View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by D-Scholarship@Pitt

Relational Machine Learning Algorithms

by

Alireza Samadianzakaria

Bachelor of Science, Sharif University of Technology, 2016

Submitted to the Graduate Faculty of

the Department of Computer Science in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

University of Pittsburgh

2021

UNIVERSITY OF PITTSBURGH

DEPARTMENT OF COMPUTER SCIENCE

This dissertation was presented

by

Alireza Samadianzakaria

It was defended on

July 7, 2021

and approved by

Dr. Kirk Pruhs, Department of Computer Science, University of Pittsburgh

Dr. Panos Chrysanthis, Department of Computer Science, University of Pittsburgh

Dr. Adriana Kovashka, Department of Computer Science, University of Pittsburgh

Dr. Benjamin Moseley, Tepper School of Business, Carnegie Mellon University

Copyright © by Alireza Samadianzakaria
2021

Relational Machine Learning Algorithms

Alireza Samadianzakaria, PhD

University of Pittsburgh, 2021

The majority of learning tasks faced by data scientists involve relational data, yet most standard algorithms for standard learning problems are not designed to accept relational data as input. The standard practice to address this issue is to join the relational data to create the type of geometric input that standard learning algorithms expect. Unfortunately, this standard practice has exponential worst-case time and space complexity. This leads us to consider what we call the Relational Learning Question: "Which standard learning algorithms can be efficiently implemented on relational data, and for those that cannot, is there an alternative algorithm that can be efficiently implemented on relational data and that has similar performance guarantees to the standard algorithm?"

In this dissertation, we address the relational learning question for the well-known problems of support vector machine (SVM), logistic regression, and k-means clustering. First, we design an efficient relational algorithm for regularized linear SVM and logistic regression using sampling methods. We show how to implement a variation of gradient descent that provides a nearly optimal approximation guarantee for stable instances. For the k-means problem, we show that the k-means++ algorithm can be efficiently implemented on relational data, and that a slight variation of adaptive k-means algorithm can be efficiently implemented on relational data while maintaining a constant approximation guarantee. On the way to developing these algorithms, we give an efficient approximation algorithm for certain sum-product queries with additive inequalities that commonly arise.

Table of Contents

List of Figures

Preface

First, I wish to thank my advisor Kirk Pruhs for his excellent guidance and teaching me how to research. Without his dedication and help, this dissertation was not possible.

The work presented in this dissertation is a result of collaboration with my advisor and multiple coauthors, and I am grateful to Mahmoud Abo-Khamis, Ryan Curtin, Sungin Im, Benjamin Moseley, and Yuyan Wang whose collaboration made this research possible. I would also like to show my gratitude to the committee members Panos Chrysanthis and Adriana Kovashka for taking time from their busy schedule to serve on my committee.

Finally, I would like to thank my friends and family members for their constant support and motivation, especially my wife Shadi for being the audience of all my practice presentations and proofreading my writings, and above all being my best friend.

The research in this dissertation is published in the following papers:

- Curtin, Ryan, Sungjin Im, Benjamin Moseley, Kirk Pruhs, Alireza Samadian. "Unconditional Coresets for Regularized Loss Minimization." In International Conference on Artificial Intelligence and Statistics, pp. 482-492. PMLR, 2020 [91].
- Abo-Khamis, Mahmoud, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. "Approximate Aggregate Queries Under Additive Inequalities." In Symposium on Algorithmic Principles of Computer Systems (APOCS), pp. 85-99. Society for Industrial and Applied Mathematics, 2021 [4].
- Abo-Khamis, Mahmoud, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. "A Relational Gradient Descent Algorithm For Support Vector Machine Training." In Symposium on Algorithmic Principles of Computer Systems (APOCS), pp. 100-113. Society for Industrial and Applied Mathematics, 2021 [5].
- Moseley, Benjamin, Kirk Pruhs, Alireza Samadian, and Yuyan Wang. "Relational Algorithms for k-means Clustering." To Appear In International Colloquium on Automata, Languages and Programming, 2021 [82].

1.0 Introduction

Kaggle surveys [1] show that the majority of learning tasks faced by data scientists involve relational data. Most commonly, the relational data is stored in tables in a relational database. The data is usually stored in a normalized form to prevent repetition, and it may have both numerical and categorical values. However, most of the machine learning algorithms need the data to be in a single table with all columns present. Furthermore, many machine learning algorithms such as linear regression and linear classifiers need the data to be numerical.

To use the traditional machine learning algorithms, the first step is a feature extraction query that consists of joining the tables and converting all columns to numerical values, which can be done with standard methods such as one-hot encoding [31]. Then the design matrix will be imported into a standard learning algorithm to train the model. Thus, conceptually, standard practice transforms a data science query to a query of the following form:

$$\textbf{Data Science Query} = \text{Standard\_Learning\_Algorithm}(\text{Design Matrix } J = T_1 \bowtie \cdots \bowtie T_m)$$

where the joins are evaluated first, and the learning algorithm is then applied to the result.

Forming the design matrix can increase the size of the join both because of the join itself and the implicit data. As an example, consider the problem of modeling user interests for YouTube videos based on comments and likes. For example, a data scientist might be interested to predict if a user likes a video based on the comments and the profile of the commenter. In this scenario, there is a table for the likes having the information about the person who likes the video and the video ID, and there is another table for the comments. The join of the two tables will have a row for each couple of comments and likes of each video, which means it will be much larger than the original tables.

As another example, you may consider a sales department that wants to predict the volume of sales for each item and each branch. In such scenarios, there might be tables for transactions, customers, items, and branches. The machine learning algorithm needs both positive cases (items that are sold in a branch to a customer) and negative cases (items that

are not sold). While the department has the information about the sold items, usually they do not store the items that they have not sold in each day. This information can be obtained by joining all other tables and excluding the combinations in the transaction table, and it can increase the size of the data dramatically.

Note that if each of the m tables in a join has n rows, then the design matrix J can have as many as n^m rows. A worst case example that can make this many rows is a join of m tables where each table T_i has three columns $(c_{2i-1}, c_{2i}, c_{2i+1})$ and n rows such that in all the rows the values of c_{2i-1} and c_{2i+1} are 0. Note that this type of join is the same as having a cross-product join. Thus, independent of the learning task, this standard practice necessarily has exponential worst-case time and space complexity as the design matrix can be exponentially larger than the underlying relational tables.

The above examples demonstrate that a relational database can be a highly compact data representation format. The size of J can be exponentially larger than the input size of the relational database [17]. Thus extracting J makes the standard practice potentially inefficient. Theoretically, there is a potential for exponential speed-up by running algorithms *directly* on the input tables of a join. However, formally defining what is a "relational" algorithm is problematic, as for each natural candidate definition there are plausible scenarios in which that candidate definition is not the "right" definition. However, for the purposes of this dissertation, it is sufficient to think of a "relational" algorithm as one whose runtime is polynomially bounded in n, m and d if the join is *acyclic*. Acyclic joins are defined formally in Section 2.3; intuitively, as we explain shortly, answering even the simplest questions on general joins is hard, and acyclicity is a commonly assumed condition that can abstract out the hardness associated with the structural complexity of the join.

Examples of problems for which there are relational algorithms are some of the aggregation queries, such as counting the number of rows or evaluating $J^T J$. These queries can be evaluated using Inside-Out algorithm in polynomial time if the join is acyclic [9, 7]. The explanation of the Inside-Out algorithm for acyclic joins can be found in Section 2.4.2. The Inside-Out algorithm is able to evaluate these queries in polynomial time in terms of the size of the input tables, which is asymptotically faster than the worst-case time complexity of any algorithm that joins the tables.

Most of the relational algorithms, including the ones discussed in this dissertation, can be extended to cyclic joins using the common technique of fractional hypertree decomposition, which is explained in Section 2.4. Luckily, most of the natural database joins are acyclic or nearly acyclic. Answering even simple queries on general (cyclic) joins, such as if the join is empty or not, is NP-Hard [50, 78]. To see this, consider the following reduction from 3-SAT: for each clause we construct a table having 3 columns, each representing one of the variables in that clause, and 7 rows that have the satisfying assignments of those variables in them. Then if the join of these tables has any row, that row would be a satisfying assignment to all of the clauses in 3-SAT.

For a general join, efficiency is commonly measured in terms of the *fractional hypertree width* of the join (denoted by "fhtw"), which measures how close the join is to being acyclic. Section 2.4 contains a formal definition of fractional hypertree width. This parameter is 1 for acyclic joins and is larger if the join is further from being acyclic. State-of-the-art algorithms for queries as simple as counting the number of rows in the design matrix have a linear dependency on n^{fhtw} in their time complexity, where n is the *maximum* number of rows in all input tables [9]. Therefore, in our study of relational learning algorithms, running in time linear in n^{fhtw} is the goal for general joins, as fundamental barriers need to be broken to be faster. Notice that this is a polynomial time complexity when fhtw is a fixed constant (i.e. nearly acyclic). The algorithms discussed in this dissertation have linear dependency on n^{fhtw}, matching the state-of-the-art.

The above definition of relational algorithms naturally leads to the algorithmic question of which problems admit relational algorithms. In this dissertation, we try to answer this question for some standard machine learning problems. More specifically, we consider linear support vector machine (SVM) and logistic regression, which are two famous linear classifiers, as well as k-means clustering which is a famous unsupervised machine learning model. Furthermore, we design a framework that can be used to approximately solve many problems relationally, such as counting the number of rows satisfying an inequality, and it can be used as a toolbox for designing other relational algorithms. This framework is also used in the algorithms we have designed for linear SVM and k-means clustering.

T_1	
f_1	f_2
1	1
2	1
3	2
4	3
5	4

T_2	
f_2	f_3
1	1
1	2
2	3
5	4
5	5

$J = T_1 \bowtie T_2$		
f_1	f_2	f_3
1	1	1
1	1	2
2	1	1
2	1	2
3	2	3

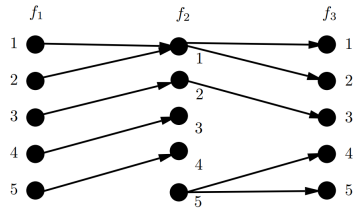

Figure 1: A Path Join With Two Tables.

1.1 An Illustrative Example

To illustrate the intuition behind the relational algorithms, consider the following special type of joins called *Path Joins*. One of the good properties of a path join is that it can be modeled as a layered directed acyclic graph (DAG). A path join $J = T_1 \bowtie \cdots \bowtie T_m$ consist of m tables, and each table T_i has two columns/features (f_i, f_{i+1}), therefore the design matrix J has $m+1$ columns. Furthermore, for simplicity, assume that each table has n rows. Then the join can be modeled as a layered DAG G in which there is one layer for each feature and one vertex v in layer i for each entry value that appears in the f_i column in either table T_{i-1} or table T_i.

Furthermore, in G, there is a directed edge between a vertex v in layer i and a vertex u in layer $i+1$ if and only if (v, u) is a row in table T_i. Then there is a one-to-one correspondence between the full paths in G, which are paths from layer 1 to layer d, and rows in the design matrix (the outcome of the join). Each node v in G is associated with weight w_v which is an entry of a table where v appears. For simplicity, think of the weights as being nonnegative. For an illustration of a path join and its analogy with DAGs, see Figure 1 which shows a specific instance in which $m = 2$ and $n = 5$. In particular, Figure 1 shows T_1, T_2, the design matrix J, and the resulting layered directed graph G.

Then in a path join, any query on the design matrix can be translated into a query over the set of paths in the corresponding DAG. For example, consider the problem of finding the nearest neighbour point. K-Nearest Neighbour (KNN) is a famous Machine Learning algorithm that can be used for regression and classification. The input for KNN consists of a collection of d dimensional points $X = (x_1, \ldots, x_N)$ with associated labels $Y = (y_1, \ldots, y_N)$, and a query point q. The output is the k closest rows in the design matrix to $q = [q_1, \ldots, q_d]$, and we measure the closeness in terms of Euclidean distance. For simplicity, in the following we consider the special case in which $k = 1$; however, the same algorithm can be generalized for any arbitrary k. The first step is replacing the values in layer i by the squared distance of them to q_i. Then 1NN query can be evaluated in two ways: (1) enumerating all paths in the DAG and find the shortest full path. (2) Use of Dijkstra's algorithm. The first solution is the same as forming the design matrix; whereas the second solution can be implemented as a relational algorithm that does not form the design matrix, and whose time complexity does not directly depend on the number of paths.

Another simple example is counting the number of rows in the design matrix. This can be done by a dynamic programming algorithm similar to Dijkstra's algorithm. We start from the left most layer and for each layer we count the number of incoming paths to each vertex. For each layer i the number of incoming path to a vertex v is the summation of the number of incoming path to all vertices u in layer $i - 1$ for which there is an edge (u, v).

1.2 The Relational Learning Question

The research question that we try to answer is the following, which we call the relational learning question:

A. Which standard learning algorithms can be implemented as relational algorithms, which informally are algorithms that are efficient when the input is in relational form?

B. And for those standard algorithms that are not implementable by a relational algorithms, is there an alternative relational algorithm that has the same performance guarantee as the standard algorithm?

C. And if we can't find an alternative relational algorithm that has the same performance guarantees to the standard algorithm, is there an alternative relational algorithm that has some reasonable performance guarantee (ideally similar to the performance guarantee for the standard algorithm)?

We address the relational learning question in the context of commonly used machine learning models. The models that we consider in this thesis are linear SVM, logistic regression, and k-means clustering. In general, one can study the relational machine learning question in the context of any machine learning algorithm that accepts geometric data or tabular data, and it can be further extended to any combinatorial problem with geometric input.

For any machine learning problem, the first question that we need to ask is whether any of the standard algorithms for this problem can be implemented relationally. An algorithm X is a relational implementation of another algorithm Y if X is a relational algorithm and on all inputs X produces the same output as does Y. On the other hand, one can show that plan A is not going to work by showing that relational implementations of the standard algorithms are not possible, or are unlikely to be possible. For example, one could show that a relational implementation would have unexpected complexity theoretic ramifications, e.g., P=NP.

When plan A fails for a problem, we look for an alternative relational algorithm that achieves a theoretical guarantee that is comparable to the theoretical guarantee achievable by the standard algorithm. If this is achieved, then to the extent that one accepts the standard algorithm because of this theoretical guarantee, one should accept the relational implementation to the same extent. Let us take as an example the case where the theoretical justification of the standard algorithm is a constant approximation guarantee; then a positive result for plan B would be a relational algorithm that outputs a different result which still has a constant approximation guarantee. An example of a problem for which plan B works is k-Means clustering. Our algorithm outputs differently than the standard adaptive k-Means algorithm, but it still provides a constant approximation. One can show that plan B is not going to work by showing that relational achieving this theoretical guarantee are not possible, or are unlikely to be possible.

If plan B fails, our final hope is to find a relational algorithm with some nonstandard theoretical guarantee. For example, if the standard algorithm achieves a constant approximation, a positive result for plan C might be be an algorithm that achieves a polylogarithmic approximation result, or achieves a constant approximation guarantee for some class of natural input instances. An example of a problem that we address, for which we adopt plan C, is a relational gradient-descent algorithm for linear SVM. The nonstandard theoretical guarantee supporting the algorithm is that it is guaranteed to converge only on stable instances, instead of on all instances.

In the following, we explain some of the related works done on relational machine learning, and then we explain our research question.

1.3 Related Works

The prior works can be divided into four subcategories: (1) designing theoretical frameworks that can be used as a toolbox for relational machine learning algorithms, (2) relational machine learning algorithms, (3) query processing systems, and (4) experimental results.

1.3.1 Theoretical Frameworks

One of the main frameworks that many relational algorithms use is SumProduct functional aggregation queries. This class of queries includes counting the number of rows in the design matrix, finding the closest point (or k closest points) to a specific point, and summing the values of one column in the design matrix. We have provided the formal definition in Section 2.4, and conceptually, in all of these queries, there is an outer operator called summation which is taken over the rows of the design matrix, and there is an inner operator called multiplication which is taken over the columns. The Inside-Out algorithm [9] can evaluate a SumProd query in time $O(md^2n^h \log n)$, where m is the number of tables, d is the number of columns, and h is the fractional hypertree width [51] of the query. Note that $h = 1$ for the acyclic joins, and thus Inside-Out is a polynomial-time algorithm for acyclic joins.

Using the SumProd queries and the Inside-Out algorithm, it is trivial to develop a relational K-Nearest Neighbour algorithm. The Inside-Out algorithm builds on several earlier papers, including [13, 41, 65, 51].

Functional aggregation query with additive inequalities (FAQ-AI) was first studied in [2]. This class of queries is similar to SumProd and SumSum queries, but the summation is taken over the rows of the design matrix that are satisfying a set of additive inequalities. For example, while counting the number of rows can be formulated as a SumProd query, counting the number of rows on one side of a hyperplane can be formulated as FAQ-AI. It is easy to formulate the computations necessary for training linear SVM or Lloyd's algorithm for K-means as FAQ-AI queries. [2] gave an algorithm to compute FAQ-AI in time $O(md^2n^{h'}\log(n))$ where h' is a relaxed hypertree width. The relaxed hypertree width h' is always greater than or equal to the hypertree width of the join query without the inequalities. However, note that one can consider any inequality condition as an infinite size table in the join. Then, h' is smaller than the hypertree width of the join query with the inequalities as input tables. The proposed algorithm for FAQ-AI has a worst-case time complexity $O(md^2n^{m/2}\log n)$ for a cross-product query. Therefore, this time complexity is better than the standard practice of forming the design matrix, which has worst-case time complexity $\Omega(dn^m)$; however, it is slower than Inside-Out which takes $O(md^2n\log n)$ for a cross-product query. Different flavors of queries with inequalities were also studied [64, 66, 8].

Many approximate algorithms for machine learning use sampling as part of their subroutine. Uniform sampling of rows from the design matrix without performing the join is considered [103]. The algorithm relies on counting the number of rows grouped by one of the input tables, which can be performed fast using SumProd queries. It is fast for the case of acyclic joins; however, it is similar to performing the join when the query is cyclic. This means for a cyclic query, the time to sample a row is the same as $\Omega(\mathrm{A})$ where A is the AGM bound of the query (See Section 2.3). Later, an algorithm for sampling from cyclic queries and join size estimation is introduced in [35]. After a linear time preprocessing, this algorithm can sample each row in the expected time of $O(A/O)$ where A is the AGM bound of the query and O is the number of rows in the design matrix. The work in [3] also introduces an algorithm for sampling from a join with a different time complexity guarantee; in partic-

ular, they consider instance optimality analysis in which they measure the time complexity of their algorithm with respect to the size of the *certificate* on the input tables instead of the size of the input tables.

1.3.2 Relational Machine learning

It has been shown that using repeated patterns in the design matrix, linear regression and factorization machines can be implemented more efficiently [89]. Later, [69] showed how to push linear regression learning through key foreign-key joins and they have experimentally shown speedup for synthetic data; however, they concluded that in some scenarios, the algorithm may not create any speedup. Furthermore, a system based on the algorithm in [69] and a relational algorithm for Naive Bayes was developed and tested in [68]. SystemF [93] is capable of pushing linear regression through arbitrary joins of multiple tables. Furthermore, [93] showed SystemF has a better performance compared to MADlib, Python, StatsModels, and R. Later [6] improved SystemF by utilizing functional dependencies between columns and experimentally evaluated the proposed algorithm against other methods such as SystemF and materializing the design matrix using Postgres over real datasets.

Using SumProd queries, the authors in [7] have introduced a unified framework for a class of problems including relational linear regression, polynomial regression, and singular value decomposition using SumProd queries. Conceptually, to solve linear regression, all we need is calculate $J^T J$ relationally and then use the resulting d by d resulting matrix explicitly. The proposed algorithm in [7] calculates $J^T J$ in time $O(d^4 m n^h \log n)$ when all columns are numerical. They have furthermore, used sparse tensor operators to handle categorical features more efficiently since a design matrix with categorical features can be seen as a sparse representation of a larger matrix where categorical features are converted to numerical values using one-hot encoding. The work in [7] builds on several earlier papers, including [69, 70, 43, 93] that all had theoretically and experimentally considered different aspects of relational linear regression. The work in [63], have further improved the time complexity of linear regression for 2 table joins using sketching and subspace embedding techniques, which provides an approximate solution for linear regression.

Relational machine learning is also considered in unsupervised models such as SVD, Gaussian Mixture Models, and K-means clustering. Let $J = U\Sigma V^T$ be the Singular Value Decomposition of J, then J^TJ can be calculated using the algorithm proposed in [7], and using J^TJ it is possible to obtain Σ and V. Note that U will be present implicitly, since the i-th row of U is $U_i = J_iV\Sigma^{-1}$.

In the case of polynomial regression, calculating the pairwise interaction of the features relationally can be time-consuming. MorpheusFI [72] makes this process faster and rewrites some of the common linear algebra operators when one of the operands is the design matrix with nonlinear (quadratic) features. This rewriting makes it possible to postpone the evaluation of nonlinear features, until the execution of the linear algebra operator and by doing them together relationally, MorpheusFI achieves speed-up.

Rk-means [38] was the first paper to give a nontrivial k-means algorithm that works on relational inputs. The paper gives an $O(1)$-approximation. The algorithm's running time has a superlinear dependency on k^d when the tables are acyclic and thus is not polynomial. Here k is the number of cluster centers and d is the dimension (a.k.a number of features) of the points; this is equivalently the number of distinct columns in the relational database.

Relational support vector machines with Gaussian kernels are studied in [102]. The algorithm utilizes Pegasos [97] which in each iteration samples a point uniformly and finds the gradient of SVM objective for that point; the algorithm in [102] finds the gradient relationally. While finding the gradient for a single point without a kernel can be done in $O(d)$, in the presence of a kernel function, this step can take $O(Nd)$; as a result, there is a speedup by implementing this step relationally. However, the number of iterations still should be proportional to the size of the design matrix, since in Pegasos every iteration samples only one row of the design matrix.

In [36], a relational algorithm is introduced for Independent Gaussian Mixture Models. Gaussian Mixture Models can be used for kernel density estimation by estimating the underlying distribution as a weighted average of k Gaussian distributions. In the case of Independent Gaussian Mixture Models, different dimensions are independent of each other and each Gaussian function can be written as the product of a set of functions, each depending on one column. Therefore, the whole probability function can be written as a SumProd query

and can be optimized relationally. [36] has also shown experimentally that this method will be faster than materializing the design matrix.

1.3.3 Query Processing and Distributed Systems

Another aspect of relational learning is designing systems and declarative languages for data analytic tasks over relational data. MADlib [59] is an open-source system that can perform various machine learning tasks inside a database, including some supervised and unsupervised learning models. Similarly, SQL4ml [75] is a system with SQL interface and it allows expressing machine learning queries in SQL. However, both MADlib and SQL4ml perform some of the machine learning optimization after doing the join, and therefore, neither is considered a relational machine learning algorithm based on our definition. A survey of declarative languages for data analysis, in general, can be found in [76].

LMFAO, introduced in [92], is a layered optimization scheme to optimize and execute multiple aggregation queries together. It has three groups of layers: converting the application to aggregation queries, logical optimization, and code optimization. The suggested optimizations in LMFAO can be applied to the existing relational algorithms for linear regression and polynomial regression. Furthermore, [92] has shown how to relationally solve problems such as training classification trees, regression trees, and obtaining mutual information of pairwise variables used for learning the structure of Bayesian networks.

A distributed version of SumProd queries for the special case of Boolean semiring is also studied in [71], and a bound on the number of rounds of communication needed for evaluating a SumProd query is presented. It has also been shown that minor changes in distributed/parallel relational database systems can make them capable of performing linear algebra operations in parallel [73].

1.3.4 Experimental

Some of the previously mentioned papers have also performed experiments on real datasets, and here we are going to enumerate some of their findings.

The experiments in [93] have compared the performance of System F which is a relational algorithm with the performance of MADlib [59], Python SStatModel,and R while performing linear regression on a retailer dataset, LastFM [33], and MovieLens [57]. For Python and R, they used PostgreSQL to join the tables before passing them to the library. They showed that System F is 3 to 200 times faster than the other libraries.

Later, the algorithm in [6] utilizes functional dependencies between columns, and it handles the categorical features and one hot encoding more efficiently. Therefore, it achieves a better run time on the retail dataset, compared to SystemF. The experiments have compared the algorithm in [6] MADlib, R, Tensorflow, and System F, and it has shown 3 to 729 time speedup for linear regression. Furthermore, they have compared their polynomial regression and factorization machine algorithm with MADlib and libFM, and their algorithm achieves more than 80 time speedup.

The performance of Rk-means [38] has also been experimentally compared with the performance of mlpack combined with postgreSQL on 3 different datasets and different values of k. They showed up to 100 times speed up while their relative error to mlpack solution was below 3 in all the datasets and values of k.

1.4 Our Results

While there are known relational algorithms for some standard machine learning problems such as training linear regression [7], there are many standard machine learning problems where relational algorithms are not known. In this thesis, we address the relational learning question for classic machine learning problems of linear SVM, logistic regression, and k-means. We now summarize these results.

1.4.1 Functional Aggregation Queries under Additive Inequality

To design an algorithm for linear SVM and k-means algorithm, we first introduce a more general framework for approximating some of the aggregate queries that are hard to com-

pute exactly. More specifically, we consider Functional Aggregation Queries under Additive Inequalities (FAQ-AI). Such queries/problems, with a smallish number of inequalities, arise naturally as subproblems in many standard learning algorithms. Before formally defining an FAQ-AI query, let us start with some examples. The first collection of examples are related to the classic Support Vector Machine problem (SVM), in which points are classified based on the side of a hyperplane that the point lies on [31, 95]. Each of the following examples can be reduced to FAQ-AI queries with one additive inequality:

- Counting the number of points correctly (or incorrectly) classified by a hyperplane.
- Finding the minimum distance of a correctly classified point to the boundary of a given hyperplane.
- Computing the gradient of the SVM objective function at a particular point.

And now we give some examples of problems related to the classic k-means clustering problem [95], in which the goal is to find locations for k centers so as to minimize the aggregate 2-norm squared distance from each point to its closest center. Each of the following examples can be reduced to FAQ-AI queries with $k-1$ inequalities:

- Evaluating the k-means objective value for a particular collection of k centers.
- Computing the new centers in one iteration of the commonly used Lloyd's algorithm.
- Computing the furthest point in each cluster from the center of that cluster.

All of these problems are readily solvable in nearly linear time in the size of the input if the input is the design matrix. Our goal is to determine whether relational algorithms exist for such FAQ-AI problems when the input is in a relational form.

In Chapter 3, we develop a framework for solving Functional Aggregation Queries with one Additive Inequality. We first show that solving FAQ-AI in general is $\#P$-Hard even for simple examples such as counting the number of rows in a cross-product join that lie on one side of a hyperplane. We also prove FAQ-AI with two or more additive inequalities are $\#P$-Hard to approximate up to any constant value.

Thus, we turn to approximately computing FAQ-AI queries. An ideal result would be what we call a *Relational Approximation Scheme* (RAS), which is a collection $\{A_\epsilon\}$ of relational algorithms, one for each real $\epsilon > 0$, such that each A_ϵ is outputs a solution that

has relative error at most ϵ. Our main result is a RAS for FAQ-AI(1) (an FAQ-AI query with only one additive inequality), that has certain natural properties defined in Chapter 3. Using the proposed RAS, it is possible to get a $1 \pm \epsilon$ approximation for queries such as

- Counting the number of points in a join lying on one side of a hyperplane, or lying inside a hypersphere.
- Finding the closest point to a given point q among the points lying on one side of a hyperplane.
- Summation of the distances of the points in a hypersphere from the center of the hypersphere (or any other given point)

1.4.2 RML Coresets

One of the main optimization algorithms for training machine learning models is gradient descent (See Chapter 2 for the definition). Unfortunately, as we show in Section 5.1, it is $\#P$-Hard to approximate the gradient of linear SVM up to any constant factor, and a similar proof can be applied for logistic regression. In fact, it can be shown that some simpler problems such as counting the number of points lying on one side of a hyperplane are also $\#P$-Hard. Therefore, instead of trying to directly find a relational implementation of gradient descent, we have investigated two different approaches:

A. Extracting a manageably small (potentially weighted) sample from the data set, and then directly solving (a weighted version of) the problem on the (weighted) sample.
B. Introducing a relational algorithm for those instances of SVM that have some stability properties.

In our first approach explained in Chapter 4, we consider a more general problem. Both logistic regression and linear SVM are special subclasses of Regularized Loss Minimization (RLM) problem [96] which can be defined as follow. The input consists of a collection $X = \{x_1, x_2, \ldots, x_n\}$ of points in $\Re^d$, and a collection $Y = \{y_1, y_2, \ldots, y_n\}$ of associated labels from $\{-1, 1\}$. Intuitively, the goal is to find a hypothesis $\beta \in \Re^d$ that is the best "linear" explanation for the labels. More formally, the objective is to minimize a function $F(\beta)$ that is a linear combination of a nonnegative nondecreasing loss function ℓ that measures the

goodness of the hypothesis, and a nonnegative regularization function r that measures the complexity of the hypothesis. That is

$$F(\beta) = \sum_{i=1}^{n} \ell(-y_i \beta \cdot x_i) + \lambda \, r(R\beta). \tag{1}$$

In the case of logistic Regression, the loss function is $\ell(z) = \log(1+\exp(z))$, and in the context of soft margin support vector machines (SVM), the loss function is $\ell(z) = \max(0, 1+z)$.

We try to extract a small sample from the data set and then directly solve the RLM problem on the weighted sample. The aspiration is that the optimal solution on the sample will be a good approximation to the optimal solution on the original data set. To achieve this aspiration, the probability that a particular point is sampled (and the weight that it is given) may need to be carefully computed as some points may be more important than other points. However, if the probability distribution is too complicated, it may not be efficiently implementable as a relational algorithm. A particularly strong condition on the sample that is sufficient for achieving this aspiration is that the sample is a *coreset*; intuitively, a sample is a coreset if *for all possible hypotheses* β, the objective value of β on the sample is very close to the objective value of β on the whole data set.

There has been work on constructing coresets for special cases of the RLM problem. In particular, sublinear coresets exist for *unregularized* logistic regression (i.e $\lambda = 0$) by making assumptions on the input. The exact assumption is technical, but intuitively the coresets are small when there is no hypothesis that is a good explanation of the labels. The work of [99] gave coresets for regularized soft-margin SVM assuming the 2-norm of the optimal β is small. Unfortunately, both of these works do not apply to general input instances. One may wonder if small coresets exist for general data sets. The work of [84] shows that there is no coreset of size $\Omega(\frac{n}{\log n})$ for unregularized logistic regression.

