large, such a hypothesis will also generalize and will perform better than random guessing over the entire distribution. We begin by describing what these properties are.

The first property is that there exists a positive integer B such that for each variable x_i, the number of positive examples in the instance with $x_i = 1$ is between $B/2$ and B, and the number of negative examples with $x_i = 1$ is between $\frac{B}{2}(1 - o(1))$ and $B(1 + o(1))$.

The first property implies that in this case the overall number of 1s in all examples is at most $2nB(1 + o(1))$, and therefore, an average example has no more than $\frac{nB(1+o(1))}{m}$ variables set to 1. If the bad negatives were typical examples, we would expect them to contain at most $\frac{nB}{m} \cdot m\mathsf{OPT}(1+o(1)) \leq nB \cdot \mathsf{OPT}(1+o(1))$ ones in total. While in general this may not necessarily be the case, we assume for this section that at least they are not *too* atypical on average. In particular, the second property we assume this instance satisfies is that the overall number of ones present in all the bad negatives is at most $n^{1+\alpha}B\mathsf{OPT}$.

Denote by $\tilde{m}$ the number of positive examples that c^{opt} classifies correctly. The third property is that $\tilde{m} \geq m/n^{o(\alpha)}$. If this dataset were our given training set then this would be redundant, as we already assume the stronger condition that the fraction of good positive examples is $1 - O(\mathsf{OPT})$.[5] However, $\tilde{m}$ will be of use in later sections, when we call this algorithm as a subroutine on instances defined by only a subset of all the variables. In other words, we show here that even if we allow c^{opt} to make more mistakes on the positive

[5]Indeed, if the original instance was sparse, we would have $\tilde{m} = m(1 - o(1))$.

examples (and in particular, to label almost all positives incorrectly!) yet make at most $m\mathsf{OPT}$ mistakes on the negatives, we are still able to weak-learn. As our analysis shows, the condition we require of $\tilde{m}$ is that the ratio $\frac{\tilde{m}}{m}$ dominates the ratio $\frac{\mathsf{OPT}}{n^{-1/3}}$. Furthermore, the ratio $\frac{\tilde{m}}{m}$ will play a role in the definition of γ, our advantage over a random guess.

An instance satisfying all the above three properties is called a $(B, \alpha, \tilde{m})$-*sparse* instance. Next, we show how to get a weak learner for such sparse instances. We first introduce the following definitions.

Definition 7.2.1. *Given an example e and a positive integer threshold θ, we define the* (e, θ)-restricted domain *to be the set of all examples whose intersection with e is strictly smaller than θ. That is, the set of examples x such that $x \cdot e < \theta$. For any hypothesis h, we define the* (e, θ)-restricted hypothesis *to be h over any example that belongs to the (e, θ) restricted domain, and "I don't know" (flipping a fair coin) over any other example. In particular, we consider the*

- (e, θ)*-restricted* $(+1)$*-hypothesis – predict* $+1$ *if the given example intersects e on less than θ variables.*
- (e, θ)*-restricted* (-1)*-hypothesis – predict* -1 *if the given example intersects e on less than θ variables.*
- (e, θ)*-restricted x_i-hypothesis – predict* $+1$ *if the given example intersects e on less than θ variables* and *has* $x_i = 1$.

We call these $n + 2$ restricted hypotheses the (e, θ)-restricted base hypotheses.

Our weak-learning algorithm enumerates over all pairs of (e, θ), where e is a negative example in our training set and θ is an integer between 1 and n. For every such pair, our algorithm checks whether any of the $n + 2$ restricted hypothesis is a $\Omega(\frac{\tilde{m}}{m} \cdot \frac{\mathsf{OPT}}{r})$-weak-learner (see Algorithm 1 below). Our next lemma proves that for $(B, \alpha, \tilde{m})$-sparse instances, this algorithm indeed finds a weak-learner. In fact, we show that for every negative example e, it suffices to consider a particular value of θ.

Lemma 7.2.2. *Suppose we are given a $(B, \alpha, \tilde{m})$-sparse instance, and that c^{opt} makes no more than a $n^{-(\frac{1}{3}+\alpha)}$ fraction of errors on the negative examples. Then there exists a bad negative example e and a threshold θ such that one of the (e, θ)-restricted base hypotheses mentioned in Definition 7.2.1 has error at most $1/2 - \gamma$ for $\gamma = \Omega(\frac{\tilde{m}}{m} \cdot \frac{\mathsf{OPT}}{r})$. Since we may assume $\mathsf{OPT} > 1/\sqrt{n}$, this implies $\gamma = \Omega(n^{-2})$. Thus Algorithm 1 outputs a hypothesis of error at most $\frac{1}{2} - \Omega(n^{-2})$.*

Algorithm 1 A weak learner for sparse instances.

Input: A $(B, \alpha, \tilde{m})$ sparse instance.

Step 1: For every negative example e in the set and every $\theta \in \{1, 2, \ldots, n\}$

Step 1a: Check if any of the (e, θ)-restricted hypotheses from Definition 7.2.1 is a weak learner with error at most $\frac{1}{2} - \Omega(n^{-2})$.

Step 1b: If Yes, then output the corresponding hypothesis and halt.

Step 2: If no restricted hypothesis is a weak learner, output failure.

Proof. Let m^+ and m^- be the number of positive and negative examples in this sparse instance, where we reserve m to refer to the size of the original dataset of which this sparse instance is a subset. As before, call examples "good" if they are classified correctly by c^{opt}, else call them "bad". We know $B = O(m\mathsf{OPT})$, because relevant variables have no more than $O(m\mathsf{OPT})$ occurrences of 1 over the negative examples. Since each good positive example has to have at least one relevant variable set to 1, it must also hold that $B = \Omega(\tilde{m}/r)$. It follows that $r\mathsf{OPT} = \Omega(\tilde{m}/m)$. We now show how to find a weak learner given a $(B, \alpha, \tilde{m})$-sparse instance, based on a bad negative example.

Consider any bad negative example e_i with t_i variables set to 1. If we sum the intersection (i.e. the dot-product) of e_i with each of the positive examples in the instance, we simply get the total number of ones in the positive examples over these t_i variables. As each variable is set to 1 between $B/2$ and B times, this sum is $B't_i$ for some $B' \in [B/2, B]$. Therefore, the expected intersection of e_i with a random positive example is $\frac{1}{m^+} \cdot t_i B'$. Set $\theta_i = \beta \cdot \frac{t_i B'}{m^+}$, where $\beta > 1$ will be chosen later suitably. Throw out any example which has more than θ_i intersection with e_i. Using Markov's inequality, we deduce that we retain at least $m^+(1 - \frac{1}{\beta})$ positive examples.

The key point of the above is that focusing on the examples that remain, none of them can contribute more than θ_i hinge-loss (attribute errors), restricting c^{opt} to the t_i variables set to 1 by e_i. On the other hand, it is possible that the number of *actual* errors over positives has increased substantially: perhaps too few of the remaining positive examples share relevant variables with e_i in order for any of the (e_i, θ_i) restricted hypotheses to be a weak learner. We now argue that this cannot happen simultaneously for all e_i.

Specifically, assume for contradiction that none of the (e_i, θ_i)-restricted base hypotheses yields a weak learner. Consider the total number of 1s contributed by the remaining negative examples over the relevant variables of e_i (the relevant variables that are set to 1 by e_i). As each bad negative contributes at most θ_i such ones, the overall contribution on the negative side is $\leq \theta_i \cdot m\mathsf{OPT}(1 + o(1)) = \beta \frac{t_i B'}{m^+} \cdot m\mathsf{OPT}(1 + o(1))$. Since none

of relevant variables set to 1 by e_i gives a weak learner, it holds that the number of 1s over the positive side of these relevant variables is no more than $2\beta\frac{m}{m^+} \cdot t_i B \cdot \mathsf{OPT}$ (see below, at the specification of the value of γ). So even if each occurrence of 1 comes from a unique positive example, we still have no more than $2\beta\frac{m}{m^+} \cdot t_i B \cdot \mathsf{OPT}$ positive examples from the (e_i, θ_i) restricted domain intersecting e_i over the relevant variables. Therefore, adding back in the positive examples *not* from the restricted domain, we have no more than $2\beta\frac{m}{m^+} \cdot t_i B \cdot \mathsf{OPT} + m^+/\beta$ positive examples that intersect e_i over the relevant variables.

Consider now a bipartite graph with the $\tilde{m}$ good positive examples on one side and the $m\mathsf{OPT}$ bad negative examples on the other side, with an edge between positive e_j and negative e_i if e_j intersects e_i over the relevant variables. Since each e_i has degree at most $2\beta\frac{m}{m^+} \cdot t_i B \cdot \mathsf{OPT} + m^+/\beta$, the total number of edges is at most $2\beta\frac{m}{m^+} B\mathsf{OPT} \sum_i t_i + m^+ \cdot m\mathsf{OPT}/\beta$, and therefore some good positive examples must have degree at most $\mathsf{OPT}[\frac{2\beta Bm}{\tilde{m}m^+} \sum_i t_i + \frac{m^+}{\beta} \cdot \frac{m}{\tilde{m}}]$. On the other hand, since we are given a $(B, \alpha, \tilde{m})$-sparse instance, we know that every good positive example intersects *at least* $\frac{B(1-o(1))}{2}$ negative examples, and moreover that $\sum_i t_i \leq n^{1+\alpha} B\mathsf{OPT}$. Putting this together we have:

$$B/2 \;\leq\; (1+o(1))\mathsf{OPT}\left[\frac{2\beta B^2 n^{1+\alpha}\mathsf{OPT}m}{\tilde{m}m^+} + \frac{m^+}{\beta} \cdot \frac{m}{\tilde{m}}\right].$$

Setting $\beta = \sqrt{\frac{(m^+)^2}{2B^2 n^{1+\alpha}\mathsf{OPT}}}$ to equalize the two terms in the sum above, we derive

$$B \leq \;\; 4\sqrt{2}(1+o(1))B \cdot \tfrac{m}{\tilde{m}} \cdot n^{(1+\alpha)/2}\mathsf{OPT}^{3/2}.$$

Thus we have $n^{1+\alpha} \cdot \frac{m^2}{\tilde{m}^2} \cdot \mathsf{OPT}^3 \geq \frac{1+o(1)}{32}$. Recall that $\tilde{m}/m \geq n^{-o(\alpha)}$, so we derive a contradiction, as for sufficiently large n it must hold that

$$\mathsf{OPT} \geq \left(\frac{1+o(1)}{32}\right)^{1/3} n^{-\frac{1+\alpha}{3}-o(\alpha)} > n^{-1/3-\alpha}.$$

In order to complete the proof, we need to verify that indeed $\beta > 1$. Recall $B = O(m\mathsf{OPT})$ and $m^+ \geq \tilde{m}$, so $m^+/m \geq n^{-o(\alpha)}$. Thus $\beta^2 = \Omega(\frac{1}{n^{1+\alpha+o(\alpha)}\mathsf{OPT}^3}) = \Omega(n^{2\alpha-o(\alpha)})$ by our assumption on OPT.

The last detail is to check what advantage do we get over a random guess. Our analysis shows that for some bad negative example e_i, the number of ones over the relevant variables on the positive side is at least $2\beta\frac{m}{m^+} \cdot t_i B \cdot \mathsf{OPT}$, whereas on the negative side, there can be at most $\beta\frac{m}{m^+} \cdot t_i B \cdot \mathsf{OPT}(1+o(1))$ ones. We deduce that at least one of the at most $\min(r, t_i)$ relevant variables set to 1 by e_i must give a gap of at

least $\frac{\beta \cdot t_i B m \cdot \mathsf{OPT}(1-o(1))}{m^+ \min(r,t_i)} > B \cdot \mathsf{OPT}(1-o(1))$ since $\beta > 1$. Finally, using the fact that $B = \Omega(\tilde{m}/r)$ we get a gap of $\Omega(\frac{\tilde{m}\mathsf{OPT}}{r})$ or equivalently an advantage of $\gamma = \Omega(\frac{\mathsf{OPT}}{r} \cdot \frac{\tilde{m}}{m})$. This advantage is trivially $\Omega(n^{-2(1+o(\alpha))})$, or, using the assumption $\mathsf{OPT} > 1/\sqrt{n}$ (for otherwise, we can apply Peleg's algorithm (Peleg [2007])), we get $\gamma = \Omega(n^{-\frac{3}{2}(1+o(\alpha))})$. □

7.2.2 General Instances

Section 7.2.1 dealt with nicely behaved (homogeneous) instances. In order to complete the proof of Theorem 7.1.2, we need to show how to reduce a general instance to such a $(B, \alpha, \tilde{m})$-sparse instance. What we show is a (simple) algorithm that partitions a given instance into sub-instances, based on the number of 1s of each example over certain variables (but without looking at the labels of the examples). It outputs a $\mathrm{polylog}(n)$-long list of sub-instances, each containing a noticeable fraction of the domain, and has the following guarantee: either some sub-instance has a trivial weak-learner (has a noticeably different number of positive versus negative examples or there is a variable with noticeable correlation), or some sub-instance is $(B, \alpha, \tilde{m})$-sparse. Formally, we prove this next lemma.

Lemma 7.2.3. *There exists a* $\mathrm{poly}((\log n)^{O(1/\alpha)}, n, m)$*-time algorithm, that gets as an input* $2m$ *labeled examples in* $\{0,1\}^n$*, and output a list of subsets, each containing* $m/\mathrm{polylog}(n)$ *examples, s.t. either some subset has a trivial weak-learner, or some subset is* $(B, \alpha, m/\mathrm{polylog}(n))$*-sparse.*

Combining the algorithm from Lemma 7.2.3 with the algorithm presented in Section 7.2.1, we get our weak-learning algorithm (see Algorithm 2). We first run the algorithm of Lemma 7.2.3, traverse all sub-instances, and check whether any has a trivial weak-learner. If not, we run the algorithm for $(B, \alpha, \tilde{m})$-sparse instances over each sub-instance. Obviously, given the one sub-instance which is sparse, we find a restricted hypothesis with $\tilde{\Omega}(n^{-2})$ advantage over a random guess.

Proof. We start by repeating the argument presented in the introduction (Section 7.1). For any relevant variable, no more than $m^-_{bad} \leq m \cdot \mathsf{OPT}(1+o(1))$ bad examples set it to 1. Therefore, as an initial step, we throw out any variable with more than this many occurrences over the negative examples, as it cannot possibly be a relevant variable. For convenience, redefine n to be the number of variables that remain. Next, we check each individual variable to determine if it itself is a weak predictor. If not, then this means each variable is set to 1 on approximately the same number of positive and negative examples.

Bucket all the variables according the number of times they are set to 1, where the j-bucket contains all the variables that are set to 1 any number of times in the range $[2^j, 2^{j+1})$. Since there are at most $\log n$ buckets, some bucket j must cover at least $\frac{m^+}{\log n}$ positive examples, in the sense that the disjunction over the *relevant* variables in this bucket agrees with at least this many good positives. So now, let $B' = 2^{j+1}$, let n' and r' be the total number of variables and the number of relevant variables in this bucket respextively. As we can ignore all examples that are identically 0 over the n' variables in this bucket, let m'^+ (resp. m'^-) be the number of positive (resp. negative) examples covered by the variables in this bucket. Our algorithm adds the remaining examples (over these n' variables) as one sub-instance to its list. Let the number of these examples be $2m'$. As before, if the number of positive examples and negative examples covered by these n' variables differ significantly, or if some variable is a weak learner (with respect to the set of examples left), then the algorithm halts. Observe that if this sub-instance is $(B', \alpha, m/\log(n))$-sparse, then we are done, no matter what other sub-instances the algorithm will add to its list.

Focusing on the remaining examples, every variable is set to 1 at most B' many times over the positive examples, so the total number of 1s, over the positive examples is $\leq n'B'$. If indeed the resulting instance is not $(B', \alpha, m/\log(n))$-sparse, then the total number of 1s over the bad negative examples is $\geq (n')^{1+\alpha}(B')\mathsf{OPT}$. So now, our algorithm throws out any example with more than $2n'B'/m'$ variables set to 1, and adds the remaining examples to the list of sub-instances. By Markov's inequality, we are guaranteed not to remove more than $1/2$ of the positive examples, so the sub-instance remaining is sufficiently large. As before, if the remaining subset of examples (over these n' variables) has a trivial weak-learner, we are done. Otherwise, the algorithm continues recursively over this sub-instance – re-buckets and then removes all examples with too many variables set to 1. Note, each time the algorithm buckets the variables, it needs to recurse over each bucket that covers at least a $1/\log(n)$ fraction of the positive examples. In the worst-case, all of the $\log(n)$ buckets cover these many positive examples, and therefore, the branching factor in each bucketing step is $\log(n)$.

We now show that the depth of the bucket-and-remove recurrence is no more than $O(1/\alpha)$. It is easy to see inductively that at the i-th step of the recursion, we retain a fraction of $m/(\log n)^i$ positive examples. Suppose that by the first i steps, no sub-instance is sparse and no weak-learner is found. Recall, if $r\mathsf{OPT} \ll 1$, we have an immediate weak-learner, so it must hold that in the i-th step, we still retain at least $n_i = 1/\mathsf{OPT}$ variables. Furthermore, as in the i-th step we did not have a sparse instance, it follows that the bad negative examples had more than $(n_i)^{1+\alpha}(B')\mathsf{OPT}$ ones before we threw out examples. Once we remove dense examples, they contain no more than $\frac{2(n_i)(B')}{m_i} \cdot m\mathsf{OPT}$ many ones.