This lower bound is discouraging, suggesting that small coresets are not possible for arbitrary inputs even for the special case of the logistic regression problem. However, the lower bound is for unregularized logistic regression. In practice, regularization is almost always used, as emphasized in the following quotes. From Chapter 5. Basic Practice of [32]: "Regularization is the most widely used approach to prevent overfitting." Quoting Maya Gupta, head of the Glassbox Machine Learning team at Google from her online course on

machine learning, "The key ingredient to making machine learning work great... is regularization" [52].

Therefore, we show that if the regularizer's effect does not become negligible as the norm of the hypothesis scales, then a uniform sample of size $\Theta(n^{1-\kappa}\Delta)$ points is with high probability a coreset where we assume $\lambda = n^{\kappa}$. Here, Δ is the VC-dimension of the loss function. Thus, coresets exists for general input instances for the RLM problem, showing regularization allows us to break through the lower bounds shown in prior work! Formally, this scaling condition says that if $\ell(-\|\beta\|) = 0$ then $r(\beta)$ must be a constant fraction of $\ell(\|\beta\|_2)$. We show that this scaling condition holds when the loss function is either logistic regression or SVM, and the regularizer is the 1-norm, the 2-norm, or 2-norm squared. For example, in the recommended case that $\kappa = 1/2$, the scaling condition ensures that a uniform sample of $\tilde{\Theta}(d\sqrt{n})$ points is with high probability a coreset when the regularizer is one of the standard ones, and the loss function is either logistic regression and SVM, as they have VC-dimension $O(d)$. Note also that uniform sampling can be reasonably implemented in all of the popular restricted access models. As a consequence, this yields a reasonable algorithm for all of the restricted access models under the assumption that a data set of size $\tilde{\Theta}(d\sqrt{n})$ can be stored, and reasonably solved in the main memory of one computer.

We complement our upper bound with two lower bounds on the size of coresets. Our lower bounds assume the 2-norm squared as the regularizer, since intuitively this is the standard regularizer for which it should be easiest to attain small coresets. We first show that our analysis is asymptotically tight for uniform sampling. That is, we show that for both logistic regression and SVM, a uniform sample of size $O(n^{1-\kappa-\epsilon})$ may not result in a coreset. We then show for both logistic regression and SVM there are instances in which every core set is of size $\Omega(n^{(1-\kappa)/5-\epsilon})$. Therefore, more sophisticated sampling methods must still have core sets whose size is in the same ballpark as is needed for uniform sampling. One might arguably summarize our results as saying that the simplest possible sampling method is nearly optimal for obtaining a coreset.

1.4.3 Training Linear SVM Using Relational Gradient Descent

In our second approach for training linear SVM, we design a relational gradient descent algorithm for a class of instances of linear SVM that we call stable instances. Unfortunately, the framework designed for approximating FAQ-AI(1) problem cannot be directly applied for training linear SVM due to a technical issue that we call *subtraction problem*. To show this, we start Chapter 5 by stating a discouraging fact that the gradient of SVM objective is NP-Hard to approximate up to any constant factor, and in fact it is hard to overcome this problem in general. However, using this framework, we still design a gradient descent algorithm for training linear SVM on the input data that are stable. More specifically, if β^* is the optimal hypothesis for SVM objective, our algorithm returns a hypothesis β_A such that its objective on a small perturbation of the points is at most $1+\epsilon$ factor of the objective of β^* on another perturbation of the points. Then we show that if the instance has some stability condition, this is sufficient to be $1+\epsilon$ factor of the optimal when the points are not perturbed.

Stability is defined formally in Chapter 5; conceptually, a data set is stable if the hypothesis explaining the dataset does not change dramatically by small movements of the data points. Some discussion of the stability of SVM instances can be found in [25].

1.4.4 Relational K-Means Clustering

One of the most famous algorithms for K-Means Clustering is the Lloyd's Algorithm in which we initialize K centers by picking some of the points using some random distribution ,and then iteratively move the centers to the center of mass of the points assigned to them. We show that calculating the center of mass for the points assigned to a center is NP-Hard and as a result it is unlikely for us to implement Lloyd's algorithm relationally.

Fortunately, there are multiple ways of constructing constant approximation coresets for k-means clustering. One of the famous methods is adaptive k-means sampling [12] in which the algorithm samples $k\log(N)$ centers using k-means++ distribution and gives a weight to each center that equals to the number of points assigned to that center. Then it can be proved that this weighted subset is a constant approximation coreset. In chapter

6, first we show how to use the rejection sampling technique [34] to implement k-means++ sampling relationally. Then we show that it is NP-Hard to approximate the weights used in adaptive k-means sampling relationally; however, we propose another weight function that can be computed relationally and has provable constant approximation guarantee similar to adaptive k-means sampling.

2.0 Preliminary

2.1 Machine Learning

Here we briefly explain the machine learning problems and techniques that are referred to in the proposal. The machine learning techniques and problems are explained thoroughly in [31] and [27].

In general, machine learning problems can be divided into two categories of supervised and unsupervised. In supervised learning, the input is a labeled data and the goal is to train a model that can predict the label for unseen data sampled from the same distribution. The supervised learning problems can be further classified into two subcategories of regression and classification, where in regression problems the labels are scalar values and in classification problems the labels are categories. An example of a regression problem is predicting house price based on the specification of the house, and an example of a classification problem is predicting if a customer is willing to buy an item or not.

Then the training problem in machine learning is optimizing a given model subject to a loss function. After training, the prediction for a new point can be done using the optimized parameters. The followings are the training problems of a few well-known machine learning models that are referred to in the proposal.

Training Linear Regression: The input is a set of d dimensional points $X = \{x_1, \ldots, x_n\}$ with associated scalar values $Y = \{y_1, \ldots, y_n\}$. The output is a d dimensional hypothesis β that minimizes

$$L(\beta) = \sum_{i=1}^{n} \|\beta \cdot x_i - y\|_2^2 .$$

The following is a closed form solution for this problem:

$$\beta^* = (A^T A)^{-1} A^T b$$

where A is a matrix with rows $x_1, \ldots, x_n$ and b is a vector with entries $y_1, \ldots, y_n$.

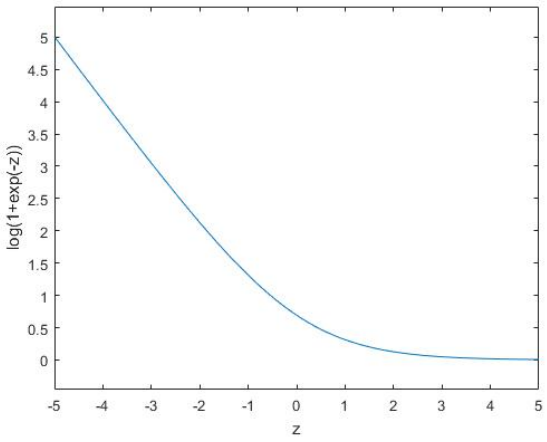

Figure 2: Logistic Regression Loss Function

Training Logistic Regression: The input is a set of d dimensional points $X = \{x_1, \ldots, x_n\}$ with associated labels $Y = \{y_1, \ldots, y_n\}$ where $y_i \in \{-1, +1\}$. Logistic regression is a linear classifier meaning it will find a hyperplane with the norm β and classifies all points on one side positive and all points on the other side negative. The goal is finding a d dimensional hypothesis β that minimizes

$$F(\beta) = \frac{1}{N} \sum_{i=1}^{n} \log\left(1 + \exp(-y_i \beta \cdot x_i)\right) + \lambda R(\beta).$$

where λ is the regularizer coefficient and $R(\beta)$ is the regularization function most often set to $\|\beta\|_2^2$ or $\|\beta\|_1$. Figure 2 shows the loss function of the logistic regression model. The loss function goes up linearly as an incorrectly classified point gets further from the hyperplane, and it goes down exponentially as a correctly classified point gets further from the hyperplane.

Training Linear SVM: The input is a set of d dimensional points $X = \{x_1, \ldots, x_n\}$ with associated labels $Y = \{y_1, \ldots, y_n\}$ where $y_i \in \{-1, +1\}$. Similar to logistic regression, linear SVM is a linear classifier meaning it will find a hyperplane with the norm β and classifies all points on one side positive and all points on the other side negative. In L1-Linear SVM

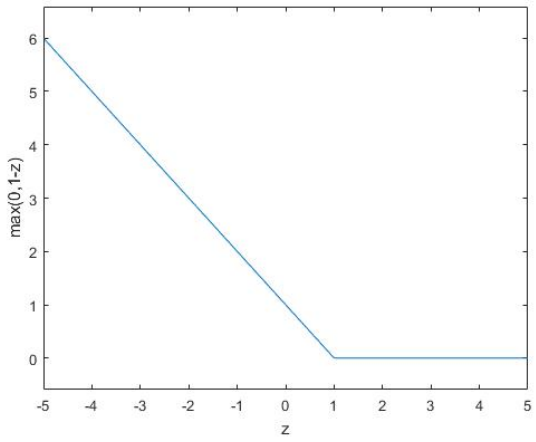

Figure 3: Hinge Loss Function

or SVM with "hinge" loss function, the goal is finding a d dimensional hypothesis β that minimizes

$$F(\beta) = \frac{1}{N}\sum_{i=1}^{n} \max(0, 1 - y_i\beta \cdot x_i) + \lambda R(\beta).$$

where λ is the regularizer coefficient and $R(\beta)$ is the regularization function most often set to $\|\beta\|_2^2$. In L2-Linear SVM or SVM with quadratic loss function the goal is finding a d dimensional hypothesis β that minimizes

$$L(\beta) = \frac{1}{N}\sum_{i=1}^{n} \max(0, 1 - y_i\beta \cdot x_i)^2 + \lambda R(\beta).$$

Figure 3 plots the hinge loss function. Note that the loss function is very similar to the logistic Regression's loss function.

K-Nearest Neighbor Problem: In K-Nearest Neighbor problem, the input is a set of d dimensional points $X = \{x_1, \ldots, x_n\}$ and a point q. The goal is finding the K nearest points to q in X subject to a distance function. Most often the distance function is L_2 distance. When the points in X are labeled, the K nearest points can be used to predict the label of the given point q.

2.1.1 Gradient Descent

Gradient descent is a first-order iterative optimization method for finding an approximate minimum of a convex function $F : \mathbb{R}^d \to \mathbb{R}$, perhaps subject to a constraint the solution lies in some convex body $\mathcal{K}$. In the gradient descent algorithm, at each descent step t the current candidate solution $\beta^{(t)}$ is updated according to the following rule:

$$\beta^{(t)} \leftarrow \beta^{(t-1)} - \eta_t G(\beta^{(t-1)}) \tag{2}$$

where η_t is the step size. In the projected gradient descent algorithm, the current candidate solution $\beta^{(t)}$ is updated according to the following rule:

$$\beta^{(t)} \leftarrow \Pi_{\mathcal{K}} \left(\beta^{(t-1)} - \eta_t G(\beta^{(t-1)})\right) \tag{3}$$

where $\Pi_{\mathcal{K}}(\alpha) = \text{argmin}_{\beta \in \mathcal{K}} \|\alpha - \beta\|_2$ is the projection of the point α to the closest point to α in $\mathcal{K}$. In (projected) gradient descent, G is $\nabla F(\beta^{(t)})$, the gradient of F at $\beta^{(t)}$. There are lots of variations of gradient descent, including variations on the step size, and variations, like stochastic gradient descent[98], in which the gradient is only approximated from a uniform sample of the data at each point.

Theorem 1 and Corollary 2 give bounds on the number of iterations on projected gradient descent to reach solutions with bounded absolute error and bounded relative error, respectively.

Theorem 1. *[24, 58] Let $\mathcal{K}$ be a convex body and F be a function such that $\|\nabla F(\beta)\|_2 \leq G$ for $\beta \in \mathcal{K}$. Let $\beta^* = \text{argmin}_{\beta \in \mathcal{K}} F(\beta)$ be the optimal solution. Let D be an upper bound on $\left\|\beta^{(0)} - \beta^*\right\|_2$, the 2-norm distance from the initial candidate solution to the optimal solution. Let $\widehat{\beta}_s = \frac{1}{s}\sum_{t=0}^{s-1} \beta^{(t)}$. Let $\eta_t = \frac{D}{G\sqrt{t}}$. Then after $T-1$ iterations of projected gradient descent, it must be the case that*

$$F(\widehat{\beta}_T) - F(\beta^*) \leq \frac{2DG}{\sqrt{T}}$$

Corollary 2. *Adopting the assumptions from Theorem 1, if $T \geq \left(\frac{4DG}{\epsilon F(\widehat{\beta}_T)}\right)^2$ then*

$$F(\widehat{\beta}_T) \leq (1+\epsilon)F(\beta^*)$$

That is, the projected gradient descent achieves relative error ϵ.

The gradient of SVM objective F is

$$\nabla F = 2\lambda\beta - \frac{1}{N} y_i \sum_{i \in \mathcal{L}} x_i \tag{4}$$

where $\mathcal{L}$ is the collection $\{i \mid \beta x_i \leq 1\}$ of indices i where x_i is currently contributing to the objective. Note that in this hinge loss function, the gradient of the points on the hyperplane $1 - \beta x = 0$ does not exist, since the gradient is not continuous at this point. In our formulation we have used the subgradient for the points on $1 - \beta x = 0$, meaning for a β on the hyperplane $1 - \beta x = 0$, we have used the limit of the gradient of the points that $1 - \beta' x > 0$ when β' goes to β. For all points that $1 - \beta' x > 0$, the gradient is x; therefore, the limit is also x.

Assume $\beta^{(0)}$ is the origin and adopt the assumptions of Theorem 1. Then $\nabla F(\beta^*) = 0$ implies for any dimension j

$$\left|\beta_j^*\right| = \left| \frac{1}{2N\lambda} \sum_{i \in \mathcal{L}} x_{ij} \right| \leq \frac{1}{2\lambda} \tag{5}$$

where the additional subscript of j refers to dimension j. And thus

$$\left\|\beta^{(0)} - \beta^*\right\|_2 \leq \|\beta^*\|_2 \leq \sqrt{d} \max_{j \in [d]} \left|\beta_j^*\right| \leq \frac{\sqrt{d}}{2\lambda} \tag{6}$$

Thus, let us define our convex body $\mathcal{K}$ to be the hypersphere with radius $\frac{\sqrt{d}}{2\lambda}$ centered at the origin. Thus for $\beta \in \mathcal{K}$,

$$\begin{aligned}
\|\nabla F(\beta)\|_2 &= \sqrt{\sum_{j \in [d]} \left(2\lambda\beta_j - \frac{1}{N} \sum_{i \in \mathcal{L}} x_{ij} \right)^2} \\
&\leq \sqrt{\sum_{j \in [d]} 4(\lambda\beta_j)^2 + 2 \left(\frac{1}{N} \sum_{i \in \mathcal{L}} x_{ij} \right)^2} && \text{Since } (a-b)^2 \leq 2a^2 + 2b^2 \\
&\leq 2\lambda \sum_{j \in [d]} |\beta_j| + \sqrt{2} \frac{1}{N} \left| \sum_{i \in \mathcal{L}} x_{ij} \right| && \text{Since } \sqrt{\sum_i a_i^2} \leq \sum_i |a_i| \\
&\leq 2d + \sqrt{2}d \\
&\leq 4d.
\end{aligned}$$

Theorem 3. *Let the convex body $\mathcal{K}$ be the hypersphere with radius $\frac{\sqrt{d}}{2\lambda}$ centered at the origin. Let $F(\beta)$ be the SVM objective function. Let $\beta^* = \operatorname{argmin}_\beta F(\beta)$ be the optimal solution. Let $\widehat{\beta}_s = \frac{1}{s}\sum_{t=0}^{s-1}\beta^{(t)}$. Let $\eta_t = \frac{1}{8\lambda\sqrt{dt}}$. Then after $T-1$ iterations of projected gradient descent, it must be the case that*

$$F(\widehat{\beta}_T) - F(\beta^*) \leq \frac{4d^{3/2}}{\lambda\sqrt{T}}$$

Theorem 4 then follows by a straightforward application of Theorem 3.

Theorem 4. *Let $F(\beta)$ be the SVM objective function. Let $\beta^* = \operatorname{argmin}_\beta F(\beta)$ be the optimal solution. Let $\widehat{\beta}_s = \frac{1}{s}\sum_{t=0}^{s-1}\beta^{(t)}$. Let $\eta_t = \frac{1}{8\lambda\sqrt{dt}}$. Then if $T \geq \left(\frac{4d^{3/2}}{\epsilon\lambda F(\widehat{\beta}_T)}\right)^2$ then the projected gradient descent guarantees that*

$$F(\widehat{\beta}_T) \leq (1+\epsilon)F(\beta^*)$$

Thus, if the algorithm returns $\widehat{\beta}$ at the first time t where $t \geq \left(\frac{4d^{3/2}}{\epsilon\lambda F(\widehat{\beta}_t)}\right)^2$, then it achieves relative error at most ϵ.

2.2 Coresets

In this section, we define some of the concepts related to coresets. The following definition of coreset can be used for regularized and unregularized machine learning problems. Given a dataset $X = (x_1, \ldots, x_n)$ (possibly with associated labels $Y = (y_1, \ldots, y_n)$) and let $F(\beta) = \sum_{x_i \in X} f_i(\beta)$ be an objective function where $f_i(\beta)$ depends only on the hypothesis β and the point x_i (and y_i). Note that in the context of regularized loss minimization, $f_i(\beta) = \ell(-y_i\beta \cdot x_i) + \lambda r(R\beta)/n$ is the contribution of point i to the objective $F(\beta)$. The following is the definition of a coreset:

Definition 5 (Coreset)**.** *For $\epsilon > 0$, an ϵ-coreset (C, U) consists of a subcollection C of $[1, n]$, and associated nonnegative weights $U = \{u_i \mid i \in C\}$, such that*

$$\forall\beta \qquad H(\beta) := \frac{\left|\sum_{i=1}^n f_i(\beta) - \sum_{i\in C} u_i f_i(\beta)\right|}{\sum_{i=1}^n f_i(\beta)} \leq \epsilon \tag{7}$$

Conceptually, one should think of u_i as a multiplicity, that is that x_i is representing u_i points from the original data set. Thus, one would expect that $\sum_{i \in C} u_i = n$; and although this is not strictly required, it is easy to observe that in the context of RLM, $\sum_{i \in C} u_i$ must be close to n (See Section 2.2).

Furthermore, the following is the definition of sensitivity which is often used in coreset construction algorithms.

Definition 6 (sensitivity)**.** *The* sensitivity *of point i is then $s_i = \sup_\beta \ f_i(\beta)/F(\beta)$, and the total sensitivity is $S = \sum_{i=1}^{n} s_i$.*

A collection X of data points is *shatterable* by a loss function ℓ if for every possible set of assignments of labels, there is a hypothesis β and a threshold t, such that for the positively labeled points $x_i \in X$ it is the case the $\ell(\beta \cdot x_i) \geq t$, and for the negatively labeled points x_i it is the case that $\ell(\beta \cdot x_i) < t$. The VC-dimension of a loss function is then the maximum cardinality of a shatterable set. It is well known that if the loci of points $x \in \Re^d$ where $\ell(\beta \cdot x) = t$ is a hyperplane then the VC-dimension is at most $d + 1$ [101]. It is obvious that this property holds if the loss function is SVM, and [83] show that it holds if the loss function is logistic regression. The regularizer does not affect the VC-dimension of a RLM problem.

Definition 7 ((σ, τ)-scaling)**.** *A loss function ℓ and a regularizer r satisfy the (σ, τ)-scaling condition if $\ell(-\sigma) > 0$, and if $\|\beta\|_2 \geq \sigma$ then $r(\beta) \geq \tau \ \ell(\|\beta\|_2)$.*

Intuitively, this condition ensures that the objective value of any correctly classified point that is near the separating hyperplane must be bounded away from zero, that is either the loss function or the regularizer must be bounded away from zero.

Theorem 8 ([45, 28])**.** *Let $(n, X, Y, \ell, r, \lambda, R, \kappa)$ be an instance of the RLM problem where the loss function has VC-dimension at most Δ. Let s'_i be an upper bound on the sensitivity s_i, let $S' = \sum_{i=1}^{n} s'_i$. Let $\epsilon, \delta \in (0, 1)$ be arbitrary. Let C be a random sample of at least $\frac{10S'}{\epsilon^2}(\Delta \log S' + \log(\frac{1}{\delta}))$ points sampled in an i.i.d fashion, where the probability that point $i \in [1, n]$ is selected each time is s'_i/S'. Let the associated weight u_i for each point $x_i \in C$ be $\frac{S'}{s'_i |C|}$. Then C and $U = \{u_i \mid x_i \in C\}$ is an ϵ-coreset with probability at least $(1 - \delta)$.*

2.3 Joins

Here we define some of the terms and explain some of the previous results from database literature and relational algorithms. Before explaining the definition of the join, we define a table/relation T to be a set of tuples $(x_1, \ldots, x_d)$ where x_i is a member of D_i, the domain of column/feature i. Therefore, we use the notation $x \in T$ for a tuple x and table T to denote the membership of x in T. Furthermore, we use C_i to denote the set of columns/features of table T_i.

Let T be a table with set of columns C and let $C' \subseteq C$, then for a tuple $x \in T$ we define the projection of x onto C' denoted by $\Pi_{C'}(x)$ to be a tuple consisting only those elements of x that are in C'. For example, let T be a table with columns (a, b, c) and let $(1, 2, 3)$ be a tuple/row in T. If $C' = a, c$, then $\Pi_{C'}(x) = (1, 3)$.

Definition 9 (Join)**.** *For a set of input tables $T_1, \ldots, T_m$, with columns $C_1, \ldots, C_m$ the join of them is defined as a table $J = T_1 \bowtie \cdots \bowtie T_m$ with columns $C = \bigcup_i C_i$ such that $x \in J$ if and only if $\Pi_{C_i}(x) \in T_i$ for all $i \in [m]$*

It is possible to model the structure of a join by a hypergraph $H = (V, E)$ such that each column of the join is vertex in V and each input table T_i is a hyperedge in E. Using such a hypergraph and the size of the input tables, [17] has introduced the following upper bound for the size of the join.

Definition 10 (Fractional edge cover number ρ^*)**.** *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph (of some query Q). Let $B \subseteq \mathcal{V}$ be any subset of vertices. A* fractional edge cover *of B using edges in $\mathcal{H}$ is a feasible solution $\vec{\lambda} = (\lambda_S)_{S \in \mathcal{E}}$ to the following linear program:*

$$\begin{aligned} \min \quad & \sum_{S \in \mathcal{E}} \lambda_S \\ s.t. \quad & \sum_{S : v \in S} \lambda_S \geq 1, \quad \forall v \in B \\ & \lambda_S \geq 0, \quad \forall S \in \mathcal{E}. \end{aligned}$$

The optimal objective value of the above linear program is called the fractional edge cover number *of B in $\mathcal{H}$ and is denoted by $\rho^*_{\mathcal{H}}(B)$. When $\mathcal{H}$ is clear from the context, we drop the subscript $\mathcal{H}$ and use $\rho^*(B)$.*

Given a join query Q, the fractional edge cover number of Q is $\rho^*_{\mathcal{H}}(\mathcal{V})$ where $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is the hypergraph of Q.

Theorem 11 (AGM-bound [17, 51])**.** *Given a join query Q over a relational database instance I, the output size is bounded by*

$$|Q(I)| \leq n^{\rho^*},$$

where ρ^* is the fractional edge cover number of Q.

Theorem 12 (AGM-bound is tight [17, 51])**.** *Given a join query Q and a non-negative number n, there exists a database instance I whose relation sizes are upper-bounded by n and satisfies*

$$|Q(I)| = \Theta(n^{\rho^*}).$$

[86] has introduced a worst-case optimal algorithm for joining tables that can perform a join in time $O(md\mathrm{A} + d^2 \sum_i n_i)$ where A is the AGM bound of the query and n_i is the cardinality of table T_i.

Relational algorithms often assume that the join query is acyclic since for a general join, even finding out if the join is empty can be NP-Hard. The followings are the definitions of acyclicity and hypertree decomposition of a join query.

Definition 13 (Hypertree decomposition)**.** *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph. A* tree decomposition *of $\mathcal{H}$ is a pair (T, χ) where $T = (V(T), E(T))$ is a tree and $\chi : V(T) \to 2^{\mathcal{V}}$ assigns to each node of the tree T a subset of vertices of $\mathcal{H}$. The sets $\chi(t)$, $t \in V(T)$, are called the* bags *of the tree decomposition. There are two properties the bags must satisfy*

(a) For any hyperedge $F \in \mathcal{E}$, there is a bag $\chi(t)$, $t \in V(T)$, such that $F \subseteq \chi(t)$.

(b) For any vertex $v \in \mathcal{V}$, the set $\{t \mid t \in V(T), v \in \chi(t)\}$ is not empty and forms a connected subtree of T.

Definition 14 (Acyclicity)**.** *A join query $J = T_1 \bowtie \cdots \bowtie T_m$ is acyclic if there exists a tree $G = (V, E)$, called the hypertree decomposition of J, such that:*

- *The set of vertices are $V = \{v_1, \ldots, v_m\}$, and*

- *for every feature $c \in C$, the set of vertices $\{v_i | c \in C_i\}$ is a connected component of G.*

In Section 2.4.2, the algorithm to test if a query is acyclic is explained. The same algorithm can return a hypertree decomposition of the acyclic query.

For non-acyclic queries, we often need a measure of how "close" a query is to being acyclic. To that end, we use *width* notions of a query.

Definition 15 (g-width of a hypergraph: a generic width notion [11])**.** *Let $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph, and $g : 2^{\mathcal{V}} \to \mathbb{R}^+$ be a function that assigns a non-negative real number to each subset of $\mathcal{V}$. The* g-width *of a tree decomposition (T, χ) of $\mathcal{H}$ is $\max_{t \in V(T)} g(\chi(t))$. The* g-width of $\mathcal{H}$ *is the* minimum *g-width over all tree decompositions of $\mathcal{H}$. (Note that the g-width of a hypergraph is a* Minimax *function.)*

Definition 16 (*Treewidth* and *fractional hypertree width* are special cases of *g-width*)**.** *Let s be the following function: $s(B) = |B| - 1$, $\forall V \subseteq \mathcal{V}$. Then the* treewidth *of a hypergraph $\mathcal{H}$, denoted by $\mathsf{tw}(\mathcal{H})$, is exactly its s-width, and the* fractional hypertree width *of a hypergraph $\mathcal{H}$, denoted by $\mathsf{fhtw}(\mathcal{H})$, is the ρ^*-width of $\mathcal{H}$.*

From the above definitions, $\mathsf{fhtw}(\mathcal{H}) \geq 1$ for any hypergraph $\mathcal{H}$. Moreover, $\mathsf{fhtw}(\mathcal{H}) = 1$ if and only if $\mathcal{H}$ is acyclic.

2.4 Functional Aggregation Queries

In this section, we define SumProd and SumSum queries which are two types of Functional Aggregation Queries. Our definitions are a little different than the definitions in [9]; however, they can be seen as special cases of the definition in [9].

Given a collection of relational tables $T_1, \ldots T_m$ with real-valued entries. Let $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$ be the design matrix that arises from the inner join of the tables. Let n be an upper bound on the number of rows in any table T_i, let N be the number of rows in J, and let d be the number of columns/features in J.

An FAQ query $Q(J)$ that is either a SumProd query or a SumSum query. We define a SumSum query to be a query of the form:

$$Q(J) = \bigoplus_{x \in J} \bigoplus_{i=1}^{d} F_i(x_i)$$

where $(R, \oplus, I_0)$ is a commutative monoid over the arbitrary set R with identity I_0. We define a SumProd query to be a query of the form:

$$Q(J) = \bigoplus_{x \in J} \bigotimes_{i=1}^{d} F_i(x_i)$$

where $(R, \oplus, \otimes, I_0, I_1)$ is a commutative semiring over the arbitrary set R with additive identity I_0 and multiplicative identity I_1. In each case, x is a row in the design matrix J, x_i is the entry in column/feature i of x, and each F_i is an arbitrary (easy to compute) function with range R.

SumProd queries and SumSum queries can be also defined grouped by one of the input tables. The result of a SumProd query grouped by table T_i is a vector with $|T_j|$ elements, such that for every row $r \in T_j$ there is a corresponding element in the result with the following value:

$$Q_r(J) = \bigoplus_{x \in J \bowtie r} \bigotimes_{i=1}^{d} F_i(x_i)$$

Note that $J \bowtie r$ is all the rows in the design matrix whose projection on the columns of T_i is r.

The followings are the definitions of commutative monoids and semirings.

Definition 17. *Fix a set S and let $\oplus$ be a binary operator $S \times S \to S$. The set S with $\oplus$ is a monoid if (1) the operator satisfies associativity; that is, $(a \oplus b) \oplus c = a \oplus (b \oplus c)$ for all $a, b, c \in S$ and (2) there is identity element $e \in S$ such that for all $a \in S$, it is the case that $e \oplus a = a \oplus e = e$. A commutative monoid is a monoid where the operator $\oplus$ is commutative. That is $a \oplus b = b \oplus a$ for all $a, b \in S$.*

Definition 18. *A semiring is a tuple* $(R, \oplus, \otimes, I_0, I_1)$. *The* $\oplus$ *operator is referred to as addition and the* $\otimes$ *is referred to as multiplication. The elements* I_0 *and* I_1 *are referred as* 0 *element and* 1 *element and both are included in* R. *The tuple* $(R, \oplus, \otimes, I_0, I_1)$ *is a semiring if*

A. *it is the case that* R *and* $\oplus$ *are a commutative monoid with* I_0 *as the identity.*
B. R *and* $\otimes$ *is a monoid with identity* I_1.
C. *the multiplication distributes over addition. That is for all* $a, b, c \in R$ *it is the case that* $a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$ *and* $(b \oplus c) \otimes a = (b \otimes a) \oplus (c \otimes a)$.
D. *the* I_0 *element annihilates* R. *That is,* $a \otimes I_0 = I_0$ *and* $I_0 \otimes a = I_0$ *for all* $a \in R$.

A commutative semiring is a semiring where the multiplication is commutative. That is, $a \otimes b = b \otimes a$ *for all* $a, b \in S$.

Inside-Out algorithm introduced in [9], can efficiently evaluate simple SumProd queries as well as SumProd queries grouped by one of the input tables. Assuming $\oplus$ and $\otimes$ can be calculated in constant time, the time complexity of Inside-Out for evaluating SumProd queries is $O(md^2n^{\text{fhtw}}\log(n))$ where m is the number of tables in the join, d is the number of columns, n is the cardinality of the largest input table (the number of rows), and fhtw is the fractional hypertree width of the query. For general SumSum queries, it is possible to use m different SumProd queries [2] and as a result they can be computed in time $O(m^2d^2n^{\text{fhtw}}\log(n))$. In the following theorem, we prove that the SumSum queries involving arithmetic summations can be computed using one SumProd query. Note that the SumSum query in the following theorem can be grouped by an input table as well.

Theorem 19. *Let* Q_f *be a function from domain of column* f *in* J *to* $\mathbb{R}$, *then the following SumSum query can be computed using a SumProd query grouped by* T_i.

$$G(J) = \sum_{X \in r \bowtie J} \sum_{f} F_i(x_f)$$

Proof. Let $S = \{(a, b) \mid a \in \mathbb{R}, b \in \mathbb{I}\}$, and for any two pairs of $(a, b), (c, d) \in S$ we define:

$$(a, b) \oplus (c, d) = (a + c, b + d)$$

and

$$(a, b) \otimes (c, d) = (ad + cb, bd).$$

Then the theorem can be proven by using the following two claims:

A. $(S, \oplus, \otimes)$ forms a commutative semiring with identity zero $I_0 = (0, 0)$ and identity one $I_1 = (0, 1)$.

B. The query $\oplus_{X \in J} \otimes_f (Q_f(x_f), 1)$ is a SumProd FAQ where the first entry of the result is $\sum_{X \in J} \sum_f Q_f(x_f)$ and the second entry is the number of rows in J.

Proof of the first claim: Since arithmetic summation is commutative and associative, it is easy to see $\oplus$ is also commutative and associative. Furthermore, based on the definition of $\oplus$ we have $(a, b) \oplus I_0 = (a + 0, b + 0) = (a, b)$.

The operator $\otimes$ is also commutative since arithmetic multiplication is commutative, the associativity of $\otimes$ can be proved by

$$\begin{aligned}
(a_1, b_1) \otimes ((a_2, b_2) \otimes (a_3, b_3)) &= (a_1, b_1) \otimes (a_2 b_3 + a_3 b_2, b_2 b_3) \\
&= (a_1 b_2 b_3 + b_1 a_2 b_3 + b_1 b_2 a_3, b_1 b_2 b_3) \\
&= (a_1 b_2 + b_1 a_2, b_1 b_2) \otimes (a_3, b_3) \\
&= ((a_1, b_1) \otimes (a_2, b_2)) \otimes (a_3, b_3)
\end{aligned}$$

Moreover, note that based on the definition of $\otimes$, $(a, b) \otimes I_0 = I_0$ and $(a, b) \otimes I_1 = (a, b)$. The only remaining property that we need to prove is the distribution of $\otimes$ over $\oplus$:

$$\begin{aligned}
(a, b) \otimes ((c_1, d_1) \oplus (c_2, d_2)) &= (a, b) \otimes (c_1 + c_2, d_1 + d_2) \\
&= (a, b) \otimes (c_1 + c_2, d_1 + d_2) \\
&= (c_1 b + c_2 b + a d_1 + a d_2, b d_1 + b d_2) \\
&= (c_1 b + a d_1, b d_1) \oplus (c_2 b + a d_2, b d_2) \\
&= ((a, b) \otimes (c_1, d_1)) \oplus ((a, b) \otimes (c_2, d_2))
\end{aligned}$$

Now we can prove the second claim: To prove the second claim, since we have already shown the semiring properties of $(S, \oplus, \otimes)$ we only need to show what is the result of $\oplus_{X \in J} \otimes_f (Q_f(x_f), 1)$. We have $\otimes_f (Q_i(x_f), 1) = (\sum_f Q_i(x_f), 1)$, therefore

$$\oplus_{X \in J} \otimes_f (Q_i(x_f), 1) = \oplus_{X \in J} (\sum_f Q_f(x_f), 1) = (\sum_{X \in J} \sum_f Q_f(x_f), \sum_{X \in J} 1)$$

where the first entry is the result of the SumSum query and the second entry is the number of rows in J. □

2.4.1 FAQ-AI

Unfortunately, the formal definition of FAQ-AI is rather cumbersome and notation heavy. To aid the reader, after the formal definitions, we give some examples of how to model some of the particular learning problems discussed earlier as FAQ-AI problems.

The input to FAQ-AI problem consists of three components:

- A collection of relational tables $T_1, \ldots T_m$ with real-valued entries. Let $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$ be the design matrix that arises from the inner join of the tables. Let n be an upper bound on the number of rows in any table T_i, let N be the number of rows in J, and let d be the number of columns/features in J.
- An FAQ query $Q(J)$ that is either a SumProd query or a SumSum query.
- A collection $\mathcal{L} = \{(G_1, L_1), \ldots (G_b, L_b)\}$ of additive inequalities, where G_i is a collection $\{g_{i,1}, g_{i,2}, \ldots g_{i,d}\}$ of d (easy to compute) functions that map the column domains to the reals, and each L_i is a real number. A row $x \in J$ satsifies the additive inequalities in $\mathcal{L}$ if for all $i \in [1, b]$, it is the case that $\sum_{j=1}^{d} g_{i,j}(x_j) \leq L_i$.

FAQ-AI(k) is a special case of FAQ-AI when the cardinality of $\mathcal{L}$ is at most k.

The output for the FAQ-AI problem is the result of the query on the subset of the design matrix that satisfies the additive inequalities. That is, the output for the FAQ-AI instance with a SumSum query is:

$$Q(\mathcal{L}(J)) = \bigoplus_{x \in \mathcal{L}(J)} \bigoplus_{i=1}^{d} F_i(x_i) \tag{8}$$

And the output for the FAQ-AI instance with a SumProd query is:

$$Q(\mathcal{L}(J)) = \bigoplus_{x \in \mathcal{L}(J)} \bigotimes_{i=1}^{d} F_i(x_i) \tag{9}$$

Here $\mathcal{L}(J)$ is the set of $x \in J$ that satisfy the additive inequalities in $\mathcal{L}$. To aid the reader in appreciating these definitions, we now illustrate how some of the SVM related problems in the introduction can be reduced to FAQ-AI(1).

Counting the number of negatively labeled points correctly classified by a linear separator: Here each row x of the design matrix J conceptually consists of a point in $\mathbb{R}^{d-1}$, whose coordinates are specified by the first $d-1$ columns in J, and a label in $\{1, -1\}$ in column d. Let the linear separator be defined by $\beta \in \mathbb{R}^{d-1}$. A negatively labeled point x is correctly classified if $\sum_{i=1}^{d-1} \beta_i x_i \leq 0$. The number of such points can be counted using SumProd query with one additive inequality as follows: $\oplus$ is addition, $\otimes$ is multiplication, $F_i(x_i) = 1$ for all $i \in [d-1]$, $F_d(x_d) = 1$ if $x_d = -1$, and $F_d(x_d) = 0$ otherwise, $g_{1,j}(x_j) = \beta_j x_j$ for $j \in [d-1]$, $g_{1,d}(x_d) = 0$, and $L_1 = 0$.

Finding the minimum distance to the linear separator of a correctly classified negatively labeled point: This distance can be computed using a SumProd query with one additive inequality as follows: $\oplus$ is the binary minimum operator, $\otimes$ is addition, $F_i(x_i) = \beta_i x_i$ for all $i \in [d-1]$, $F_d(x_d) = 1$ if $x_d = -1$, and $F_d(x_d) = 0$ otherwise, $g_{1,j}(x_j) = \beta_j x_j$ for $j \in [d-1]$, $g_{1,d}(x_d) = 0$, and $L_1 = 0$.

2.4.2 The Inside-Out Algorithm for Acyclic SumProd Queries

In this section, we explain how to obtain a hypertree decomposition of an acyclic join, and then explain (a variation of) the Inside-Out algorithm from [9] for evaluating a SumProd query $Q(J) = \oplus_{x \in J} \otimes_{i=1}^{d} F_i(x_i)$ over a commutative semiring $(R, \oplus, \otimes, I_0, I_1)$ for acyclic join $J = T_1 \bowtie \cdots \bowtie T_m$. A call to Inside-Out may optionally include a root table T_r.

Let C_i denote the set of features in T_i and let $C = \bigcup_i C_i$. Furthermore, given a set of features C_i and a tuple x, let $\Pi_{C_i}(x)$ be the projection of x onto C_i.

Algorithm to Compute Hypertree Decomposition:

A. Initialize graph $G = (V, \emptyset)$ where $V = \{v_1, \ldots, v_m\}$.

B. Repeat the following steps until $|T| = 1$:

   a. Find T_i and T_j in T such that every feature of T_i is either not in any other table of T or is in T_j. If there exists no T_i and T_j with this property, then the query is cyclic.

   b. Remove T_i from T and add the edge (v_i, v_j) to G.

Inside-Out Algorithm:

A. Compute the hypertree decomposition $G = (V, E)$ of J.

B. Assign each feature c in J to an arbitrary table T_i such that $c \in C_i$. Let A_i denote the features assigned to T_i in this step.

C. For each table T_i, add a new column/feature Q_i. For all the tuples $x \in T_i$, initialize the value of column Q_i in row x to $Q_i^x = \bigotimes_{j \in A_i} F_j(x_j)$. Note that if $A_i = \emptyset$ then $Q_i^x = I_1$.

D. Repeat until G has only one vertex

   a. Pick an arbitrary edge (v_i, v_j) in G such that v_i is a leaf and $i \neq r$.

   b. Let $C_{ij} = C_i \cap C_j$ be the shared features between T_i and T_j.

   c. Construct a temporary table T_{ij} that has the features $C_{ij} \cup \{Q_{ij}\}$.

   d. If $C_{ij} = \emptyset$, then table T_{ij} only has the column/feature Q_{ij} and one row, and its lone entry is set to $\oplus_{x \in T_i} Q_i^x$. Otherwise, iterate through the y such that there exists an $x \in T_i$ for which $\Pi_{C_i}(x) = y$, and add the row (y, Q_{ij}^y) to table T_{ij} where: Q_{ij}^y is set to the sum, over all rows $x \in T_i$ such that $C_{ij}(x) = y$, of Q_i^x.

   e. For all the tuples $(x, Q_j^x) \in T_j$, let $y = \Pi_{C_{ij}}(x)$, and update Q_j^x by

$$Q_j^x \leftarrow Q_j^x \otimes Q_{ij}^y.$$

      If $(y, Q_{ij}^y) \notin T_{ij}$, set $Q_j^x = I_0$.

   f. Remove vertex v_i and edge (v_i, v_j) from G.

E. At the end, when there is one vertex v_r left in G, return the value

$$\bigoplus_{(x, Q_r^x) \in T_r} Q_r^x$$

When we use the Inside-Out algorithm in the context of an approximation algorithm in Chapter 3, it is important that the sum Q_{ij}^y computed in step d is computed using a balanced binary tree, so that if k items are being summed, the depth of the expression tree is at most $\lceil \log k \rceil$.

One way to think about step 4 of the algorithm is that it is updating the Q_j values to what they would be if, what good old CLRS [37] calls a relaxation in the description of the Bellman-Ford shortest path algorithm, was applied to every edge in a particular bipartite graph $G_{i,j}$. In $G_{i,j}$ one side of the vertices are the rows in T_i, and the other side are the rows in T_j, and there a directed edge (x, y) from a vertex/row in T_i to a vertex/row in T_j if they have equal projections onto $C_{i,j}$. The length P_j^y of edge (x, y) is the original value of Q_j^y before the execution of step 4. A relaxation step on a directed edge (x, y) is then $Q_j^y = (P_j^y \otimes Q_i^x) \oplus Q_j^y$. Therefore, the result of step 4 of Inside-Out is the same as relaxing every edge in $G_{i,j}$. Although Inside-Out does not explicitly relax every edge. Inside-Out exploits the structure of $G_{i,j}$, by grouping together rows in T_i that have the same projection onto $C_{i,j}$, to be more efficient.

Note that as it is mentioned in [9], we can slightly modify the same algorithm and apply it to cyclic joins as well. To do so, we need to find the hypertree decomposition of the join (or an approximation to it) and for each vertex in the hypertree decomposition of the join, we create a table that is the join of the original input tables assigned to that vertex. Then we use these tables and the Inside-Out algorithm for acyclic queries to compute the SumProd query.

3.0 Functional Aggregation Queries Under Additive Constraint

In this section, we consider the problem of evaluating Functional Aggregate Queries (FAQ's) subject to additive constraints. We start by showing in Section 3.1 that the FAQ-AI(1) problem is $\#P$-hard, even for the problem of counting the number of rows in the design matrix for a cross-product join. Therefore, a relational algorithm for FAQ-AI(1) queries is extraordinarily unlikely as it would imply $P = \#P$.

Thus, we turn to approximately computing FAQ-AI queries. An ideal result would be what we call a *Relational Approximation Scheme* (RAS), which is a collection $\{A_\epsilon\}$ of relational algorithms, one for each real $\epsilon > 0$, such that each A_ϵ is outputs a solution that has relative error at most ϵ. Our main result is a RAS for FAQ-AI(1) queries that have certain natural properties, which we now define.

Definition 20.

- *An operator $\odot$ has bounded error if it is the case that when $x/(1+\delta_1) \leq x' \leq (1+\delta_1)x$ and $y/(1+\delta_2) \leq y' \leq (1+\delta_2)y$ then $(x \odot y)/((1+\delta_1)(1+\delta_2)) \leq x' \odot y' \leq (1+\delta_1)(1+\delta_2)(x \odot y)$.*
- *An operator introduces no error if it is the case that when $x/(1+\delta_1) \leq x' \leq (1+\delta_1)x$ and $y/(1+\delta_2) \leq y' \leq (1+\delta_2)y$ then $(x \odot y)/(1+\max(\delta_1, \delta_2)) \leq x' \odot y' \leq (1+\max(\delta_1, \delta_2))(x \odot y)$..*
- *An operator $\odot$ is repeatable if for any two non-negative integers k and j and any non-negative real δ such that $k/(1+\delta) \leq j \leq (1+\delta)k$, it is the case that for every $x \in R$, $(\bigodot^k x)/(1+\delta) \leq \bigodot^j x \leq (1+\delta) \bigodot^k x$.*
- *An operator $\odot$ is monotone if it is either monotone increasing or monotone decreasing. The operator $\odot$ is monotone increasing if $x \odot y \geq \max(x, y)$. The operator $\odot$ is monotone decreasing if $x \odot y \leq \min(x, y)$.*

Theorem 21. *There is a RAS to compute a SumSum FAQ-AI(1) query over a commutative monoid $(R, \oplus, I_0)$ if:*

- *The domain R is a subset of reals $\mathbb{R}$.*
- *The operators $\oplus$ and $\otimes$ can be computed in polynomial time.*

- $\oplus$ *introduces no error.*
- $\oplus$ *is repeatable.*

Theorem 22. *There is a RAS to compute a SumProd FAQ-AI(1) query over a commutative semiring* $(R, \oplus, \otimes, I_0, I_1)$ *if:*

- *The domain* R *is* $\mathbb{R}^+ \cup \{I_0\} \cup \{I_1\}$.
- $I_0, I_1 \in \mathbb{R}^+ \cup \{+\infty\} \cup \{-\infty\}$
- *The operators* $\oplus$ *and* $\otimes$ *can be computed in polynomial time.*
- $\oplus$ *introduces no error.*
- $\otimes$ *has bounded error.*
- $\oplus$ *is monotone. An operator* $\odot$ *is monotone if it is either monotone increasing or monotone decreasing.*
- *The log of the aspect ratio of the query is polynomially bounded. The aspect ratio is the ratio of the maximum, over every possible submatrix of the design matrix, of the value of the query on that submatrix, to the minimum, over every possible submatrix of the design matrix, of the value of the query on that submatrix.*

In Section 2.4.2 we review the Inside-Out algorithm for SumProd queries over acyclic joins, as our algorithms will use the Inside-Out algorithm.

In Section 3.2, we explain how to obtain a RAS for a special type of FAQ-AI(1) query, an Inequality Row Counting Query, that counts the number of rows in the design matrix that satisfy a given additive inequality. A even more special case of an Inequality Row Counting query is counting the number of points that lie on a given side of a given hyperplane. Our algorithm for Inequality Row Counting can be viewed as a reduction to the problem of evaluating a general SumProd query (without any additive inequalities), over a more complicated type of semiring, that we call a *dynamic programming semiring*. In a dynamic programming semiring the base elements can be thought of as arrays, and the summation and product operations are designed so that the SumProd query computes a desired dynamic program. In the case of Inequality Row Counting, our SumProd query essentially ends up implementing the dynamic program for Knapsack Counting from [42]. Given the widespread utility of dynamic programming as an algorithm design technique, it seems to us likely that

the use of dynamic programming semirings will be useful in designing relational algorithms for other problems. Connections between semirings and dynamic programming have certainly been observed in the past. Nevertheless, the references we could find mostly observed that some algorithms can be generalized to work over semirings, for example, Dijkstra's algorithm is known to work over certain types of semirings [81]. We couldn't find references in the literature that designed semirings to compute particular dynamic programs (although it would hardly be shocking if such references were existed, and we welcome pointers if reviewers know of any such references).

The time complexity of our algorithm for Inequality Row Counting is at most $O\left(\frac{m^6 \log^4 n}{\epsilon^2}\right)$ times the time complexity of the algorithm in [9] for evaluating an SumProd query without additive inequalities, which is $O(d^2 mn \log n)$ if the join is acyclic. Thus, for acyclic joins, the running time of our algorithm is $O\left(\frac{m^7 d^2 n \log^5 n}{\epsilon^2}\right)$. Note that in most natural instances of interest, n is orders of magnitude larger than d or m. Thus, it is important to note that the running time of our algorithm is nearly linear in the dominant parameter n. Finally, we end Section 3.2 by showing how we use this Inequality Row Counting RAS to obtain a RAS for SumSum FAQ-AI(1) queries covered by Theorem 21.

In Section 3.3 we explain how to generalize our RAS for SumProd FAQ-AI(1) queries covered by Theorem 22.

In Section 3.4 we show several applications of our main results to obtain RAS for several natural problems. And we give a few examples where our main result does not apply.

In Section 3.5 we show that the problem of obtaining $O(1)$-approximation for a row counting query with two additive inequalities is NP-hard, even for acyclic joins. This shows that our result for FAQ-AI(1) cannot be extended to FAQ-AI(2), and that a relational algorithm with bounded relative error for row counting with two additive inequalities is quite unlikely, as such an algorithm would imply P = NP.

3.1 NP-Hardness of FAQ-AI(1)

Theorem 23. *The problem of evaluating a FAQ-AI(1) query is #P-Hard.*

Proof. We prove the #-hardness by a reduction from the $\#P$-hard Knapsack Counting problem. An instance of Knapsack consists of a collection $W = \{w_1, \ldots, w_d\}$ of nonnegative integer weights, and a nonnegative integer weight C. The output should be the number of subsets of W with aggregate weight at most C.

We construct the instance of FAQ-AI as follows. We creating d tables. Each table T_i has one column and two rows, with entries 0 and w_i. Then $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_d$ is the cross product join of the tables. We define β to be the d dimensional vector with 1's in all dimensions, and the additive inequality to $\beta \cdot x \leq C$. Then note that there is then a natural bijection between the rows in J that satisfy this inequality the subsets of W with aggregate weight at most C. □

3.2 Algorithm for Inequality Row Counting

The *Inequality Row Counting* is a special case of SumProd FAQ-AI(1) in which the SumProd query $Q(\mathcal{L}(J)) = \sum_{x \in J} \prod_{i=1}^{d} 1$ counts the number of rows in the design matrix that satisfy the constraints $\mathcal{L}$, which consists of one additive constraint $\sum_i g_i(x_i) \leq L$, over a join $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$. In Section 3.2.1, we design a SumProd query over a dynamic programming semiring that computes $Q(\mathcal{L}(J))$ exactly in exponential time. Then in Section 3.2.2 we explain how to apply standard sketching techniques to obtain a RAS. We finish in Section 3.2.3 by explaining how to use our algorithm for Inequality Row Counting to obtain to obtain a RAS SumSum query covered by Theorem 21.

3.2.1 An Exact Algorithm

We first define a commutative semiring $(S, \sqcup, \sqcap, E_0, E_1)$ as follows:

- The elements of the base set S are finite multisets of real numbers. Let $\#A(e)$ denote the frequency of the real value e in the multiset A and let it be 0 if e is not in A. Thus, one can also think of A as a set of pairs of the form $(e, \#A(e))$.
- The additive identity E_0 is the empty set $\emptyset$.
- The addition operator $\sqcup$ is the union of the two multisets; that is $A = B \sqcup C$ if and only if for all real values e, $\#A(e) = \#B(e) + \#C(e)$.
- The multiplicative identity is $E_1 = \{0\}$.
- The multiplication operator $\sqcap$ contains the pairwise sums from the two input multisets; that is, $A = B \sqcap C$ if and only if for all real values e, $\#A(e) = \sum_{i \in \mathbb{R}} (\#B(e-i) \cdot \#C(i))$. Note that this summation is well-defined because there is a finite number of values for i such that $B(e-i)$ and $C(i)$ are non-zero.

Lemma 24. *$(S, \sqcup, \sqcap, E_0, E_1)$ is a commutative semiring.*

Proof. To prove the lemma, we prove the following claims in the order in which they appear.

A. $A \sqcup B = B \sqcup A$

B. $A \sqcup (B \sqcup C) = (A \sqcup B) \sqcup C$

C. $A \sqcup E_0 = A$

D. $A \sqcap B = B \sqcap A$

E. $A \sqcap (B \sqcap C) = (A \sqcap B) \sqcap C$

F. $A \sqcap E_0 = E_0$

G. $A \sqcap E_1 = A$

H. $A \sqcap (B \sqcup C) = (A \sqcap B) \sqcup (A \sqcap C)$

First, we show $\sqcup$ is commutative and associative and $A \sqcup E_0 = A$. By definition of $\sqcup$, $C = A \sqcup B$ if and only if for all $e \in C$ we have $\#C(e) = \#A(e) + \#B(e)$. Since the summation is commutative and associative, $\sqcup$ would be commutative and associative as well. Moreover, note that if $B = E_0 = \emptyset$ then $\#B(e) = 0$ for all values of e and as a result $\#C(e) = \#A(e)$ which means $C = A$.

Now we can show that $\sqcap$ is commutative and associative. By the definition of $\sqcap$, $C = A \sqcap B$ if and only if for all values of e, $\#C(e) = \sum_{i \in \mathbb{R}} (\#A(e-i) \cdot \#B(i))$, since we are taking

the summation over all values:

$$\begin{aligned}\#C(e) &= \sum_{i\in\mathbb{R}}(\#A(e-i)\cdot\#B(i)) \\ &= \sum_{i\in\mathbb{R}}(\#A(i)\cdot\#B(e-i))\end{aligned}$$

The last line is due to the definition of $B \sqcap A$, which means $\sqcap$ is commutative.

To show claim (5), let $D = A \sqcap (B \sqcap C)$ and $D' = (A \sqcap B) \sqcap C$:

$$\begin{aligned}\#D(e) &= \sum_{i\in\mathbb{R}} \#A(e-i)\cdot\Big(\sum_{j\in\mathbb{R}}\#B(i-j)\cdot\#C(j)\Big) \\ &= \sum_{i,j\in\mathbb{R}} \#A(e-i)\cdot\#B(i-j)\cdot\#C(j)\end{aligned}$$

By setting $i' = e - j$ and $j' = e - i$, we obtain:

$$\begin{aligned}\#D(e) &= \sum_{i',j'\in\mathbb{R}} \#A(j')\cdot\#B(i'-j')\cdot\#C(e-i') \\ &= \sum_{i'\in\mathbb{R}}\Big(\sum_{j'\in\mathbb{R}}\#A(j')\cdot\#B(i'-j')\Big)\cdot\#C(e-i') = \#D'(e),\end{aligned}$$

which means $\sqcap$ is associative, as desired.

Now we prove $A \sqcap E_0 = E_0$ and $A \sqcap E_1 = A$. The claim (6) is easy to show since for all e, $\#E_0(e) = 0$ then for all real values e $\sum_{i\in\mathrm{dist}(A)}(\#A(i)\cdot\#E_0(e-i)) = 0$; therefore, $A \sqcap E_0 = E_0$. For claim (7), we have $\sum_{i\in\mathrm{dist}(E_1)}(\#E_1(i)\cdot\#A(e-i)) = (\#E_1(0)\cdot\#A(e)) = \#A(e)$; therefore, $A \sqcap E_1 = A$.

At the end, all we need to show is the distributive law, which means we need to show $A \sqcap (B \sqcup C) = (A \sqcap B) \sqcup (A \sqcap C)$. Let $D = A \sqcap (B \sqcup C)$ and $D' = (A \sqcap B) \sqcup (A \sqcap C)$. We have,

$$\begin{aligned}\#D(e) &= \sum_{i\in\mathbb{R}} \#A(e-i)\cdot(\#B(i)+\#C(i)) \\ &= \sum_{i\in\mathbb{R}}(\#A(e-i)\cdot\#B(i)) + (\#A(e-i)\cdot\#C(i)) \\ &= \sum_{i\in\mathbb{R}}(\#A(e-i)\cdot\#B(i)) + \sum_{j\in\mathbb{R}}(\#A(e-j)\cdot\#C(j)) = \#D'(e)\end{aligned}$$

□

Lemma 25. *The SumProd query* $\widehat{Q}(J) = \sqcup_{x \in J} \sqcap_{i=1}^{d} F_i(x_i)$*, where* $F_i(x_i) = \{g_i(x_i)\}$*, evaluates to the multiset* $\{\sum_i g_i(x_i) \mid x \in J\}$ *the aggregate of the* g_i *functions over the rows of* J*.*

Proof. Based on the definition of $\sqcap$ we have,

$$\sqcap_{i=1}^{d} F_i(x_i) = \sqcap_{i=1}^{d} \{g_i(x_i)\} = \left\{ \sum_{i=1}^{d} g_i(x_i) \right\}$$

Then we can conclude:

$$\sqcup_{x \in J} \sqcap_{i=1}^{d} F_i(x_i) = \sqcup_{x \in J} \left\{ \sum_{i=1}^{d} g_i(x_i) \right\} = \left\{ \sum_{i=1}^{d} g_i(x_i) \mid x \in J \right\}$$

□

Thus, the inequality row count is the number of elements in the multiset returned by $\widehat{Q}(J)$ that are at most L.

3.2.2 Applying Sketching

For a multiset A, let $\triangle A(t)$ denote the number of elements in A that are less than or equal to t. Then the ϵ-sketch $\mathbb{S}_\epsilon(A)$ of a multiset A is a multiset formed in the following manner: For each integer $k \in [1, \lfloor \log_{1+\epsilon} |A| \rfloor]$ there are $\lfloor (1+\epsilon)^k \rfloor - \lfloor (1+\epsilon)^{k-1} \rfloor$ copies of the $\lfloor (1+\epsilon)^k \rfloor$ smallest element $x_k \in A$; that is, $x_k = \triangle A(\lfloor (1+\epsilon)^k \rfloor)$. Note that $|\mathbb{S}_\epsilon(A)|$ may be less that $|A|$ as the maximum value of k is $\lfloor \log_{1+\epsilon} |A| \rfloor$. We will show in Lemma 26 that sketching preserves $\triangle A(t)$ within $(1+\epsilon)$ factor.

Lemma 26. *For all* $t \in \mathbb{R}$*, we have* $(1-\epsilon)\triangle A(t) \leq \triangle \mathbb{S}_\epsilon(A)(t) \leq \triangle A(t)$*.*

Proof. Let $A' = \mathbb{S}_\epsilon(A)$. Note that since we are always rounding the weights up, every item in A that is larger than t will be larger in A' as well. Therefore, $\triangle A'(t) \leq \triangle A(t)$. We now show the lower bound. Recall that in the sketch, every item in the sorted array A with an index in the interval $((1+\epsilon)^i, (1+\epsilon)^{i+1}]$ (or equivalently $((1+\epsilon)^i, \lfloor (1+\epsilon)^{i+1} \rfloor$) will be rounded to $A[\lfloor (1+\epsilon)^{i+1} \rfloor]$. Let i be the integer such that $(1+\epsilon)^i < \triangle A(t) \leq (1+\epsilon)^{i+1}$, then the only

items that are smaller or equal to t in A and are rounded to have a weight greater than t in A' are the ones with index between $(1+\epsilon)^i$ and $j = \triangle A(t)$. Therefore,

$$\begin{aligned}\triangle A(t) - \triangle A'(t) \leq & j - (1+\epsilon)^i \leq (1+\epsilon)^{i+1} - (1+\epsilon)^i \\ = & \epsilon(1+\epsilon)^i \leq \epsilon \triangle A(t),\end{aligned}$$

which shows the lower bound of $\#\triangle A(t)$ as claimed. □

Then our algorithm runs the Inside-Out algorithm, with the operation $\sqcup$ replaced by an operation $\bigcirc$, defined by $A \bigcirc B = \mathbb{S}_\alpha(A \sqcup B)$, and with the operation $\sqcap$ replaced by an operation $\odot$, defined by $A \odot B = \mathbb{S}_\alpha(A \sqcap B)$, where $\alpha = \Theta(\frac{\epsilon}{m^2 \log(n)})$. That is, the operations $\bigcirc$ and $\odot$ are the sketched versions of $\sqcup$ and $\sqcap$. That is, Inside-Out is run on the query $\tilde{Q}(J) = \bigcirc_{x \in J} \odot_{i=1}^{d} F_i(x_i)$, where $F_i(x_i) = \{g_i(x_i)\}$.

Because $\bigcirc$ and $\odot$ do not necessarily form a semiring, Inside-Outside may not return $\tilde{Q}(J)$. However, Lemma 27 bounds the error introduced by each application of $\bigcirc$ and $\odot$. This makes it possible in Theorem 28 to bound the error of Inside-Out's output.

Lemma 27. *Let $A' = \mathbb{S}_\beta(A)$, $B' = \mathbb{S}_\gamma(B)$, $C = A \sqcup B$, $C' = A' \sqcup B'$, $D = A \sqcap B$, and $D' = A' \sqcap B'$. For all $t \in \mathbb{R}$, we have:*

A. $(1 - \max(\beta, \gamma))\triangle C(t) \leq \triangle C'(t) \leq \triangle C(t)$

B. $(1 - \beta - \gamma)\triangle D(t) \leq \triangle D'(t) \leq \triangle D(t)$

Proof. By the definition of $\sqcup$, we know $\#C'(t) = \#A'(t) + \#B'(t)$; Thus, we have:

$$\begin{aligned}\triangle C'(t) &= \sum_{\tau \leq t} \#C'(\tau) = \sum_{\tau \leq t} \#A'(\tau) + \sum_{\tau \leq t} \#B'(\tau) \\ &= \triangle A'(t) + \triangle B'(t)\end{aligned}$$

Similarly, we have $\triangle C(t) = \triangle A(t) + \triangle B(t)$. Then by Lemma 26 we immediately have the first claim.