Thus, the fraction of ones over the bad negatives that survive each removal step is no more than $n_i^{-\alpha} \cdot \frac{m}{m_i}$. As $1/\mathsf{OPT} > n^{1/3}$, this fraction is at most $n^{-\alpha/3}(\log n)^i < n^{-\alpha/6}$ (for the first $O(1/\alpha)$ iterations). Hence, after $6/\alpha$ iterations, some relevant variable must be a weak-learner.

To complete the proof, note that we take no more than $(\log n)^{6/\alpha}$ bucket-and-remove steps. Each such step requires $\mathrm{poly}(n, m)$ time for the bucketing, removal and checking for weak-learner. We conclude that the run-time of this algorithm is $\mathrm{poly}((\log n)^{1/\alpha}, n, m)$. □

Algorithm 2 A weak learner for general instances.

Input: A set of $2m$ training examples.

Step 1: If any individual variable or the constant hypotheses is a weak learner, output it and halt.

Step 2: Remove any variable which has more than $2m\mathsf{OPT}$ 1's over the negative examples.

Step 3: Bucket the remaining variables such that bucket j contains variables with density in $[2^j, 2^{j+1})$.

Step 4: For every bucket which covers at least a $\log n$ fraction of the positive examples

Step 4a: Run the algorithm for sparse instances on this bucket. If a weak learner is obtained, output it and halt.

Step 4b: Let B' be the density (2^{j+1}) in this bucket, n' be the number of variables in the bucket and $2m'$ be the total number of examples with respect to this bucket (ignoring the ones which are identically zero over the n' variables). Remove all the examples which have more than $2n'B'/m'$ 1's over this bucket. Repeat steps 1-4 on this new instance.

7.2.3 Strong Learning

Given Theorem 7.1.2, we now prove the main theorem (Theorem 7.1.1) by plugging the weak-learner into an off-the-shelf boosting algorithm for the agnostic case. We use the `ABoostDI` algorithm from Feldman [2010], which converts any algorithm satisfying Theorem 7.1.2 into one satisfying Theorem 7.1.1. The result in Feldman [2010] gives a boosting technique for (η, γ)-weak learners. In our context an (η, γ)-weak learner is an algorithm which with respect to to any distribution $\mathcal{D}$, with high probability, produces a hypothesis of error $\leq \frac{1}{2} - \gamma$, whenever $\mathrm{OPT}_{disj} \leq \frac{1}{2} - \eta$.

Theorem 7.2.4 (Feldman [2010], Theorem 3.5)**.** *There exists an algorithm* `ABoostDI` *that, given a (η, γ)-weak learner, for every distribution D and $\epsilon > 0$, produces, with high probability, a hypothesis h such that $err_D(h) \leq \frac{\mathrm{OPT}_{disj}}{1-2\eta} + \epsilon$. Furthermore, the running time of the algorithm is $T \cdot poly(\frac{1}{\gamma}, \frac{1}{\epsilon})$, where T is the running time of the weak learner.*

As an immediate corollary, we set $\eta = \frac{1}{2} - \frac{1}{2} \cdot n^{-1/3-\alpha}$ and obtain an hypothesis h such that $\mathbf{err}_D(h) \leq 2n^{1/3+\alpha}\mathsf{OPT} + \epsilon$. This concludes the proof of Theorem 7.1.1. We note that as an alternative to `ABoostDI`, one can also use the boosting algorithm of Kalai et al. [2008], followed by another boosting algorithm of Gavinsky [2003], to get the result in Theorem 7.1.1.

Chapter 8

Learning using Local Membership Queries

As mentioned in Chapter 6, the membership query model (PAC+MQ) is a powerful extension of the basic PAC model and we now have efficient PAC+MQ algorithms for many classes which seem out of reach of current techniques in the PAC model. Most notably are the works of Kushilevitz and Mansour [1993], Bshouty [1993] on learning decision trees and the celebrated result of Jackson [1997] on learning DNF formulas under the uniform distribution. Despite these and several other interesting theoretical results, the membership query model has not been received enthusiastically by machine learning practitioners. Of course, there is the obvious difficulty of getting labelers to perform their task while the learning algorithm is being executed. But another, and probably more significant, reason for this disparity is that quite often, the queries made by these algorithms are for labels of points that do not look like typical points sampled from the underlying distribution. This was observed by Lang and Baum [1992], where experiments on handwritten characters and digits revealed that the query points generated by the algorithms often had no structure and looked meaningless to the human eye. This can cause problems for the learning algorithm as it may receive noisy labels for such query points.

Motivated by the above observations, in this thesis we propose a model of membership queries where the learning algorithm is restricted to query labels of points that "look" like points drawn from the distribution. We focus our attention to the case when the instance space is the boolean cube, i.e. $X = \{-1, 1\}^n$, or $X = \{0, 1\}^n$. However, similar models could be defined in the case when X is some subset of $\mathbb{R}^n$. Suppose x is a *natural* example, *i.e.* one that was received as part of the training dataset (through the oracle $\mathsf{EX}(f, D)$). We restrict the learning algorithm to make queries x', where x and x' are close in Hamming

distance. More precisely, we say that a membership query x' is r-local with respect to a point x, if the Hamming distance, $|x - x'|_H$, is at most r.

One can imagine settings where these queries could be realistic, yet powerful. Suppose you want to learn a hypothesis that predicts a particular medical diagnosis using patient records. It could be helpful if the learning algorithm could generate a new medical record and query its label. However, if the learning algorithm is entirely unconstrained, it might come up with a record that looks gibberish to any doctor. On the other hand, if the query chosen by the learning algorithm is obtained by changing an existing record in a few locations (local query), it is more likely that a doctor may be able to make sense of such a record. In fact, this might be a powerful way for the learning algorithm to identify the most important features of the record.

It is interesting to study what power these local membership queries add to the learning setting. At the two extremes, are the PAC model (with 0-local queries), and MQ-model (with n-local queries). It can be easily observed that using only 1-local queries, the class of parities can be learned in polynomial time even in the presence random classification noise. This problem is known to be notoriously difficult in the PAC learning setting (Blum et al. [2003]). At the same time, most PAC+MQ algorithms we are aware of, such as the algorithms for learning decision trees (Bshouty [1993]) and for learning DNF formulas (Jackson [1997]), rely crucially on using MQs in a strongly non-local way. Also, it is easy to show that in a formal sense, allowing a learner to make 1-local queries gives it strictly more power than in the PAC setting. In fact, essentially the same argument can be used to show that $r + 1$-local queries are more powerful than r-local queries. These separation results can be easily proved under standard cryptographic assumptions, and are presented in Section 8.6.

Our results are for learning on *log-Lipschitz* distributions over the boolean cube, which we denote by $\{-1, 1\}^n$ (or sometimes by $\{0, 1\}^n$). We say that a distribution, D, over the boolean cube, $X = \{b_0, b_1\}^n$ is α-*log-Lipschitz* if the logarithm of the density function is $\log(\alpha)$-Lipschitz with respect to the Hamming distance. A straightforward implication of a distribution being α-log-Lipschitz is that for any two points x and x' which differ in only one bit, $D(x)/D(x') \leq \alpha$. Intuitively, this means that points that are close to each other cannot have vastly different frequencies. Frequency of a point reflects (and sometimes defines) its "naturalness" and so, in a sense, our assumption on the distribution is the same as the assumption underlying the use of local queries. The notion of log-Lipschitzness is a natural one and its variants have been studied before in different contexts (Feldman and Schulman [2012], Koltun and Papadimitriou [2007]). Furthermore, log-Lipschitz distributions contain a wide variety of popularly studied distributions as special cases. For example, the uniform distribution is *log-Lipschitz* with $\alpha = 1$. For constant α, log-Lipschitz

distributions have the property that changing $O(\log(n))$ bits can change the weight of a point by at most a polynomial factor. Such distributions include product distributions when the mean of each bit is some constant bounded away from ± 1 (or $0, 1$). Convex combinations of α-log-Lipschitz distributions are also α-log-Lipschitz. They also include smooth distributions which have previously been studied for designing learning algorithms (Kalai et al. [2009b]).

Our Results: We give several learning algorithms for general log-Lipschitz distributions and for the special case of product/uniform distributions. Our main result for the log-Lipschitz distributions is that sparse[1] polynomials are efficiently learnable with membership queries that are logarithmically local.

Theorem 8.0.5. *The class of t-sparse polynomials (with real coefficients) over $\{0,1\}^n$ is efficiently learnable under the class of α-log-Lipschitz distributions, for any constant α, by a learning algorithm that only uses $O(\log(n) + \log(t))$-local membership queries.*

An important subclass of sparse polynomials is $O(\log n)$-depth decision trees. Richer concept classes are also included in the class of sparse polynomials. This includes the class of disjoint $\log(n)$-DNF expressions and $\log$-depth decision trees, where each node is a monomial (rather than a variable). A special case of such decision trees is $O(\log(n))$-term DNF expressions.

When the polynomials represent boolean functions this algorithm can easily be made to work in the presence of *persistent* random classification noise, as described in Section 8.7.3.

For the special case of constant bounded product distributions we show that polynomial-size decision trees are efficiently learnable.

Theorem 8.0.6. *Let $\mathcal{P}$ be the class of product distributions over $X = \{-1,1\}^n$, such that the mean of each bit is bounded away from -1 and 1 by a constant. Then, the class of polynomial-size decision trees is learnable with respect to the class of distributions $\mathcal{P}$, by an algorithm that uses only $O(\log(n))$-local membership queries.*

We also consider polynomial size DNF which are known to be learnable in PAC+MQ.

Theorem 8.0.7. *The class of polynomial sized* DNF *formulas is learnable under the uniform distribution using $O(\log(n))$-local queries in time $n^{O(\log\log n)}$.*

[1] Sparsity refers to the number of non-zero coefficients.

The results in this chapter are based on work in Awasthi et al. [2013b].

Techniques: All our results are based on learning polynomials. It is well known that $\log$-depth decision trees can be expressed as sparse polynomials of degree $O(\log(n))$.

Our results on learning sparse polynomials (Section 8.2) under log-Lipschitz distributions rely on being able to identify all the important monomials (those with non-zero coefficient) of low-degree, using $O(\log(n))$-local queries. We identify a *set* of monomials, the size of which is bounded by a polynomial in the required parameters, which includes all the important monomials. The crucial idea is that using $O(\log(n))$-local queries, we can identify given a subset of variables $S \subseteq [n]$, whether the function on the remaining variables (those in $[n] \setminus S$), is zero or not. We use the fact that the distribution is log-Lipschitz to show that performing L_2 regression over the set of monomials will give a good hypothesis.

For uniform (or product) distributions (Sections 8.4; 8.3 and 8.7.2), we can make use of Fourier techniques and numerous algorithms based on them. A natural approach to the problem is to try to adapt the famous algorithm of Kushilevitz and Mansour [1993] for learning decision trees (the KM algorithm) to the use of local MQs. The KM algorithm relies on a procedure that isolates all Fourier coefficients that share a common prefix and computes the sum of their squares. Isolation of coefficients that share a prefix of length k requires k-local MQs and therefore we cannot use the KM algorithm directly. Instead we isolate Fourier coefficients that contain a certain set of variables and we grow these sets variable-by-variable as long as the sum of squares of coefficients in them is large enough. Using k-local MQs it is possible to grow sets up to size k. More importantly, the use of prefixes in the KM algorithm ensures that Fourier coefficients are split into disjoint collections and therefore not too many collections will be relevant. In our case the collections of coefficients are not disjoint and so to prove that our algorithm will not take superpolynomial time we rely on strong concentration properties of the Fourier transform of decision trees.

For the case of DNF formulas, we use the result of Feldman [2012], Kalai et al. [2009b] which shows that one can learn a DNF formula given its heavy logarithmic-degree Fourier coefficients. To recover those coefficients we use the same algorithm as in the decision tree case. However in this case a more delicate analysis is required to obtain even the $n^{O(\log\log n)}$ running time bound we give in Theorem 8.0.7. We rely on a concentration bound by Mansour [1992] that shows that the total weight of Fourier coefficients of degree d decays exponentially as d grows. We remark that Mansour also gives a PAC+MQ algorithm for learning DNF running in time $n^{O(\log\log n)}$ but aside from the use of the concentration bound our algorithm and analysis are different (the dependence on the error ϵ in

our algorithm is also substantially better).

All known efficient algorithms for learning DNF under the uniform distribution rely on agnostic learning of parities using the KM algorithm (or a related algorithm of Levin [1993]) Blum et al. [1994], Jackson [1997]. In the agnostic case one cannot rely on the concentration properties crucial for our analysis and therefore it is unclear whether poly-size DNF formulas can be learned efficiently from logarithmically-local MQs. As some evidence of the hardness of this problem, in Section 8.5 we show that for a constant k, k-local queries do not help in agnostic learning under the uniform distribution.

One point to note is that under α-log-Lipschitz distributions for a constant α, the main difficulty is designing algorithms which are faster than time $n^{O(\log(n))}$. Designing an $n^{O(\log(n))}$ time algorithm is trivial for decision trees and DNF formulas. In fact, one does not even require local-membership queries to do this. This follows from the observation that *agnostic* learning of $O(\log(n))$-size parities is easy in $n^{O(\log n)}$ time.

Related work: Models that address the problems that arise when membership queries are answered by humans have been studied before. The work of Blum et al. [1998] proposed a noise model wherein membership queries made on points lying in the low probability region of the distribution are unreliable. For this model the authors design algorithms for learning an intersection of two halfspaces in $\mathbb{R}^n$ and also for learning a very special subclass of monotone DNF formulas. Our result on learning sparse polynomials can be compared with that of Schapire and Sellie [1996], who provided an algorithm to learn sparse polynomials over GF(2) under arbitrary distributions in Angluin's *exact* learning model. However, their algorithm is required to make membership queries that are not local. Bshouty [1993] gave an algorithm for learning decision trees using membership queries. In both these cases, it seems unlikely that the algorithms can be modified to use only local membership queries, even for the class of locally smooth distributions.

There has been considerable work investigating learnability beyond the PAC framework. We consider our results in this body of work. Many of these models are motivated by theoretical as well as real-world interest. On the one hand, it is interesting to study the minimum extra power one needs to add to the PAC setting, to make the class of polynomial-size decision trees or DNF formulas efficiently learnable. The work of Aldous and Vazirani [1990] studies models of learning where the examples are generated according to a Markov process. An interesting special case of such models is when examples are generated by a random walk on $\{-1, 1\}^n$. For this model Bshouty et al. [2005] give an algorithm for learning DNF formulas (see also Jackson and Wimmer [2009] for more recent developments). One could simulate random walks of length up to $O(\log(n))$ using local membership queries, but adapting their DNF learning algorithm to our model runs into

the same issues as adapting the KM algorithm. The work of Kalai et al. [2009b] provided polynomial time algorithms for learning decision trees and DNF formulas in a framework where the learner gets to see examples from a *smoothed* distribution.[2] Their model was inspired by the celebrated smoothed analysis framework of Spielman and Teng [2004]. On the other hand, other models have been proposed to capture plausible settings when the learner may indeed have more power than in the PAC-setting. These situations arise for example in scientific studies where the learner may have more than just *black-box* access to the function. Two recent examples in this line of work are the learning using injection queries of Angluin et al. [2006], and learning using restriction access of Dvir et al. [2012].

8.1 Notation and Preliminaries

Notation: In this chapter we will assume that the instance space X is the Boolean hypercube. In Sections 8.4 and 8.3 we will use $X = \{-1,1\}^n$, as we apply Fourier techniques. In Section 8.2, we will use $X = \{0,1\}^n$ (the class of sparse polynomials over $\{0,1\}^n$ is different from sparse polynomials over $\{-1,1\}^n$). For real valued functions, $h : X \to \mathbb{R}$, we use squared loss as the error measure instead of the $0/1$ loss, *i.e.* $\mathbf{err}_D(f,h) = \mathbb{E}_{x\sim D}[(f(x)-h(x))^2]$.

For some bit vector x (where bits may be $\{0,1\}$ or $\{-1,1\}$), and any subset $S \subseteq [n]$, x_S denotes the bits of x corresponding to the variables, $i \in S$. The set $\bar{S}$ denotes the set $[n] \setminus S$. For two disjoint sets, S, T, $x_S x_T$ denote the variables corresponding to the set $S \cup T$. In particular, $x_S x_{\bar{S}} = x$.

If D is a distribution over X, for a subset S, D_S denotes the marginal distribution over variables in the set S. Let b_S denote a function, $b_S : S \to \{b_0, b_1\}$, (where $\{b_0, b_1\} = \{0,1\}$ or $\{b_0, b_1\} = \{-1,1\}$). Then, $x_S = b_S$, denotes that for each $i \in S, x_i = b_S(i)$, thus the variables in the set S are set to the values defined by the function b_S. Let $\pi : X \to \{0,1\}$ denote some property (e.g. $\pi(x) = 1$, if $x_S = b_S$ and $\pi(x) = 0$ otherwise). The distribution $(D|\pi)$, denotes the conditional distribution, given that $\pi(x) = 1$, *i.e.* the property holds.

Local Membership Queries: For any point x, we say that a query x' is r-local with respect to x if the Hamming distance, $|x - x'|_H$ is at most r. In our model, we only allow

[2]The notion of *smoothness* in the work of Kalai et al. is not related to our notion of log-Lipschitzness. They consider product distributions where each bit has mean that is chosen randomly from a range bounded away from ± 1 by a constant.

algorithms to make queries that are r-local with respect to some example that it received by querying $\mathsf{EX}(f, D)$, an oracle that returns a random example from D labeled according to f. We think of examples coming through $\mathsf{EX}(f, D)$ as *natural* examples. Thus, the learning algorithm draws a set of natural examples from $\mathsf{EX}(f, D)$ and then makes queries that are close to some example from this set. The queries are made to the membership oracle, $\mathsf{MQ}(f)$, which on receiving a query x, returns $f(x)$. Formally, we define learning using r-local membership queries as follows:

Definition 8.1.1 (PAC+r-local MQ Learning)**.** *Let X be the instance space, C a concept class over X, and $\mathcal{D}$ a class of distributions over X. We say that C is PAC-learnable using r-local membership queries with respect to distribution class, $\mathcal{D}$, if there exist a learning algorithm, $\mathcal{L}$, such that for every $\epsilon > 0$, $\delta > 0$, for every distribution $D \in \mathcal{D}$ and every target concept $f \in C$, the following hold:*

1. *$\mathcal{L}$ draws a sample, $\mathcal{S}$, of size $m = \mathrm{poly}(n, 1/\delta, 1/\epsilon)$ using example oracle, $\mathsf{EX}(f, D)$*
2. *Each query, x', made by $\mathcal{L}$ to the membership query oracle, $\mathsf{MQ}(f)$, is r-local with respect to some example, $x \in \mathcal{S}$*
3. *$\mathcal{L}$ outputs a hypothesis, h, that satisfies with probability at least $1 - \delta$, $\mathbf{err}_D(h, f) \leq \epsilon$*
4. *The running time of $\mathcal{L}$ (hence also the number of oracle accesses) is polynomial in n, $1/\epsilon$, $1/\delta$ and the output hypothesis, h, is polynomially evaluable.*

Log-Lipschitz Distributions: Since we want to talk about log-Lipschitz distributions over $\{-1, 1\}^n$ and $\{0, 1\}^n$ both, we consider $X = \{b_0, b_1\}^n$ and state the properties of interest in general terms. We say that a distribution, D, over $X = \{b_0, b_1\}^n$ is α-log-Lipschitz, for $\alpha \geq 1$, if for every pair $x, x' \in X$, with Hamming distance, $|x - x'|_H = 1$, it holds that $|\log(D(x)) - \log(D(x'))| \leq \log(\alpha)$.

We will repeatedly use the following useful properties of α-log-Lipschitz distributions. The proof of these are easy and hence are omitted.

Fact 8.1.2. *Let D be an α-log-Lipschitz distribution over $X = \{b_0, b_1\}^n$. Then the following are true:*

1. *For $b \in \{b_0, b_1\}$, $\frac{1}{1+\alpha} \leq \Pr_D[x_i = b] \leq \frac{\alpha}{1+\alpha}$.*
2. *For any subset, $S \subseteq [n]$, the marginal distribution, $D_{\bar{S}}$ is α-log-Lipschitz.*

3. *For any subset $S \subseteq [n]$, and for any property, π_S, that depends only on variables x_S (e.g. $x_S = b_S$), the marginal (with respect of $\bar{S}$) of the conditional distribution, $(D|\pi_S)_{\bar{S}}$ is α-log-Lipschitz.*

4. *(As a corollary of the above three)* $\left(\frac{1}{1+\alpha}\right)^{|S|} \leq \Pr_D[x_S = b_S] \leq \left(\frac{\alpha}{1+\alpha}\right)^{|S|}$.

8.2 Learning Sparse Polynomials under Log-Lipschitz Distributions

In this section, we consider the problem of learning t-sparse polynomials with coefficients over $\mathbb{R}$ (or $\mathbb{Q}$), when the domain is restricted to $\{0,1\}^n$. In this case, we may as well assume that the polynomials are multi-linear. We assume that the absolute values of the coefficients are bounded by B, and hence the polynomials take values in $[-tB, tB]$, on the domain $\{0,1\}^n$. For a subset $S \subseteq [n]$, let $\xi_S(x) = \prod_{i \in S} x_i$, thus $\xi_S(x)$ is the monomial corresponding to the variables in the set S. Note that any t-sparse multi-linear polynomial can be represented as,

$$f(x) = \sum_S c_S \xi_S(x),$$

where $c_S \in \mathbb{R}$, $|\{S \mid c_S \neq 0\}| \leq t$, and $|c_S| \leq B$ for all S. Let $\mathbb{R}^n_{t,B}[X]$ denote the class of multi-linear polynomials over n variables with coefficients in $\mathbb{R}$, where at most t coefficients are non-zero and all coefficients have magnitude at most B.

We assume that we have an infinite precision computation model for reals.[3] Also, since the polynomials may take on arbitrary real values, we use squared loss as the notion of error. For a distribution, D over $\{0,1\}^n$, the squared loss between polynomials, f and h, is $\mathbb{E}_{x \sim D}[(f(x) - h(x))^2]$. Our main result is:

Theorem 8.2.1. *The class $\mathbb{R}^n_{t,B}[X]$, is learnable with respect the class of α-log-Lipschitz distributions over $\{0,1\}^n$, using $O(\log(n/\epsilon)+\log(t/\epsilon))$-local* MQ*s and in time $poly((ntB/\epsilon)^\alpha, \log(n/\delta))$. The output hypothesis is a multi-linear polynomial, h, such that, with probability $(1-\delta)$, $\mathbb{E}_{x \sim D}[(h(x) - f(x))^2] \leq \epsilon$.*

Recall that for a subset, S, x_S denotes the variables that are in S; and that $\bar{S}$ denotes the set $[n] \setminus S$. Let $f_S(x_{\bar{S}})$ denote the multi-linear polynomial defined only on variables in

[3]The case when we have bounded precision can be handled easily since our algorithms run in time polynomial in B, but is more cumbersome.

Algorithm: LEARNING t-SPARSE POLYNOMIALS

inputs: d, θ, oracles $\mathsf{EX}(f, D)$, (local)$\mathsf{MQ}(f)$

1. **let** $\mathcal{S} = \emptyset$
2. **repeat** (while some new set is added to $\mathcal{S}$)
   (a) For every $S' \in \mathcal{S}$, $|S'| \leq d-1$ and for every $j \in [n] \setminus S'$
      i. **let** $S = S' \cup \{j\}$
      ii. **if** $\Pr_{D_{\bar{S}}}[f_S(x_{\bar{S}}) \neq 0] \geq \theta$, **then** $\mathcal{S} = \mathcal{S} \cup \{S\}$
3. Perform regression to identify a polynomial $h = \sum_S h[S]\xi_S(x)$, that minimizes $\mathbb{E}[(f(x) - h(x))^2]$, subject to:
   (a) $h[S] = 0$ for $S \notin \mathcal{S}$.
   (b) $\sum_S |h[S]| \leq tB$

output $h(x)$

Figure 8.1: Algorithm: Learning t-Sparse Polynomials

$x_{\bar{S}}$,

$$f_S(x_{\bar{S}}) = \sum_{T \subseteq \bar{S}} c_{S \cup T} \xi_T(x_{\bar{S}})$$

The learning algorithm is shown in Figure 8.1. Here, we describe the high-level idea of the proof of Theorem 8.2.1. The details of the argument are provided in Section 8.7.1. Algorithm 8.1 outputs a hypothesis that approximates the polynomial f.

Truncation: First, we show that there are low-degree polynomials that approximate the multi-linear polynomial, f, up to arbitrary (inverse polynomial) accuracy. These polynomials are the truncations of f itself. Let f^d denote the multi-linear polynomial obtained from f by discarding all the terms of degree at least $d+1$. Note that f^d is multi-linear and t-sparse, and has coefficients of magnitude at most B. Thus,

$$f^d(x) = \sum_{\substack{S \subseteq [n] \\ |S| \leq d}} c_S \xi_S(x)$$

Now, observe that because D is locally α-smooth, the probability that $\xi_S(x) = 1$ is at most $(\alpha/(1+\alpha))^{|S|}$ (see Fact 8.1.2). Thus, the probability that at least one term of degree $\geq d+1$ in f is non-zero, is at most $t(\alpha/(1+\alpha))^d$ by a union bound. Thus, $\Pr_{x\sim D}[f(x) \neq f^d(x)] \leq t(\alpha/(1+\alpha))^d$. Also, since $|f(x)| \leq tB$ and $|f^d(x)| \leq tB$, this implies that $\mathbb{E}_{x\sim D}[(f(x) - f^d(x))^2] \leq 4t^3B^2(\alpha/(1+\alpha))^d$. By choosing d appropriately, when α is a constant this quantity can be made arbitrarily (inverse polynomial) small.

Step 3 of the Algorithm (see Fig. 8.1) identifies all the important coefficients of the polynomial, f. Suppose, we could guarantee that the set, $\mathcal{S}$, contains all coefficients S, such that $c_S \neq 0$ and $|S| \leq d$, *i.e.* all non-zero coefficients of f^d are identified. This guarantees that the regression in step 4 will give a good approximation to f, since the error of the hypothesis obtained by regression has to be smaller than $\mathbb{E}_{x\sim D}[(f(x) - f^d(x))^2]$. (The generalization guarantees are fairly standard and are described in the long version.)

Identifying Important Monomials: In order to test whether or not a monomial, S, is important, the algorithm checks whether $\Pr_{D_{\bar{S}}}[f_S(x_{\bar{S}}) \neq 0] \geq \theta$. Here, $D_{\bar{S}}$ is the marginal distribution over the variables $x_{\bar{S}}$. We assume that this test can be performed perfectly accurately. (The analysis using samples is standard by applying appropriate Chernoff-Hoeffding bounds.)

In Lemma 8.7.2, we show that if the polynomial $f_S(x_{\bar{S}})$ has a non-zero coefficient of degree at most $d-|S|$, then the probability that $f_S(x_{\bar{S}}) \neq 0$ is at least $(1/(1+\alpha))^{d+\log(t)}$. In Lemma 8.7.3, we show that if $f_S(x_{\bar{S}})$ has no non-zero coefficient of degree less than $d' = O((d+\ln(t))\ln(1+\alpha))$, then $f_S(x_{\bar{S}}) \neq 0$ with probability at most $0.5(1/(1+\alpha))^{d+\log(t)}$. Thus, we will never add any subset S, unless there is some co-efficient T in f of size at most $d' = O((d+\ln(t))\ln(1+\alpha))$ and $S \subseteq T$. However, the number of such T is at most t, and each such set can have at most $2^{d'}$ subsets. This bounds the total number of subsets the algorithm may add to $\mathcal{S}$, and hence, also the running time of the algorithm (to polynomial in the required parameters).

Note that sampling, $x_{\bar{S}}$ according to $D_{\bar{S}}$ is trivial, just draw random example from $\mathsf{EX}(f, D)$ and ignore the variables x_S. Let U_S denote the uniform distribution over variables in x_S. Then we have,

$$f_S(x_{\bar{S}}) = \mathbb{E}_{x_S\sim U_S}[2^{|S|}\prod_{i\in S}(2x_i - 1)f(x)] \tag{8.1}$$

The variables in $x_{\bar{S}}$ are fixed, the expectation is only taken over the uniform distribution over variables in x_S. Notice that for any $i \in S$, since $x_i \in \{0,1\}$, $\mathbb{E}_{x_S}[(2x_i - 1)] = 0$ and $\mathbb{E}_{x_S}[(2x_i - 1)x_i] = 1/2$. Thus, in the RHS of (8.1), if $S \not\subseteq T$, $\mathbb{E}_{x_S}[2^{|S|}\prod_{i\in S}(2x_i - 1)\xi_T(x)] = 0$, and if $S \subseteq T$, $\mathbb{E}_{x_S}[2^{|S|}\prod_{i\in S}(2x_i - 1)\xi_T(x)] = \xi_{T\setminus S}(x_{\bar{S}})$. Thus, the relation in (8.1) is true. Also, this means that if the example, x, is received by querying the oracle,

$\mathsf{EX}(f, D)$, $f_S(x_{\bar{S}})$ can be obtained by making $O(|S|)$-local membership queries to the oracle $\mathsf{MQ}(f)$.

The complete details of the proof appear in Section 8.7.1.

8.3 Learning Decision Trees under the Uniform Distribution

In this section, we present an algorithm for learning t-leaf decision trees (of arbitrary depth) under the uniform distribution. Although, the uniform distribution is a special case of product distributions considered in Section 8.7.2, the exposition is simpler and conveys the high-level ideas better.

We use standard results from Fourier analysis; Kushilevitz and Mansour [1993] proved the following useful properties of the Fourier spectrum of decision trees. Let f be a function that is represented by a t-leaf decision tree, then:

1. For any set $S \subseteq [n]$, $|\hat{f}(S)| \leq t/2^{|S|}$.
2. $L_1(f) = \sum_{S \subseteq [n]} |\hat{f}(S)| \leq t$.

Using the above relations, we can immediately prove the following useful (and well-known) fact.

Fact 8.3.1. *Suppose f is boolean function that is represented by t-leaf decision tree. Then, for any $\tau > 0$, $\sum_{S,|S| \geq \log(t^2/\tau)} \hat{f}(S)^2 \leq \tau$*

Proof. Consider,

$$\begin{aligned}\sum_{S,|S| \geq \log(t^2/\tau)} \hat{f}(S)^2 &\leq \max_{S,|S| \geq \log(t^2/\tau)} |\hat{f}(S)| \cdot \left(\sum_{T,|T| \geq \log(t^2/\tau)} |\hat{f}(S)| \right) \\ &\leq t \cdot (\tau/t^2) \cdot L_1(f) \leq \tau\end{aligned}$$

□

The algorithm in Figure 8.2 learns t-leaf decision trees under the uniform distribution. For simplicity of presentation, we assume that the expectations used in the algorithm and also the Fourier coefficients can be computed *exactly*. It is easy to see that using

Algorithm: LEARNING DECISION TREES

inputs: d, θ, oracles $\mathsf{EX}(f, U)$, (local)-$\mathsf{MQ}(f)$

1. **let** $\mathcal{S} = \{\emptyset\}$
2. **for** $i = 1, \ldots, d$
   (a) **for** every $S' \in \mathcal{S}$, $|S'| = i - 1$ and for every $j \in [n] \setminus S'$
      i. **let** $S = S' \cup \{j\}$
      ii. (L_2 Test) **if** $\mathbb{E}_{x \sim U}[f_S(x)^2] > \theta^2$, **then** $\mathcal{S} = \mathcal{S} \cup \{S\}$
3. **let** $h(x) = \sum_{S \in \mathcal{S}} \hat{f}(S)\chi_S(x)$

output: $\mathrm{sign}(h(x))$

Figure 8.2: Algorithm: Learning Decision Trees under the Uniform Distribution

standard applications of Chernoff-Hoeffding bounds, the guarantees of the algorithm hold even when the expectations and values of the Fourier coefficients can only be computed approximately. The main step in Algorithm 8.2 that requires some explanation is how to compute the quantity $\mathbb{E}_{x \sim U}[f_S(x)^2]$ to check if it is greater than θ^2. We refer to this as the L_2 Test.

L_2 **Test**: Let $x \in \{-1, 1\}^n$, and recall that for $S \subseteq [n]$, $f_S(x) = \sum_{T \supseteq S} \hat{f}(T)\chi_{T \setminus S}(x)$, and that this can be computed by using the fact that,

$$f_S(x_{\bar{S}}) = \mathbb{E}_{x_S \sim U_S}[\chi_S(x) f(x)]$$

Given a point $(x, f(x))$, we observe that the expectation $\mathbb{E}_{x_S \sim U_S}[f(x)\chi_S(x)]$ can be computed using $2^{|S|}$, $|S|$-local membership queries with respect to x (only the bits in S need to be flipped). The quantity $\mathbb{E}_{x \sim U}[f_S(x)^2]$ can thus be computed easily using only $|S|$-local membership queries and taking a sample from $\mathsf{EX}(f, D)$.

High-Level Overview of Proof: Fact 8.3.1 showed that the Fourier mass (sum of squares of the Fourier coefficients) of t-leaf decision trees is concentrated on low degree terms. Parseval's identity implies that this is sufficient to construct a polynomial, $h(x)$, that is a good ℓ_2 approximation to the decision tree, f, i.e. $\mathbb{E}_{x \sim U}[(h(x) - f(x))^2] \leq \epsilon$. Also, **?**] showed that since $L_1(f)$ is bounded, most of the Fourier mass is concentrated on a small (polynomially many) number of terms.