Let $D'' = A \sqcap B'$, Based on the definition of $\sqcap$ we have:

$$\begin{aligned}\triangle D''(t) &= \sum_{\tau \leq t} \#D''(t) = \sum_{\tau \leq t} \sum_{v \in \mathbb{R}} (\#A(v) \cdot \#B'(\tau - v)) \\ &= \sum_{v \in \mathbb{R}} \sum_{\tau \leq t} (\#A(v) \cdot \#B'(\tau - v)) = \sum_{v \in \mathbb{R}} (\#A(v) \cdot \triangle B'(t - v))\end{aligned}$$

Therefore, using Lemma 26 we have: $(1-\gamma)\triangle D(t) \leq \triangle D''(t) \leq (1+\gamma)\triangle D(t)$. We can similarly replace A to A' in D'' and get $(1-\beta)\triangle D''(t) \leq \triangle D'(t) \leq (1+\beta)\triangle D''(t)$ which proves the second claim. □

Theorem 28. *Our algorithm achieves an* $(1+\epsilon)$*-approximation to the Row Count Inequality query* $Q(\mathcal{L}(J))$ *in time* $O(\frac{1}{\epsilon^2}(m^3\log^2(n))^2(d^2mn^h\log(n)))$.

Proof. We first consider the approximation ratio. Inside-Out on the query $\tilde{Q}(J)$ performs the same semiring operations as does on the query $\widehat{Q}(J)$, but it additionally applies the α-Sketching operation over each partial results, meaning the algorithm applies α-Sketching after steps d,e, and 5. Lets look at each iteration of applying steps d and e. Each value produced in the steps d and 5 is the result of applying $\bigcirc$ over at most n^m different values (for acyclic queries it is at most n). Using Lemma 27 and the fact that the algorithm applies $\bigcirc$ first on each pair and then recursively on each pair of the results, the total accumulated error produced by each execution of steps d and 5 is $m\log(n)\alpha$. Then, since the steps d and e will be applied once for each table, and e accumulates the errors produced for all tables, the result of the query will be $(m^2\log(n)+m)\alpha$-Sketch of $\widehat{Q}(J)$.

We now turn to bounding the running time of our algorithm. The time complexity of Inside-Out is $O(md^2n^{\text{fhtw}}\log n)$ when the summation and product operators take a constant time [9]. Since each multiset in a partial result has at most m^n members in it, an α-sketch of the partial results will have at most $O(\frac{m\log n}{\alpha})$ values, and we compute each of $\bigcirc$ and $\odot$ in time $O\left(\left(\frac{m\log n}{\alpha}\right)^2\right)$. Therefore, our algorithm runs in time $O(\frac{1}{\epsilon^2}(m^3\log^2(n))^2(d^2mn^h\log(n)))$. □

3.2.3 SumSum FAQ-AI(1)

In this section, we prove Theorem 21, that there is a RAS for SumSum FAQ-AI(1) queries covered by the theorem. Our algorithm for SumSum queries uses our algorithm for Inequality Row Counting. Consider the SumSum $Q(\mathcal{L}(J)) = \oplus_{x\in\mathcal{L}(J)} \oplus_{i=1}^{d} F_i(x_i)$, where $\mathcal{L}$ consists of one additive constraint $\sum_i g_i(x_i) \leq L$, over a join $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$.

SumSum Algorithm: For each table T_j, we run our Inside-Out on the Inequality Row Counting query $\tilde{Q}(J)$, with the root table being T_j, and let $\tilde{T}_j$ be the resulting table just

before step 5 of Inside-Out is executed. From the resulting tables, one can compute, for every feature $i \in [d]$ and for each possible value of x_i for $x \in J$, a $(1+\epsilon)$-approximation $U(x_i)$ to the number of rows in the design matrix that contain value x_i in column i, by aggregating over the row counts in any table $\tilde{T}_j$ that contains feature i. Then one can evaluate $Q(\mathcal{L}(J))$ by

$$\bigoplus_{i=1}^{d} \bigoplus_{x_i \in D(i)} \bigoplus_{j=1}^{U(x_i)} F_i(x_i)$$

where $D(i)$ is the domain of feature i.

Note that $\oplus$ operator is assumed to be repeatable, meaning if we have a $(1 \pm \epsilon)$ approximation of $U(x_i)$ then the approximation error of $\bigoplus_{j=1}^{U(x_i)} F_i(x_i)$ is also $(1 \pm \epsilon)$. Therefore, our SumSum algorithm is a $(1+\epsilon)$-approximation algorithm because the only error introduced is caused by our Inequality Row Counting algorithm. The running time is $O(\frac{1}{\epsilon^2}(m^3 \log^2(n))^2(d^2 m^2 n^h \log(n)))$ because we run m Inequality Row Counting algorithm m times.

3.3 SumProd FAQ-AI(1)

In this section, we prove Theorem 22, that there is a RAS for SumProd FAQ-AI(1) queries covered by the theorem. Our RAS for such queries generalizes our RAS for Inequality Row Counting. Consider the SumProd query $Q(\mathcal{L}(J)) = \oplus_{x \in \mathcal{L}(J)} \otimes_{i=1}^{d} F_i(x_i)$. where $\mathcal{L}$ consists of the single additive constraint $\sum_{i=1}^{d} g_i(x_i) \leq L$. We again first give an exact algorithm that can be viewed as a reduction to a SumProd query over a dynamic programming semiring, and then apply sketching.

3.3.1 An Exact Algorithm

We first define a structured commutative semiring $(S, \sqcup, \sqcap, E_0, E_1)$ derived from the $(R, \oplus, \otimes, I_0, I_1)$ as follows:

- The base set S are finite subsets A of $\mathbb{R} \times (R - \{I_0\})$ with the property that $(e, v) \in A$ and $(e, u) \in A$ implies $v = u$; so there is only one tuple in A of the form $(e, *)$. One can can interpret the value of v in a tuple $(e, v) \in A$ as a (perhaps fractional) multiplicity of e.
- The additive identity E_0 is the empty set $\emptyset$.
- The multiplicative identity E_1 is $\{(0, I_1)\}$.
- For all $e \in \mathbb{R}$, define $\#A(e)$ to be v if $(e, v) \in A$ and I_0 otherwise.
- The addition operator $\sqcup$ is defined by $A \sqcup B = C$ if and only if for all $e \in \mathbb{R}$, it is the case that $\#C(e) = \#A(e) \oplus \#B(e)$.
- The multiplication operator $\sqcap$ is defined by $A \sqcap B = C$ if and only if for all $e \in \mathbb{R}$, it is the case that $\#C(e) = \bigoplus_{i \in \mathbb{R}} \#A(e - i) \otimes \#B(i)$.

Lemma 29. *If $(R, \oplus, \otimes, I_0, I_1)$, is a commutative semiring then $(S, \sqcup, \sqcap, E_0, E_1)$ is a commutative semiring.*

Proof. We prove the following claims respectively:

A. $A \sqcup B = B \sqcup A$

B. $A \sqcup (B \sqcup C) = (A \sqcup B) \sqcup C$

C. $A \sqcup E_0 = A$

D. $A \sqcap B = B \sqcap A$

E. $A \sqcap (B \sqcap C) = (A \sqcap B) \sqcap C$

F. $A \sqcap E_0 = E_0$

G. $A \sqcap E_1 = A$

H. $A \sqcap (B \sqcup C) = (A \sqcap B) \sqcup (A \sqcap C)$

Based on the definition, $C = A \sqcup B$ if and only if $\#C(e) = \#A(e) \oplus \#B(e)$; since $\oplus$ is commutative and associative, $\sqcup$ will be commutative and associative as well. Furthermore, $\#A(e) \oplus \#E_0(e) = \#A(e) \oplus I_0 = \#A(e)$; therefore, $A \sqcup E_0 = A$.

Let $C = A \sqcap B$, using the commutative property of $\otimes$ and change of variables, we can prove the fourth claim:

$$\begin{aligned}
\#C(e) &= \bigoplus_{i \in \mathbb{R}} \#A(e-i) \otimes \#B(i) \\
&= \bigoplus_{i \in \mathbb{R}} \#B(i) \otimes \#A(e-i) \\
&= \bigoplus_{j \in \mathbb{R}} \#B(e-j) \otimes \#A(j)
\end{aligned}$$

Similarly, using the change of variables $j' = i - j$ and $i' = j$, and semiring properties of the $\otimes$ and $\oplus$, we have:

$$\begin{aligned}
&\bigoplus_{i \in \mathbb{R}} \#A(e-i) \otimes (\bigoplus_{j \in \mathbb{R}} \#B(i-j) \otimes \#C(j)) \\
=&\bigoplus_{i \in \mathbb{R}} \bigoplus_{j \in \mathbb{R}} (\#A(e-i) \otimes \#B(i-j)) \otimes \#C(j) \\
=&\bigoplus_{i' \in \mathbb{R}} (\bigoplus_{j' \in \mathbb{R}} \#A(e-i'-j') \otimes \#B(j')) \otimes \#C(i')
\end{aligned}$$

Therefore, $A \sqcap (B \sqcap C) = (A \sqcap B) \sqcap C$.

The claim $A \sqcap E_0 = E_0$ can be proved by the fact that $\#E_0(i) = I_0$ for all the elements and $\#A(e-i) \otimes I_0 = I_0$. In addition, we have $\bigoplus_{i \in \mathbb{R}} \#A(e-i) \otimes \#E_1(i) = \#A(e)$ because, $\#E_1(i) = I_0$ for all nonzero values of i and it is I_1 for $e = 0$; therefore, $A \sqcap E_1 = A$.

Let $D = A \sqcap (B \sqcup C)$, the last claim can be proved by the following:

$$\begin{aligned}
\#D(e) &= \bigoplus_{i \in \mathbb{R}} \#A(e-i) \otimes (\#B(i) \oplus \#C(i)) \\
&= \bigoplus_{i \in \mathbb{R}} \big((\#A(e-i) \otimes \#B(i)) \oplus (\#A(e-i) \otimes \#C(i))\big) \\
&= \big(\bigoplus_{i \in \mathbb{R}} (\#A(e-i) \otimes \#B(i))\big) \oplus \big(\bigoplus_{i \in \mathbb{R}} (\#A(e-i) \otimes \#C(i))\big)
\end{aligned}$$

where the last line is the definition of $(A \sqcap B) \sqcup (A \sqcap C)$. $\square$

For each column $i \in [d]$, we define the function $\mathcal{F}_i$ to be $\{(g_i(x_i), F_i(x_i))\}$ if $F_i(x_i) \neq I_0$ and the empty set otherwise. Our algorithm for computing $Q(\mathcal{L}(J))$ runs the Inside-Out algorithm on the SumProd query:

$$\widehat{Q} = \sqcup_{x \in J} \sqcap_{i=1}^{d} \mathcal{F}_i(x_i)$$

and returns $\bigoplus_{e \leq L} \#\widehat{Q}(e)$.

Lemma 30. *This algorithm correctly computes $Q(\mathcal{L}(J))$.*

Proof. We can rewrite the generated FAQ as follows:

$$\begin{aligned}
\hat{Q} &= \sqcup_{x \in J} \sqcap_{i=1}^{d} \mathcal{F}_i(x_i) \\
&= \sqcup_{x \in J} \sqcap_{i=1}^{d} \{(g_i(x_i), F_i(x_i))\} \\
&= \sqcup_{x \in J} \left\{ \left(\sum_{i=1}^{d} g_i(x_i), \bigotimes_{i=1}^{d} F_i(x_i) \right) \right\}
\end{aligned}$$

Then the operator $\sqcup$ returns a set of pairs (e, v) such that for each value e, $v = \#\hat{Q}(e)$ is the aggregation using $\oplus$ operator over the rows of J where $\sum_{i=1}^{d} g_i(x_i) = e$. More formally,

$$\#\hat{Q}(e) = \bigoplus_{x \in J, \sum_i g_i(x_i) = e} \left(\bigotimes_{i=1}^{d} F_i(x_i) \right)$$

Therefore, the value returned by the algorithm is

$$\begin{aligned}
\bigoplus_{e \leq L} \#\hat{Q}(e) &= \bigoplus_{e \leq L} \bigoplus_{x \in J, \sum_i g_i(x_i) = e} \left(\bigotimes_{i=1}^{d} F_i(x_i) \right) \\
&= \bigoplus_{x \in \mathcal{L}(J)} \bigotimes_{i=1}^{d} F_i(x_i)
\end{aligned}$$

□

3.3.2 Applying Sketching

For a set $A \in S$ define $\triangle A(\ell)$ to be $\oplus_{e \leq \ell} \# A(e)$. Note that $\triangle A(\ell)$ will be monotonically increasing if $\oplus$ is monotonically increasing, and it will be monotonically decreasing if $\oplus$ is monotonically decreasing.

Conceptually an ϵ-sketch $\mathbb{S}_\epsilon(A)$ of an element $A \in S$ rounds all multiplicities up to an integer power of $(1+\epsilon)$. Formally the ϵ-sketch $\mathbb{S}_\epsilon(A)$ of A is the element A' of S satisfying

$$\#A'(e) = \begin{cases} \bigoplus_{L_k < e \leq U_k} \#A(e) & \text{if } \exists k \; e = U_k \\ I_0 & \text{otherwise} \end{cases}$$

where

$$L_0 = \min\{e \in \mathbb{R} | \triangle A(e) \leq 0\}$$

and for $k \neq 0$

$$L_k = \min\{e \in \mathbb{R} | \rho(1+\epsilon)^{k-1} \leq \triangle A(e) \leq \rho(1+\epsilon)^k\}$$

and where

$$U_0 = \max\{e \in \mathbb{R} | \triangle A(e) \leq 0\}$$

and for $k \neq 0$

$$U_k = \max\{e \in \mathbb{R} | \rho(1+\epsilon)^{k-1} \leq \triangle A(e) \leq \rho(1+\epsilon)^k\}$$

where $\rho = \min\{\#A(e) \mid \#e \in \mathbb{R} \text{ and } \#A(e) > 0\}$. For the special case that $\#A(e) \leq 0$ for all $e \in \mathbb{R}$, we only have L_0 and U_0. Note that the only elements of R that can be zero or negative are I_0 and I_1; therefore, in this special case, $\#A(e)$ for all the elements e is either I_0 or I_1.

Lemma 31. *For all $A \in S$, for all $e \in \mathbb{R}^+$, if $A' = \mathbb{S}_\epsilon(A)$ then*

$$\triangle A(e)/(1+\epsilon) \leq \triangle A'(e) \leq (1+\epsilon)\triangle A(e)$$

Proof. Since $\triangle A(e)$ is monotone, the intervals $[L_k, U_k]$ do not have any overlap except over the points L_k and U_k, and if the $\triangle A(e)$ is monotonically increasing, then $L_k = U_{k+1}$; and if $\triangle A(e)$ is monotonically decreasing, then $L_k = U_{k-1}$.

For any integer j we have:

$$\begin{aligned}\triangle A(U_j) &= \bigoplus_{i\le U_j} \#A(i) = \bigoplus_{k\le j} \bigoplus_{L_k<i\le U_k} \#A(i) = \bigoplus_{k\le j} \#A'(U_k) \\ &= \bigoplus_{i\le U_j} \#A'(i) = \triangle A'(U_j)\end{aligned} \tag{10}$$

Now, first we assume $\triangle A(e)$ is monotonically increasing and prove the lemma. After that, we do the same for the monotonically decreasing case. Given a real value e, let k be the integer such that $e \in (L_k, U_k]$. Then using the definition of U_k and Equality (10) we have:

$$\begin{aligned}\triangle A(e)/(1+\epsilon) &\le \triangle A(U_k)/(1+\epsilon) = \triangle A(U_{k-1}) = \triangle A'(U_{k-1}) \\ &\le \triangle A'(e) \le \triangle A'(U_k) = \triangle A(U_k) \\ &= (1+\epsilon)\triangle A(U_{k-1}) \le (1+\epsilon)\triangle A(e)\end{aligned}$$

Note that in the above inequalities, for the special case of $k = 0$, we can use L_k instead of U_{k-1}. Similarly, for a monotonically decreasing case we have:

$$\begin{aligned}\triangle A(e)/(1+\epsilon) &\le \triangle A(U_{k-1})/(1+\epsilon) = \triangle A(U_k) = \triangle A'(U_k) \\ &\le \triangle A'(e) \le \triangle A'(U_{k-1}) = \triangle A(U_{k-1}) \\ &= (1+\epsilon)\triangle A(U_k) \le (1+\epsilon)\triangle A(e)\end{aligned}$$

□

Then our algorithm runs the Inside-Out algorithm, with the operation $\sqcup$ replaced by an operation $\bigcirc$, defined by $A \bigcirc B = \mathbb{S}_\alpha(A \sqcup B)$, and with the operation $\sqcap$ replaced by an operation $\odot$, defined by $A \odot B = \mathbb{S}_\alpha(A \sqcap B)$, where $\alpha = \frac{\epsilon}{m^2 \log(n)+m}$. That is, the operations $\bigcirc$ and $\odot$ are the sketched versions of $\sqcup$ and $\sqcap$. Our algorithm returns $\triangle A(L) = \oplus_{e\le L} \#A(e)$.

Because $\bigcirc$ and $\odot$ do not necessarily form a semiring, Inside-Outside may not return $\bigcirc_{x\in J} \odot_{i=1}^m F_i(x_i)$. However, Lemma 32 bounds the error introduced by each application of $\bigcirc$ and $\odot$. This makes it possible in Theorem 22 to bound the error of Inside-Out's output.

Lemma 32. *Let $A' = \mathbb{S}_\beta(A)$, $B' = \mathbb{S}_\gamma(B)$, $C = A \sqcup B$, $C' = A' \sqcup B'$, $D = A \sqcap B$, and $D' = A' \sqcap B'$. Then, for all $e \in \mathbb{R}$ we have:*

A. $\frac{\triangle C(e)}{1+\max(\beta,\gamma)} \leq \triangle C'(e) \leq (1+\max(\beta,\gamma))\triangle C(e)$

B. $\frac{\triangle D(e)}{(1+\beta)(1+\gamma)} \leq \triangle D'(e) \leq (1+\beta)(1+\gamma)\triangle D(e)$

Proof. The first claim follows from the assumption that $\oplus$ does not introduce any error and it can be proved by the following:

$$\begin{aligned}
&(\triangle A(e) \oplus \triangle B(e))/(1+\max(\beta,\gamma)) \\
\leq &\#C(e) = \triangle A'(e) \oplus \triangle B'(e) \\
\leq &(1+\max(\beta,\gamma))(\triangle A(e) \oplus \triangle B(e))
\end{aligned}$$

The second claim can be also proved similarly; based on definition of $\sqcap$, we have

$$\begin{aligned}
\triangle D(e) &= \bigoplus_{j\leq e}\bigoplus_{i\in\mathbb{R}}(\#A(j-i)\otimes\#B(i)) \\
&= \bigoplus_{i\in\mathbb{R}}\bigoplus_{j\leq e}(\#A(j-i)\otimes\#B(i)) \\
&= \bigoplus_{i\in\mathbb{R}}(\#B(i)\otimes\bigoplus_{j\leq e}\#A(j-i)) \\
&= \bigoplus_{i\in\mathbb{R}}(\#B(i)\otimes\triangle A(e-i))
\end{aligned}$$

Let $D'' = A' \sqcap B$, then based on the approximation guarantee of $\triangle A'(e)$ and the error properties of $\otimes$ and $\oplus$, we have

$$\triangle D(e)/(1+\beta) \leq \triangle D''(e) \leq (1+\beta)\triangle D(e)$$

Then the second claim follows by replacing B with B' in D'' and repeating the above step. □

Now we can prove the existence of an algorithm for approximating SumProd FAQ-AI(1) queries.

Proof of Theorem 22. We first consider the approximation ratio. Inside-Out on the query $\tilde{Q}(J)$ performs the same semiring operations as does on the query $\widehat{Q}(J)$, but it additionally applies the α-Sketching operation over each partial results, meaning the algorithm applies α-Sketching after steps d,e, and 5. Lets look at each iteration of steps d and e. Each value produced in the steps d and 5 is the result of applying $\bigcirc$ over at most n^m different values (for acyclic queries it is at most n). Using Lemma 27 and the fact that the algorithm applies $\bigcirc$ first on each pair and then recursively on each pair of the results, the total accumulated error produced by each execution of steps d and 5 is $m\log(n)\alpha$. Then, since the steps d and e will be applied once for each table, and e accumulates the errors produced for all tables, the result of the query will be $(m^2\log(n)+m)\alpha$-Sketch of $\widehat{Q}(J)$.

We now turn to bounding the running time of our algorithm. The time complexity of Inside-Out is $O(md^2n^{\text{fhtw}}\log n)$ when the summation and product operators take a constant time [9]. The size of each partial result set $A \in S$, after applying α-sketching, will depend on the smallest positive value of $\triangle A(e)$ and the largest value of $\triangle A(e)$. Let β and γ be the minimum and maximum positive real value of SumProd query over all possible sub-matrices of the design matrix, the smallest and largest value of $\triangle A(e)$ for all partial results A would be β and γ respectively; therefore, the size of the partial results after applying α-Sketching is at most $O(\frac{\log(\gamma/\beta)}{\alpha})$. As a result, we compute each of $\bigcirc$ and $\odot$ in time $O\left(\left(\frac{\log(\gamma/\beta)}{\alpha}\right)^2\right)$. Therefore, our algorithm runs in time $O(\frac{1}{\epsilon^2}(m^2\log(n)\log(\frac{\gamma}{\beta}))^2(d^2mn^h\log(n)))$ and the claim follows by the assumption that the log of the aspect ratio, $\log(\frac{\gamma}{\beta})$, is polynomially bounded. □

3.4 Example Applications

In this section, we give example applications of our results.

Inequality Row Counting: Some example problems for which we can use our Inequality Row Counting to obtain a RAS in a straightforward manner:

- Counting the number of points on one side of a hyperplane, say the points x satisfy $\beta \cdot x \leq L$.

- Counting the number of points within a hypersphere of radius r centered at a point y. The additive constraint is $\sum_{i=1}^{d}(x_i - y_i)^2 \leq r^2$.
- Counting the number of points in an axis parallel ellipsoid, say the points x such that $\sum_{i=1}^{d} \frac{x_i^2}{\alpha_i^2}$ for some d dimensional vector α.

SumSum FAQ-AI(1) Queries Some examples of problems that can be reduced to SumSum FAQ-AI(1) queries and an application of Theorem 21 gives a RAS:

- Sum of 1-norm distances from a point y to points on one side of a hyperplane. The SumSum query is $\sum_{x \in J} \sum_{i=1}^{d} |x_i - y_i|$. One can easily verify that the addition introduces no error and is repeatable.
- Sum of 2-norm squared of points in a axis-parallel ellipsoid. The SumSum query is $\sum_{x \in J} \sum_{i=1}^{d} x_i^2$.
- Number of nonzero entries of points on one side of a hyperplane. The SumSum query is $\sum_{x \in J} \sum_{i=1}^{d} \mathbb{1}_{x_i \neq 0}$.

SumProd FAQ-AI(1) Queries Some examples of problems that can be reduced to SumProd FAQ-AI(1) queries and an application of Theorem 22 gives a RAS:

- Finding the minimum 1-norm of any point in a hypersphere H of radius r centered at a point y. The SumProd query is $\min_{x \in J} \sum_{i=1}^{d} |x_i|$. Note $(\mathbb{R}^+ \cup \{0\} \cup \{+\infty\}, \min, +, +\infty, 0)$ is a commutative semiring. The multiplication operator in this semiring, which is addition, has bounded error. The addition operator, which is minimum, introduces no error and is monotone. The aspect ratio is at most $(\max_{x \in J} \sum_{i=1}^{d} |x_i|)/(\min_{x \in J} \min_{i \in [d] | x_i \neq 0} |x_i|)$, and thus the log of the aspect ratio is polynomially bounded.
- Finding the point on the specified side of a hyperplane H that has the maximum 2-norm distance from a point y. The SumProd query is $\max_{x \in J} \sum_{i=1}^{d} (y_i - x_i)^2$. Note that this computes the point with the maximum 2-norm squared distance. One cannot directly write a SumProd query to compute the point with the 2-norm distance; we need to appeal to the fact that the closest point is the same under both distance metrics. Note $(\mathbb{R}^+ \cup \{0\} \cup \{-\infty\}, \max, +, -\infty, 0)$ is a commutative semiring. The multiplication operator in this semiring, which is addition, has bounded error. The addition operator, which is maximum, introduces no error and is monotone. The aspect ratio is at

most $(\max_{x\in J}\sum_{i=1}^{d}|x_i|)/(\min_{x\in J}\min_{i\in[d]|x_i\neq 0}|x_i|)$, and thus the log of the aspect ratio is polynomially bounded.

Snake Eyes: Some examples of problems for which our results apparently do not apply:

- Finding the minimum distance of any point on a specified side of a specified hyperplane H to H. For instance, consider the problem of finding a point x where $\beta \cdot x \geq L$ and $x \cdot \beta$. The natural SumProd query is $\min_{x\in J}\sum_{i=1}^{d} x_i\beta_i$. Note that some of the $x_i\beta_i$ terms may be negative, so this does not fulfill the condition that the domain has to be over the positive reals. And this appears to be a nontrivial issue because a good approximation of s and t does not in general allow one to compute a good approximation of $s - t$. We have been call this the subtraction problem. Using a variation of the proof of Theorem 33 one can show that approximating this query within $O(1)$ factor is NP-hard.
- Sum of entries of the points lying on one side of a hyperplane. The natural SumSum query is $\sum_{x\in J}\sum_{i=1}^{d} x_i$. Again, as some of the x_i terms may be be negative, we run into the subtraction problem again.
- Aggregate 2-norms of the rows in the design matrix. The natural query is $\sum_{x\in J}\left(\sum_{i=1}^{d} x_i^2\right)^{1/2}$, which is neither a SumSum or a SumProd query.

3.5 NP-hardness of FAQ-AI(2) Approximation

Theorem 33. *For all $c \geq 1$, it is NP-hard to c-approximate the number of rows in the design matrix (even for a cross-product join) that satisfy two (linear) additive inequalities. Therefore, it is NP-hard to c-approximate FAQ-AI(2).*

Proof. We reduce from the Partition problem, where the input is a collection $W = \{w_1, w_2, ..., w_m\}$ of positive integers, and the question is whether one can partition W into two parts with equal aggregate sums. From this instance, we create m tables, $T_1, T_2, \ldots, T_m$, where each T_i has a single columns and has two rows with entries w_i and $-w_i$. Let J be the cross product of these tables. Note that J has exactly 2^m rows and each row $x \in J$ contains either w_i or $-w_i$ for every i, which can be naturally interpreted as a partitioning that places each item i

in one part or the other, depending on the sign of w_i. The two (linear) additive inequalities are $(1, 1, \ldots, 1) \cdot x \geq 0$ and $(-1, -1, \ldots, -1) \cdot x \geq 0$. Then the solution to the Row Counting SumProd query subject to these two constraints is the number of ways to partition W into two parts of equal aggregate sum. $\square$

4.0 Coresets for Regularized Loss Minimization

In this section, we consider the design and mathematical analysis of sampling-based algorithms for regularized loss minimization (RLM) problems on large data sets [96]. Common regularizers are 1-norm, 2-norm, and 2-norm squared [30]. The parameter $\lambda \in \Re$ is ideally set to balance the risks of overfitting and underfitting. We will assume that λ is proportional to n^κ for some $0 < \kappa < 1$, capturing the range of the most commonly suggested regularizers. In particular, it is commonly recommended to set λ to be proportional to $\Theta(\sqrt{n})$ [96, 85]. For this choice of λ, if there was a true underlying distribution from which the data was drawn in an i.i.d. manner, then there is a guarantee that the computed β will likely have vanishing relative error with respect to the ground truth [96, Corollary 13.9] [85, Corollary 3]. The parameter R is the maximum 2-norm of any point in X. Note that the regularizer must scale with R if it is to avoid having a vanishing effect as the point set X scales.[1]

One popular method to deal with large data sets is to extract a manageably small (potentially weighted) sample from the data set, and then directly solve the (weighted version of) RLM problem on the (weighted) sample. Thus, the research question of fundamental importance is if small coresets exist for RLM problems in general, and for regularized logistic regression and regularized SVM – further, if they can be efficiently computed within the common restricted access models.

Our main result is that if the regularizer's effect does not become negligible as the norm of the hypothesis scales then a uniform sample of size $\Theta(n^{1-\kappa}\Delta)$ points is with high probability a coreset. Here, Δ is the VC-dimension of the loss function. Thus, coresets exists for general input instances for the RLM problem, showing regularization allows us to break through the lower bounds shown in prior work! Formally, this scaling condition says that if $\ell(-\|\beta\|) = 0$ then $r(\beta)$ must be a constant fraction of $\ell(\|\beta\|_2)$. We show that this scaling condition holds when the loss function is either logistic regression or SVM, and the regularizer is the 1-norm, the 2-norm, or 2-norm squared. For example, in the recommended case that $\kappa = 1/2$, the

[1] To see this note that if we multiplied each coordinate of each point x_i by a factor of c, the optimal hypothesis β would decrease by a factor of c, Thus, decreasing the value of all of the standard regularizers.

scaling condition ensures that a uniform sample of $\tilde{\Theta}(d\sqrt{n})$ points is with high probability a coreset when the regularizer is one of the standard ones, and the loss function is either logistic regression and SVM, as they have VC-dimension $O(d)$. Note also that uniform sampling can be reasonably implemented in all of the popular restricted access models. Therefore, this yields a reasonable algorithm for all of the restricted access models under the assumption that a data set of size $\tilde{\Theta}(d\sqrt{n})$ can be stored, and reasonably solved in the main memory of one computer.

We complement our upper bound with two lower bounds on the size of coresets. Our lower bounds assume the 2-norm squared as the regularizer, since intuitively this is the standard regularizer for which it should be easiest to attain small coresets. We first show that our analysis is asymptotically tight for uniform sampling. That is, we show that for both logistic regression and SVM, a uniform sample of size $O(n^{1-\kappa-\epsilon})$ may not result in a coreset. We then show for both logistic regression and SVM there are instances in which every core set is of size $\Omega(n^{(1-\kappa)/5-\epsilon})$. This means that more sophisticated sampling methods must still have core sets whose size is in the same ballpark as is needed for uniform sampling. One might arguably summarize our results as saying that the simplest possible sampling method is nearly optimal for obtaining a coreset.