The main insight here is that, these terms on which most of the Fourier mass is concentrated, can be identified using only $O(\log(n))$-local membership queries. It is relatively easy to see that any coefficient for which $|\hat{f}(S)| \geq \theta$ will be identified correctly by the test in line 2.(a).ii. (Figure 8.2). We show that the quantity $|\mathcal{S}|$ never grows too large. To show this, we prove that if any coefficient is inserted in $\mathcal{S}$ in line 2.(a).ii, it must be a subset of some coefficient of large magnitude. This follows quite easily by observing that $\mathbb{E}_{x\sim U}[f_S(x)^2] = \sum_{T\supseteq S} \hat{f}(T)^2$ and using the fact that $L_1(f)$ is bounded.

The rest of the section is devoted to a formal proof of the above overview.

Claim 8.3.2. *Suppose that S is such that $|\hat{f}(S)| \geq \theta$ and $|S| \leq d$, then $S \in \mathcal{S}$.*

Proof. First observe that for any subset $S' \subseteq S$, it holds that $\mathbb{E}[f_{S'}(x)^2] \geq \theta^2$. This follows immediately by observing that

$$\mathbb{E}[f_{S'}(x)^2] = \sum_{T\supseteq S'} \hat{f}(T)^2 \geq \hat{f}(S)^2 \geq \theta^2$$

It follows by a simple induction argument that at iteration i, $\mathcal{S}$ contains every subset of S of size at most i, for which $\mathbb{E}[f_S(x)^2] \geq \theta^2$. And, hence $S \in \mathcal{S}$. □

Claim 8.3.3. *If $S \in \mathcal{S}$, then there exists a $S' \supseteq S$ such that $\hat{f}(S') \geq \theta^2/t$.*

Proof. Since $S \in \mathcal{S}$, we know that $\mathbb{E}[f_S(x)^2] = \sum_{T\supseteq S} \hat{f}(S)^2 \geq \theta^2$. But observe that,

$$\sum_{T\supseteq S} \hat{f}(T)^2 \leq \left(\sum_{T\supseteq S} |\hat{f}(T)|\right) \cdot \max_{T\supseteq S} |\hat{f}(T)|$$

The above inequality simply states the fact that $L_2(f_S) \leq L_1(f_S)L_\infty(f_S)$. Since f is a t-leaf decision tree, $\sum_{T\supseteq S} |\hat{f}(T)| \leq L_1(f) \leq t$. The claim now follows immediately. □

Using the above claims, it is easy to show our main theorem.

Theorem 8.3.4. *Algorithm in Fig. 8.2 run with parameters $d = \log(2t^2/\epsilon)$ and $\theta = \epsilon/(2t)$, outputs a hypothesis, $\mathrm{sign}(h(x))$, where $\mathbf{err}_U(\mathrm{sign}(h(x), f) \leq \epsilon$. The running time is $\mathrm{poly}(t, n, 1/\epsilon)$ and the algorithm only makes $\log(2t^2/\epsilon)$-local queries to the membership oracle $\mathsf{MQ}(f)$.*

Proof. First, we recall that for a t-leaf decision tree, $|\hat{f}(S)| \leq t/2^{|S|}$ **?**]. Thus, if $|\hat{f}(S)| \geq \theta^2/t$, then $|S| \leq 2\log(t/\theta)$. Using Parseval's identity (see Section 8.1), we know that the number of Fourier coefficients that have magnitude greater than θ^2/t is at most t^4/θ^2.

Consider the set $\mathcal{S}$ constructed by the algorithm (Fig. 8.2) at the end of d iterations. If $S \in \mathcal{S}$, then there must exist some $T \supseteq S$ such that $|\hat{f}(T)| \geq \theta^2/t$ (Claim 8.3.3). But there can be at most t^2/θ^4 such terms and each is of size at most $2\log(t/\theta)$. Hence, the $|\mathcal{S}| \leq (t^2/\theta^4)2^{2\log(t/\theta)} = t^4/\theta^6$.

For any coefficient, such that $|\hat{f}(S)| \geq \theta$, it must be that $|S| \leq \log(t/\theta) \leq d$. Claim 8.3.2 shows that all such coefficients are included in $\mathcal{S}$. Thus, $\max_{S \notin \mathcal{S}} |\hat{f}(S)| \leq \theta$. Hence, $\sum_{S \notin \mathcal{S}} \hat{f}(S)^2 \leq \sum_{S \notin \mathcal{S}} |\hat{f}(S)| \cdot \max_{S \notin \mathcal{S}} |\hat{f}(S)| \leq L_1(f) \cdot \theta \leq \theta t$. But $\mathbb{E}[(h(x) - f(x))^2] = \sum_{S \notin \mathcal{S}} \hat{f}(S)^2$ and also notice that $\Pr_{x \sim U}[\text{sign}(h(x)) \neq f(x)] \leq \mathbb{E}_{x \sim U}[(h(x) - f(x))^2]$ (since $f(x)$ only takes values ± 1). □

8.4 Learning DNF Formulas under the Uniform Distribution

In this section, we present an algorithm for learning polynomial size DNF formulas under the uniform distribution. We will frequently use the following facts about DNF formulas.

1. For every size-s DNF formula f, there exists a size-s, DNF formula g with terms of size l, such that $\|f - g\|_2^2 = \mathbb{E}[(f(x) - g(x))^2] \leq \frac{4s}{2^l}$. This follows by setting g to the DNF formula obtained by dropping all terms of size greater than l from f.
2. Let f be an l-term DNF formula. Then $\sum_{|S|>t} \hat{f}^2(S) \leq 2^{\frac{-t}{10l}}$. This fact follows from the proof of Lemma 3.2 in Mansour [1992].

We will further use the following result from Kalai et al. [2009b], Feldman [2012]:

Theorem 8.4.1 (Feldman [2012], Kalai et al. [2009b])**.** *If f is a size-s DNF formula, then there exists an efficient randomized algorithm,* LearnDNF *that given access to the heavy, low-degree Fourier coefficients, i.e. the set $\{\hat{f}(S) \mid |S| \leq \log(s/\epsilon), |\hat{f}(S)| \geq (\epsilon/(4s))\}$, outputs a hypothesis, h, such that $\mathbf{err}_U(h, f) \leq \epsilon$.*

The algorithm to obtain all *heavy, low-degree terms* is presented in Figure 8.3. The output of the algorithm, LearnDNF($\mathcal{S}$) is obtained by doing the following: (i) estimate all the Fourier coefficients, $\hat{f}(S)$, for $S \in \mathcal{S}$ (ii) use the algorithm from Theorem 8.4.1 to obtain h. The rest of the section is devoted to the proof of the following Theorem.

Theorem 8.4.2. *The class of size-s DNF formulas is learnable in time $(s/\epsilon)^{O(\log\log(s/\epsilon))}$ using $O(\log(s/\epsilon))$-local membership queries under the uniform distribution.*

Algorithm: LEARNING DNF FORMULAS

inputs: d, θ, oracles $\mathsf{EX}(f, U)$, (local)-$\mathsf{MQ}(f)$

1. **let** $\mathcal{S} = \{\emptyset\}$
2. **for** $i = 1, \ldots, d$
   (a) **for** every $S' \in \mathcal{S}$, $|S'| = i - 1$ and for every $j \in [n] \setminus S'$
      i. **let** $S = S' \cup \{j\}$
      ii. (L_2 Test) **if** $\mathbb{E}_{x \sim U}[f_S(x)^2] > \theta^2$, **then** $\mathcal{S} = \mathcal{S} \cup \{S\}$

output: $h = \mathsf{LearnDNF}(\mathcal{S})$

Figure 8.3: Algorithm: Learning DNF formulas under the Uniform Distribution

When $d = log(s/\epsilon)$ and $\theta = \epsilon/4s$, we argue that every Fourier coefficient that has magnitude at least θ is included in $\mathcal{S}$ and also that $|\mathcal{S}|$ is $(s/\epsilon)^{O(\log\log(s/\epsilon))}$. The first part follows from the following claim

Claim 8.4.3. *Suppose that S is such that $|\hat{f}(S)| \geq \theta$ and $|S| \leq d$, then $S \in \mathcal{S}$.*

Proof. First observe that for any subset $S' \subseteq S$, it holds that $\mathbb{E}[f_{S'}(x)^2] \geq \theta^2$. This follows immediately by observing that

$$\mathbb{E}[f_{S'}(x)^2] = \sum_{T \supseteq S'} \hat{f}(T)^2 \geq \hat{f}(S)^2 \geq \theta^2$$

It follows by a simple induction argument that at iteration i, $\mathcal{S}$ contains every subset of S of size at most i, for which $\mathbb{E}[f_S(x)^2] \geq \theta^2$. And, hence $S \in \mathcal{S}$. □

We now prove the second claim which bounds the size of $\mathcal{S}$.

Claim 8.4.4. *After running algorithm 8.3, $|\mathcal{S}| = (s/\epsilon)^{O(\log\log(s/\epsilon))}$.*

of Claim 8.4.4. For a set S, denote $\|f_S\|^2 = \sum_{T \supseteq S} \hat{f}^2(T)$. If $S \in \mathcal{S}$, then we know that $\|f_S\|^2 \geq \theta^2$. Thus, it follows that $|\mathcal{S}| \leq (1/\theta^2) \sum_{S:|S| \leq d} \|f_S\|^2$. Now, $\sum_{S:|S| \leq d} \|f_S\|^2 = \sum_{d'=1}^{d} \sum_{S:|S|=d'} \|f_S\|^2$. We will show that for each d', $\sum_{S:|S|=d'} \|f_S\|^2 \leq ns2^{O(d' \log(d'))}$.

Then,

$$\sum_{S:|S|=d'} \|f_S\|^2 = \sum_{S:|S|=d'} \sum_{T \supseteq S} \hat{f}^2(T)$$
$$= \sum_{T:|T| \geq d'} \binom{|T|}{d'} \hat{f}^2(T)$$
$$= \sum_{t=d'}^{n} \sum_{T:|T|=t} \binom{t}{d'} \hat{f}^2(T)$$

For a given t, consider the inner summation $\sum_{T:|T|=t} \binom{t}{d'} \hat{f}^2(T)$. We will first apply Fact 1, to say that f is close to some g_t which is an l_t-term DNF formula (we will define the value of l_t shortly). Hence, we get that,

$$\sum_{T:|T|=t} \binom{t}{d'} \hat{f}^2(T) \leq \binom{t}{d'} \sum_{T:|T|=t} \hat{g}_t^2(T) + \frac{4s}{2^{l_t}}$$

Next we use Fact 2 to claim that $\sum_{T:|T|=t} \hat{g}_t^2(T) \leq 2^{\frac{-t}{10l_t}}$. So we have

$$\sum_{T:|T|=t} \binom{t}{d'} \hat{f}^2(T) \leq \binom{t}{d'} \left(2^{\frac{-t}{10l_t}} + \frac{4s}{2^{l_t}} \right)$$

Setting $l_t = C\sqrt{t}$ and differentiating, we get that the term is maximized at $t = O(d'^2)$ and the maximum value is $s2^{O(d' \log d')}$. Since there are at most n such terms we get that $\sum_{S:|S|=d'} ||f_S||^2 = ns2^{O(d' \log d')}$. Finally, we have

$$\sum_{S:|S| \leq d} ||f_S||^2 \leq \sum_{d'=1}^{d} ns2^{O(d' \log(d'))} = nds2^{O(d \log d)}.$$

□

The proof of Theorem 8.4.2 follows immediately using Claims 8.4.3, 8.4.4 and Theorem 8.4.1.

8.5 Lower Bound for Agnostic Learning

In this section, we prove that any concept class, C, efficiently *agnostically* learnable over the uniform distribution with (constant) k-local MQs is also efficiently *agnostically* learnable from random examples alone. This result can be compared with that of Feldman

[2008], where it is shown that membership queries do not help for distribution-independent agnostic learning.

We remark that 1-local MQs suffice for learning parities of any size with random classification noise. At the same time when learning from random examples alone agnostic learning of parities can be reduced to learning parities with random classification noise (Feldman et al. [2006]). However this reduction does not lead to an agnostic algorithm for learning parities with 1-local MQs since it is highly-nonlocal: the (noisy) label of every example is influenced by labels of points chosen randomly and uniformly from the whole hypercube.

Our reduction is based on embedding the unknown function, f, in a higher dimensional domain such that the original points are mapped to points that are at least at distance $2k+1$ apart (and in particular that no single point in the domain is k-close to more than one of the original points). A crucial property of this embedding is that, up to scaling it preserves the correlation of any function with f. The embedding is achieved using a linear binary error-correcting code, specifically we use the classic binary BCH code (Hocquenghem [1959], Bose and Ray-Chaudhuri [1960]).

Theorem 8.5.1. *For any constant, k, if a concept class C is learnable agnostically under the uniform distribution in the* PAC+k*-local* MQ *model, then C is also agnostically learnable in the* PAC *model.*

Proof. We begin by describing the required properties of the error correcting code. For every integer t and m that is a power of two, there exists a binary BCH code that maps a binary string x of length $m - 1 - (t-1)\log m$ to a binary string $z = x \cdot e(x)$ of length m and has distance of $2t$. In particular, if we denote the length of message x by n then for any k we can obtain a code with a codeword of length $m \leq n + k \log n$ and distance $2k + 1$.

Given a function $f : \{-1,1\}^n \to \{-1,1\}$ we define a function $f_e : \{-1,1\}^m \to \{-1,0,1\}$ as follows: $f_e(z) = f(x)$ if $z = x \cdot e(x)$ for some $x \in \{-1,1\}^n$ and $f_e(z) = 0$ otherwise. Here the value 0 is interpreted as function being equal to a random and independent coin flip.

We first note the following properties of this embedding.

- For every function $g : \{-1,1\}^n \to \{-1,1\}$,

$$\mathbb{E}_{x \cdot y \sim U_m}[f_e(x \cdot y)g(x)] = 2^{n-m}\mathbb{E}_{x \sim U}[f(x)g(x)] \ .$$

- For every function $h : \{-1,1\}^m \to \{-1,1\}$,

$$\mathbb{E}_{z\sim U_m}[f_e(z)h(z)] = 2^{n-m}\mathbb{E}_{x\sim U_n}[f(x)h(x\cdot e(x))] \ .$$

We can now describe the reduction from agnostic learning with k-local MQs to learning from random examples alone. Let C be a concept class, let f denote an unknown target function and ϵ denote the error parameter.

The main idea is to simulate random examples and k-local queries to f_e using random examples alone of f. The simulation requires observing that points in $\{-1,1\}^m$ can be split into a set Z which includes all codewords together with Hamming balls of radius t around them and the rest of points (which we denote by $\bar{Z}$). We simulate a random example of $f_e(x)$ as follows:

1. Flip a coin with probability of heads equal to $\beta = |Z|/2^m$.

2. If the outcome is 1: ask for a random example and denote it by (x,ℓ). Choose a random point z' in the Hamming ball of radius k around $x\cdot e(x)$. If $z' = x\cdot e(x)$ then return the example (z',ℓ) otherwise return (z',b) where b is a random coin flip.

3. If the outcome is 0: sample a random point in $\bar{Z}$ and output (z,b) where b is a random coin flip. One can sample randomly from $\bar{Z}$ as follows: sample a random point in z in $\{-1,1\}^m$, use a decoding algorithm for the BCH code to obtain a message x. If the decoding algorithm failed, that is $x\cdot e(x)$ is not within distance k of z then return z. Otherwise, try again.

It is important to note that the expected number of tries of this algorithm is $2^m/|\bar{Z}| = 2^m/(2^m - 2^n\beta_{m,k}) = 1/(1-2^{n-m}\beta_{m,k})$, where $\beta_{m,k}$ denotes the size of the Hamming ball of radius k. We can always assume that $2^{n-m}\beta_{m,k} \leq 2/3$ by for example adding 1 to m (this does not affect the code, increases the term 2^{m-n} by 2 and $\beta_{m,k}$ by at most $(1+k/(m+1))$). Therefore, with high probability, the simulation step will not require more than a logarithmic number of tries. Note that the BCH code we chose is efficiently decodable from up to k errors (Massey [1969]).

Now we simulate a k-local MQ z as follows. If z is k-close to random example we generated in $\bar{Z}$ we return a random coin flip since there are no non-zero values of f_e within distance k of any point in $\bar{Z}$. If z is k-close to random example we generated in Z then it can only be k-close to $x\cdot e(x)$ in the Hamming ball of which z was generated and for which we have an example (x,ℓ). This means that we can easily answer this MQ: if $z = x\cdot e(x)$ then we return label ℓ, otherwise a random coin flip.

On these simulated examples we run the agnostic learning algorithm $\mathcal{A}$ for C on $\{-1,1\}^m$ with $\epsilon' = \epsilon \cdot 2^{n-m} \geq \epsilon/n^k$. Let h denote the hypothesis returned by the algorithm.

Let $c^* = argmax_{c \in C}\{\mathbb{E}_U[f(x)c(x)]\}$ and let $\Delta = \mathbb{E}_U[f(x)c^*(x)]$. Then we know that

$$\mathbb{E}_{x \cdot y \sim U_m}[f_e(x \cdot y)c^*(x)] = 2^{n-m}\Delta.$$

By the agnostic guarantee of $\mathcal{A}$, we know that

$$\mathbb{E}_{z \sim U_m}[f_e(z)h(z)] \geq 2^{n-m}\Delta + \epsilon' = 2^{n-m}(\Delta + \epsilon) \ .$$

Now, again by the properties of the embedding,

$$\mathbb{E}_{x \sim U_n}[f(x)h(x \cdot e(x))] = 2^{m-n}\mathbb{E}_{z \sim U_m}[f_e(z)h(z)] \geq \Delta + \epsilon \ .$$

Hence $h'(x) = h(x \cdot e(x))$ is a valid hypothesis for agnostic learning of C.