Related Work on Coresets: The most closely related prior work is probably [84], who considered coresets for *unregularized* logistic regression; i.e, the regularization parameter $\lambda = 0$. [84] showed that are data sets for which there do not exist coresets of sublinear size, and then introduced a parameter μ of the instances that intuitively is small when there is no hypothesis that is a good explanation of the labels, and showed that a coreset of size roughly linear in μ can be obtain by sampling each point with a uniform probability plus a probability proportional to its ℓ_2^2 leverage score (which can be computed from a singular value decomposition of the points). This result yields an algorithm, for the promise problem in which μ is known a priori to be small (but it is not clear how to reasonably compute μ), and it is reasonably implementable in the MPC model, and with two passes over the data in the streaming model. It seems unlikely that this algorithm is implementable in the relational model due to the complex nature of the required sampling probabilities. Contemporaneously with our research, [99] obtained results similar in flavor to those of [84]. [99] also showed

that small coresets exist for certain types of RLM instances; in this case, those in which the norm of the optimal hypothesis is small. Therefore, for normalized logistic rregression, [99] shows that when the 2-norm of the optimal β is bound by μ, coresets of size $\tilde{O}(\mu^2 n^{1-\kappa})$ can be obtained by sampling a point with probability proportional to its norm divided by its ordinal position in the sorted order of norms. Therefore, again this yields an algorithm for the promise problem in which μ is known a priori to be small (and again it is not clear how to reasonably compute μ). Due to the complex nature of the probabilities, it is not clear that this algorithm is reasonably implementable in any of the restricted access models that we consider. Thus, from our perspective, there are three key differences between the results of [84] and [99] and our positive result: *(1)* our result applies to all data sets *(2)* we use uniform sampling, and thus *(3)* our sampling algorithm is implementable in all of the restricted access models that we consider.

Surveys of the use of coresets in algorithmic design can be found in [83] and in [55, Chapter 23]. The knowledge that sampling with a probability at least proportional to sensitivity yields a coreset has been used for at least a decade as it is used by [40]. Coresets were used for partitioned clustering problems, such as k-means [54, 79, 19]. Coresets for hard margin SVM are known [56]. These coresets have an approximation guarantee on the quality of the margin to the hyperplane. Unfortunately, these ideas are not applicable to soft-margin SVM.

Coresets have been used the Minimum Enclosing Ball (MEB) problem [55]. Coresets for MEB are the basis for the Core Vector Machine approach to unregularized kernelized SVM [100]. Several strong coresets for computing balls are known [29, 20]. We note that while there is a reduction from the kernelized SVM to MEB, the reduction is not approximation preserving, and thus the existence of coresets for MEB does not imply the existence of coresets for SVM.

Coresets have also been used for submodular optimization [80], clustering [21], Baysian Logistic Regression [61] and in the design of streaming algorithms (e.g. [87]), as well as distributed algorithms (e.g. [77]).

4.1 Upper Bound for Uniform Sampling

In this section, we show that uniform sampling can be used to construct a coreset for regularized loss minimization.

Theorem 34. *Let $(n, X, Y, \ell, r, \lambda, R, \kappa)$ be an instance of the RLM problem where ℓ and r satisfy the (σ, τ)-scaling condition and the loss function has VC-dimension at most Δ. Let $S' = \frac{n}{\tau\lambda} + \frac{\ell(\sigma)}{\ell(-\sigma)} + 1$. A uniform sample of $q = \frac{10S'}{\epsilon^2}(\Delta \log S' + \log(\frac{1}{\delta}))$ points, each with an associated weight of $u = n/q$, is an ϵ-coreset with probability at least $1 - \delta$.*

Proof. With an aim towards applying Theorem 8 we start by upper bounding the sensitivity of an arbitrary point. To this end, consider an arbitrary $i \in [1, n]$ and an arbitrary hypothesis β. First, consider the case that $R \left\|\beta\right\|_2 \geq \sigma$. In this case:

$$
\begin{aligned}
\frac{f_i(\beta)}{F(\beta)} &= \frac{\ell(-y_i\beta \cdot x_i) + \frac{\lambda}{n} r(R\beta)}{\sum_j \ell(-y_j\beta \cdot x_j)) + \lambda\, r(R\beta)} \\
&\leq \frac{\ell(|\beta \cdot x_i|) + \frac{\lambda}{n} r(R\beta)}{\sum_j \ell(-y_j\beta \cdot x_j) + \lambda\, r(R\beta)} && \text{[As the loss function is nondecreasing]} \\
&\leq \frac{\ell(|\beta \cdot x_i|) + \frac{\lambda}{n} r(R\beta)}{\lambda\, r(R\beta)} && \text{[As the loss function is nonnegative]} \\
&\leq \frac{\ell(|\beta \cdot \beta| \frac{R}{\left\|\beta\right\|_2}) + \frac{\lambda}{n} r(R\beta)}{\lambda\, r(R\beta)} && \text{[As maximum is when } x_i = \beta \frac{R}{\left\|\beta\right\|_2}\text{]} \\
&\leq \frac{\ell(R\ \left\|\beta\right\|_2)}{\lambda\, r(R\beta)} + \frac{1}{n} \\
&\leq \frac{\ell(R\ \left\|\beta\right\|_2)}{\lambda\, \tau\, \ell(R \left\|\beta\right\|_2)} + \frac{1}{n} && \text{[By } (\sigma, \tau) \text{ scaling assumption and} \\
& && \text{the assumption } R \left\|\beta\right\|_2 \geq \sigma\text{]} \\
&\leq \frac{1}{\tau\lambda} + \frac{1}{n}
\end{aligned}
$$

Next, consider the case that $R\left\|\beta\right\|_2 < \sigma$. In this case:

$$\begin{aligned}
\frac{f_i(\beta)}{F(\beta)} &= \frac{\ell(-y_i\beta\cdot x_i) + \frac{\lambda}{n}r(R\beta)}{\sum_j \ell(-y_j\beta\cdot x_j) + \lambda\, r(R\beta)} \\
&\leq \frac{\ell(|\beta\cdot x_i|) + \frac{\lambda}{n}r(R\beta)}{\sum_j \ell(-|\beta\cdot x_j|) + \lambda r(R\beta)} && \text{[As the loss function is nondecreasing]} \\
&\leq \frac{\ell(|\beta\cdot\beta|\frac{R}{\|\beta\|_2}) + \frac{\lambda}{n}r(R\beta)}{\sum_j \ell(-|\beta\cdot\beta|\frac{R}{\|\beta\|_2}) + \lambda\, r(R\beta)} && \text{[As maximum is when } x_i = \beta\frac{R}{\|\beta\|_2}\text{]} \\
&\leq \frac{\ell(R\left\|\beta\right\|_2) + \frac{\lambda}{n}r(R\beta)}{\sum_j \ell(-R\left\|\beta\right\|_2) + \lambda\, r(R\beta)} \\
&\leq \frac{\ell(R\ \left\|\beta\right\|_2)}{\sum_j \ell(-R\ \left\|\beta\right\|_2)} + \frac{1}{n} && \text{[As } a,b,c,d \geq 0 \text{ implies } \frac{a+b}{c+d} \leq \frac{a}{c} + \frac{b}{d}\text{]} \\
&\leq \frac{\ell(\sigma)}{\sum_j \ell(-\sigma)} + \frac{1}{n} && \text{[By assumption } R\left\|\beta\right\|_2 < \sigma\text{]} \\
&\leq \frac{\ell(\sigma)}{n\,\ell(-\sigma)} + \frac{1}{n}
\end{aligned}$$

Thus, the sensitivity of every point is at most $\frac{1}{\tau\lambda} + \frac{\ell(\sigma)}{n\,\ell(-\sigma)} + \frac{1}{n}$, and the total sensitivity S is at most $\frac{n}{\tau\lambda} + \frac{\ell(\sigma)}{\ell(-\sigma)} + 1$. The claim follows by Theorem 8. □

Corollary 35. *Let $(n, X, Y, \ell, r, \lambda, R, \kappa)$ be an instance of the RLM problem where the loss function ℓ is logistic regression or SVM, and the regularizer r is one of the 1-norm, 2-norm, or 2-norm squared. Let $S' = \frac{12n}{\lambda} + 6 = 12n^{1-\kappa} + 6$. A uniform sample of $q = \frac{10S'}{\epsilon^2}((d+1)\log S' + \log(\frac{1}{\delta}))$ points, each with an associate weight of $u = \frac{n}{q}$, is an ϵ-coreset with probability at least $1-\delta$.*

Proof. Since the VC-dimension of logistic regression and SVM is at most $d+1$, it is enough to show that the scaling condition holds in each case. First, consider logistic regression. Let $\sigma = 1$. Then we have $l(-1) = \log(1+\exp(-1)) \neq 0$. In the case that $r(\beta) = \left\|\beta\right\|_2$ it is sufficient to take $\tau = \frac{1}{2}$ as $\ell(z) = \log(1+\exp(z)) \leq 2z$ when $z \geq 1$. Similarly its sufficient to take $\tau = \frac{1}{2}$ when the regularizer is the 2-norm squared, as $\ell(z) = \log(1+\exp(z)) \leq 2z^2$ when $z \geq 1$. As $\left\|\beta\right\|_1 \geq \left\|\beta\right\|_2$ it is also sufficient to take $\tau = \frac{1}{2}$ when the regularizer is the 1-norm. Therefore, total sensitivity is bounded by $\frac{2n}{\lambda} + 6$ in all of these cases.

Now consider SVM. Let $\sigma = 1/2$. Then $l(-1/2) = 1/2 \neq 0$. In the case that $r(\beta) = \|\beta\|_2$ it is sufficient to take $\tau = \frac{1}{3}$ as $\ell(z) = 1 + z \leq 3z$ when $z \geq \frac{1}{2}$; $\tau = \frac{1}{3}$ will be also sufficient when the regularizer is the 1-norm since $\|\beta\|_1 \geq \|\beta\|_2$.

Furthermore, if $\|\beta\|_2 \geq 1$, then $\|\beta\|_2^2 \geq 4\|\beta\|_2$; therefore, in the case that $r(\beta) = \|\beta\|_2^2$, it is sufficient to take $\tau = \frac{1}{12}$. Therefore, total sensitivity is bounded by $\frac{12n}{\lambda} + 4$. □

The implementation of uniform sampling and the computation of R in the streaming and MPC models are trivial. Uniform sampling and the computation of R in the relational model can be implemented without joins because both can be done using.

We note that in several other papers (e.g. [83]) coreset construction can be applied recursively to obtain very small coresets. We cannot apply the previous theorem recursively because after sampling and re-weighting, the regularizer stays the same; however, the number of points is less and the weight of loss function for each point is scaled. To see that it is not possible to resample the new instance, it is enough to divide the new error function by the weight of each sample and get an unweighted instance such that its regularizer has a small coefficient; now, having a coreset for the weighted sample is similar to having a coreset for this unweighted sample with a small regularizer. Of course, it can also be seen that the theorem cannot be applied recursively because it would contradict our lower bound in the next section as well.

4.2 Uniform Sampling Lower Bound

In this section we show in Theorem 37 that our analysis of uniform sampling is tight up to poly-logarithmic factors. Before stating the lower bound, we make the following observation about the total weight of any possible coreset.

Observation 36. *Assume that $\ell(0) \neq 0$, as is the case for logistic regression and SVM. If (C, U) is an ϵ-coreset then $(1-\epsilon)n \leq \sum_{i \in C} u_i \leq (1+\epsilon)n$.*

Proof. Applying the definition of coreset in the case that β is the hypothesis with all 0 components, it must be the case that $\left|\sum_{i=1}^n \ell(0) - \sum_{i \in C} u_i \ell(0)\right| \leq \epsilon \sum_{i=1}^n \ell(0)$, or equivalently $\left|n - \sum_{i \in C} u_i\right| \leq \epsilon n$. □

Note that in the special case that each u_i is equal to a common value u, as will be the case for uniform sampling, setting each $u_i = 1$ and scaling λ down by a factor of u, would result in the same optimal hypothesis β.

Theorem 37. *Assume that the loss function is either logistic regression or SVM, and the regularizer is 2-norm squared. Let $\epsilon, \gamma \in (0,1)$ be arbitrary. For all sufficiently large n, there exists an instance I_n of n points such that with probability at least $1 - 1/n^{\gamma/2}$ it will be the case that for a uniform sample C of $c = n^{1-\gamma}/\lambda = n^{1-\kappa-\gamma}$ points, there is no weighting U that will result in an ϵ-coreset.*

Proof. The instance I_n consists of points located on the real line, so the dimension $d = 1$. A collection A of $n - (\lambda n^{\gamma/2})$ points is located at $+1$, and the remaining $\lambda n^{\gamma/2}$ points are located at -1; call this collection of points B. All points are labeled $+1$. Note $R = 1$.

Let C be the random sample of c points, and U an arbitrary weighting of the points in C. Note that U may depend on the instantiation of C. Our goal is to show that with high probability, (C, U) is not an ϵ-coreset. Our proof strategy is to first show that because almost all of the points are in A, it is likely that C contains only points from A. Then we want to show that, conditioned on $C \subseteq A$, that C cannot be a coreset for any possible weighting. We accomplish this by showing that $\lim_{n\to\infty} H(\beta) = 1$ when $\beta = n^{\gamma/4}$.

We now show that one can use a standard union bound to establish that it is likely that $C \subseteq A$. To accomplish this, let E_i be the probability that the the i^{th} point selected to be in C is not in A.

$$\Pr[C \subseteq A] = 1 - \Pr\left[\vee_{i\in C} E_i\right] \geq 1 - |C|\frac{|B|}{n}$$
$$= 1 - \frac{n^{1-\gamma}}{\lambda}\frac{\lambda n^{\gamma/2}}{n} = 1 - \frac{1}{n^{\gamma/2}}$$

Now we show if $C \subseteq A$ and n is large enough, then (C, U) cannot be an ϵ-coreset for any collection U of weights. To accomplish this, consider the the hypothesis $\beta_0 = n^{\gamma/4}$. From the definition of coreset, it is sufficient to show that $H(\beta_0)$, defined as

$$H(\beta_0) = \frac{|\sum_{i\in P} f_i(\beta_0) - \sum_{i\in C} u_i f_i(\beta_0)|}{\sum_{i\in P} f_i(\beta_0)} \tag{11}$$

is greater than ϵ. We accomplish this by showing that the limit as n goes to infinity of $H(\beta_0)$ is 1. Applying Observation 36 we can conclude that

$$H(\beta_0) \geq \frac{|\sum_{i\in P} \ell_i(\beta_0) - \sum_{i\in C} u_i \ell_i(\beta_0)| - \epsilon\lambda \|\beta_0\|_2^2}{\sum_{i\in P} \ell_i(\beta_0) + \lambda \|\beta_0\|_2^2} \tag{12}$$

Then, using the fact that A and B is a partition of the points and $C \subseteq A$ we can conclude that

$$\begin{aligned} &H(\beta_0) \geq \\ &\frac{|\sum_{i\in A} \ell_i(\beta_0) + \sum_{i\in B} \ell_i(\beta_0) - \sum_{i\in C} u_i \ell_i(\beta_0)| - \epsilon\lambda \|\beta_0\|_2^2}{\sum_{i\in A} \ell_i(\beta_0) + \sum_{i\in B} \ell_i(\beta_0) + \lambda \|\beta_0\|_2^2} \\ &= \frac{\left|\frac{\sum_{i\in A} \ell_i(\beta_0)}{\sum_{i\in B} \ell_i(\beta_0)} + 1 - \frac{\sum_{i\in C} u_i \ell_i(\beta_0)}{\sum_{i\in B} \ell_i(\beta_0)}\right| - \frac{\epsilon\lambda\|\beta_0\|_2^2}{\sum_{i\in B} \ell_i(\beta_0)}}{\frac{\sum_{i\in A} \ell_i(\beta_0)}{\sum_{i\in B} \ell_i(\beta_0)} + 1 + \frac{\lambda\|\beta_0\|_2^2}{\sum_{i\in B} \ell_i(\beta_0)}} \end{aligned} \tag{13}$$

We now need to bound various terms in equation (13). Let us first consider logistic regression. Note that

$$\sum_{i\in B} \ell_i(\beta_0) = |B| \log(1 + \exp(n^{\gamma/4})) \geq |B| n^{\gamma/4} = \lambda n^{3\gamma/4} \tag{14}$$

Therefore,

$$\lim_{n\to\infty} \frac{\lambda \|\beta_0\|_2^2}{\sum_{i\in B} \ell_i(\beta_0)} \leq \lim_{n\to\infty} \frac{\lambda n^{\gamma/2}}{\lambda n^{3\gamma/4}} = 0 \tag{15}$$

Moreover, note that

$$\lim_{n\to\infty} \sum_{i\in A} \ell_i(\beta_0) \tag{16}$$

$$= \lim_{n\to\infty} |A| \log(1 + \exp(-n^{\gamma/4})) \tag{17}$$

$$\leq \lim_{n\to\infty} n \exp(-n^{\gamma/4}) = 0 \tag{18}$$

Finally, by Observation 36, we have,

$$\lim_{n\to\infty} \sum_{i\in C} u_i \ell_i(\beta_0) \leq \lim_{n\to\infty} (1+\epsilon) n \exp(-n^{\gamma/4}) = 0 \tag{19}$$

Combining equations (14), (15), (16), and (19), we the expression in equation (13) converges to 1 as $n \to \infty$. Thus, for sufficiently large n, $H(\beta_0) > \epsilon$and thus (C, U) is not an ϵ-coreset.

We now need to bound various terms in equation (13) for SVM. First note that

$$\sum_{i \in B} \ell_i(\beta_0) = |B|(1 + n^{\gamma/4}) \geq |B| n^{\gamma/4} = \lambda n^{3\gamma/4} \tag{20}$$

Therefore,

$$\lim_{n \to \infty} \frac{\lambda \left\|\beta_0\right\|_2^2}{\sum_{i \in B} \ell_i(\beta_0)} \leq \lim_{n \to \infty} \frac{\lambda n^{\gamma/2}}{\lambda n^{3\gamma/4}} = 0 \tag{21}$$

Moreover, note that

$$\lim_{n \to \infty} \sum_{i \in A} \ell_i(\beta_0) = \lim_{n \to \infty} |A| \max(0, 1 - n^{\gamma/4}) = 0 \tag{22}$$

Finally, by Observation 36, we have that:

$$\lim_{n \to \infty} \sum_{i \in C} u_i \ell_i(\beta_0) \leq \lim_{n \to \infty} (1 + \epsilon) n \max(0, 1 - n^{\gamma/4}) = 0 \tag{23}$$

Combining equations (20), (21), (22), and (23), we the expression in equation (13) converges to 1 as $n \to \infty$. Thus, for sufficiently large n, $H(\beta_0) > \epsilon$and thus (C, U) is not an ϵ-coreset. □

4.3 General Lower Bound on Coreset Size

This section is devoted to proving the following theorem:

Theorem 38. *Assume that the loss function is either logistic regression or SVM, and the regularizer is 2-norm squared. Let $\epsilon, \gamma \in (0, 1)$ be arbitrary. For all sufficiently large n, there exists an instance I_n of n points such that I_n does not have an ϵ-coreset of size $O(n^{(1-\kappa)/5-\gamma})$*

4.3.1 Logistic Regression

The goal of this subsection is to prove Theorem 38 when the loss function is logistic regression. The lower bound instance I_n consists of a collection of n positively-labeled points in $\Re^3$ uniformly spaced around a circle of radius 1 centered at $(0, 0, 1)$ in the plane $z = 1$. Note that $R = \sqrt{2}$. However, for convenience, we will project I_n down into a collection X of points in the plane $z = 0$. Therefore, the resulting instance, which we call the circle instance, consists of n points uniformly spread around the unit circle in $\Re^2$, and for a hypothesis $\beta = (\beta_x, \beta_y, \beta_z)$, $F(\beta)$ is now $\sum_{x_i \in X} \ell(-y_i((\beta_x, \beta_y) \cdot x_i + \beta_z)) + 2\lambda \|\beta\|_2^2$. This means β_z can be thought of as an offset or bias term, that allows hypotheses in $\Re^2$ that do not pass through the origin.

Fix a constant $c > 0$ and a subset C of X that has size $k = c\frac{n^{1/5-\gamma}}{\lambda^{1/5}} = cn^{(1-\kappa)/5-\gamma}$ as a candidate coreset. Let U be an arbitrary collection of associated weights. Toward finding a hypothesis that violates equation (7), define a *chunk* A to be a collection of $\frac{n}{4k}$ points in the middle of $\frac{n}{2k}$ consecutive points on the circle that are all not in C. That is, no point in the chunk A is in C, and no point in the next $\frac{n}{8k}$ points in either direction around the circle are in C. Its easy to observe that, by the pigeon principle, a chunk A must exist. Now let $\beta_A = (\beta_x, \beta_y, \beta_z)$ be the hypothesis where $(\beta_x, \beta_y) \cdot x_i + \beta_z = 0$ for the two points $x_i \in X \setminus A$ that are adjacent to the chunk A, that predicts A incorrectly (and thus that predicts the points $X \setminus A$ correctly), and where $\|\beta_A\|_2 = \sqrt{\frac{n^{1-\gamma}}{k\lambda}}$. To establish Theorem 38 we want to show that equation (7) is not satisfied for the hypothesis β_A. By Observation 36 it is sufficient to show that the limit as $n \to \infty$ of:

$$\frac{\left|\sum_{x_i \in X} \ell_i(\beta_A) - \sum_{x_i \in C} u_i \ell_i(\beta_A)\right| - 2\epsilon\lambda \|\beta_A\|_2^2}{\sum_{x_i \in X} \ell_i(\beta_A) + \lambda \|\beta_A\|_2^2} = \frac{\left|1 - \frac{\sum_{x_i \in C} u_i \ell_i(\beta_A)}{\sum_{x_i \in X} \ell_i(\beta_A)}\right| - \frac{2\epsilon\lambda\|\beta_A\|_2^2}{\sum_{x_i \in X} \ell_i(\beta_A)}}{1 + \frac{\lambda\|\beta_A\|_2^2}{\sum_{x_i \in X} \ell_i(\beta_A)}}$$

is 1. To accomplish this, it is sufficient to show that the limits of the ratios in the second expression approach 0, which we do in the next two lemmas.

Lemma 39. $\lim_{n\to\infty} \frac{\lambda\|\beta_A\|_2^2}{\sum_{x_i \in X} \ell_i(\beta_A)} = 0.$

Proof. As the $\frac{n}{4k}$ points in A have been incorrectly classified by β_A, we know that $\ell_i(\beta_A) \geq \log 2$ for $x_i \in A$. Thus:

$$\lim_{n\to\infty} \frac{\lambda \left\|\beta_A\right\|_2^2}{\sum\limits_{x_i \in X} \ell_i(\beta_A)} \leq \lim_{n\to\infty} \frac{\lambda \frac{n^{1-\gamma}}{k\lambda}}{\frac{n}{4k}\log 2} = \lim_{n\to\infty} \frac{4}{n^{\gamma}\log 2} = 0$$

□

Let d_i be the distance between x_i and the line that passes through the first and last points in the chunk A. Let θ_i be the angle formed by the the ray from the origin through x_i and the ray from the origin to them middle point in A. Let $\theta = \max_{i\in A} \theta_i = \frac{2\pi}{n}\frac{n}{8k} = \frac{\pi}{4k}$. We then make two algebraic observations.

Observation 40. *For all $x_i \in X$, $d_i \left\|\beta_A\right\|_2 /2 \leq |(\beta_x, \beta_y) \cdot x_i + \beta_z| \leq d_i \left\|\beta_A\right\|_2$.*

Proof. It is well known that

$$d_i = \frac{|(\beta_x, \beta_y) \cdot x_i + \beta_z|}{\sqrt{\beta_x^2 + \beta_y^2}}$$

Therefore,

$$|(\beta_x, \beta_y) \cdot x_i + \beta_z| = d_i \sqrt{\beta_x^2 + \beta_y^2} \leq \left\|\beta_A\right\| d_i$$

Now we need to show $\left\|\beta_A\right\| d_i/2 \leq |(\beta_x, \beta_y) \cdot x_i + \beta_z|$. Note that there are two points (points adjacent to A) $x_j = (a', b')$ for which $(\beta_x, \beta_y) \cdot x_j + \beta_z = 0$. Consider one of them. We have:

$$\begin{aligned} 0 &= \beta_x a' + \beta_y b' + \beta_z \\ &\geq \beta_z - |\beta_x a' + \beta_y b'| \\ &\geq \beta_z - \sqrt{\beta_x^2 + \beta_y^2}\sqrt{a'^2 + b'^2} \end{aligned}$$

Since the points are over a circle of size 1 we have $\sqrt{a'^2 + b'^2} = 1$. Therefore,

$$\beta_x^2 + \beta_y^2 \geq \beta_z^2$$

Thus, we can conclude:

$$|(\beta_x, \beta_y) \cdot x_i + \beta_z| = d_i \sqrt{\beta_x^2 + \beta_y^2} \geq \frac{d_i}{\sqrt{2}} \left\|\beta_A\right\| \geq \frac{d_i}{2} \left\|\beta_A\right\|$$

□

Observation 41. *For all $x_i \in X$, $d_i = |\cos(\theta_i) - \cos(\theta)|$.*

Lemma 42. $\lim_{n\to\infty} \frac{\sum_{x_i\in C} u_i \ell_i(\beta_A)}{\sum_{x_i\in X} \ell_i(\beta_A)} = 0.$

Proof. We have:

$$\begin{aligned}
&\lim_{n\to\infty} \frac{\sum_{x_i\in C} u_i \ell_i(\beta_A)}{\sum_{x_i\in X} \ell_i(\beta_A)} \\
&= \lim_{n\to\infty} \frac{\sum_{x_i\in C} u_i \log\left(1+\exp(-((\beta_x,\beta_y)\cdot x_i+\beta_z))\right)}{\sum_{x_i\in X} \ell_i(\beta_A)} \\
&\le \lim_{n\to\infty} \frac{\sum_{x_i\in C} u_i \log\left(1+\exp(-\frac{d_i\|\beta_A\|_2}{2})\right)}{\sum_{x_i\in A} \ell_i(\beta_A)} && \text{By Observation 40} \\
&\le \lim_{n\to\infty} \frac{\sum_{x_i\in C} u_i \log\left(1+\exp(-\frac{\|\beta_A\|_2}{2}(\cos\theta-\cos\theta_i))\right)}{\sum_{x_i\in A} \ell_i(\beta_A)} && \text{By Observation 41} \\
&\le \lim_{n\to\infty} \frac{\sum_{x_i\in C} u_i \exp(-\frac{\|\beta_A\|_2}{2}(\cos\theta-\cos\theta_i))}{\sum_{x_i\in A} \ell_i(\beta_A)} && \text{Since } \log(1+x)\le x \\
&\le \lim_{n\to\infty} \frac{\sum_{x_i\in C} u_i \exp(-\frac{\|\beta_A\|_2}{2}(\cos\frac{\pi}{4k}-\cos\frac{\pi}{2k}))}{\sum_{x_i\in A} \ell_i(\beta_A)} && \text{Since maximizer is when } \theta_i=\frac{\pi}{2k}
\end{aligned}$$

Using the Taylor expansion of $\cos(x) = \sum_{i=0}^{\infty}(-1)^i \frac{x^{2i}}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots$, we have $\cos(\frac{\pi}{4k}) - \cos(\frac{\pi}{2k}) \ge \frac{1}{2}\left((\frac{\pi}{2k})^2 - (\frac{\pi}{4k})^2\right) - O(\frac{1}{k^4}) = (\frac{3\pi^2}{32k^2}) - O(\frac{1}{k^4})$. Plugging this inequality, we derive

$$\begin{aligned}
\frac{\sum_{x_i\in C} u_i \ell_i(\beta_A)}{\sum_{x_i\in P} \ell_i(\beta_A)} &\le \frac{\sum_{x_i\in C} u_i \exp(-\frac{\|\beta_A\|}{2}((\frac{3\pi^2}{32k^2}) - O(\frac{1}{k^4})))}{\sum_{x_i\in A} \ell_i(\beta_A)} \\
&= \frac{\sum_{x_i\in C} u_i \exp(-\frac{n^{2/5}}{2\sqrt{c}\lambda^{2/5}}(\frac{3\pi^2\lambda^{2/5}}{32c^2 n^{2/5-2\gamma}} - O(\frac{\lambda^{4/5}}{n^{4/5-4\gamma}})))}{\sum_{x_i\in A} \ell_i(\beta_A)} \\
&= \frac{\sum_{x_i\in C} u_i \exp(-\alpha n^{2\gamma} + O(\frac{\lambda^{2/5}}{n^{2/5-4\gamma}}))}{\sum_{x_i\in A} \ell_i(\beta_A)},
\end{aligned}$$

where $\alpha > 0$ is a constant. Since all points in A are miss-classified, we have $\ell_i(\beta_A) \geq \log 2$ for all of them. Using this fact and Observation 36, we have:

$$\frac{\sum\limits_{x_i \in C} u_i \ell_i(\beta_A)}{\sum\limits_{x_i \in P} \ell_i(\beta_A)} \leq \frac{(1+\epsilon) n \exp(-\alpha n^{2\gamma} + O(\frac{\lambda^{2/5}}{n^{2/5-4\gamma}}))}{\frac{n}{4k} \log 2}$$

Finally, using the fact that $k = c\frac{n^{1/5-\gamma}}{\lambda^{1/5}}$ and taking the limit, we conclude:

$$\lim_{n\to\infty} \frac{\sum\limits_{x_i \in C} u_i \ell_i(\beta_A)}{\sum\limits_{x_i \in P} \ell_i(\beta_A)} \leq \lim_{n\to\infty} \frac{4k(1+\epsilon) \exp(-\alpha n^{2\gamma} + O(\frac{\lambda^{2/5}}{n^{2/5-4\gamma}}))}{\log 2} = 0$$

□

4.3.2 SVM

The goal of this subsection is to prove Theorem 38 when the loss function is SVM. For the sake of contradiction, suppose an ϵ-coreset (C, u) of size k exists for the circle instance. We fix A to be a chunk. Similar to logistic regression, we set β_A as the parameters of the linear SVM that separates A from P/A such that the model predicts A incorrectly and predicts the points P/A as positive correctly and $\|\beta_A\|_2 = \sqrt{\frac{n^{1-\gamma}}{k\lambda}} = \frac{n^{2/5}}{\sqrt{c}\lambda^{2/5}}$.