Finally the running time of this simulation is $poly(n) \cdot T(n, n^k/\epsilon)$ where $T(n, n^k/\epsilon)$ is the running time of $\mathcal{A}$. In particular, if T is polynomial then for a constant k this simulation takes polynomial time. □

In particular, the above theorem implies that it is highly unlikely that the class of parities (even of size $O(\log(n))$) will be efficiently agnostically learnable using k-local MQs. The class of parities is of particular interest, because an efficient agnostic algorithm for learning $O(\log(n))$ sized parities would yield an efficient DNF-learning algorithm.

8.6 Separation Results

In this section, we show that PAC+r-local MQ model is strictly more powerful than the PAC model, assuming that one-way functions exist. In the following discussion we show that even 1-local membership queries are more powerful than the standard PAC setting.

In this section, we assume that we are working with the domain $\{0,1\}^n$, rather than $\{-1,1\}^n$. Let $\mathcal{F}_n = \{f_s : \{0,1\}^n \to \{0,1\}\}_{s \in \{0,1\}^n}$ be a pseudo-random family of functions. It is well-known that such families can be constructed under the assumptions that one-way functions exist (Goldreich et al. [1986]). Let $A_1, \ldots, A_n$ be a balanced partition (each A_i is approximately the same size) of $\{0,1\}^n$ that is easily computable. For example, if the strings in $\{0,1\}^n$ are lexicographically ordered, then A_i contains strings with rank in the range $[(i-1)2^n/n, i2^n/n)$. For an $n+1$ bit string x, x_{-1} denotes the

n-length suffix of x. Then, for some string s, define the function $g_s : \{0,1\}^{n+1} \to \{0,1\}$ as follows:

$$g_s(x) = \begin{cases} f_s(x_{-1}) & \text{If } x_1 = 0 \\ f_s(x_{-1}) \oplus s_i & \text{If } x_1 = 1 \text{ and } x_{-1} \in A_i \end{cases}$$

Define $\mathcal{G}_{n+1} = \{g_s : \{0,1\}^{n+1} \to \{0,1\}\}_{s \in \{0,1\}^n}$. We show below that the class $\mathcal{G}_{n+1}$ is not learnable in the PAC setting, but is learnable in the PAC+1-local MQ model under the uniform distribution.

Theorem 8.6.1. *Assuming that one-way functions exist, the class $\mathcal{G}_{n+1}$ is not learnable in the* PAC *model, but is learnable in the* PAC+1*-local* MQ *model, under the uniform distribution.*

Proof. First, we show that $\mathcal{G}_{n+1}$ is learnable in the PAC+1-local MQ model. Let $1x$ and $0x$ be the two strings of length $n+1$, with suffix $x \in \{0,1\}^n$. Then for any $g_s \in \mathcal{G}_{n+1}$, $g_s(1x) \oplus g_s(0x) = s_i$, if $x \in A_i$. Thus, drawing a random example from U and making a one local query reveals one bit of the string s. By drawing $O(n \log(n))$ random examples, all the bits of the string s can be recovered with high probability. Thus, revealing the function g_s itself.

On the other hand, in the PAC model, the probability that seeing two examples $1x$ and $0x$ is exponentially small. Thus, all the labels appear perfectly random (since f_s is from a pseudorandom family). Thus, no learning is possible in the PAC model. □

In fact, the above construction also shows that the random walk learning model of Bshouty et al. [2005] is also more powerful than the PAC learning setting, assuming that one way function exist. Bshouty et al. [2005] had already shown that the random walk model is provably weaker than the full MQ model assuming that one-way functions exist. In fact, essentially the same argument also shows that full MQ is more powerful than PAC+$o(n)$-local MQ. The following simple concept class (which is the same as that of Bshouty et al.) shows the necessary separation.

Let e^i be the vector that has 1 in position i, and 0s elsewhere. Again, let $\mathcal{F}_n = \{f_s : \{0,1\}^n \to \{0,1\}\}_{s \in \{0,1\}^n}$ be the pseudorandom family of functions. Then define, $\mathcal{G}'_n = \{g_s\}$ as follows:

$$g_s(x) = \begin{cases} s_i & \text{If } x = e^i \\ f_s(x) & \text{Otherwise} \end{cases}$$

Theorem 8.6.2. *The concept class $\mathcal{G}'_n$ is learnable in the full* MQ *model, but not in* PAC+$o(n)$*-local* MQ *model under the uniform distribution.*

Proof. It is easy to see that by making membership queries to the points, $e^1, \ldots, e^n$, the string s is revealed and hence also the function g_s. On the other hand, random points from the Boolean cube have Hamming weight $\Omega(n)$, except with exponentially small probability. Thus, $o(n)$-local MQs are of no use to query the points e^i. The labels for any point obtained from the distribution, or using $o(n)$-local MQs are essentially random. Hence, $\mathcal{G}'_n$ is not learnable in the PAC+$o(n)$-local MQ model. □

8.7 Additional proofs

8.7.1 Learning Sparse Polynomials under Log-Lipschitz Distributions

Proof of Theorem 8.2.1
First, we have the following useful general lemma.

Lemma 8.7.1. *Let f be a t-sparse multi-linear polynomial defined over any field, $\mathbb{F}$, with a non-zero constant term, c_0. Let D be any α-log-Lipschitz distribution over $\{0,1\}^n$, then*

$$\Pr_D[f(x) \neq 0] \geq \left(\frac{1}{1+\alpha}\right)^{\log_2(t)}$$

Proof. We prove this by induction on the number of variables, n. When $n = 1$, the only possible polynomials are $f(x_1) = c_0 + c_1 x_1$. Then $f(x) = 0$ if and only if $x_1 = 1$ and $c_1 = -c_0$ (since $c_0 \neq 0$). Note that when D is α-log-Lipschitz, $\Pr[x_1 = 1] \leq \alpha/(1+\alpha)$ (see Fact 8.1.2). Thus, $\Pr_D[f(x_1) \neq 0] \geq 1/(1+\alpha)$. (And the sparsity is 2, and $\log(2) = 1$.) Thus the base case is verified.

Let f be any multi-linear polynomial defined over n variables. Suppose there exists a variable, without loss of generality, say x_1, such that $c_1 x_1$ is a term in f, where $c_1 \neq 0$. Then we can write f as follows:

$$f(x) = f_{-1}(x) + x_1 f_1(x)$$

where f_{-1} and f_1 are both multi-linear polynomials over $n-1$ variables and both have a non-zero constant term. (The constant term of f_{-1} is just c_0, and f_1 has constant term c_1.) Then note that $1/(1+\alpha) \leq \Pr_D[x_1 = b | x_{-1}] \leq \alpha/(1+\alpha)$, for both $b = 1$ and $b = 0$. Now, it is easy to see that $\Pr_D[f(x) \neq 0] \geq \Pr_D[x_1 = 0|x_{-1}] \Pr_D[f_{-1}(x) \neq 0] \geq (1/(1+\alpha)) \Pr[f_{-1}(x) \neq 0]$.

To see that $\Pr_D[f(x) \neq 0] \geq (1/(1+\alpha)) \Pr_D[f_1(x) \neq 0]$ consider the following: Fix x_{-1}, if $\Pr[f_1(x) \neq 0]$, then for at least one setting of x_1, it must be the case that $f(x) \neq 0$.

Thus, conditioned on x_{-1}, $\Pr_D[f(x) \neq 0 | x_{-1}] \geq (1/(1+\alpha))\delta(f_{-1}(x) \neq 0)$ (here $\delta(\cdot)$ is the indicator function). Thus, $\Pr_D[f(x) \neq 0] \geq (1/(1+\alpha)) \Pr_D[f_{-1}(x) \neq 0]$. However, at least one of f_{-1}, f_1 must have sparsity at most $t/2$, thus by induction we are done.

In the case, that there is no x_i such that $c_i x_i$ (with $c_i \neq 0$) appears in f as a term, let f_0 be the polynomial obtained from f by setting $x_1 = 0$ and f_1 be the polynomial obtained from f by setting $x_1 = 1$. Note that both f_0 and f_1 have constant term $c_0 \neq 0$ and sparsity at most t, but they have one fewer variable than f. Thus, $\Pr_D[f_b(x) \neq 0] \geq (1/(1+\alpha))^{\log_2(t)}$, for $b = 0, 1$. However, note that

$$\begin{aligned} \Pr_D[f(x) \neq 0] &= \Pr_D[x_1 = 0] \Pr_{D_0}[f_0(x) \neq 0] + \Pr_D[x_1 = 1] \Pr_{D_1}[f_1(x) \neq 0] \\ &\geq \left(\frac{1}{1+\alpha}\right)^{\log(t)} \end{aligned}$$

This completes the induction. □

Using Lemma 8.7.1, we can now show that step 3 in algorithm 8.1 correctly identifies all the important monomials (monomials of low-degree with non-zero coefficients in f).

Lemma 8.7.2. *Suppose $S \subseteq [n]$, such that $f_S(x)$ has a monomial of degree at most $d - |S|$ with non-zero coefficient. Then,*

$$\Pr_{D_{\bar{S}}}[f_S(x_{\bar{S}}) \neq 0] \geq \left(\frac{1}{1+\alpha}\right)^{d-|S|+\log(t)}$$

Proof. Note that, since D is a α-log-Lipschitz distribution, $D_{\bar{S}}$ is also a α-log-Lipschitz distribution (see Fact 8.1.2). Let S' be a subset of $\bar{S}$, such that $\xi_{S'}(x_{\bar{S}})$ is the smallest degree monomial in $f_S(x_{\bar{S}})$ with non-zero coefficient. Then, since $D_{\bar{S}}$ is α-log-Lipschitz, $\Pr_{D_{\bar{S}}}[\xi_{S'}(x_{\bar{S}}) = 1] \geq (1/(1+\alpha))^{|S'|} \geq (1/(1+\alpha))^{d-|S|}$.

Now, the conditional distribution $(D_{\bar{S}} | \xi_{S'}(x_{\bar{S}}) = 1)$ is not necessarily α-log-Lipschitz, but the marginal distribution with respect to variables $(\overline{S \cup S'})$, $(D_{\bar{S}} | (\xi_{S'}(x_{\bar{S}}) = 1))_{(\overline{S \cup S'})}$, is indeed α-log-Lipschitz (see Fact 8.1.2). Let $f_S^{S'}(x_{(\overline{S \cup S'})})$ be the polynomial obtained from f_S by setting $x_i = 1$ for each $i \in S'$. Note that the constant term of $f_S^{S'}$ is non-zero and it is t-sparse, and is only defined on the variables in $-(S \cup S')$. Hence, by applying Lemma 8.7.1 to $f_S^{S'}$ and the marginal (w.r.t the variables $\bar{S'}$) of the conditional distribution $(D_{\bar{S}} | \xi_{S'}(x_{\bar{S}}) = 1)$, *i.e.* $(D_{\bar{S}} | \xi_{S'}(x) = 1)_{(\overline{S \cup S'})}$, we get the required result. □

Next, we show the following simple lemma (proof is in Section 8.7.1) that will allow us to conclude that step 3 of the algorithm never adds too many terms.

Lemma 8.7.3. *If each term of f_S has degree at least d', then the probability that $f_S(x) \neq 0$ is at most $t(\alpha/(1+\alpha))^{d'}$.*

Proof. Note that each monomial of f_S has degree at least d'. Under any α-log-Lipschitz distribution, the probability that a monomial of degree d' is not-zero is at most $(\alpha/(1+\alpha))^{d'}$ (see Fact 8.1.2). Since, $D_{\bar{S}}$ is a α-log-Lipschitz distribution, by a simple union bound we get the required result. □

Now in order to get an ϵ-approximation in terms of squared error, using the argument about truncation, it suffices to choose $d = \log(4t^3B^2/\epsilon)/\log((1+\alpha)/\alpha)$, and consider the truncation α. For this value of d, if θ is set to $1/(4t^3B^2)^{2\log(1+\alpha)/\log((1+\alpha)/\alpha)}$, using Lemma 8.7.2, we are sure that all the monomials in f of degree at most d that have non-zero coefficients are identified in step 3 of the algorithm. Note that θ is still inverse polynomial in $(ntB/\epsilon)^\alpha$.

Finally, we note that if d' is set to $\log(2t/\theta)/\log((1+\alpha)/\alpha)$, then for any subset, S, if the monomial with the least degree in f_S, has degree at least d', then $\Pr_{D_{\bar{S}}}[f_S(x) \neq 0] \leq \theta/2$. In particular, this means that if a set, S, with $|S| \leq d$, is such that the smallest monomial, $\xi_T(x)$ in f for which $S \subseteq T$, is such that $|T| \geq d + d'$, then S will never be added to $\mathcal{S}$ by the algorithms. The fact that this probability was $\theta/2$ (instead of exactly θ), means that sampling can be used carry out the test in the algorithm to reasonable accuracy. Finally, observe that $t2^{d+d'}$ is still polynomial in $(ntB/\epsilon)^\alpha$. Thus, the total number of sets added in $\mathcal{S}$, can never be more than polynomially many.

Generalization The generalization argument is pretty standard and so we just present an outline. First, we observe that it is fine to discretize real numbers to some Δ, where Δ is inverse polynomial in $(ntB/\epsilon)^\alpha$, without blowing up the squared loss. Now, the regression in the algorithm requires that the sum of absolute values of the coefficients of the polynomial, h, be at most tB. Thus, we can view this as distributing tB/Δ blocks over 2^n possible coefficients (in fact the number of coefficients is smaller). The total number of such discretized polynomials is at most $2^{\mathrm{poly}((ntB/\epsilon)^\alpha)}$. Thus, it suffices to minimize the squared error on a (reasonably large) sample.

8.7.2 Learning Decision Trees under Product Distributions

In this section, we prove that the class of t-leaf decision trees can be learned under the class of product distributions, where each bit has mean bounded away from -1 and 1. Let $\mu = (\mu_1, \ldots, \mu_n)$ denote a product distribution over $X = \{-1, 1\}^n$, where $\mathbb{E}_{x\sim\mu}[x_i] = \mu_i \in [-1+2c, 1-2c]$, for some constant $c \in (0, 1/2]$. We use Fourier analysis using the

modified basis for the product distribution. We begin by introducing required notation for using Fourier techniques.

Fourier Analysis over μ: Let $\mu = (\mu_1, \ldots, \mu_n)$ be the product distribution over $X = \{-1, 1\}^n$, where $\mathbb{E}_{x\sim\mu}[x_i] = \mu_i$. Define,

$$\chi_S^\mu(x) = \prod_{i\in S} \frac{x_i - \mu_i}{\sqrt{(1-\mu_i^2)}}.$$

Then, it is easy to observe that for any two sets $S_1 \neq S_2$, $\mathbb{E}_{x\sim\mu}[\chi_{S_1}^\mu(x)\chi_{S_2}^\mu] = 0$ and that, for any set S, $\mathbb{E}_{x\sim\mu}[\chi_S^\mu(x)^2] = 1$. Thus, the set of functions $\langle\chi_S^\mu(x)\rangle_{S\subseteq[n]}$ forms an orthonormal basis for functions defined on $\{-1, 1\}^n$ under the distribution μ. For any function $f : \{-1, 1\}^n \to \mathbb{R}$, the Fourier coefficients under distribution μ are defined as $\hat{f}^\mu(S) = \mathbb{E}_{x\sim\mu}[f(x)\chi_S^\mu(x)]$. The following is Parseval's identity in this basis:

$$\mathbb{E}_{x\sim\mu}[f(x)^2] = \sum_{S\subseteq[n]} \hat{f}^\mu(S)^2 \tag{8.2}$$

In particular, when f is a boolean function, i.e. with range $\{-1, 1\}$, the sum of Fourier coefficients is 1.

Let $L_1^\mu(f) = \sum_{S\subseteq[n]} |\hat{f}^\mu(S)|$, $L_2^\mu(f) = \sum_{S\subseteq[n]} \hat{f}^\mu(S)^2$ and $L_\infty^\mu(f) = \max_{S\subseteq[n]} |\hat{f}^\mu(S)|$ denote the 1, 2 and ∞ norm of the Fourier spectrum under distribution μ. Also let $L_0^\mu(f) = |\{S \mid \hat{f}^\mu(S) \neq 0\}|$ denote the number of non-zero Fourier coefficients of f. We will frequently use the following useful observations:

1. $L_2^\mu(f) \leq L_1^\mu(f) \cdot L_\infty^\mu(f)$
2. $L_2^\mu(f) \leq L_0^\mu(f) \cdot (L_\infty^\mu(f))^2$

Decision Tree Learning Algorithm

We present a high-level overview of our algorithm and a formal statement of the main result, before providing full details. The Algorithm is described in Figure 8.4.