Our goal is to show Eqn. (7) tends to 1 as n grows to infinity. We can break the cost function of linear SVM into two parts:

$$F_{P,1}(\beta_A) := \sum_{x_i \in P} \ell_i(\beta_A) + 2\lambda \|\beta_A\|_2^2$$

where $\ell_i(\beta_A) = \max(1 - \beta_A x_i y_i, 0) = \max(1 - ((\beta_x, \beta_y) \cdot x_i + \beta_z) y_i, 0)$. Then, we determine the limit of the following quantities as n grows to infinity.

Lemma 43. *For the circle instance P, if (C, u) is an ϵ-coreset of P with size $k = c\frac{n^{1/5-\gamma}}{\lambda^{1/5}}$ for linear SVM, and A is a chunk, then we have,*

A. $\lim\limits_{n\to\infty} \frac{\lambda\|\beta_A\|_2^2}{\sum\limits_{x_i \in P} \ell_i(\beta_A)} = 0$;

B. $\lim\limits_{n\to\infty} \frac{\sum\limits_{x_i \in C} u_i \ell_i(\beta_A)}{\sum\limits_{x_i \in P} \ell_i(\beta_A)} = 0$.

Using this lemma, which we will prove soon, we can prove Theorem 38 for the linear SVM: The definition of coreset allows us to choose any β, so we can set $\beta = \beta_A$ for a chunk A. Then, by Observation 36, Eqn. (7) simplifies to:

$$\frac{|\sum_{x_i \in X} f_i(\beta_A) - \sum_{x_i \in C} u_i f_i(\beta_A)|}{\sum_{x_i \in X} f_i(\beta_A)} \geq \frac{|\sum_{x_i \in X} \ell_i(\beta_A) - \sum_{x_i \in C} u_i \ell_i(\beta_A)| - 2\epsilon\lambda \left\|\beta_A\right\|_2^2}{\sum_{x_i \in X} \ell_i(\beta_A) + \lambda \left\|\beta_A\right\|_2^2}$$
$$= \frac{|1 - \frac{\sum_{x_i \in C} u_i \ell_i(\beta_A)}{\sum_{x_i \in X} \ell_i(\beta_A)}| - \frac{2\epsilon\lambda\|\beta_A\|_2^2}{\sum_{x_i \in X} \ell_i(\beta_A)}}{1 + \frac{\lambda\|\beta_A\|_2^2}{\sum_{x_i \in X} \ell_i(\beta_A)}},$$

which tends to 1 as $n \to \infty$ by Lemma 43. This implies that (C, u) is not an ϵ-coreset for the circle instance, which is a contradiction. This completes the proof of Theorem 38 for SVM.

4.3.3 Proof of Lemma 43

The remainder of this section is devoted to proving Lemma 43. The proof is very similar to the proof of Lemma 39 and 42.

Proof. **of Claim 1 in Lemma 43** We know for all points in A, $\ell_i(\beta_A) \geq 1$ this is because all of them have been incorrectly classified. We also know that since A is a chunk, $|A| = \frac{n}{4k}$.

Therefore, we have $\sum_{x_i \in A} \ell_i(\beta_A) \geq \frac{n}{4k}$. We also know $\left\|\beta_A\right\|_2 = \sqrt{\frac{n^{1-\gamma}}{k\lambda}}$, so we can conclude

$$\frac{\lambda \left\|\beta_A\right\|_2^2}{\sum_{x_i \in X} \ell_i(\beta_A)} \leq \frac{\lambda \left\|\beta_A\right\|_2^2}{\sum_{x_i \in A} \ell_i(\beta_A)} \leq \frac{\lambda \left\|\beta_A\right\|_2^2}{\frac{n}{4k}} = \frac{\lambda \frac{n^{1-\gamma}}{k\lambda}}{\frac{n}{4k}} = \frac{4}{n^\gamma}.$$

The lemma follows by taking the limit of the above inequality. □

Proof. **of Claim 2 in Lemma 43**. Using Observation 40 and the fact that all points in the coreset are predicted correctly by β_A we have:

$$\frac{\sum_{x_i \in C} u_i \ell_i(\beta_A)}{\sum_{x_i \in X} \ell_i(\beta_A)} \leq \frac{\sum_{x_i \in C} u_i \max\left(0, 1 - \frac{\|\beta_A\| d_i}{2}\right)}{\sum_{x_i \in A} \ell_i(\beta_A)}$$

Then, by Observation 41 we have

$$\frac{\sum_{x_i \in C} u_i \ell_i(\beta_A)}{\sum_{x_i \in X} \ell_i(\beta_A)} \leq \frac{\sum_{x_i \in C} u_i \max\left(0, 1 - \frac{\|\beta_A\|}{2}(\cos\theta - \cos\theta_i)\right)}{\sum_{x_i \in A} \ell_i(\beta_A)}$$

By definition of chunk, we know all the points in C are at least $\frac{n}{4k}$ away from the center of A, which means the closest point in C to chunk A is at least $\frac{n}{8k}$ points away, we have $\theta_i \geq \theta + \frac{2\pi}{n}\frac{n}{8k} = \frac{\pi}{2k}$. Therefore,

$$\frac{\sum\limits_{x_i \in C} u_i \ell_i(\beta_A)}{\sum\limits_{x_i \in X} \ell_i(\beta_A)} \leq \frac{\sum\limits_{x_i \in C} u_i \max\left(0, 1 - \frac{\|\beta_A\|}{2}(\cos\frac{\pi}{4k} - \cos\frac{\pi}{2k})\right)}{\sum\limits_{x_i \in A} \ell_i(\beta_A)}$$

Using the Taylor expansion of $\cos(x) = \sum\limits_{i=0}^{\infty}(-1)^i \frac{x^{2i}}{(2i)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \ldots$, we have,

$$\cos(\frac{\pi}{4k}) - \cos(\frac{\pi}{2k}) \geq \frac{1}{2}\big((\frac{\pi}{2k})^2 - (\frac{\pi}{4k})^2\big) - O(\frac{1}{k^4}) = (\frac{3\pi^2}{32k^2}) - O(\frac{1}{k^4})$$

Therefore, we derive,

$$\begin{aligned}
\frac{\sum\limits_{x_i \in C} u_i \ell_i(\beta_A)}{\sum\limits_{x_i \in X} \ell_i(\beta_A)} &\leq \frac{\sum\limits_{x_i \in C} u_i \max\left(0, 1 - \frac{\|\beta_A\|}{2}((\frac{3\pi^2}{32k^2}) - O(\frac{1}{k^4}))\right)}{\sum\limits_{x_i \in A} \ell_i(\beta_A)} \\
&= \frac{\sum\limits_{x_i \in C} u_i \max\left(0, 1 - \frac{n^{2/5}}{2c\lambda^{2/5}}(\frac{3\pi^2\lambda^{2/5}}{32cn^{2/5-2\gamma}} - O(\frac{\lambda^{4/5}}{n^{4/5-4\gamma}}))\right)}{\sum\limits_{x_i \in A} \ell_i(\beta_A)} \\
&= \frac{\sum\limits_{x_i \in C} u_i \max\left(0, 1 - \alpha n^{2\gamma} + O(\frac{\lambda^{2/5}}{n^{2/5-4\gamma}})\right)}{\sum\limits_{x_i \in A} \ell_i(\beta_A)}
\end{aligned}$$

For large enough n, we have $\max\left(0, 1 - \alpha n^{2\gamma} + O(\frac{\lambda^{2/5}}{n^{2/5-4\gamma}})\right) = 0$. Therefore, by taking the limit we have:

$$\lim_{n\to\infty} \frac{\sum\limits_{x_i \in C} u_i \ell_i(\beta_A)}{\sum\limits_{x_i \in X} \ell_i(\beta_A)} \leq \lim_{n\to\infty} \frac{\sum\limits_{x_i \in C} u_i \max\left(0, 1 - \alpha n^{2\gamma} + O(\frac{\lambda^{2/5}}{n^{2/5-4\gamma}})\right)}{\sum\limits_{x_i \in A} \ell_i(\beta_A)} = 0$$

□

5.0 Relational Gradient Descent Algorithm For Support Vector Machine Training

In this section, we address the relational learning question within the context of gradient descent algorithms for the classic (soft-margin linear) Support Vector Machine (SVM) training problem. SVM is identified as one of the five most important learning problems in [31], and is covered in almost all introductory machine learning textbooks. Gradient descent is probably the most commonly used computational technique for solving convex learning optimization problems [98]. So plan A is to find a relational implementation of gradient descent for the SVM objective. And if plan A fails, plan B is to find a relational descent algorithm that has the same performance guarantee as gradient descent. And finally, if both plan A fail and plan B fail, plan C is to find a relational algorithm that has some other reasonable performance guarantee.

We start by making some observations about the gradient

$$\nabla F = 2\lambda\beta - \frac{1}{N}\sum_{i\in\mathcal{L}} y_i x_i \tag{24}$$

of the SVM objective function F. First note the term $2\lambda\beta$ is trivial to compute, so let us focus on the term $G = \frac{1}{N}\sum_{i\in\mathcal{L}} y_i x_i$. Firstly, only those points x_i that satisfy the additive constraint $\mathcal{L}$ contribute to the gradient. Now let us focus on a particular dimension, and use x_{ik} to refer to the value of point x_i in dimension k. Let $L_k^- = \{i \mid i \in \mathcal{L} \text{ and } y_i x_{ik} < 0\}$ denote those points that satisfy $\mathcal{L}$ and whose the gradient in the k^{th} coordinate has a negative sign. Conceptually, each point in L_k^- pushes the gradient in dimension k up with "force" proportional to its value in dimension k. Let $L_k^+ = \{i \mid x_i \in \mathcal{L} \text{ and } y_i x_{ik} > 0\}$ denote those points that satisfy $\mathcal{L}$ and whose the gradient in the k^{th} coordinate has positive sign. And conceptually, each point in L_k^+ pushes the gradient in dimension k down with "force" proportional to its value in dimension k.

Next, we note that $G = \frac{1}{N}\sum_{i\in\mathcal{L}} y_i x_i$ is a FAQ-AI(1) query for which we have given a relational approximation scheme (RAS). The results in Chapter 3 can be applied to obtain a RAS to compute a $(1+\epsilon)$ approximation $\widehat{G}_k^+$ to $G_k^+ = \frac{1}{N}\sum_{i\in L_k^+} y_i x_{ik}$, and a RAS to compute

a $(1+\epsilon)$ approximation $\widehat{G}_k^-$ to $G_k^- = \frac{1}{N}\sum_{i \in L_k^-} y_i x_{ik}$. However, the results in Chapter 3 cannot be applied to get a RAS for computing a $(1+\epsilon)$-approximation to $G = G_k^- + G_k^+$, as it suffers from the subtraction problem. Conceptually, the subtraction problem is the fact that good approximations of scalars a and b are generally insufficient to deduce a good approximation of $a - b$.

Thus, an additional reason for our interest in relational algorithms to compute the (perhaps approximate) gradient of the SVM objective function is that we want to use it as test case to see if there is some way that we can surmount/circumvent the subtraction problem, and obtain a relational algorithm with a reasonable performance guarantee, ideally using techniques that are applicable to other problems in which this subtraction problem arises.

We start with a discouraging negative result that shows that we cannot surmount the subtraction problem in the context of computing the gradient of the SVM objective problem. In particular, we show in Section 5.1 that computing an $O(1)$ approximation to the partial derivative in a specified specified dimension is $\#P$-hard, even for acyclic joins. This hardness result kills the plan A as a relational algorithm to compute the gradient would imply $P = \#P$. This also makes it hard to imagine plan B working out since, assuming $P \neq \#P$, a relational algorithm cannot even be sure that it is even approximately headed in the direction of the optimal solution, and thus its not reasonable to expect that we could find a relational algorithm to compute some sort of "pseudo-gradient" that would guarantee convergence on all instances.

Thus, it seems we have no choice but to fall back to plan C. That is, we have to try to circumvent (not surmount) the subtraction problem. After some reflection, one reasonable interpretation of our $\#P$-hardness proof is that it shows that computing the gradient is hard on unstable instances. In this context, intuitively, an instance is stable if a nearly optimal solution remains nearly optimal if the points are perturbed slightly. Intuitively, one would expect real world instances, where there is a hypothesis β that explains the labels reasonably well, to be relatively stable (some discussion of the stability of SVM instances can be found in [25]). And for instances where there isn't a hypothesis that explains the labels reasonably well, it probably doesn't matter what hypothesis the algorithm returns, as it will likely be discarded by the data scientist anyways. Thus, our plan C will be to seek a gradient descent

algorithm that has a similar convergence guarantee to gradient descent on stable instances.

Long story short, the main result of this chapter is that this plan C works out. That is, we give a relational algorithm that computes a "pseudo-gradient" that guarantees convergence for stable instances at a rate comparable to that achieved by using the actual gradient.

Stability analysis, similar in spirit to our results for linear SVM, has been considered before in clustering problems [90, 26, 74, 10, 22, 39, 67, 88, 14, 18, 23]. [26, 74] shows that instances of the Max Cut, in which a multiplicative perturbation of the edge weights does not change the optimal answer, are easier to solve. Furthermore, the NP-hard k-means, k-medians and k-centers clustering problems are polynomially solvable for instances in which changing the distances of the points by a multiplicative factor of at most 2 does not change the optimal solution [14, 18, 23]

The algorithm design can be found in Section 5.2, and the algorithm analysis can be found in Section 5.3. Postponing for the moment our formal definition of stability, we state our main result in Theorem 44. The reader should compare Theorem 44 to the analysis of gradient descent in Theorem 4.

Theorem 44. *Let X be an (α, δ, γ)-stable SVM instance formed by an acyclic join. Let $\beta^* = \text{argmin}_\beta F(\beta)$ be the optimal solution. Then there is a relational algorithm that can compute a pseudo-gradient in time $O(\frac{m}{\epsilon^2}(m^3 \log^2(n))^2(d^2 mn \log(n)))$, where $\epsilon = \min(\frac{\delta}{8}, \alpha)$. After $T = \left(\frac{256 d^{3/2}}{\lambda \delta F(\beta_a, X_a)}\right)^2$ iterations of projected descent using this pseudo-gradient there is a relational algorithm that can compute in time $O(\frac{1}{\epsilon^2}(m^3 \log^2(n))^2(d^2 mn \log(n)))$ a hypothesis $\widehat{\beta}$ such that:*

$$F(\widehat{\beta}, X) \leq (1 + \gamma) F(\beta^*, X)$$

Our main takeaway point is that in a broader context, we believe that our results suggest that this sort of stability analysis would likely yield useful insights in the context of designing relational algorithms for other learning problems in which the subtraction problem arises.

5.1 Hardness of Gradient Approximation

Lemma 45. *It #P hard to $O(1)$-approximate the partial derivative of the SVM objective function in a specified dimension.*

Proof. We reduce the decision version of the counting knapsack problem to the problem of approximating the gradient of SVM. The input to the decision counting knapsack problem is a set of weights $W = \{w_1, w_2, \ldots, w_m\}$, a knapsack size L, and an integer k. The output of the problem is whether there are k different combinations of the items that fit into the knapsack.

We create $m + 1$ tables, each with two columns. The columns of the first m table are (Key, E_i) for T_i and the rows are

$$T_i = \{(1, 0), (1, w_i/L), (0, 0)\}.$$

The last table has two columns (Key, Value), and it has two rows $(1, 1), (0, -k)$. Note that if we take the join of these tables, there will be $m + 2$ columns: $(\text{Key}, \text{Value}, E_1, E_2, \ldots, E_m)$.

Let $\beta = (0, 0, 1, 1, \ldots, 1)$ and $\lambda = 0$, so β is 0 on the columns Key and Value and 1 everywhere else. Then we claim, if the gradient of F on the second dimension (Value) is nonnegative, then the answer to the original counting knapsack is true, otherwise, it is false.

To see the reason, consider the rows in J: there are 2^m rows in the design matrix that have $(1, 1)$ in the first two dimensions and all possible combinations of the knapsack items in the other dimensions. More precisely, the concatenation of $(1, 1)$ and w_S for every $S \in [m]$ where w_S is the vector that has w_i/L in the i-th entry if item i is in S or 0 otherwise. Furthermore, J has a single special row with values $(0, -k, 0, 0, \ldots, 0)$. Letting G_2 be the gradient of SVM on the second dimension (column Value), we have,

$$G_2 = \sum_{x \in J : 1 - \beta x \geq 0} x_2$$

For the row with Key = 1 for each $S \in [m]$, we have $1 - \beta x = 1 - \sum_{i \in S} w_i/L \geq 0$ if and only if the items in S fits into the knapsack and $x_2 = 1$. For the single row with Key = 0, we have $1 - \beta x = 1$, and its value on the second dimension is $x_2 = k$. Therefore,

$$G_2 = C_L(w_1, \ldots, w_m) - k$$

where C_L is the number of subsets of items fitting into the knapsack of size L. This means if we could approximate the gradient up to any constant factor, we would be able to determine if G_2 is positive or negative, and as a result we would be able to answer the (decision version of) counting knapsack problem, which is $\#P$-hard. □

5.2 Algorithm Design

5.2.1 Review of Row Counting with a Single Additive Constraint

We now summarize the algorithmic results from Section 3.2 for two different problems, that we will use as a black box.

In the first problem the input is a collection $T_1, \ldots, T_m$ of tables, a label $\ell \in \{-1, +1\}$, and an additive inequality $\mathcal{L}$ of the form $\sum_{j \in [d]} g_j(x_j) \geq R$, where each function g_j can be computed in constant time. The output consists of, for each $j \in [d]$ and $e \in D(j)$, where $D(j)$ is the domain of column/feature j, the number $C^{\ell}_{j,v}$ of rows in the design matrix $J = T_1 \bowtie \ldots \bowtie T_m$ that satisfy constraint $\mathcal{L}$, that have label ℓ, and that have value v in column j. The Row Counting Algorithm introduced in Section 3.2 computes a $(1+\epsilon)$-approximation for each such $\widehat{C}^{\ell}_{j,v}$ to each $C^{\ell}_{j,v}$, and that runs in time $O(\frac{m}{\epsilon^2}(m^3 \log^2(n))^2(d^2 m n^h \log(n)))$

In the second problem the input is a collection $T_1, \ldots, T_m$ of tables, a label $\ell \in \{-1, +1\}$, and an expression in the form of $\sum_{j \in [d]} g_j(x_j)$, where the g_j functions can be computed in constant time. The output consists of, for each $k \in [0, \log_{1+\epsilon} N]$, maximum value of H_k such that the number of points in the design matrix $J = T_1 \bowtie \ldots \bowtie T_m$ with label $\ell \in \{-1, 1\}$ satisfying the additive inequality $\sum_{j \in [d]} g_j(x_j) \geq H_k$ is at least $\lfloor (1+\epsilon)^k \rfloor$. Then Section 3.2, gives an algorithm for this problem, which we will call the Generalized Row

Counting Algorithm. The algorithm runs in time $O(\frac{1}{\epsilon^2}(m^3\log^2(n))^2(d^2mn^h\log(n)))$. Using the result of the algorithm, for any scalar distance H, it is possible to obtain a row count $\hat{N}(H)$ such that $N(H)/(1+\epsilon) \leq \hat{N}(H) \leq N(H)$, where $N(H)$ is the number of points in the design matrix with label ℓ satisfying the inequality $\sum_{j\in[d]} g_j(x_j) \geq H_k$.

5.2.2 Overview of Our Approach

Recall from the introduction that the difficulty arises when a $\widehat{G}_k^+$ is approximately equal to $-\widehat{G}_k^-$. In this case, it would seem that by appropriately perturbing one of L_1^- or L_1^+ by a relatively small amount one could force $G = \widehat{G}^- + \widehat{G}^+$ for this perturbed instance. In which case, if we used $2\lambda\beta^{(t)} + (\widehat{G}^- + \widehat{G}^+)$ as the pseudo-gradient, then it would be the true gradient for a slightly perturbed instance. However, this is not quite right, as there is an additional issue. If we perturb a point x_i, then the sign of $1 - y_i\beta x_i$ may change, which means this point's contribution to the gradient may discontinuously switch between 0 and $-y_i x_i$. To address this issue, when computing the pseudo-gradient, we use a new instance X' that excludes points that are "close" to the separating hyperplane $1 - y_i\beta x_i = 0$. That is, X' excludes every point that can change sides of the hyperplane in an ϵ-perturbation of each coordinate. This will allow us to formally conclude that if we used $2\lambda\beta^{(t)} + (\widehat{G}^- + \widehat{G}^+)$, where $\widehat{G}^-$ and $\widehat{G}^+$ are defined on X', as the pseudo-gradient, then it would be the true gradient for a slightly perturbed instance. After the last descent step, we choose the final hypothesis to be the ϵ-perturbation of any computed hypothesis $\beta^{(t)}$, $t \in [0, T]$ that minimizes the SVM objective.

In the analysis, we interpret the sequence $\beta^{(0)}, \beta^{(1)}, \ldots, \beta^{(T)}$ as solving an online convex optimization problem, and apply known techniques from this area.

5.2.3 Pseudo-gradient Descent Algorithm

Firstly, in linear time it is straight-forward to determine if the points in X lie in $[-1, 1]$, and if not, to rescale so that they do; this can be accomplished by, for each feature, dividing all the values of that feature in all input tables by the maximum absolute value of that feature. The initial hypothesis $\beta^{(0)}$ is the origin. For any vector v, let $u = |v|$ be a vector such that its entries are the absolute values of v, meaning for all j $u_j = |v_j|$.

Algorithm to Compute the Pseudo-gradient:

A. Run the Row Counting Algorithm to compute, for each $j \in [d]$ and $v \in D(j)$, a $(1+\epsilon)$ approximation $\widehat{C}^-_{j,v}$ to $C^-_{j,v}$, which is the number of rows in $x \in J$ with negative label, satisfying $1 + \beta^{(t)} \cdot x \geq \epsilon \left|\beta^{(t)}\right| \cdot |x|$.

B. Run the Row Counting Algorithm to compute, for each $j \in [d]$ and $v \in D(j)$, a $(1+\epsilon)$ approximation $\widehat{C}^+_{j,v}$ to $C^+_{j,v}$, which is the number of rows in $x \in J$ with positive label, satisfying $1 - \beta^{(t)} \cdot x \geq \epsilon \left|\beta^{(t)}\right| \cdot |x|$.

C. For all $k \in [d]$, compute $\widehat{G}^-_k = \sum_{v \in D(k): v<0} v\, \widehat{C}^-_{k,v} - \sum_{v \in D(k): v \geq 0} v\, \widehat{C}^+_{k,v}$.

D. For all $k \in [d]$, compute $\widehat{G}^+_k = \sum_{v \in D(k): v \geq 0} v\, \widehat{C}^-_{k,v} - \sum_{v \in D(k): v<0} v\, \widehat{C}^+_{k,v}$.

E. The pseudo-gradient is then

$$\widehat{G} = \frac{\widehat{G}^- + \widehat{G}^+}{N} + \lambda \beta^{(t)}$$

Algorithm for a Single Descent Step: The next hypothesis $\beta^{(t+1)}$ is

$$\beta^{(t+1)} = \Pi_{\mathcal{K}}(\beta^{(t)} - \eta_{t+1}\widehat{G})$$

Here $\eta_t = \frac{1}{\lambda\sqrt{dt}}$ and $\Pi_{\mathcal{K}}(\beta)$ is the projection of β onto a hypersphere $\mathcal{K}$ centered at the origin with radius $\frac{\sqrt{d}}{2\lambda}$. Note that $\Pi_{\mathcal{K}}(\beta)$ is β if $\|\beta\|_2 \leq \frac{\sqrt{d}}{2\lambda}$ and $\frac{\sqrt{d}}{2\lambda\|\beta\|_2}\beta$ otherwise.

Algorithm to Compute the Final Hypothesis: After $T-1$ descent steps, the algorithm calls the Generalized Row Counting twice for each $t \in [0, T-1]$, with the following inputs:

- $\ell = 1$ and additive expression $1 - \beta^{(t)} \cdot x_i - \epsilon|\beta^{(t)}| \cdot |x_i|$
- $\ell = -1$ and additive expression $1 + \beta^{(t)} \cdot x_i - \epsilon|\beta^{(t)}| \cdot |x_i|$

Note that both of these expressions are equivalent to $1 - y_i\beta^{(t)} \cdot x_i - \epsilon|\beta^{(t)}| \cdot |x_i|$. Let the array H^+ be the output for the first call and H^- be the output for the second call. Note that H^+ and H^- are monotonically decreasing by the the definition of the Generalized Row Counting algorithm. Let L^+ be the largest k such that $H^+_k \geq 0$ and L^- be the largest k such that $H^-_k \geq 0$.

The algorithm then returns as its final hypothesis $\widehat{\beta}$, the hypothesis $\beta^{(\widehat{t})}$ where $\widehat{t}$ is defined by:

$$\widehat{t} = \underset{t \in [T]}{\operatorname{argmin}} \widehat{F}(\beta^{(t)}, X) \tag{25}$$

where

$$\begin{aligned} \widehat{F}(\beta^{(t)}, X) =& \frac{1}{N} \left(\sum_{k=0}^{L^+-1} (1+\epsilon)^k (H_k^+ - H_{k+1}^+) + (1+\epsilon)^{L^+} H_{L^+}^+ \right) \\ &+ \frac{1}{N} \left(\sum_{k=0}^{L^--1} (1+\epsilon)^k (H_k^- - H_{k+1}^-) + (1+\epsilon)^{L^-} H_{L^-}^- \right) + \lambda \left\| \beta^{(t)} \right\|_2^2 \end{aligned} \tag{26}$$

Note that the values L^-, L^+, H^+ and H^- in the definition of $\widehat{F}$, in equation (26), all depend upon t, which we suppressed to make the notation somewhat less ugly.

5.3 Algorithm Analysis

In Section 5.3.1 we prove Theorem 47 which bounds the convergence of our project pseudo-gradient descent algorithm in a rather nonstandard way by applying known results on online convex optimization [24, 58]. In Section 5.3.2 we introduce our definition of stability and then prove Theorem 44.

5.3.1 Perturbation Analysis

Before stating Theorem 47 we need some definitions.

Definition 46.

- *A point p is an ϵ-perturbation of point q if every component of p is within $(1+\epsilon)$ factor of the corresponding component of q. Meaning in each dimension j we have $(1-\epsilon)q \leq p \leq (1+\epsilon)q$*
- *A point set X_a is an ϵ-perturbation of a point set X_b if there is a bijection between X_a and X_b such that every point in X_a is an ϵ-perturbation of its corresponding point in X_b.*
- *Let $\beta^* = \operatorname{argmin}_\beta F(\beta, X)$ to be the optimal solution at X.*

- *For any ϵ-perturbation X_a of X, define $\beta_a^* = \text{argmin}_\beta F(\beta, X_a)$ to be the optimal solution at X_a.*
- *For a given hypothesis β, we call a point x with label y* close *if there is some ϵ-perturbation x' of x such that $1 - y\beta x' < 0$; otherwise it is called* far. *In other words, a point x with label y is close if $1 - y\beta \cdot x < \epsilon \, |\beta| \cdot |x|$*

Theorem 47. *Assume our projected pseudo-gradient descent algorithm ran for $T-1$ descent steps. Then for all hypotheses $\beta \in \mathbb{R}^d$ there exist ϵ-perturbations X_a and X_b of X such that*

$$F(\widehat{\beta}, X_a) \leq (1+\epsilon)F(\beta, X_b) + \frac{32d^{3/2}}{\lambda\sqrt{T}}$$

To prove Theorem 47, our main tool is a result from the online convex optimization literature [24, 58].

Theorem 48. *[24, 58] Let $g_1, g_2, ..., g_T : \mathbb{R}^n \to \mathbb{R}$ be G-Lipschitz functions over a convex region $\mathcal{K}$, i.e., $||\nabla g_t(\beta)|| \leq G$ for all $\beta \in \mathcal{K}$ and all t. Then, starting at point $\beta^{(0)} \in \mathbb{R}^n$ and using the update rule of $\beta^{(t)} \leftarrow \Pi_{\mathcal{K}}\left(\beta^{(t-1)} - \eta_t \nabla g_{t-1}(\beta^{(t-1)})\right)$, with $\eta = \frac{D}{G\sqrt{t}}$ for $T-1$ steps, we have*

$$\frac{1}{T}\sum_{t=0}^{T-1} g_t(\beta^{(t)}) \leq \frac{1}{T}\sum_{t=0}^{T-1} g_t(\beta^*) + \frac{2DG}{\sqrt{T}} \tag{27}$$

for all β^* with $||\beta^{(0)} - \beta^*|| \leq D$.

To apply this Theorem 48, we set $g_t = F(\beta^{(t)}, X^{(t)}, Y)$, where $X^{(t)}$ is an ϵ-perturbation of X, such that the pseudo-gradient at X is equal to the true gradient at $X^{(t)}$. We establish the existence of $X^{(t)}$ in Lemma 49. Thus, our projected pseudo-gradient descent algorithm updates the hypothesis exactly the same as stated in Theorem 47 (assuming that we use the same upper bounds on D and G). Then in definition 50 we identify the ϵ-permutation Z that minimizes $F(\beta, Z)$, and then in Lemma 51 bound the relative error between $\widehat{F}(\beta, X)$ and $F(\beta, Z)$. Finally, this will allow use in Lemma 52 and Lemma 53 we show the existence of X_b and X_a, respectively, that will allow us to conclude the proof of Theorem 47.

Lemma 49. *In every descent step t, the computed pseudo-gradient $\widehat{G}$ is the exact gradient of $F(\beta^{(t)}, X^{(t)})$ for some point set $X^{(t)}$ that is an ϵ-perturbation of X.*

Proof. To prove the claim, we show how to find a desired $X^{(t)}$ – this is only for the sake of the proof and the algorithm doesn't need to know $X^{(t)}$. We call any point x with label y "far" if it satisfies the inequality

$$1 - y\beta^{(t)} \cdot x \geq \epsilon \left|\beta^{(t)}\right| \cdot |x| \tag{28}$$

, otherwise we call the point "close". Note that for a far point there is no ϵ-perturbation to make the derivative of the loss function 0. That is, for any point x with label y, if $1 - y\beta \cdot x \geq \epsilon \sum_{j\in[d]} |\beta_j| \, |x_j|$, then we have $1 - y\beta x' \geq 0$ for any x' that is ϵ-perturbation of x. To see this, note that we have $1 - y\beta x' = 1 - \sum_{k=1}^{d} \beta_k x'_k \geq 1 - \sum_{k=1}^{d} (\beta_k x_k + |\beta_k| \, |x_k|) \geq 0$ because of x' being ϵ-perturbation of x. On the other hand, for all the close points there exists a perturbation x' such that $1 - y\beta^{(t)} \cdot x' < 0$. We first perturb all of the close points such that they do not have any effect on the gradient.

Next, we need to show a perturbation of the far points for which the $\widehat{G}$ is the gradient of the loss function. Let X_f^+ and X_f^- be the set of far points with positive and negative labels. Let $X_f = X_f^+ \cup X_f^-$. We show the perturbation for each dimension k separately. Based on definition of $\widehat{G}_k^+$ and $\widehat{G}_k^-$ we have:

$$\begin{aligned}
\widehat{G}_k^+ + \widehat{G}_k^- &= \sum_{v\in D(k)} v \, \widehat{C}_{k,v}^- - \sum_{v\in D(k)} v \, \widehat{C}_{k,v}^+ \\
&= \sum_{v\in D(k)} v \, (1 \pm \epsilon) C_{k,v}^- - \sum_{v\in D(k)} v \, (1 \pm \epsilon) C_{k,v}^+
\end{aligned}$$

Note that $C_{k,v}^+$ is the number of points in X_f^+ with value v in dimension k. Therefore,

$$\begin{aligned}
\widehat{G}_k^+ + \widehat{G}_k^- &= \sum_{v\in D(k)} v \, (1 \pm \epsilon) C_{k,v}^- - \sum_{v\in D(k)} v \, (1 \pm \epsilon) C_{k,v}^+ \\
&= \sum_{x_i\in X_f^-} (1 \pm \epsilon) x_{i,k} - \sum_{x_i\in X_f^+} (1 \pm \epsilon) x_{i,k} \\
&= - \sum_{x_i\in X_f} (1 \pm \epsilon) y_i x_{i,k}
\end{aligned}$$

where the last term is $N \frac{\partial L(\beta^{(t)}, X^{(t)})}{\partial \beta_k^{(t)}}$ where $X^{(t)}$ an ϵ-perturbation of X. □

Definition 50. *Let $Z^{(t)}$ be an ϵ-perturbation of X such that for all $z_i \in Z^{(t)}$ and for all dimensions k*

$$z_{i,k} = \begin{cases} (1-\epsilon)x_{i,k} & y_i\beta_k^{(t)} \geq 0 \\ (1+\epsilon)x_{i,k} & y_i\beta_k^{(t)} < 0 \end{cases}$$

Note that this ϵ-perturbation minimizes $F(\beta^{(t)}, Z^{(t)})$.

Lemma 51. $\frac{1}{1+\epsilon}F(\beta^{(t)}, Z^{(t)}) \leq \widehat{F}(\beta^{(t)}, X) \leq F(\beta^{(t)}, Z^{(t)})$.

Proof. Consider a value t and let $N^+(\tau) = \left|\{x_i \mid y_i = +1 \text{ and } 1 - \beta^{(t)} \cdot x_i - \epsilon\left|\beta^{(t)}\right| \cdot |x_i| \geq \tau\}\right|$, and $N^-(\tau) = \left|\{x_i \mid y_i = -1 \text{ and } 1 + \beta^{(t)} \cdot x_i - \epsilon\left|\beta^{(t)}\right| \cdot |x_i| \geq \tau\}\right|$.

Before proving the lemma, we prove the following claim: $F(\beta^{(t)}, Z^{(t)}) = \frac{1}{N}\int_{\tau=0}^{\infty} N^+(\tau)\mathrm{d}\tau + \frac{1}{N}\int_{\tau=0}^{\infty} N^-(\tau)\mathrm{d}\tau + \lambda\left\|\beta^{(t)}\right\|^2$.

Note that based on the definition of $Z^{(t)}$ it is the case that $1 - y_i\beta^{(t)} \cdot z_i = 1 - y_i\beta^{(t)} \cdot x_i - \epsilon\left|\beta^{(t)}\right| \cdot |x_i|$; therefore, $N^+(\tau) = \left|\{y_i = +1 \in Z^{(t)} \text{ and } 1 - y_i\beta^{(t)} \cdot z_i \geq \tau\}\right|$ and $N^-(\tau) = \left|\{y_i = -1 \in Z^{(t)} \text{ and } 1 - y_i\beta^{(t)} \cdot z_i \geq \tau\}\right|$. Hence,

$$\begin{aligned} L(\beta^{(t)}, Z^{(t)}) &= \frac{1}{N}\sum_i \max(0, 1 - y_i\beta \cdot z_i) = \frac{1}{N}\sum_{i:1-y_i\beta\cdot z_i \geq 0} 1 - y_i\beta \cdot z_i \\ &= \frac{1}{N}\sum_{i:1-y_i\beta\cdot z_i \geq 0}\int_{\tau=0}^{1-y_i\beta\cdot z_i} \mathrm{d}\tau = \frac{1}{N}\int_{\tau=0}^{\infty}\sum_{i:1-y_i\beta\cdot z_i \geq \tau} \mathrm{d}\tau \\ &= \frac{1}{N}\int_{\tau=0}^{\infty}(N^+(\tau) + N^-(\tau))\mathrm{d}\tau \end{aligned}$$

Therefore,

$$F(\beta^{(t)}, Z^{(t)}) = \frac{1}{N}\int_{\tau=0}^{\infty} N^+(\tau)\mathrm{d}\tau + \frac{1}{N}\int_{\tau=0}^{\infty} N^-(\tau)\mathrm{d}\tau + \lambda\left\|\beta^{(t)}\right\|^2 \tag{29}$$

The number of points with label ℓ satisfying $1 - \ell\beta^{(t)} \cdot x_i - \epsilon\left|\beta^{(t)}\right| \cdot |x_i| \geq \tau$ for any $\tau \in [H_k^\ell, H_{k+1}^\ell)$ is in the range $[\lfloor(1+\epsilon)^k\rfloor, \lfloor(1+\epsilon)^{(k+1)}\rfloor)$. Therefore, the claim follows by replacing $N^+(\tau)$ in Equation (29) with $(1+\epsilon)^k$ for all the values of $\tau \in [H_k^+, H_{k+1}^+)$ and replacing $N^-(\tau)$ in (29) with $(1+\epsilon)^k$ for all the values of $\tau \in [H_k^-, H_{k+1}^-)$. □

Lemma 52. *For all hypothesis β, there exists an ϵ-perturbation X_b of X such that*

$$\min_s F(\beta^{(s)}, Z^{(s)}) \leq F(\beta, X_b) + \frac{2DG}{\sqrt{T}}$$

Proof. By Theorem 48

$$\frac{1}{T}\sum_{t=0}^{T-1} F(\beta^{(t)}, X^{(t)}) \leq \frac{1}{T}\sum_{t=0}^{T-1} F(\beta, X^{(t)}) + \frac{2DG}{\sqrt{T}} \tag{30}$$

Then

$$\min_s F(\beta^{(s)}, Z^{(s)}) \leq \frac{1}{T}\sum_{t=0}^{T-1} F(\beta^{(t)}, Z^{(t)}) \leq \frac{1}{T}\sum_{t=0}^{T-1} F(\beta^{(t)}, X^{(t)}). \tag{31}$$

The first inequality follows since the minimum is less than the average, and the second inequality follows from the definition of $Z^{(t)}$. Let $u = \text{argmax}_t F(\beta, X^{(t)})$, and $X_b = X^{(u)}$. Then

$$\frac{1}{T}\sum_{t=0}^{T-1} F(\beta, X^{(t)}) \leq \max_t F(\beta, X^{(t)}) = F(\beta, X_b) \tag{32}$$

Thus, combining lines (30), (31) and (32) we can conclude that:

$$\min_s F(\beta^{(s)}, Z^{(s)}) \leq F(\beta, X_b) + \frac{2DG}{\sqrt{T}} \tag{33}$$

□

Lemma 53. *There exists an ϵ-perturbation X_a of X such that*

$$F(\widehat{\beta}, X_a) \leq (1+\epsilon)\min_s F(\beta^{(s)}, Z^{(s)})$$

Proof. Let $X_a = Z^{(\widehat{t})}$ where

$$\begin{aligned}
F(\widehat{\beta}, X_a) &\leq (1+\epsilon)\widehat{F}(\widehat{\beta}, X) && \text{By Lemma 51} \\
&= (1+\epsilon)\min_s \widehat{F}(\beta^{(s)}, X) && \text{By definition of } \widehat{\beta} \\
&\leq (1+\epsilon)\min_s F(\beta^{(s)}, Z^{(s)}) && \text{By Lemma 51}
\end{aligned}$$

□

5.3.2 Stability Analysis

Our formal definition of stability, which we give in Definition 54 while not unnatural, is surely not the first natural formalization that one would think of. Our formal definition is more or less forced on us, which leads to the type of non-traditional approximation achieved in Theorem 47.

Definition 54. *An SVM instance X is (α, δ, γ)-stable for $\delta \leq 1$ if for all X_a and X_b that are α-perturbations of X it is the case that:*

- *β_a^* is a $(1+\delta)$ approximation to the optimal objective value at X_b, that is, $F(\beta_a^*, X_b) \leq (1+\delta)\min_\beta F(\beta, X_b)$.*
- *If β_a is $(1+2\delta)$ approximation to the optimal SVM objective value at X_a then β_a is a $(1+\gamma)$ approximation to the optimal SVM objective value at X_b. That is if $F(\beta_a, X_a) \leq (1+2\delta)\min_\beta F(\beta, X_a)$ then $F(\beta_a, X_b) \leq (1+\gamma)\min_\beta F(\beta, X_b)$*

Proof of Theorem 44. Let $\epsilon \leq \min(\delta/8, \alpha)$.

$$
\begin{aligned}
F(\widehat{\beta}, X_a) &\leq (1+\epsilon)F(\beta_a^*, X_b) + \frac{32d^{3/2}}{\lambda\sqrt{T}} && X_a \text{ and } X_b \text{ come from Theorem 47} \\
&= (1+\epsilon)(1+\delta)F(\beta_a^*, X_a) + \frac{32d^{3/2}}{\lambda\sqrt{T}} && \text{By definition of stability} \\
&= (1+\epsilon)(1+\delta)F(\beta_a^*, X_a) + \frac{\delta}{8}F(\widehat{\beta}, X_a) && \text{By definition of } T \\
&\leq \frac{(1+\delta)(1+\epsilon)}{1-\delta/8}F(\beta_a^*, X_a) && \text{By algebra} \\
&\leq (1+2\delta)F(\beta_a^*, X_a) && \text{by definition of } \epsilon
\end{aligned}
$$

Finally since $\widehat{\beta}$ is $(1+2\delta)$ approximate solution at X_a, by the definition of stability, $\widehat{\beta}$ is a $(1+\gamma)$ approximate solution at X. □

6.0 Relational Algorithms for K-Means Clustering

In this chapter, we design a relational algorithm for k-means clustering. It has a polynomial time complexity for acyclic joins which can be generalized to general joins with a linear dependency on n^{fhtw}. Recall that state-of-the-art algorithms for queries as simple as counting the number of rows in the design matrix have linear dependency on n^{fhtw}; therefore, running in time linear in n^{fhtw} is the goal, as fundamental barriers need to be broken to be faster. Notice that this is polynomial time when fhtw is a fixed constant (i.e., nearly acyclic).

The input to the k-means problem consists of a collection S of points in a Euclidean space and a positive integer k. A feasible output is k points $c_1, \ldots, c_k$, which we call centers. The objective is to choose the centers to minimize the aggregate squared distance from each original point to its nearest center. Recall extracting all data points could take time exponential in the size of a relational database. Thus, the problem is to find the cluster centers without fully realizing all data points the relational data represents.

Although there are algorithms for k-means clustering that can accept relational input [38], there is no algorithm with polynomial time complexity. Therefore, it remains unanswered whether there is a relational algorithm for k-means and what algorithmic techniques we can use for this problem.

Overview of Results: The main result of this chapter is the following.

Theorem 55. *Given an acyclic join query $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$ where the design matrix J has N rows and d columns. Let n be the maximum number of rows in any table. Then there is a randomized algorithm running in time polynomial in d, n and k that computes an $O(1)$ approximate k-means clustering solution with high probability.*

For cyclic joins, we have the same theorem, however, the time complexity has a linear dependence on n^{fhtw}. To illustrate the challenges for finding such an algorithm as described in the prior theorem, consider the following theorem.

Theorem 56. *Given an acyclic join query $J = T_1 \bowtie T_2 \bowtie \cdots \bowtie T_m$ where the design matrix J has N rows and d columns. Given k centers $c_1, \ldots, c_k$, let J_i be the set of points in J*

that are closest to c_i for $i \in [k]$. It is #P-Hard to compute $|J_i|$ for $k \geq 2$ and NP-Hard to approximate $|J_i|$ to any factor for $k \geq 3$.

The proof of this theorem can be found in Section 6.1.1. We show it by reducing a NP-Hard problem to the problem of determining if J_i is empty or not. Counting the points closest to a center is a fundamental building block in almost all k-means algorithms. Moreover, as we show in Theorem 57, even performing one iteration of the classic Lloyd's algorithm is $\#P$-Hard. Together this necessitates the design of new techniques to address the main theorem, shows that seemingly trivial algorithms are difficult relationally, and suggests computing a coreset is the right approach for the problem as it is difficult to cluster the data directly.

Theorem 57. *Given an acyclic join and two centers, it is #P-hard to compute the center of mass for the points assigned to each center.*

Proof. We prove the theorem using a reduction from a decision version of the counting knapsack problem. The input to the counting knapsack problem consists of a the set $W = \{w_1, \ldots, w_n\}$ of positive integer weights, a knapsack size L, and a count D. The problem is to determine whether there are at least D subsets of W with aggregate weight at most L. The points in our instance of k-means will be given relationally. We construct a join query with $n+1$ columns/attributes, and n tables. All tables have one column in common and each has an additional distinct column. More specifically, the i-th table has 2 columns (d_i, d_{n+1}) and three rows $\{(w_i, -1), (0, -1), (0, D)\}$. Note that the join has 2^n rows with -1 in dimension $n+1$, and one row with values $(0, 0, \ldots, 0, D)$. The rows with -1 in dimension $d+1$ have all the subsets of $\{w_1, \ldots, w_n\}$ in their first n dimensions. Let the two centers for k-means problem be any two centers c_1 and c_2 such that a point x is closer to c_1 if it satisfies $\sum_{d=1}^{n} x_d < L$ and closer to c_2 if it satisfies $\sum_{d=1}^{n} x_d > L$. Note that the row $(0, 0, \ldots, 0, D)$ is closer to c_1. Therefore, the value of dimension $n+1$ of the center of mass for the tuples that are closer to c_1 is $Y = (D - C)/C$ where C is the actual number of subsets of W with aggregate weight at most L. If Y is negative, then the number of solutions to the counting knapsack instance is at least D. □

Overview of Techniques: We first compute a **coreset** of all points in J. That is, a collection of points with weights such that if we run an $O(1)$ approximation algorithm on this weighted set, we will get a $O(1)$ approximate solution for all of J. To do so, we sample points according to the principle in k-means++ algorithm and assign weights to the points sampled. The number of points chosen will be $\Theta(k \log N)$. Any $O(1)$-approximate weighted k-means algorithm can be used on the coreset to give Theorem 55.

k-means++: k-means++ is a well-known k-means algorithm [15, 12]. The algorithm iteratively chooses centers $c_1, c_2, \ldots$. The first center c_1 is picked uniformly from J. Given that $c_1, \ldots, c_{i-1}$ are picked, a point x is picked as c_i with probability $P(x) = \frac{L(x)}{Y}$ where $L(x) = \min_{j \in [i-1]}(\|x - c_j\|_2^2)$ and $Y = \sum_{x \in J} L(x)$. Here $[i-1]$ denotes $\{1, 2, \ldots, i-1\}$.

Say we sample $\Theta(k \log N)$ centers according to this distribution, which we call the **k-means++ distribution**. It was shown in [12] that if we cluster the points by assigning them to their closest centers, the total squared distance between points and their cluster centers is at most $O(1)$ times the optimal k-means cost with high probability. Note that this is not a feasible k-means solution because more than k centers are used. However, leveraging this, the work showed that we can construct a coreset by weighting these centers according to the number of points in their corresponding clusters.

We seek to mimic this approach with a relational algorithm. Let us focus on one iteration where we want to sample the center c_i given $c_1, \ldots, c_{i-1}$ according to the k-means++ distribution. Consider the assignment of every point to its closest center in $c_1, \ldots, c_{i-1}$. Notice that the k-means++ probability is determined by this assignment. Indeed, the probability of a point being sampled is the cost of assigning this point to its closest center $(\min_{j \in [i-1]} \|x - c_j\|_2^2)$ normalized by Y. Y is the summation of this cost over all points.

The relational format makes this distribution difficult to compute without the design matrix J. It is hard to efficiently characterize which points are closest to which centers. The assignment *partitions* the data points according to their closest centers, where each partition may not be easily represented by a compact relational database (unlike J).

A Relational k-means++ Implementation: Our approach will sample every point according to the k-means++ distribution without computing this distribution directly. Instead,

we use **rejection sampling** [34], which allows one to sample from a "hard" distribution P using an "easy" distribution Q. Rejection sampling works by sampling from Q first, then reject the sample with another probability used to bridge the gap between Q and P. The process is repeated until a sample is accepted. In our setting, P is the k-means++ distribution, and we need to find a Q which could be sampled from efficiently with a relational algorithm (without computing J). Rejection sampling theory shows that for the sampling to be efficient, Q should be close to P point-wise to avoid high rejection frequency. In the end, we will *perfectly simulate* the k-means++ algorithm.

We now describe the intuition for designing such a Q. Recall that P is determined by the assignment of points to their closest centers. We will approximate this assignment up to a factor of $O(i^2 d)$ when sampling the i^{th} center c_i, where d is the number of columns in J. Intuitively, the approximate assignment makes things easier since for any center we can easily find the points assigned to it using an efficient relational algorithm. Then Q is found by normalizing the squared distance between each point and its assigned center.

The approximate assignment is designed as follows. Consider the d-dimensional Euclidean space where the data points in J are located. The algorithm divides space into a **laminar** collection of **hyper-rectangles**[1] (i.e., $\{x \in \mathcal{R}^d : v_j \leq x_j \leq w_j, j = 1, \ldots, d\}$, here x_j is the value for feature f_j). We assign each hyper-rectangle to a center. A point assigns itself to the center that corresponds to the *smallest* hyper-rectangle containing that point.

The key property of hyper-rectangles that benefits our relational algorithm is: we can efficiently represent all points from J inside any hyper-rectangle by removing some entries in each table from the original database and taking the join of all tables. For example, if a hyper-rectangle has constraint $v_j \leq x_j \leq w_j$, we just remove all the rows with value outside of range $[v_j, w_j]$ for column f_j from the tables containing column f_j. The set of points assigned to a given center can be found by adding and subtracting a laminar set of hyper-rectangles, where each hyper-rectangle can be represented by a relational database.

Weighting the Centers: We have sampled a good set of cluster centers. To get a coreset, we need to assign weights to them. As we have already mentioned, assuming $P \neq \#P$,

[1] A laminar set of hyper-rectangles means any two hyper-rectangles from the set either have no intersection, or one of them contains the other.

the weights cannot be computed relationally. In fact, they cannot be approximated up to any factor in polynomial time unless $P = NP$. Rather, we design an alternative relational algorithm for computing the weights. Each weight will not be an approximate individually, but we prove that the weighted centers form an $O(1)$-approximate coreset in aggregate.

The main algorithmic idea is that for each center c_i we generate a collection of hyperspheres around c_i containing geometrically increasing numbers of points. The space is then partitioned into these hyperspheres where each partition contains a portion of points in J. Using the algorithm from Chapter 3, we then sample a polylog sized collection of points from each partition, and use this subsample to estimate the fraction of the points in this partition which are closer to c_i than any other center. The estimated weight of c_i is aggregated accordingly.

Chapter Organization: We begin with some special cases which help the reader build intuition. In Section 6.1 we give a warm-up by showing how to implement 1-means++ and 2-means++ (i.e. initialization steps of k-means++). In this section, we also prove Theorem 56 as an example of the limits of relational algorithms. In Section 6.2 we give the k-means++ algorithm via rejection sampling. Section 6.3 shows an algorithm to construct the weights and then analyze this algorithm. Many of the technical proofs appear in the appendix due to space.

6.1 Warm-up: Efficiently Implementing 1-means++ and 2-means++

This section is a warm-up to understand the combinatorial structure of relational data. We will show how to do k-means++ for $k \in \{1, 2\}$ (referred to as 1- and 2-means++) on a simple join structure. We will also show the proof of Theorem 56 which states that counting the number of points in a cluster is a hard problem on relational data.

First, let us consider relationally implementing 1-means++ and 2-means++. For better illustration, we consider a path join which is a special case of acyclic join. The relational algorithm used will be generalized to work on more general join structures when we move to the full algorithm in Section 6.2.

Recall from Section 1.1 that in a path join, each table T_i has two features/columns f_i, and f_{i+1}. Tables T_i and T_{i+1} then share a common column f_{i+1}. Assume for simplicity that each table T_i contains n rows. The design matrix $J = T_1 \bowtie T_2 \bowtie \ldots \bowtie T_m$ has $d = m+1$ features, one for each feature (i.e. column) in the input tables. See Figure 1 for an illustration of a path join as a layered DAG.

Even with this simple structure, the size of the design matrix J could still be exponential in the size of database - J could contain up to $n^{m/2}$ rows , and $dn^{m/2}$ entries. Thus, the standard practice could require time and space $\Omega(mn^{m/2})$ in the worst case.

A Relational Implementation of 1-means++: Implementing the 1-means++ algorithm is equivalent to generating a full path uniformly at random from a DAG G as illustrated in Section 1.1. We generate this path by iteratively picking a row from table $T_1, \ldots, T_m$, corresponding to picking an arc pointing from layer f_1 to f_2, f_2 to f_3, ..., such that concatenating all picked rows (arcs) will give a point in J (full path in G).

To sample a row from T_1, for every row $r \in T_1$, consider $r \bowtie J$, which is all rows in J whose values in columns (f_1, f_2) are equivalent to r. Let the function $F_1(r)$ denote the total number of rows in $r \bowtie J$. This is also the number of full paths passing the arc r. Then, every $r \in T_1$ is sampled with probability $\frac{F_1(r)}{\sum_{r' \in T_1} F_1(r')}$, notice $\sum_{r' \in T_1} F_1(r')$ is the total number of full paths. Let the picked row be r_1.

After sampling r_1, we can conceptually throw away all other rows in T_1 and focus only on the rows in J that uses r_1 to concatenate with rows from other tables (i.e., $r_1 \bowtie J$). For any row $r \in T_2$, let the function $F_2(r)$ denote the number of rows in $r \bowtie r_1 \bowtie J$, also equivalent to the total number of full paths passing arc r_1 and r. We sample every r with probability $\frac{F_2(r)}{\sum_{r' \in T_2} F_2(r')}$. Notice that $\sum_{r' \in T_2} F_2(r') = F_1(r_1)$, the number of full paths passing arc r_1. Repeat this procedure until we have sampled a row in the last table T_m: for table T_i and $r \in T_i$, assuming we have sampled $r_1, \ldots, r_{i-1}$ from $T_1, \ldots, T_{i-1}$ respectively, throw away all the other rows in previous tables and focus on $r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$. $F_i(r)$ is the number of rows in $r \bowtie r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$ and r is sampled with probability proportional to $F_i(r)$. It is easy to verify that every full path is sampled uniformly.

For every table T_i we need to find the function $F_i(\cdot)$ which is defined on all its rows. There are m such functions. For each $F_i(\cdot)$, we can find all $F_i(r)$ values for $r \in T_i$ using a one-pass dynamic programming and then sample according to the values. Repeating this procedure m rounds completes the sampling process. This gives a polynomial time algorithm.

A Relational Implementation for 2-means++: Assume $x = (x_1, \ldots, x_d)$ is the first sampled center and now we want to sample the second center. By k-means++ principles, any row $r \in J$ is sampled with probability $\frac{\|r-x\|^2}{\sum_{r' \in J} \|r'-x\|^2}$. For a full path in G corresponding to a row $r \in J$ we refer to $\|r - x\|^2$ as the **aggregated cost** over all d nodes/features.

Similar to 1-means++, we pick one row in each table from T_1 to T_m and putting all the rows together gives us the sampled point. Assume we have sampled the rows $r_1, r_2, \ldots, r_{i-1}$ from the first $i-1$ tables and we focus on all full paths passing $r_1, \ldots, r_{i-1}$ (i.e., the new design matrix $r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$). In 1-means++, we compute $F_i(r)$ which is the total number of full paths passing arc $r_1, \ldots, r_{i-1}, r$ (i.e., $r \bowtie r_1 \bowtie \ldots \bowtie r_{i-1} \bowtie J$.) and sample $r \in T_i$ from a distribution normalized using $F_i(r)$ values. In 2-means++, we define $F_i(r)$ to be the summation of aggregated costs over all full paths which pass arcs $r_1, \ldots, r_{i-1}, r$. We sample $r \in T_i$ from a distribution normalized using $F_i(r)$ values.

It is easy to verify the correctness. Again, each $F_i(\cdot)$ could be computed using a one-pass dynamic programming which gives the values for all rows in T_i when we sample from T_i. This would involve m rounds of such computations and give a polynomial algorithm.

6.1.1 Hardness of Relationally Computing the Weights

Here we prove Theorem 56. We focus on showing that given a set of centers, counting the number of points in J that is closest to any of them is $\#P$-hard. Then we also prove that it is NP-hard to approximate the center weights for three centers. We prove $\#P$-Hardness by a reduction from the well known $\#P$-hard Knapsack Counting problem. The input to the Knapsack Counting problem consists of a set $W = \{w_1, \ldots, w_h\}$ of nonnegative integer weights, and a nonnegative integer L. The output is the number of subsets of W with aggregate weight at most L. To construct the relational instance, for each $i \in [h]$, we define the tables T_{2i-1} and T_{2i} as follows:

T_{2i-1}	
f_{2i-1}	f_{2i}
0	0
0	w_i

T_{2i}	
f_{2i}	f_{2i+1}
0	0
w_i	0

Let centers c_1 and c_2 be arbitrary points such that points closer to c_1 than c_2 are those points p for which $\sum_{i=1}^{d} p_i \leq L$. Then there are 2^h rows in J, since w_i can either be selected or not selected in feature $2i$. The weight of c_1 is the number of points in J closer to c_1 than c_2, which is in turn exactly the number of subsets of W with total weight at most L.

Now we prove the second part of Theorem 56 that given an acyclic database and a set of centers $c_1, \ldots, c_k$, it is NP-Hard to approximate the number of points assigned to each center when $k \geq 3$. We prove it by reduction from Subset Sum. In Subset Sum problem, the input is a set of integers $A = w_1, \ldots, w_m$ and an integer L, the output is true if there is a subset of A such that its summation is L. We create the following acyclic schema. There are m tables. Each table T_i has a single unique column x_i with two rows $w_i, 0$. Then the join of the tables has 2^m rows, and it is a cross product of the rows in different tables in which each row represents one subset of A.

Then consider the following three centers: $c_1 = (\frac{L-1}{m}, \frac{L-1}{m}, \ldots, \frac{L-1}{m})$, $c_2 = (\frac{L}{m}, \ldots, \frac{L}{m})$, and $c_1 = (\frac{L+1}{m}, \frac{L+1}{m}, \ldots, \frac{L+1}{m})$. The Voronoi diagram that separates the points assigned to each of these centers consists of two parallel hyperplanes: $\sum_i x_i = L - 1/2$ and $\sum_i x_i = L + 1/2$ where the points between the two hyperplanes are the points assigned to c_2. Since all the points in the design matrix have integer coordinates, the only points that are between these two hyperplanes are those points for which $\sum_i x_i = L$. Therefore, the approximation for the number of points assigned to c_2 is nonzero if and only if the answer to Subset Sum is true.

6.2 The k-means++ Algorithm

In this section, we describe a relational implementation of the k-means++ algorithm. It is sufficient to explain how the center c_i is picked given the previous centers $c_1, \ldots, c_{i-1}$. Recall that the k-means++ algorithm picks a point x to be c_i with probability $P(x) = \frac{L(x)}{Y}$ where $L(x) = \min_{j \in [i-1]} \|x - c_j\|_2^2$ and $Y = \sum_{x \in J} L(x)$ is a normalizing constant.

The implementation consists of two parts. The first part, described in Section 6.2.2.1, shows how to partition the d-dimensional Euclidean space into a laminar set of hyper-rectangles (referred to as **boxes** hereafter) that are generated around the previous centers. The second part, described in Section 6.2.2.2, samples according to the "hard" distribution P using rejection sampling and an "easy" distribution Q.

Conceptually, we assign every point in the design matrix J to an *approximately* nearest center among $c_1, \ldots, c_{i-1}$. This is done by assigning every point in J to one of the centers contained in the *smallest* box this point belongs to. Then Q is derived using the squared distance between the points in J and their assigned centers.

For illustration, we first show the special case of when $k = 3$, and then we proceed to the general case.

6.2.1 Relational Implementation of 3-means++

Recall that the 3-means++ algorithm picks a point x to be the third center c_3 with probability $P(x) = \frac{L(x)}{Y}$ where $L(x) = \min(\|x - c_1\|_2^2, \|x - c_2\|_2^2)$ and $Y = \sum_{x \in J} L(x)$ is a normalizing constant. Conceptually, think of P as being a 'hard" distribution to sample from.

Description of the Implementation: The implementation first constructs two identically sized axis-parallel hypercubes/boxes b_1 and b_2 centered around c_1 and c_2 that are **as large as possible** subject to the constraints that the side lengths have to be non-negative integral powers of 2, and that b_1 and b_2 cannot intersect. Such side lengths could be found since we may assume c_1 and c_2 have integer coordinates or they are sufficiently far away from each other that we can scale them and increase their distance. Conceptually, the

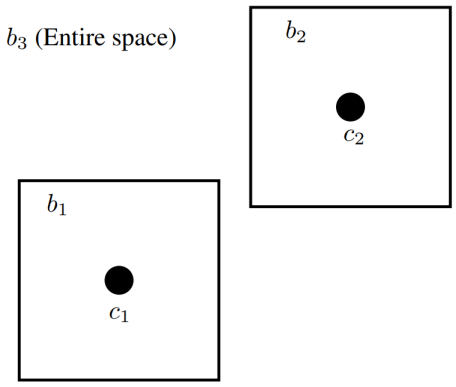

Figure 4: Boxes Used For Sampling Third Center.

implementation also considers a box b_3 that is the whole Euclidean space.