Truncation: We show that a t-leaf decision tree, when truncated to logarithmic depth, is still a very good (inverse polynomially close) approximation to the original decision tree. This observation can be used to show that it suffices to identify low-degree (logarithmic) "heavy" Fourier coefficients of f, with respect to the distribution, μ, and also that the number of such terms is not too large (at most polynomial). Note that this is not

Algorithm: LEARNING DECISION TREES

inputs: d, θ, oracles $\mathsf{EX}(f, \mu)$, $\mathsf{MQ}(f)$

\# f is a t-leaf decision tree

\# μ is a product distribution over $\{-1, 1\}^n$, $\mu_i \in [-1 + 2c, 1 - 2c]$

1. **let** $\mathcal{S} = \{\emptyset\}$
2. **for** $i = 1, \ldots, d$
   (a) **for** every $S' \in \mathcal{S}$, $|S'| = i - 1$ and for every $j \in [n] \setminus S'$
      i. **let** $S = S' \cup \{j\}$
      ii. (L_2 Test) **if** $\mathbb{E}_{x \sim \mu}[f_S(x)^2] > \theta^2$, **then** $\mathcal{S} = \mathcal{S} \cup \{S\}$
3. **let** $h(x) = \sum_{S \in \mathcal{S}} \hat{f}^{\mu}(S) \chi^{\mu}_S(x)$

output: $\mathrm{sign}(h(x))$

Figure 8.4: Algorithm: Learning Decision Trees under Product Distributions

as simple as in the case of the uniform distribution, because it is not straightforward to bound $L_1^{\mu}(f) = \sum_{S \subseteq [n]} |\hat{f}^{\mu}(S)|$. (When μ is the uniform distribution, this is bounded by t.) Properties of such truncated decision trees were also used by Kalai et al. [2009b] in the smoothed analysis setting.

A t-leaf decision tree can be though of as t (not disjoint) paths from root to leaves. A truncation of a decision tree at depth d, is a decision tree where for each path of length more than d, only the prefix (from root) of length d is preserved. Note that this may collapse several paths to the same prefix, possibly reducing the number of leaves. A new leaf is added at the end of this path and labeled arbitrarily as -1 or $+1$.

For any function g, we denote by $\mathcal{S}_g^{\mu}$, the set of non-zero Fourier coefficients of g, with respect to the product distribution, μ, *i.e.* $\mathcal{S}_g^{\mu} = \{T \subseteq [n] \mid \hat{g}(T) \neq 0\}$.

We prove two useful properties of the truncated decision trees with respect to product distribution. These appear as formal statements in Lemmas 8.7.4 and 8.7.5. Similar observations were also used by Kalai et al. [2009b] to prove learning of decision trees in the smoothed analysis setting.

(i) Truncation at logarithmic depth is a good approximation (inverse polynomial) to the

original decision tree.

(ii) The number of nonzero Fourier coefficients of the truncated decision tree, $|\mathcal{S}_g^\mu|$ is small (polynomial).

Lemma 8.7.4. *Let f be a t-leaf decision tree, let μ be a product distribution over $X = \{-1,1\}^n$ such that $\mu_i \in [-1+2c, 1-2c]$. Then for every $\tau > 0$, there exists a t-leaf decision tree of depth at most $\log(t/\tau)/\log(1/(1-c))$, such that $\Pr_{x\sim\mu}[g(x) \neq f(x)] \leq \tau$*

Proof. Let g be the decision tree obtained by truncating f at depth d. The new leaves added at depth d can be labeled arbitrarily. Now, the points x for which $g(x) \neq f(x)$ are precisely those, for which g would lead to the newly added leaf node at depth d. But since $\mathbb{E}_{x\sim\mu}[x_i] \in [-1+2c, 1-2c]$, the probability that a random point from μ reaches such a node is at most $(1-c)^d$. The number of new leaf nodes added cannot be more than t (since any truncation only reduces the number of leaves). Thus, $\Pr_{x\sim\mu}[g(x) \neq f(x)] \leq t(1-c)^d$. When, $d = \log(t/\tau)/\log(1/(1-c))$ we get the result. □

Lemma 8.7.5. *Let g be a decision tree of depth d and t leaves; then the number of non-zero Fourier coefficients of g is at most $t \cdot 2^d$ and each is of size at most d.*

Proof. We consider any path in g from root to leaf, and let P denote the subset of indexes corresponding to the variable that occur in the path. First, we expand decision tree g as a polynomial.

$$g(x) = \sum_{\text{path } P} y_P \prod_{i\in P} \frac{1+\sigma_{P,i}x_i}{2},$$

where $\sigma_{P,i}$ is $+1$ or -1, depending on whether the path leading out of node labeled x_i on path P was labeled $+1$ or -1, and y_P is the label of the leaf at the end of the path P.

The only nonzero coefficients in g are of the from $\prod_{i\in T} x_i$ for some $T \subseteq P$ for some path P. This also means that the only non-zero Fourier coefficients can be those corresponding to such subsets. This is because $\mathbb{E}_{x\sim\mu}[\chi_T^\mu(x)\prod_{i\in S} x_i] = 0$, unless $T \subseteq S$ (because μ is a product distribution). Since the number of paths in g is at most t and the length of each path is at most d, we get the required result. □

Lemma 8.7.6. *Let f be a t-leaf decision tree, let g be a truncation of f to depth $\log(4t/\tau)/\log(1/(1-c))$. Then,*

$$\sum_{S, S\notin\mathcal{S}_g^\mu} \hat{f}_\mu(S)^2 \leq \tau$$

Proof. Let g be a truncation of f at depth $\log(4t/\tau)/\log(1/(1-c))$. Let $\mathcal{S}_g^\mu$ denote the set of non-zero Fourier coefficients of g under distribution μ. Using Lemma 8.7.4, we know that $\Pr_{x\sim\mu}[f(x) \neq g(x)] \leq \tau/4$, hence $\mathbb{E}[(f(x)-g(x))^2] \leq \tau$. Now, by Parseval's identity:

$$\begin{aligned}\tau &\geq \mathbb{E}_{x\sim\mu}[(f(x)-g(x))^2] \\ &= \sum_{S\subseteq\mathcal{S}_g} (\hat{f}(S)-\hat{g}(S))^2 + \sum_{S\notin S_g} \hat{f}(S)^2 \\ &\geq \sum_{S\notin S_g} \hat{f}(S)^2\end{aligned}$$

The proof is complete by observing that every coefficient $S \in \mathcal{S}_g^\mu$ satisfies $|S| \leq \log(4t/\tau)/\log(1/(1-c))$ by Lemma 8.7.5. □

L_2 **Test**: As in the case of uniform distribution, we write $f(x)$ as:

$$f(x) = f^\mu_{-S}(x) + \chi^\mu_S(x) f^\mu_S(x),$$

where, $f^\mu_{-S}(x) = \sum_{T,S\not\subseteq T} \hat{f}^\mu(T)\chi^\mu_T(x)$ and $f^\mu_S(x) = \sum_{T\supseteq S} \hat{f}^\mu(T)\chi^\mu_{T\setminus S}(x)$. Then as in the case of uniform distribution, $f_S(x) = f_S(x_{-S}) = \mathbb{E}_{x_S\sim\mu_S}[f(x)\chi^\mu_S(x)]$, where now x_S is drawn from the restriction μ_S of the product distribution to the bits x_S. Note that for any given point x, $f_S(x)$ can be computed easily using $2^{|S|}$ membership queries that are $|S|$-local (since only the bits x_S need to be changed). We point out that there is a subtle point in the case of product distributions. Recall that $f_S(x) = \mathbb{E}_{x_S\sim\mu_S}[f(x)\chi^\mu_S(x)]$. In the case when μ is the uniform distribution, the parity functions, χ_S are $\{-1,1\}$ valued, and so $f_S(x) \in [-1,1]$. Thus, application of Chernoff-Hoeffding bounds is straightforward. In the case, of product distributions the range of $\chi^\mu_S(x)$ can be $[-\prod_{i\in S}((1-|\mu_i|)/(\sqrt{1-\mu_i^2})), \prod_{i\in S}((1+|\mu_i|)/(\sqrt{1-\mu_i^2}))]$. Since, we never consider sets S that are larger than $O(\log(n/\epsilon))$, the range of f_S in our case is still polynomially bounded and arbitrarily good (inverse polynomial) estimates to the true expectation of $\mathbb{E}_{x\sim\mu}[f_S(x)^2]$ can be obtained by taking a sample and applying Chernoff-Hoeffding bounds. Thus, to simplify the presentation, we assume we can compute the expectation (in Line 2.a.ii in Fig. 8.4) and the Fourier coefficients exactly.

Theorem 8.7.7 is the statement of the formal result about learning decision trees under product distributions. The main ideas are similar to the proof in the case of uniform distribution; but, the proof is more involved as explained above.

Theorem 8.7.7. *Algorithm in Fig. 8.4 with parameters* $\theta = \sqrt{\epsilon/(2t(8t/\epsilon)^{1/\log(1/(1-c))})}$, $d = \log(8t/\epsilon)/\log(1/(1-c))$, *outputs a hypothesis* $\mathrm{sign}(h(x))$, *such that* $\mathbf{err}_\mu(\mathrm{sign}(h(x)), f) \leq$

ϵ. The running time of the algorithm is polynomial in n, t and $1/\epsilon$ and the algorithm makes only $O(\log(nt/\epsilon))$-local membership queries to the oracle $\mathsf{MQ}(f)$.

The rest of this section is devoted to the proof of Theorem 8.7.7.

Claim 8.7.8. *If S is such that $|\hat{f}^{\mu}(S)| \geq \theta$ and $|S| \leq d$, then $S \in \mathcal{S}$.*

Proof. This proof is thee same as the proof of Claim 8.3.2. □

Claim 8.7.9. *If $S \in \mathcal{S}$, then there exists $S' \supseteq S$, such that $\hat{f}^{\mu}(S')^2 \geq (\theta^2/2)/(t \cdot (8t/\theta^2)^{1/\log(1/(1-c))})$ and $|S'| \leq \log(8t/\theta^2)/\log(1/(1-c))$.*

Proof. Let $\tau = \theta^2/2$ and let g' be the decision tree obtained by truncation of f as described in Lemma 8.7.6. Then, by Lemma 8.7.5, we know the depth of g' is $\log(8t/\theta^2)/\log(1/(1-c))$ and that $\mathcal{S}^{\mu}_{g'}$ is of size at most $t \cdot 2^{\log(8t/\theta^2)/\log(1/(1-c))} = t \cdot (8t/\theta^2)^{1/\log(1/(1-c))}$. Also, by Lemma 8.7.6 we know that $\sum_{T \notin \mathcal{S}^{\mu}_{g'}} \hat{f}^{\mu}(T)^2 \leq \theta^2/2$, and hence if S passes the L_2-test, i.e. $\sum_{T \supseteq S} \hat{f}^{\mu}(T)^2 \geq \theta^2$, it must be that $\sum_{T \supseteq S, T \in \mathcal{S}^{\mu}_{g'}} \hat{f}^{\mu}(T)^2 \geq \theta^2/2$. Hence, there must be some set S' of size at most $\log(8t/\theta^2)/\log(1/(1-c))$ for which $\hat{f}^{\mu}(S')^2 \geq (\theta^2/2)/(t \cdot (8t/\theta^2)^{1/\log(1/(1-c))})$. □

Proof of Theorem 8.7.7. Let g be the truncation of the target decision tree, f, to depth d. Then using Lemma 8.7.6, we know that $\sum_{S \notin \mathcal{S}^{\mu}_{g'}} \hat{f}^{\mu}(S)^2 \leq \epsilon/2$. Now, every coefficient in $S \in \mathcal{S}^{\mu}_{g}$ for which $|\hat{f}^{\mu}(S)| \geq \theta$ is in $\mathcal{S}$ (see Algorithm 8.4 and Claim 8.7.8). $|\mathcal{S}^{\mu}_{g'}| \leq t2^d$. Tedious calculations show that $\sum_{S \in \mathcal{S}^{\mu}_{g'}, |\hat{f}(S)| < \theta} \hat{f}(S)^2 \leq t2^d\theta^2 \leq \epsilon/2$. Thus, $\sum_{S \in \mathcal{S}} \hat{f}(S)^2 \geq \sum_{S \in \mathcal{S} \cap \mathcal{S}^{\mu}_{g'}} \hat{f}(S)^2 \geq 1 - \epsilon$. This implies by Parseval, that $\mathbb{E}_{x \sim \mu}[(h(x) - f(x))^2] \leq \epsilon$, where $h(x)$ is as defined in Algorithm 8.4.

The only thing remaining to show is that $|\mathcal{S}|$ always remains bounded by $\mathrm{poly}(t, n, 1/\epsilon)$. This can be shown easily using Claim 8.7.9, since if $S \in \mathcal{S}$, there exists $S' \supseteq S$, such that $|S'| \leq \log(8t/\theta^2)/\log(1/(1-c))$ and $\hat{f}^{\mu}(S)^2 \geq (\theta^2/2)/(t \cdot (8t/\theta^2)^{1/\log(1/(1-c))}$. Thus, the magnitude of $\hat{f}(S')^2$ is at least $1/\mathrm{poly}(t, n, 1/\epsilon)$, so by Parseval there can be at most $\mathrm{poly}(t, n, 1/\epsilon)$. Also the size of $|S'|$ is $O(\log(tn/\epsilon))$, thus the total number of irrelevant subsets added to $\mathcal{S}$ is at most $\mathrm{poly}(t, n, 1/\epsilon)$. □

8.7.3 Learning under Random Classification Noise

In this section, we show how the algorithms for learning decision trees can be implemented even with access to a noisy oracle. The learning algorithm we use is allowed queries to

the membership oracle, $\mathsf{MQ}(f)$, therefore we consider a persistent random noise model. An easy way to conceptualize this model is as follows: Let $\zeta : \{-1,1\}^n \to \{-1,1\}$ be a function where for each $x \in \{-1,1\}^n$, the value of $\zeta(x) = 1$ with probability $1-\eta$ and -1 with probability η, independently. Once this noise function, ζ, has been fixed, we assume that we have access to the function: $f^\eta = f \cdot \zeta$, rather than the function f. We show how the tests mentioned in this section can be implemented using $\mathsf{EX}(f^\eta, D)$ and $\mathsf{MQ}(f^\eta)$, rather than $\mathsf{EX}(f, D)$ and $\mathsf{MQ}(f)$.

Non-Zero Test

Recall that we are interested in estimating $\Pr[f_S(x) \neq 0]$, where $S \subseteq [n]$, and

$$f_S(x) = \mathbb{E}_{x_S \sim U_S}[f(x)\chi_S(x)] \tag{8.3}$$

Instead, if we have access to f^η, we are able to compute,

$$f_S^\eta(x) = \mathbb{E}_{x_S \sim U_S}[f^\eta(x)\chi_S(x)]$$

Although, the random classification noise is persistent and fixed according to ζ, for the purpose of analysis it is easier to imagine that for each x, $\zeta(x)$ is only determined when the algorithm makes a query for the point x (or x is drawn by $\mathsf{EX}(f^\eta, D)$). Lemma 8.7.10 allows us to conclude that the L_2 test can be performed using access to f^η instead of f. The lemma assumes that $\zeta(x)$ is chosen independently, each time x is queried, *i.e.* the noise is not *persistent*. However, we show later that our algorithm queries each example only once, so the noise may as well have been persistent.

Lemma 8.7.10. *The following are true:*

1. $\Pr_{x,\zeta}[f_S^\eta(x) \neq 0] \geq (1-p_0) + \frac{(2\eta-1)^2 c_0}{2^{3|S|/2}} \Pr[f_S(x) \neq 0]$
2. $\Pr_{x,\zeta}[f_S^\eta(x) \neq 0] \leq (1-p_0) + \Pr[f_S(x) \neq 0]$

Here, c_0 is an absolute constant, p_0 depends only on $|S|$ and η. The probability is taken over the choice of $x \sim D$ and choice of ζ.

Proof. Note that $f_S(x) = \mathbb{E}_{x_S \sim U_S}[f(x)\chi_S(x)]$, and so $f_S(x)$ is evaluated by using $2^{|S|}$ different values of $f(x)$. For every x, $f(x) \in \{-1,1\}$, and hence if $f_S(x) = 0$, it must be that the $2^{|S|}$ values used in the expectation have exactly $2^{|S|-1}$ $+1$s and -1s each.

On the other hand, if $f_S(x) \neq 0$, then the number of $+1$s is different than -1s. If $f_S(x) \neq 0$, without loss of generality, we only consider the case when $f_S(x) > 0$, so that there are more $+1$s than -1s. Thus, we are left with the following combinatorial question:

Suppose we begin with $2k$ variables, $x_1, \ldots, x_{2k}$, where each x_i is $+1$ or -1. Let k_1 be the number of $+1$s and $2k - k_1$ is the number of -1s. We will assume throughout that $k \geq 2$. We perform the following process, each x_i is left as is with probability $1 - \eta$ and its sign flipped with probability η, independently. Let x'_i be the values of the resulting variables, and let $X' = \sum_i x'_i$. Let p_i^k denote the probability that X' is 0 having started with $(k+i)$ $+1$s and $(k-i)$ -1s. Thus, p_0^k is the probability of getting a 0, when we start with equal number of $+1$s and -1s.

Then the following are true:

1. p_i^k decreases as i increases.
2. $p_0^k - p_1^k \geq (2\eta - 1)^2 c_0 / k^{3/2}$ for some absolute constant c_0.

For proof of the above facts see Lemmas 8.7.12 and 8.7.13 below, though it should be fairly clear that the conclusions make sense. When $\eta = 1/2$, the initial values are irrelevant of the x_i are irrelevant and each $x'_i = \pm 1$ with probability $1/2$, but for $\eta < 1/2$, if one started with the sum $\sum_i x_i = 0$, it is more likely that $\sum_i x'_i = 0$, than if one started from some value, $\sum_i x_i$ that was greater than 0.

We apply the above to the setting when $k = 2^{|S|-1}$. We drop the superscripts $p_0^{2^{|S|-1}}$ and $p_1^{2^{|S|-1}}$ in the rest of this discussion. First, imagine that we have fixed the variables x_{-S} so that the expectation (8.3) is only a function of the noise function ζ. If $f_S(x_{-S}) = 0$, then $\Pr_\zeta[f_S^\eta(x_{-S}) = 0] = p_0$. On the other hand, if $f_S(x_{-S}) \neq 0$, then $0 \leq \Pr_\zeta[f^\eta(x_{-S}) = 0] \leq p_1$. So, we have the following:

$$\begin{aligned}
\Pr_{x,\zeta}[f_S^\eta(x) \neq 0] &\geq \Pr_x[f_S(x) \neq 0](1 - p_1) + \Pr_x[f_S(x) = 0](1 - p_0) \\
&= (1 - p_0) + (p_0 - p_1) \Pr_{x \sim D}[f_S(x) \neq 0]
\end{aligned}$$

On the other hand,

$$\begin{aligned}
\Pr_{x,\zeta}[f_S^\eta(x) \neq 0] &\leq \Pr_x[f_S(x) \neq 0] + (1 - p_0) \Pr_x[f_S(x) \neq 0] \\
&\leq (1 - p_0) + \Pr_x[f_S(x) \neq 0]
\end{aligned}$$

This completes the proof of the assertion. □

We note that this allows us to distinguish between the cases where $\Pr_{x\sim D}[f_S(x) \neq 0] \geq \alpha$ from $\Pr_{x\sim D}[f_S(x) \neq 0] \leq \beta$, as long as $\alpha - \beta$ is sufficiently large. This can be done by choosing $\beta = \alpha \cdot (2\eta - 1)^2 c_0/(2 \cdot 2^{3|S|/2})$, and then computing the value $\Pr_{x\sim D}[f_S^\eta(x) \neq 0]$. Note that p_0 can be computed exactly, if the size $|S|$ and the noise rate η are known. We assume that the noise rate is known; if not, the standard trick of binary searching the noise rate can be employed. Note that these tests can be carried out to high accuracy from samples. Now, in the case when D is an locally α-smooth distribution for constant α, any two points x and x' drawn from $\mathsf{EX}(f, D)$ will have Hamming distance $\Omega(n)$ with very high probability. The local queries to $\mathsf{MQ}(f^\eta)$ are only made for points that are at Hamming distance $O(\log(n))$ from sampled points (see Fact 8.1.2). Thus, with very high probability, the queries made to compute $f_S^\eta(x)$ and $f_S^\eta(x')$ do not have any point in common, *i.e.* no example is queried twice by the learning algorithm. So we can employ Lemma 8.7.10 as if the noise was chosen independently each time a point was queried.

L_2 Test

Recall that $f_S^\eta(x_{-S}) = \frac{1}{2^{|S|}} \sum_{x_S \in \{-1,1\}^{|S|}} [f^\eta(x_S x_{-S})]$. For a fixed x_{-S}, $f^\eta(x_{-S})$ is a random variable depending only on the noise function ζ. Let $2^{|S|} f_S(x) = 2k$, where $2k$ is some even integer in the range $[-2^{|S|}, 2^{|S|}]$. Let $k_1 = 2^{|S|-1} + k = 2^{|S|-1}(1 + f_S(x))$ and $k_2 = 2^{|S|-1} - k = 2^{|S|-1}(1 - f_S(x))$, so that $2^{|S|} f_S(x)$ is a sum of k_1, $+1$s and k_2, -1s. Let $Z_1 \sim \mathsf{Bin}(k_1, \eta)$ and $Z_2 \sim \mathsf{Bin}(k_2, \eta)$ be binomial random variables. Then $2^{|S|} f^\eta(x_{-S}) = 2^{|S|} f_S(x) - 2Z_1 + 2Z_2$. This follows immediately from the definition of the noise model. The following can then be verified by straightforward calculations,

$$\mathbb{E}_\zeta[f_S^\eta(x_{-S})] = (1 - 2\eta) f_S(x)$$
$$\mathbb{E}_\zeta[f_S^\eta(x_{-S})^2] = (1 - 2\eta)^2 f_S(x)^2 + 2^{-|S|+1}\eta(1 - \eta)$$

Thus, if we can obtain accurate estimates of $\mathbb{E}_{x\sim D}[f_S^\eta(x)^2]$, we can also obtain accurate estimates of $\mathbb{E}_{x\sim D}[f_S(x)^2]$. Again, as in the previous case, we observe that the algorithm (with high probability) never makes a query twice for the same example. Thus, we can assume that the noise model is in fact not persistent. It is clear that $\mathbb{E}_{x\sim D}[f_S^\eta(x)^2]$ can be estimated highly accurately by sampling.

The proof of the following two lemmas are elementary and are omitted.

Lemma 8.7.11. *Suppose, $X_0 = 0$. Consider the following random walk, $X_{i+1} = X_i$ with probability $1 - \alpha$, $X_{i+1} = X_i + 2$, with probability $\alpha/2$ and $X_{i+1} = X_i - 2$, with probability $\alpha/2$, where $\alpha \in [0, 1/2]$. Then, for $i \geq 0$, $\Pr[X_n = 0] - \Pr[X_n = 2]$ is a decreasing function of α.*

The idea of the proof is to notice that the probability, $\Pr[X_n = 2j]$ follows a bell shaped curve, and the curve gets steeper (more mass is concentrated at 0) as α goes to 0.

Lemma 8.7.12. *Let $x_1, \ldots, x_{2n}$, be such that $x_1 = \cdots = x_{n+d} = 1$ and $x_{n+d+1} = x_{n+i+2} = \cdots x_{2n} = -1$. The sign of each x_i is flipped independently with probability $\eta < 1/2$, to get x'_i. Let p_d^n be the probability that the $\sum_i x'_i = 0$. Then for $d \geq 0$, as d increases, p_d^n decreases.*

This expresses the quite obvious idea that if the probability of flipping is less than half, then the further from 0 the initial sum ($\sum_i x_i$), the less likely it is that $\sum_i x'_i = 0$.

Lemma 8.7.13. *Let $x_1, \ldots, x_{2n}$, be such that $x_1 = \cdots x_{n+1} = 1$ and $x_{n+2} = \cdots = x_{2n} = -1$. The sign of each x_i is flipped independently with probability $\eta < 1/2$, to get x'_i. Let p_1 denote the probability that $\sum_i x'_i = 0$. Let $y_1, \ldots, y_{2n}$ be such that, $y_1 = \cdots y_n = 1$ and $y_{n+1} = \cdots = y_{2n} = -1$. Then, let y'_i be obtained by flipping y_i independently with probability $\eta < 1/2$, and let p_0 denote the probability that $\sum_i y'_i = 0$. Then $p_0 - p_1 \geq (2\eta - 1)^2 c_0 / n^{3/2}$, for some absolute constant c_0.*

Proof. First we leave aside the values, x'_n, x'_{n+1}, y'_n and y'_{n+1}. The remaining variables, both in the case of x_is and y_is, were obtained by starting with exactly $(n-1)$ $+1$s and $(n-1)$ -1s and flipping each independently with probability $\eta < 1/2$. We can form pairs of $(+1, -1)$, to get a random variable $z_i = x'_i + x'_{n+1+i}$, $i = 1, \ldots, n-1$, where $z_i = 0$ with probability $\eta^2 + (1-\eta)^2 > 1/2$, $z_i = +2$ with probability $\eta(1-\eta)$ and $z_i = -2$ with probability $\eta(1-\eta)$. (A similar argument can be made in the case of y'_is.) We can view the sum of these z_i random variables as a random walk described in Lemma 8.7.11, where $X_{i+1} = X_i$ with probability $\eta^2 + (1-\eta)^2$ and $X_{i+1} = X_i + 2$, with probability $\eta(1-\eta)$ and $X_{i+1} = X_i - 2$, with probability $\eta(1-\eta)$. Now, $p_1 = \Pr[X_{n-1} = 0](2\eta(1-\eta)) + \Pr[X_{n-1} = 2]\eta^2 + \Pr[X_{n-1} = -2](1-\eta)^2$. On the other hand, $p_0 = \Pr[X_{n-1} = 0](\eta^2 + (1-\eta)^2) + \Pr[X_{n-1} = 2]\eta(1-\eta) + \Pr[X_{n-1} = -2]\eta(1-\eta)$. Noticing that, $\Pr[X_{n-1} = 2] = \Pr[X_{n-1} = -2]$, we get that $p_0 - p_1 = (2\eta - 1)^2(\Pr[X_{n-1} = 0] - \Pr[X_{n-1} = 2])$. But this difference is a decreasing function of $\alpha = 1 - (\eta^2 + (1-\eta)^2)$. But, even when $\alpha = 1/2$, i.e. $\eta = 1/2$, this difference is given by,

$$\begin{aligned}
\Pr[X_{n-1} = 0] - \Pr[X_{n-1} = 2] &= \frac{1}{2^{2n-2}} \left(\binom{2n-2}{n-1} - \binom{2n-2}{n} \right) \\
&= \frac{1}{2^{2n}} \binom{2n-2}{n-1} \left(1 - \frac{n-2}{n} \right)
\end{aligned}$$

The claim now follows easily. □

Chapter 9

Conclusions

Several interesting directions and open problems come out of this thesis. We discuss them below.

In Chapter 3 we introduced and studied a notion of clustering stability called weak deletion-stability. We gave a PTAS for k-median and k-means clustering with runtime dependence exponential in $1/\alpha$. Also, recall that the k-median problem restricted only to weakly-stable instances has no FPTAS. So the fact that our algorithm's runtime has super-polynomial dependence in $1/\alpha$ is unavoidable. Nonetheless, one might still hope to do better. In particular, one major runtime expense of our algorithm comes from handling expensive clusters by brute-force guessing or sampling. Can one improve the runtime by doing something more clever for expensive clusters? It is worth noting that for the stability conditions of Balcan et al. [2009a], Voevodski et al. [2010] develop an especially efficient implementation with good performance (in terms of both accuracy and speed) on real-world protein sequence datasets.

A different open problem lies in the relation to results of Ostrovsky et al. [2006]. Their motivating question was to analyze the performance of Lloyd-type methods over stable instances. Is it possible that weak deletion-stability is sufficient for a version of the k-means heuristic to converge to the optimal clustering? Finally, we present the high level question of extending this line of work to other problems. Can stability assumptions, preferably ones of a mild nature, allow us to bypass NP-hardness results of other problems? One particularly intriguing direction is the problem of Sparsest-Cut, for which no PTAS or constant-approximation algorithm is known, yet a powerful heuristics based on spectral techniques work remarkably well in practice (Shi and Malik [1997]).

In Chapter 5 we motivated and studied a model for designing local interactive clus-

tering algorithms. We designed efficient algorithms for data sets satisfying a natural stability property. It would be interesting to relax the condition on η that is required by our algorithms in the η-merge model. We would also like to develop algorithms in the unrestricted-merge model for arbitrary request sequences.

It is also important to study additional properties of an interactive clustering algorithm that are desirable in practice. For instance, one might require that the algorithm never output an intermediate clustering with error larger than the error of the initial clustering. Our algorithms for the unrestricted-merge model have this property, but the ones for the η-merge model do not. Finally, a clustering algorithm may process the split/merge requests in batches, collected over a period of time. One can potentially design algorithms in this batch setting with better run-time bounds.

In Chapter 7 we presented an algorithm for learning the class of disjunctions in the case that $\mathsf{OPT} < n^{-(1/3+\alpha)}$, achieving an error rate of $O(n^{1/3+\alpha} \cdot \mathsf{OPT}) + \epsilon$. The natural open question is whether one can improve this bound. For example, can one achieve weak agnostic learning for $\mathsf{OPT} = n^{-1/4}$? Or, can one improve the bounds as a function of the number of relevant variables, e.g., making only a factor $O(r^{0.9})$ times more mistakes than the best disjunction?

An intriguing open question is whether one can extend this technique for other concept classes. For example, consider the class of linear separators over $\{0,1\}^n$ with weights in $\{0,1\}$ (i.e., majority vote or "k of r" functions). Here we do not know even how to achieve weak learning for $\mathsf{OPT} = n^{-0.99}$. The algorithm presented in this thesis for disjunctions uses the fact that in order for individual variables not to be weak hypotheses themselves, the bad negative examples must in some sense "point" in the direction of the target vector (they must have a high dot-product with the target function vector if we view the disjunction as a linear threshold function) to a substantially greater extent than the positive examples do. E.g., if a typical positive example has t relevant variables set to 1, then the typical bad negative example must have t/OPT relevant variables set to 1. For the case of majority-vote functions, the difficulty with this approach is that instead all we can say is that if the positive examples have $r/2 + t$ relevant variables set to 1, then the typical bad negative examples should have at least $r/2 + t/\mathsf{OPT}$ relevant variables set to 1, which might not be such a distinction in a multiplicative sense.

On a more general note, our work here uses somewhat non-traditional hypotheses, by using the examples themselves to define "slices" of the data (focusing on those examples with no more than a certain θ dot-product with some given negative example). Perhaps this might be useful for other learning problems.

In Chapter 8 we introduced the local membership query model, with the goal of study-

ing query algorithms that may be useful in practice. With the rise of crowdsourcing tools, it is increasingly possible to get human labelers for a variety of tasks. Thus, membership queries beyond the standard active learning paradigm could prove to be useful to increase the efficiency and accuracy of learning. In order to make use of human labelers, it is necessary to make queries that make sense to them. In some ways, our algorithms can be understood as searching for higher-dimensional (deeper) features using queries that modify the examples locally.

Our model of local membership queries is also a very natural and simple theoretical model. There are several interesting open questions: (i) can the class of t-leaf decision trees (without depth restriction) be learned under the class of log-Lipschitz distributions? (ii) is the class of DNF formulas learnable in polynomial time, at least under the uniform distribution? Another interesting question is whether a general purpose boosting algorithm exists that only uses α-log-Lipschitz distributions. This looks difficult since most boosting algorithms decrease weights of points substantially[1].

It is also interesting to see whether agnostic learning of any interesting concept classes is possible in this learning model. Our results show that constant local queries are not useful for agnostic learning. However can $O(\log(n))$-local queries help in learning $O(\log(n))$-sized parities in the agnostic setting? We observe that learning the class of $O(\log(n))$-sized parities and the class of decision-trees is equivalent in the agnostic learning setting (even under locally smooth distributions), since weak and strong agnostic learning is equivalent even with respect to a fixed distribution (Kalai and Kanade [2009], Feldman [2010]). Agnostic learning $O(\log(n))$-sized parities (even with respect to a fixed distribution) would also imply (PAC) learning DNF in our model with local membership queries (with respect to the same distribution) (Kalai et al. [2009a]).

[1]Note that Smooth-boosting Servedio [2003] does not use distributions that are log-Lipschitz.

Bibliography

D. Achlioptas and F. McSherry. On spectral learning of mixtures of distributions. In *Proceedings of the 18th Annual Conference on Learning Theory*, 2005. 4

N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *Advances in Neural Information Processing Systems*, 2009. 2.2

D. Aldous and U. Vazirani. A markovian extension of valiant's learning model. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, 1990. 8

D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1998. 4

Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75 (2):87–106, 1987. 6.0.5

Dana Angluin and Philip Laird. Learning from noisy examples. *Mach. Learn.*, 2(4), April 1988. 6.0.5, 6.0.5, 6.0.7

Dana Angluin, James Aspnes, Jiang Chen, and Wu Yinghua. Learning a circuit by injecting values. In *STOC*, 2006. 8

S. Arora and R. Kannan. Learning mixtures of arbitrary gaussians. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, 2001. 4

Sanjeev Arora, Prabhakar Raghavan, and Satish Rao. Approximation schemes for Euclidean k-medians and related problems. In *STOC*, 1998. 2, 1

D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2007. 2.2, 2.2, 2.2.2

David Arthur and Sergei Vassilvitskii. How slow is the k-means method? In *Proceedings of the twenty-second annual symposium on Computational geometry*, SCG '06, 2006. 2.2

David Arthur, Bodo Manthey, and Heiko Röglin. Smoothed analysis of the k-means method. *J. ACM*, 58(5), October 2011. 2.2

V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k-median and facility location problems. *SIAM J. Comput.*, 33(3):544–562, 2004. 2

Vijay Arya, Naveen Garg, Rohit Khandekar, Adam Meyerson, Kamesh Munagala, and Vinayaka Pandit. Local search heuristic for k-median and facility location problems. In *Proc. 33rd Annual ACM Symposium on Theory of Computing (STOC)*, 2001. 3

Peter Auer and Manfred K. Warmuth. Tracking the best disjunction. In *Proc. 36th Symposium on Foundations of Computer Science*, pages 312–321, 1995. 7

Pranjal Awasthi and Reza Bosagh Zadeh. Supervised clustering. In *NIPS*, 2010. 4.2.2, 5.3

Pranjal Awasthi, Avrim Blum, and Or Sheffet. Stability yields a PTAS for k-median and k-means clustering. FOCS, 2010a. 3.2

Pranjal Awasthi, Avrim Blum, and Or Sheffet. Improved guarantees for agnostic learning of disjunctions. In *COLT*, 2010b. 7, 7.1

Pranjal Awasthi, Avrim Blum, and Or Sheffet. Center-based clustering under perturbation stability. *Inf. Process. Lett.*, 112(1-2), January 2012. 3.7

Pranjal Awasthi, Maria-Florina Balcan, and Konstantin Voevodski. Local algorothms for interactive clustering. In *Manuscript*, 2013a. 5

Pranjal Awasthi, Vitaly Feldman, and Varun Kanade. Learning using local membership queries. In *COLT*, 2013b. 8

Mihai Bādoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *STOC*, pages 250–257, 2002. ISBN 1-58113-495-9. 3

M.-F. Balcan, A. Blum, and S. Vempala. A discriminative framework for clustering via similarity functions. In *Proceedings of the 40th ACM Symposium on Theory of Computing*, 2008a. 4.4.1, 4.4.3

M.-F. Balcan, A. Blum, and A. Gupta. Clustering under approximation stability. In *Journal of the ACM*, 2013. (document), 3.7, 3, 3.2

Maria-Florina Balcan and Avrim Blum. Clustering with interactive feedback. In *ALT*, 2008. 1.1, 1.1.1, 4, 4.2, 4.2.1, 4.2.2, 4.2.3, 4.3.3, 4.4

Maria-Florina Balcan and Mark Braverman. Finding low error clusterings. In *COLT*, 2009. 3.3.2, 3.4

Maria-Florina Balcan and Pramod Gupta. Robust hierarchical clustering. In *COLT*, 2010. 5.4.3

Maria-Florina Balcan and Yingyu Liang. Clustering under perturbation resilience. *ICALP*, 2012. 3.7

Maria-Florina Balcan, Avrim Blum, and Santosh Vempala. A discriminative framework for clustering via similarity functions. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, 2008b. 5, 5.2

Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. Approximate clustering without the approximation. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2009a. 1.1, 1.1.1, 2, 3.1, 3.2, 3.3, 3.3.2, 3.3.2, 9

Maria-Florina Balcan, Heiko Röglin, and Shang-Hua Teng. Agnostic clustering. In *ALT*, pages 384–398, 2009b. 3.3.2

Mikhail Belkin and Kaushik Sinha. Polynomial learning of distribution families. In *FOCS*, 2010. 4

Yonatan Bilu and Nathan Linial. Are stable instances easy? In *ICS*, 2010. 3.7

A. Blum, A. Frieze, R. Kannan, and S. Vempala. A polynomial-time algorithm for learning noisy linear threshold functions. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, 1996. 6.0.7

A. Blum, A. Kalai, and H. Wasserman. Noise-tolerant learning, the parity problem, and the statistical query model. *Journal of the ACM*, 50(4):506–519, 2003. 8

Avrim Blum, Merrick Furst, Jeffrey Jackson, Michael Kearns, Yishay Mansour, and Steven Rudich. Weakly learning dnf and characterizing statistical query learning using fourier analysis. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, STOC '94, 1994. 8

Avrim Blum, Prasad Chalasani, Sally A. Goldman, and Donna K. Slonim. Learning with unreliable boundary queries. *J. Comput. Syst. Sci.*, 56, April 1998. 8

R. C. Bose and D. K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68–79, 1960. 8.5

Constantinos Boulis and Mari Ostendorf. Combining multiple clustering systems. In *In 8th European conference on Principles and Practice of Knowledge Discovery in Databases(PKDD), LNAI 3202*, 2004. 5

S. Charles Brubaker and Santosh Vempala. Isotropic pca and affine-invariant clustering. *CoRR*, abs/0804.3575, 2008. 4

David Bryant and Vincent Berry. A structured family of clustering and tree construction methods. *Adv. Appl. Math.*, 27(4), November 2001. 5

Nader H. Bshouty. Exact learning via the monotone theory (extended abstract). In *FOCS*, 1993. 1.2, 6.0.5, 8, 8

Nader H. Bshouty, Elchanan Mossel, Ryan O'Donnell, and Rocco A. Servedio. Learning dnf from random walks. *J. Comput. Syst. Sci.*, 71, October 2005. 8, 8.6

R.D. Carr, S. Doddi, G. Konjevod, and M. Marathe. On the red-blue set cover problem. In *Proceedings of the Eleventh Annual ACM-SIAM symposium on Discrete algorithms (SODA)*, pages 345–353, 2000. ISBN 0-89871-453-2. 2

M. Charikar, S. Guha, E. Tardos, and D. B. Shmoy. A constant-factor approximation algorithm for the k-median problem. In *ACM Symposium on Theory of Computing*, 1999a. 2

Moses Charikar, Sudipto Guha, Éva Tardos, and David B. Shmoys. A constant-factor approximation algorithm for the k-median problem. In *Proc. 31st Annual ACM Symposium on Theory of Computing (STOC)*, 1999b. 3

Bo Dai, Baogang Hu, and Gang Niu. Bayesian maximum margin clustering. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, 2010. 5

S. Dasgupta. Learning mixtures of gaussians. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999. 4

Sanjoy Dasgupta. The hardness of k-means clustering. Technical report, University of California at San Diego, 2008. URL `http://cseweb.ucsd.edu/Dienst/UI/2.0/Describe/ncstrl.ucsd_cse/CS2008-0916`. 3, 5

W. Fernandez de la Vega, Marek Karpinski, Claire Kenyon, and Yuval Rabani. Approximation schemes for clustering problems. In *STOC '03: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, 2003. 2, 3

Ilias Diakonikolas, Ryan O'Donnell, Rocco A. Servedio, and Yi Wu. Hardness results for agnostically learning low-degree polynomial threshold functions. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '11, 2011. 6.0.7

Z. Dvir, A. Rao, A. Wigderson, and A. Yehudayoff. Restriction access. In *ITCS*, 2012. 8

Michelle Effros and Leonard J. Schulman. Deterministic clustering with data nets. *ECCC*, (050), 2004. 3

Dan Feldman and Leonard J. Schulman. Data reduction for weighted and outlier-resistant clustering. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, 2012. 8

V. Feldman. Learning dnf expressions from fourier spectrum. In *COLT*, 2012. 8, 8.4, 8.4.1

V. Feldman, P. Gopalan, S. Khot, and A. Ponnuswami. New results for learning noisy parities and halfspaces. In *FOCS*, 2006. 8.5

Vitaly Feldman. *Efficiency and computational limitations of learning algorithms*. PhD thesis, Cambridge, MA, USA, 2007. AAI3251269. 6.0.7

Vitaly Feldman. On the power of membership queries in agnostic learning. In *COLT*, 2008. 8.5

Vitaly Feldman. Optimal hardness results for maximizing agreements with monomials. *SICOMP*, 39(2), 2009. Extended abstract in CCC 2006. 6.0.7, 7

Vitaly Feldman. Distribution-specific agnostic boosting. In *1st Symposium on Innovations in Computer Science (ICS)*, pages 241–250, 2010. 7.1, 7.2.3, 7.2.4, 9

Vitaly Feldman, Venkatesan Guruswami, Prasad Raghavendra, and Yi Wu. Agnostic learning of monomials by halfspaces is hard. *SIAM J. Comput.*, 41(6):1558–1590, 2012. 6.0.7, 7

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, EuroCOLT '95, 1995. 6.0.4, 6.0.6

Dmitry Gavinsky. Optimally-smooth adaptive boosting and application to agnostic learning. *J. Mach. Learn. Res.*, 4:101–117, 2003. ISSN 1532-4435. 7.1, 7.2.3

Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33:792–807, 1986. ISSN 0004-5411. doi: http://doi.acm.org/10.1145/6490.6503. URL `http://doi.acm.org/10.1145/6490.6503`. 6, 8.6

Sudipto Guha and Samir Khuller. Greedy strikes back: Improved facility location algorithms. In *Journal of Algorithms*, pages 649–657, 1998. 3

Venkatesan Guruswami and Prasad Raghavendra. Hardness of learning halfspaces with noise. In *Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '06, 2006. 6.0.7

Sariel Har-Peled and Soham Mazumdar. On coresets for k-means and k-median clustering. In *STOC*, pages 291–300, 2004. ISBN 1-58113-852-0. 3, 1

J.A. Hartigan. Statistical theory in clustering. *Journal of Classification*, 2(1):63–76, 1985. ISSN 0176-4268. doi: 10.1007/BF01908064. 2.4, 2.4.1

David Haussler. Decision theoretic generalizations of the pac model for neural net and other learning applications. *Inf. Comput.*, 100(1), September 1992. 6.0.6

Katherine A. Heller and Zoubin Ghahramani. Bayesian hierarchical clustering. In *ICML*, 2005. 5, 5.4.2

A. Hocquenghem. Codes correcteurs d'erreurs (in french). *Chiffres*, 2:147–156, 1959. 8.5

Mary Inaba, Naoki Katoh, and Hiroshi Imai. Applications of weighted voronoi diagrams and randomization to variance-based k-clustering: (extended abstract). In *Proc. 10th Symp. Comp. Geom.*, pages 332–339, 1994. ISBN 0-89791-648-4. 2.2, 3.3.2, 3.6

J. Jackson. An efficient membership-query algorithm for learning DNF with respect to the uniform distribution. *Journal of Computer and System Sciences*, 55:414–440, 1997. 1.2, 6.0.5, 8, 8

Jeffrey C. Jackson and Karl Wimmer. New results for random walk learning. In *COLT*, 2009. 8

Kamal Jain, Mohammad Mahdian, and Amin Saberi. A new greedy approach for facility location problems (extended abstract). In *Proc. 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 731–740, 2002. 2, 3

N. Jardine and R. Sibson. Mathematical taxonomy. *New York*, 1971. 4.4.1

Adam Kalai and Varun Kanade. Potential-based agnostic boosting. In *NIPS*, 2009. 9

Adam Kalai, Varun Kanade, and Yishay Mansour. Reliable agnostic learning. In *COLT*, 2009a. 9

Adam Tauman Kalai, Adam R. Klivans, Yishay Mansour, and Rocco A. Servedio. Agnostically learning halfspaces. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '05, 2005. 6.0.7

Adam Tauman Kalai, Yishay Mansour, and Elad Verbin. On agnostic boosting and parity learning. In *STOC '08: Proceedings of the 40th annual ACM symposium on Theory of computing*, pages 629–638, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-047-0. doi: http://doi.acm.org/10.1145/1374376.1374466. 7.1, 7.2.3

Adam Tauman Kalai, Alex Samorodnitsky, and Shang-Hua Teng. Learning and smoothed analysis. In *Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '09, 2009b. 1.2.1, 8, 8, 8.4, 8.4.1, 8.7.2

Adam Tauman Kalai, Ankur Moitra, and Gregory Valiant. Efficiently learning mixtures of two gaussians. In *STOC*, 2010. 4

R. Kannan, H. Salmasian, and S. Vempala. The spectral method for general mixture models. In *Proceedings of the 18th Annual Conference on Learning Theory*, 2005. 4

Tapas Kanungo, David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A local search approximation algorithm for k-means clustering. In *Proc. 18th Symp. Comp. Geom.*, 2002. 2, 3

Michael Kearns and Leslie Valiant. Cryptographic limitations on learning boolean formulae and finite automata. *JOURNAL OF THE ACM*, 41:433–444, 1994. 1.2, 6.0.3

Michael J. Kearns and Umesh V. Vazirani. *An introduction to computational learning theory*. MIT Press, Cambridge, MA, USA, 1994. 7.2.1

Michael J. Kearns, Robert E. Schapire, and Linda M. Sellie. Toward efficient agnostic learning. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT '92, 1992. 6.0.6, 6.0.7

J. Kleinberg. An impossibility theorem for clustering. In *Advances in Neural Information Processing Systems 15: Proceedings of the 2002 Conference*, page 463. The MIT Press, 2003. 4.4.1

Adam R. Klivans, Philip M. Long, and Rocco A. Servedio. Learning halfspaces with malicious noise. In *ICALP*, 2009. 6.0.7

Vladlen Koltun and Christos H. Papadimitriou. Approximately dominating representatives. *Theor. Comput. Sci.*, 371(3), February 2007. 8

Akshay Krishnamurthy, Sivaraman Balakrishnan, Min Xu, and Aarti Singh. Efficient active algorithms for hierarchical clustering. *ICML*, 2012. 5

Amit Kumar and Ravindran Kannan. Clustering with spectral norm and the k-means algorithm. In *Proceedings of the 2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, FOCS '10, 2010. 1.1, 3.7

Amit Kumar, Yogish Sabharwal, and Sandeep Sen. A simple linear time (1+ ϵ) - approximation algorithm for k-means clustering in any dimensions. In *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2004. 2, 1, 3.4, 3.8.2

E. Kushilevitz and Y. Mansour. Learning decision trees using the Fourier spectrum. *SIAM Journal on Computing*, 22(6):1331–1348, 1993. 1.2, 6.0.5, 8, 8, 8.3

K. Lang and E. Baum. Query learning can work poorly when a human oracle is used. *IEEE International Joint Conference on Neural Networks*, 1992. 8

L. Levin. Randomness and non-determinism. *Journal of Symbolic Logic*, 58(3):1102–1103, 1993. 8

N. Littlestone. Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In *Proc. 4th Conference on Computational Learning Theory*, pages 147–156, Santa Cruz, California, 1991. Morgan Kaufmann. 7

Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2(4), 1987. 1.2.1, 7

S.P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inform. Theory*, 28(2):129–137, 1982. 2

Y. Mansour. An $o(n^{\log\log n})$ learning algorithm for dnf under the uniform distribution. In *COLT*, 1992. 8, 2

J. Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. Inform. Theory*, 15: 122–127, 1969. 8.5

Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems, 1984. 2.2

Ankur Moitra and Gregory Valiant. Settling the polynomial learnability of mixtures of gaussians. In *FOCS*, 2010. 4

R. Ostrovsky and Y. Rabani. Polynomial time approximation schemes for geometric k-clustering. In *FOCS*, 2000. ISBN 0-7695-0850-2. 3

Rafail Ostrovsky, Yuval Rabani, Leonard J. Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. In *Proc. 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2006. 1.1, 1.1.1, 2, 3.1, 3.2, 3.3, 3.3.1, 3.3.2, 3.4, 3.6, 3.7, 3, 9

David Peleg. Approximation algorithms for the label-covermax and red-blue set cover problems. *J. of Discrete Algorithms*, 5(1):55–64, 2007. ISSN 1570-8667. doi: http://dx.doi.org/10.1016/j.jda.2006.03.008. 1.2.1, 6.0.7, 7, 7.2, 7.2.1

Robert E. Schapire. The strength of weak learnability. *Mach. Learn.*, 5(2), July 1990. 6.0.6, 6.0.4, 6.0.6

Robert E. Schapire and Linda M. Sellie. Learning sparse multivariate polynomials over a field with queries and counterexamples. In *COLT*, pages 17–26, 1996. 6.0.5, 8

Rocco A. Servedio. Smooth boosting and learning with malicious noise. *J. Mach. Learn. Res.*, 4:633–648, December 2003. ISSN 1532-4435. doi: http://dx.doi.org/10.1162/153244304773936072. URL `http://dx.doi.org/10.1162/153244304773936072`. 1

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22:888–905, 1997. 9

Daniel A. Spielman and Shang-Hua Teng. Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time. *J. ACM*, 51(3), 2004. 2.2, 8

Matus Telgarsky and Sanjoy Dasgupta. Agglomerative bregman clustering. *ICML*, 2012. 5, 5.4.2

L. G. Valiant. A theory of the learnable. *Commun. ACM*, 27(11), 1984. 1.2, 6, 6.0.1, 6

V. N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons Inc., 1998. 4.2.3

Andrea Vattani. k-means requires exponentially many iterations even in the plane. In *Proceedings of the 25th annual symposium on Computational geometry*, SCG '09, 2009. 2.2

Konstantin Voevodski, Maria Florina Balcan, Heiko Roglin, ShangHua Teng, and Yu Xia. Efficient clustering with limited distance information. In *Proc. 26th UAI*, 2010. 9

X. Wu, V. Kumar, J. Ross Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. The top ten algorithms in data mining. *Knowledge and Information Systems*, 2008. 2.2

Reza Bosagh Zadeh and Shai Ben-David. A Uniqueness Theorem for Clustering. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, 2009. 4.4.1

Reza Bosagh Zadeh and Shai Ben-David. Axiomatic Characterizations of Single-Linkage. In *Journal of Machine Learning Research*, 2011. 4.4.1

Shi Zhong. Generative model-based document clustering: a comparative study. *Knowledge and Information Systems*, 2005. 5