To define our "easy" distribution Q, for each point x define $R(x)$ to be

$$R(x) = \begin{cases} \|x - c_1\|_2^2 & x \in b_1 \\ \|x - c_2\|_2^2 & x \in b_2 \\ \|x - c_1\|_2^2 & x \in b_3 \text{ and } x \notin b_1 \text{ and } x \notin b_2 \end{cases}$$

In the above definition, note that when $x \notin b_1$ and $x \notin b_2$, the distance of x to both centers are relatively similar; therefore, we can assign x to either of the centers – here we have assigned it to c_1. Then $Q(x)$ is defined to be $\frac{R(x)}{Z}$, where $Z = \sum_{x \in J} R(x)$ is a normalizing constant. The implementation then repeatedly samples a point x with probability $Q(x)$. After sampling x, the implementation can either (A) reject x, and then resample or (B) accept x, which means setting the third center c_3 to be x. The probability that x is accepted after it is sampled is $\frac{L(x)}{R(x)}$, and as a result, the probability that x is rejected is $1 - \frac{L(x)}{R(x)}$.

It is straightforward to see how to compute b_1 and b_2 (note that b_1 and b_2 can be computed without any relational operations), and how to compute $L(x)$ and $R(x)$ for a particular point x. Thus, the only non-straight-forward part is the sampling of a point x with probability $Q(x)$. Before explaining this step, we need to explain a simple lemma that describes a subroutine used by the sampling algorithm.

Lemma 58. *Given a point $y \in \mathcal{R}^d$ and a hyper-rectangle $b = \{x \in \mathcal{R}^d : v_i \leq x_i \leq w_i, i = 1, \ldots, d\}$ where v and w are constant vectors, we let $J \cap b$ denote the data points represented by rows of J that also fall into b. It is possible to compute the summation of the squared distance of the points in $J \cap b$ from y grouped by the table T_j in time $O(md^2 n^{fhtw} \log(n))$.*

Proof. We can use a SumSum query grouped by table T_i. First, note that $J \cap b$ can be represented by a join query. Let $b(T_i)$ be all the rows in T_i that are located inside the projection of b onto the columns of T_i. Then, as b is an axis-parallel hyper-rectangle, we have $J \cap b = b(T_1) \bowtie \cdot \bowtie b(T_m)$. Therefore, any SumSum query on $J \cap b$ can be computed using Inside-Out algorithm on $b(T_1) \bowtie \cdot \bowtie b(T_m)$. Then the following SumSum query is the summation of the squared distance of the points in $J \cap b$ from y:

$$\sum_{p \in J \cap b} \|p - y\|_2^2 = \sum_{p \in J \cap b} \sum_{i=1}^{d} (p_i - y_i)^2.$$

Based on Theorem 19, we can compute this query in time $O(md^2 n^{\text{fhtw}} \log(n))$. □

Using the above axis-parallel boxes and the subroutine explained in Lemma 58, we can explain the sampling algorithm:

- The implementation uses a SumProd query to compute the aggregate 2-norm squared distance from c_1 constrained to points in b_3 (all the points) and grouped by table T_1 using Lemma 58. Let the resulting vector be C. Therefore, C_r is the aggregate 2-norm squared distance from c_1 of all rows in the design matrix that are extensions of row r in T_1.
- Then the implementation uses a SumProd query to compute the aggregated 2-norm squared distance from c_2, constrained to points in b_2, and grouped by T_1. Let the resulting vector be D. Notice that an axis-parallel box constraint can be expressed as a collection of axis-parallel hyperplane constraints, and for every axis-parallel constraint, it is easy to remove the points not satisfying it from the join by filtering one of the input tables having that dimension/feature. Then the sum product query is the same as the sum product query in the previous step.

- Then the implementation uses a SumProd query to compute the aggregated 2-norm squared distance from c_1, constrained to points in b_2, and grouped by T_1 Let the resulting vector be E.
- Then pick a row r of T_1 with probability proportional to $C_r - E_r + D_r$.
- The implementation then replaces T_1 by a table consisting only of the picked row r.
- The implementation then repeats this process on table T_2, then table T_3 etc.
- At the end, J will consist of one point/row x, where the probability that a particular point x ends up as this final row is $Q(x)$. To see this, note that in the iteration performed for T_i, $C - E$ is the aggregate 2-norm squared distances to c_1 for all points not in b_2 grouped by T_i, and D is the aggregated squared distances of the points in b_2 to c_2 grouped by T_i.

We now claim that this implementation guarantees that $c_3 = x$ with probability $P(x)$. We can see this using the standard rejection sampling calculation. At each iteration of sampling from Q, let $S(x)$ be the event that point x is sampled and $A(x)$ be the event that x is accepted. Then,

$$\mathbf{Pr}[S(x) \text{ and } A(x)] = \mathbf{Pr}[A(x)] \mid S(x)] \cdot \mathbf{Pr}[S(x)] = \frac{L(x)}{R(x)} Q(x) = \frac{L(x)}{Z}$$

Thus, x is accepted with probability proportional to $L(x)$, as desired.

As the number of times that the implementation has to sample from Q is geometrically distributed, the expected number of times that it will have to sample is the inverse of the probability of success, which is $\max_x \frac{R(x)}{L(x)}$. It is not too difficult to see (we prove it formally in Lemma 61) that $\max_x \frac{R(x)}{L(x)} = O(d)$. It takes $3m$ SumProd queries to sample from Q. Therefore, the expected running time of our implementation of 3-means++ is $O(md\Psi(n, d, m))$ where $\Psi(n, d, m) = md^2 n^{\text{fhtw}} \log(n)$ is the time required for computing a single SumProd query.

6.2.2 General Algorithm

6.2.2.1 Box Construction

Here we explain the algorithm for constructing a set of laminar boxes given the centers sampled previously. The construction is completely combinatorial. It only uses the given centers and we do not need any relational operation for the construction.

Algorithm Description: Assume we want to sample the i^{th} point in k-means++. The algorithm maintains two collections $\mathcal{G}_i$ and $\mathcal{B}_i$ of tuples. Each tuple consists of a box and a point in that box, called the **representative** of the box. This point is one of the previously sampled centers. One can think of the tuples in $\mathcal{G}_i$ as "active" ones that are subject to changes and those in $\mathcal{B}_i$ as "frozen" ones that are finalized, thus removed from $\mathcal{G}_i$ and added to $\mathcal{B}_i$. When the algorithm terminates, $\mathcal{G}_i$ will be empty, and the boxes in $\mathcal{B}_i$ will be a laminar collection of boxes that we use to define the "easy" probability distribution Q.

The initial tuples in $\mathcal{G}_i$ consist of one *unit hyper-cube* (side length is 1) centered at each previous center c_j, $j \in [i-1]$, with its representative point c_j. Up to scaling of initial unit hyper-cubes, we can assume that initially no pair of boxes in $\mathcal{G}_i$ intersect. This property of $\mathcal{G}_i$ is maintained throughout the process. Initially $\mathcal{B}_i$ is empty. Over time, the implementation keeps growing the boxes in $\mathcal{G}_i$ in size and moves tuples from $\mathcal{G}_i$ to $\mathcal{B}_i$.

The algorithm repeats the following steps in rounds. At the beginning of each round, there is no intersection between any two boxes in $\mathcal{G}_i$. The algorithm performs a doubling step where it **doubles** every box in $\mathcal{G}_i$. Doubling a box means each of its $d-1$ dimensional face is moved twice as far away from its representative. Mathematically, a box whose representative point is $y \in \mathcal{R}^d$ may be written as $\{x \in \mathcal{R}^d : y_i - v_i \leq x_i \leq y_i + w_i, i = 1, \ldots, d\}$ $(v_i, w_i > 0)$. This box becomes $\{x \in \mathcal{R}^d : y_i - 2v_i \leq x_i \leq y_i + 2w_i, i = 1, \ldots, d\}$ after doubling.

After doubling, the algorithm performs the following operations on intersecting boxes until there are none. The algorithm iteratively picks two arbitrary intersecting boxes from $\mathcal{G}_i$. Say the boxes are b_1 with representative y_1 and b_2 with representative y_2. The algorithm executes a **melding** step on (b_1, y_1) and (b_2, y_2), which has the following procedures:

- Compute the smallest box b_3 in the Euclidean space that contains both b_1 and b_2.

- Add (b_3, y_1) to $\mathcal{G}_i$ and delete (b_1, y_1) and (b_2, y_2) from $\mathcal{G}_i$.
- Check if b_1 (or b_2) is a box created by the doubling step at the beginning of the current round and has not been melded with other boxes ever since. If so, the algorithm computes a box b_1' (resp. b_2') from b_1 (resp. b_2) by **halving** it. That is, each $d-1$ dimensional face is moved so that its distance to the box's representative is halved. Mathematically, a box $\{x \in \mathcal{R}^d : y_i - v_i \leq x_i \leq y_i + w_i, i = 1, \ldots, d\}$ $(v_i, w_i > 0)$, where vector y is its representative, becomes $\{x \in \mathcal{R}^d : y_i - \frac{1}{2}v_i \leq x_i \leq y_i + \frac{1}{2}w_i, i = 1, \ldots, d\}$ after halving. Then (b_1', y_1) (or (b_2', y_2)) is added to $\mathcal{B}_i$. Otherwise do nothing.

Notice that melding decreases the size of $\mathcal{G}_i$.

The algorithm terminates when there is one tuple (b_0, y_0) left in $\mathcal{G}_i$, at which point the algorithm adds a box that contains the whole space with representative y_0 to $\mathcal{B}_i$. Note that during each round of doubling and melding, the boxes which are added to $\mathcal{B}_i$ are the ones that after doubling were melded with other boxes, and they are added with their shapes before the doubling step.A pseudocode of this algorithm can be found in the appendix.

Lemma 59. *The collection of boxes in $\mathcal{B}_i$ constructed by the above algorithm is laminar.*

Proof. Note that right before each doubling step, the boxes in $\mathcal{G}_i$ are disjoint and that is because the algorithm in the previous iteration melds all the boxes that have intersection with each other. We prove by induction that at all times, for every box b in $\mathcal{B}_i$ there exist a box b' in $\mathcal{G}_i$ such that $b \subseteq b'$. Since the boxes added to $\mathcal{B}_i$ in each iteration are a subset of the boxes in $\mathcal{G}_i$ before the doubling step and they do not intersect each other, laminarity of $\mathcal{B}_i$ is a straight-forward consequence.

Initially $\mathcal{B}_i$ is empty and therefore the claim holds. Assume in some arbitrary iteration ℓ this claim holds right before the doubling step, then after the doubling step since every box in $\mathcal{G}_i$ still covers all of the area it was covering before getting doubled, the claim holds. Furthermore, in the melding step, every box b_3 that is resulted from melding of two boxes b_1 and b_2 covers both b_1 and b_2; therefore, b_3 will cover b_1 and b_2 if they are added to $\mathcal{B}_i$, and if a box in $\mathcal{B}_i$ was covered by either of b_1 or b_2, it will be still covered by b_3. □

The collection of boxes in $\mathcal{B}_i$ can be thought of as a tree where every node corresponds to a box. The root node is the entire space. In this tree, for any box b', among all boxes

included by b', we pick the inclusion-wise *maximal* boxes and let them be the **children** of b'. Thus, the number of boxes in $\mathcal{B}_i$ is $O(i)$ since the tree has i leaves, one for each center.

6.2.2.2 Sampling

To define our easy distribution Q, for any point $x \in J$, let $b(x)$ be the minimal box in $\mathcal{B}_i$ that contains x and $y(x)$ be the representative of $b(x)$. Define $R(x) = \|x - y(x)\|_2^2$, and $Q(x) = \frac{R(X)}{Z}$ where $Z = \sum_{x \in J} R(x)$ normalizes the distribution. We call $R(x)$ the **assignment cost** for x. We will show how to sample from the target distribution $P(\cdot)$ using $Q(\cdot)$ and rejection sampling, and how to implement this designed sampling step relationally.

Rejection Sampling: The algorithm repeatedly samples a point x with probability $Q(x)$, then either (A) rejects x and resamples, or (B) accepts x as the next center c_i and finishes the sampling process. After sampling x, the probability of accepting x is $\frac{L(x)}{R(x)}$, and that of rejecting x is $1 - \frac{L(x)}{R(x)}$. Notice that here $\frac{L(x)}{R(x)} \leq 1$ since $R(x) = \|x - y(x)\|_2^2 \geq \min_{j \in [i-1]} \|x - c_j\|_2^2$.

If $S(x)$ is the the event of initially sampling x from distribution Q, and $A(x)$ is the event of subsequently accepting x, the probability of choosing x to be c_i in one given round is:

$$\Pr[S(x) \text{ and } A(x)] = \Pr[A(x) \mid S(x)] \Pr[S(x)] = \frac{L(x)}{R(x)} Q(x) = \frac{L(x)}{Z}$$

Thus, the probability of x being the accepted sample is proportional to $L(x)$, as desired.

We would like $Q(\cdot)$ to be close to $P(\cdot)$ point-wise so that the algorithm is efficient. Otherwise, the acceptance probability $\frac{L(x)}{R(x)}$ is low and it might keep rejecting samples.

Relational Implementation of Sampling: We now explain how to relationally sample a point x with probability $Q(x)$. The implementation heavily leverages Lemma 58, which states for a given box b^* with representative y^*, the cost of assigning all points in $r \bowtie J \cap b^*$ to y^* for each row $r \in T_i$ can be computed in polynomial time using a SumProd query grouped by T_i. Recall that we assign all points in J to the representative of the smallest box they belong to. We show that the total assignment cost is computed by evaluating SumProd queries on the boxes and then adding/subtracting the query values for different boxes.

Following the intuition provided in Section 6.1, the implementation generates a single row from table $T_1, T_2, \ldots, T_m$ sequentially. The concatenation of these rows (or the join of

them) gives the sampled point x. It is sufficient to explain assuming that we have sampled $r_1, \ldots, r_{\ell-1}$ from the first $\ell-1$ tables, how to implement the generation of a row from the next table T_ℓ. Just like 1- and 2-means++ in Section 6.1, the algorithm evaluates a function $F_\ell(\cdot)$ defined on rows in T_ℓ using SumProd queries, and samples r with probability $\frac{F_\ell(r)}{\sum_{r' \in T_\ell} F_\ell(r')}$. Again, we focus on $r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$, denoting the points in J that uses the previously sampled rows. The value of $F_\ell(r)$ is determined by points in $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$.

To ensure that we generate a row according to the correct distribution Q, we define the function $F_\ell(\cdot)$ as follows. Let $F_\ell(r)$ be the total assignment cost of all points in $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$. That is, $F_\ell(r) = \sum_{x \in r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J} R(x)$. Notice that the definition of the function $F_\ell(\cdot)$ is very similar to 2-means++ apart from that each point is no longer assigned to a given center, but the representative of the smallest box containing it.

Let $G(r, b^*, y^*)$ denote the cost of assigning all points from $r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$ that lies in box b^* to a center y^*. By replacing J in Lemma 58 by $r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$, we can compute all $G(r, b^*, y^*)$ values in polynomial time using one SumProd query grouped by T_ℓ. The value $F_\ell(r)$ can be expanded into subtraction and addition of $G(r, b^*, y^*)$ terms. The expansion is recursive. For a box b_0, let $H(r, b_0) = \sum_{x \in r \bowtie r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J \cap b_0} R(x)$. Notice that $F_\ell(r) = H(r, b_0)$ if b_0 is the entire Euclidean space. Pick any row $r \in T_\ell$. Assume we want to compute $H(r, b_0)$ for some tuple $(b_0, y_0) \in \mathcal{B}_i$.

Recall that the set of boxes in $\mathcal{B}_i$ forms a tree structure. If b_0 has no children this is the base case - $H(r, b_0) = G(r, b_0, y_0)$ by definition since all points in b_0 must be assigned to y_0. Otherwise, let $(b_1, y_1), \ldots, (b_q, y_q)$ be the tuples in $\mathcal{B}_i$ where $b_1, \ldots, b_q$ are children of b_0. Notice that, by definition, all points in $b_0 \setminus (\bigcup_{j \in [q]} b_j)$ are assigned to y_0. Then, one can check that the following equation holds for any r:

$$H(r, b_0) = G(r, b_0, y_0) - \sum_{j \in [q]} G(r, b_j, y_0) + \sum_{j \in [q]} H(r, b_j)$$

Starting with setting b_0 as the entire Euclidean space, the equation above could be used to recursively expand $H(\cdot, b_0) = F_\ell(\cdot)$ into addition and subtraction of $O(|\mathcal{B}_i|)$ number of $G(\cdot, \cdot, \cdot)$ terms, where each term could be computed with one SumProd query by Lemma 58.

Runtime Analysis of the Sampling: We now discuss the running time of the sampling algorithm simulating k-means++. These lemmas show how close the probability distribution we compute is compared to the k-means++ distribution. This will help bound the running time.

Lemma 60. *Consider the box construction algorithm when sampling the i^{th} point in the k-means++ simulation. Consider the end of the j^{th} round where all melding is finished but the boxes have not been doubled yet. Let b be an arbitrary box in $\mathcal{G}_i$ and $h(b)$ be the number of centers in b at this time. Let c_a be an arbitrary one of these $h(b)$ centers. Then:*

A. The distance from c_a to any $d-1$ dimensional face of b is at least 2^j.

B. The length of each side of b is at most $h(b) \cdot 2^{j+1}$.

Proof. The first statement is a direct consequence of the definition of doubling and melding since at any point of time the distance of all centers in a box is at least 2^j. To prove the second statement, we define the assignment of the centers to the boxes as following. Consider the centers inside each box b right before the doubling step. We call these centers the centers assigned to b and denote the number of them by $h'(b)$. When two boxes b_1 and b_2 are melding into box b_3, we assign their assigned centers to b_3.

We prove that each side length of b is at most $h'(b)2^{j+1}$ by induction on the number j of executed doubling steps. Since $h'(b) = h(b)$ right before each doubling, this will prove the second statement. The statement is obvious in the base case, $j = 0$. The statement also clearly holds by induction after a doubling step as j is incremented and the side lengths double and the number of assigned boxes do not change. It also holds during every meld step because each side length of the newly created larger box is at most the aggregate maximum side length of the smaller boxes that are moved to $\mathcal{B}_i$, and the number of assigned centers in the newly created larger box is the aggregate of the assigned centers in the two smaller boxes that are moved to $\mathcal{B}_i$. Note that since for any box b all the assigned centers to b are inside b at all times, $h'(b)$ is the number of centers inside b before the next doubling. □

This lemma bounds the difference of the two probability distributions.

Lemma 61. *Consider the box generation algorithm when sampling the ith point in the k-means++ simulation. For all points x, $R(x) \leq O(i^2 d) \cdot L(x)$.*

Proof. Consider an arbitrary point x. Let c_ℓ, $\ell \in [i-1]$, be the center that is closest to x under the 2-norm distance. Assume j is minimal such that just before the $(j+1)$-th doubling round, x is contained in a box b in $\mathcal{G}_i$. We argue about the state of the algorithm at two times, the time s just before doubling round j and the time t just before doubling round $j+1$. Let b be a minimal box in $\mathcal{G}_i$ that contains x at time t, and let y be the representative for box b. Notice that we assign x to the representative of the smallest box in $\mathcal{B}_i$ that contains it, so x will be assigned to y. Indeed, none of the boxes added into $\mathcal{B}_i$ before time t contains x by the minimality of j, and when box b gets added into $\mathcal{B}_i$ (potentially after a few more doubling rounds) it still has the same representative y. By Lemma 60 the squared distance from from x to r is at most $(i-1)^2 d 2^{2j+2}$. Therefore, it is sufficient to show that the squared distance from x to c_ℓ is $\Omega(2^j)$.

Let b' be the box in $\mathcal{G}_i$ that contains c_ℓ at time s. Note that x could not have been inside b' at time s by the definition of t and s. Then by Lemma 60 the distance from c_ℓ to the edge of b' at time t is at least 2^{2j-2}, and hence the distance from c_ℓ to x is also at least 2^{2j-2} as x is outside of b'. □

The following theorem bounds the running time.

Theorem 62. *The expected time complexity for running k' iterations of this implementation of k-means++ is $O(k'^4 dm\Psi(n,d,m))$ where $\Psi(n,d,m) = md^2 n^{fhtw} \log(n)$.*

Proof. When picking center c_i, a point x can be sampled with probability $Q(x)$ in time $O(mi\Psi(n,m,d))$. This is because the implementation samples one row from each of the m tables. To sample one row, we evaluate $O(|\mathcal{B}_i|)$ SumProd queries, each in $O(\Psi(n,m,d))$ time. As mentioned earlier, $\mathcal{B}_i$ can be thought of as a tree of boxes with $i-1$ leaves, so $|\mathcal{B}_i| = O(i)$.

By Lemma 61, the probability of accepting any sampled x is $\frac{L(x)}{R(x)} = \frac{1}{O(i^2 d)}$. The expected number of sampling from Q until getting accepted is $O(i^2 d)$. Thus, the expected time of finding c_i is $O(i^3 dm\Psi(n,m,d))$. Summing over $i \in [k']$, we get $O(k'^4 dm\Psi(n,m,d))$. □

6.3 Weighting the Centers

Our algorithm samples a collection C of $k' = \Theta(k \log N)$ centers using the k-means++ sampling described in the prior section. We give weights to the centers to get a coreset.

Ideally, we would compute the weights in the standard way. That is, let w_i denote the number of points that are closest to point c_i among all centers in C. These pairs of centers and weights (c_i, w_i) are known to form a coreset. Unfortunately, as stated in Theorem 56, computing such w_i's even approximately is NP hard. Instead, we will find a different set of weights which still form a coreset and are computable.

Next, we describe a relational algorithm to compute a collection W' of weights, one weight $w'_i \in W'$ for each center $c_i \in C$. The proof that the centers with these alternative weights (c_i, w'_i) also form a coreset is postponed until the appendix.

Algorithm for Computing Alternative Weights: Initialize the weight w'_i for each center $c_i \in C$ to zero. In the d-dimensional Euclidean space, for each center $c_i \in C$, we generate a collection of hyperspheres (also named **balls**) $\{B_{i,j}\}_{j \in [\lg N]}$, where $B_{i,j}$ contains approximately 2^j points from J. The space is then partitioned into $\{B_{i,0}, B_{i,1} - B_{i,0}, B_{i,2} - B_{i,1}, \ldots\}$. For each partition, we will sample a small number of points and use this sample to estimate the number of points in this partition that are closer to c_i than any other centers, and thus aggregating w'_i by adding up the numbers. Fix small constants $\epsilon, \delta > 0$. The following steps are repeated for $j \in [\lg N]$:

- Let $B_{i,j}$ be a ball of radius $r_{i,j}$ centered at c_i. Find a $r_{i,j}$ such that the number of points in $J \cap B_{i,j}$ lies in the range $[(1-\delta)2^j, (1+\delta)2^j]$. This step can be done using Theorem 28.
- Let τ be a constant that is at least 30. A collection $T_{i,j}$ of $\frac{\tau}{\epsilon^2} k'^2 \log^2 N$ "test" points are independently sampled following the same **approximately uniform** distribution with replacement from every ball $B_{i,j}$. Here an "approximately uniform" distribution means one where every point p in $B_{i,j}$ is sampled with a probability $\gamma_{p,i,j} \in [(1-\delta)/|B_{i,j}|, (1+\delta)/|B_{i,j}|]$ on each draw. This can be accomplished efficiently similar to the techniques used in Theorem 28. Further elaboration is given in the Section 6.3.1.
- Among all sampled points $T_{i,j}$, find $S_{i,j}$, the set of points that lie in the **"donut"** $D_{i,j} =$

$B_{i,j} - B_{i,j-1}$. Then the cardinality $s_{i,j} = |S_{i,j}|$ is computed.

- Find $t_{i,j}$, the number of points in $S_{i,j}$ that are closer to c_i than any other center in C.
- Compute the ratio $f'_{i,j} = \frac{t_{i,j}}{s_{i,j}}$ (if $s_{i,j} = t_{i,j} = 0$ then $f'_{i,j} = 0$).
- If $f'_{i,j} \geq \frac{1}{2k'^2 \log N}$ then w'_i is incremented by $f'_{i,j} \cdot 2^{j-1}$, else w'_i stays the same.

At first glance, the algorithm appears naive: w'_i can be significantly underestimated if in some donuts only a small portion of points are closest to c_i, making the estimation inaccurate based on sampling. However, in Section 6.4, we prove the following theorem which shows that the alternative weights computed by our algorithm actually form a coreset.

Theorem 63. *The set of centers C, along with the computed weights W', form an $O(1)$-approximate coreset with high probability.*

The running time of a naive implementation of this algorithm would be dominated by the sampling of the test points. Sampling a single test point can be accomplished with m applications of the algorithm in Section 3.2 and setting the approximation error to $\delta = \epsilon/m$. Recall the running time of the algorithm from Section 3.2 is $O\left(\frac{m^6 \log^4 n}{\delta^2} \Psi(n, d, m)\right)$ where $\Psi(n, d, m) = md^2 n^{\text{fhtw}} \log(n)$. Thus, the time to sample all test points is $O\left(\frac{k'^2 m^9 \log^6 n}{\epsilon^4} \Psi(n, d, m)\right)$. Substituting for k', and noting that $N \leq n^m$, we obtain a total time for a naive implementation of $O\left(\frac{k^2 m^{11} \log^8 n}{\epsilon^4} \Psi(n, d, m)\right)$.

6.3.1 Uniform Sampling From a Hypersphere

To uniformly sample a point from inside a ball, it is enough to show how we can count the number of points located inside a ball grouped by a table T_i. Because, if we can count the number of points grouped by the input tables, then we can use a similar technique to the one used in Section 6.2 to sample. Unfortunately, similar to the proof of Theorem 23, we can show that it is $\#P$-Hard to count the number of points inside a ball; however, it is possible to obtain a $1 \pm \delta$ approximation of the number of points as we showed in Theorem 28. Below we briefly explain how to use the algorithm in Section 3.2 for counting the number of points inside a hypersphere.

Given a center c and a radius R, the goal is approximating the number of tuples $x \in J$ for which $\sum_i (c^i - x^i)^2 \leq R$. Note that $\sum_i (c^i - x^i)^2 \leq R$ is an additive inequality and therefore

the problem we are considering is an inequality row counting instance. Recall that the algorithm in Section 3.2 uses some real number multisets and it defines two operators of $\oplus$ and $\otimes$ that form a semiring with multisets. Then it runs Inside-Out [9] algorithm combined with some sketching techniques to reduce the size of the partial results. At the end, the algorithm in Section 3.2 returns an array where in j-th entry it has the smallest value r for which there are $(1+\delta)^j$ tuples $x \in J$ satisfying the additive inequality $\|x-c\|_2^2 \leq r^2$.

The query can also be executed grouped by one of the input tables since Inside-Out itself can execute SumProd queries grouped by one of the input tables. Therefore, using this polynomial approximation scheme, we can calculate the conditioned marginalized probability distribution with multiplicative $(1 \pm \delta)$. Therefore, using m queries, it is possible to sample a tuple from a ball with probability distribution $\frac{1}{n}(1 \pm m\delta)$ where n is the number of points inside the ball. In order to get a sample with probability $\frac{1}{n}(1 \pm \epsilon)$, all we need is to set $\delta = \epsilon/m$; hence, the time complexity for sampling each tuple will be $O\Big(\frac{m^9 \log^4(n)}{\epsilon^2}\Psi(n,d,m)\Big)$

6.4 Analysis of the Weighting Algorithm

The goal of this subsection is to prove Theorem 63 which states that the alternative weights form an $O(1)$-approximate coreset with high probability. Throughout our analysis, "with high probability" means that for any constant $\rho > 0$ the probability of the statement not being true can be made less than $\frac{1}{N^\rho}$ asymptotically by appropriately setting the constants in the algorithm.

Intuitively, if a decent fraction of the points in each donut are closer to center c_i than any other center, then Theorem 63 can be proven by using a straight-forward application of Chernoff bounds to show that each alternate weight w_i' is likely close to the true weight w_i. The conceptual difficultly is if only a very small portion of points in a donut $D_{i,j}$ are closer to c_i than any other points, in which case the estimated $f_{i,j}' < \frac{1}{2k'^2 \log N}$ and thus the "uncounted" points in $D_{i,j}$ would contribute no weight to the computed weight w_i'. We call this the **undersampled** case. If many docuts around a center i are undersampled, the computed weight w_i' may well poorly approximate the actual weight w_i.

To address this, we need to prove that omitting the weight from these uncounted points does not have a significant impact on the objective value. We break our proof into four parts. The first part, described in Section 6.4.1, involves conceptually defining a fractional weight w_i^f for each center $c_i \in C$. Each point has a weight of 1, and instead of giving all this weight to its closest center, we allow fractionally assigning the weight to various "near" centers. w_i^f is then the aggregated weight over all points for c_i. The second part, described in Section 6.4.2, establishes various properties of the fractional weight that we will need. The third part, described in subsubsection 6.4.3, shows that each fractional weight w_i^f is likely to be closely approximated the computed weight w_i'. The fourth part, described in Section 6.4.4, shows that the fractional weights of the centers in C form a $O(1)$-approximate coreset. Section 6.4.4 also contains the proof of Theorem 63.

6.4.1 Defining the Fractional Weights

To define the fractional weights we first define an auxiliary directed acyclic graph $G = (S, E)$ where there is one node in S corresponding to each row in J. For the rest of this section, with a little abuse of notation, we use S to denote both the nodes in graph G, and the set of d-dimensional data points in the design matrix. Let p be an arbitrary point in $S - C$. Let $\alpha(p)$ denote the subscript of the center closest to p, i.e., if $c_i \in C$ is closest to p then $\alpha(p) = i$. Let $D_{i,j}$ be the donut around c_i that contains p. If $D_{i,j}$ is not undersampled then p will have one outgoing edge (p, c_i). Hence, let us now assume that $D_{i,j}$ is undersampled. Defining the outgoing edges from p in this case is a bit more complicated.

Let $A_{i,j}$ be the points $q \in D_{i,j}$ that are closer to c_i than any other center in C (i.e., $\alpha(q) = i$). If $j = 1$ then $D_{i,1}$ contains only the point p, and the only outgoing edge from p goes to c_i. As a result, let us assume the remaining case of $j > 1$. Let c_h the center that is closest to the most points in $D_{i,j-1}$, the next donut in toward c_i from $D_{i,j}$. That is $c_h = \text{argmax}_{c_j \in C} \sum_{q \in D_{i,j-1}} \mathbb{1}_{\alpha(q)=c_j}$. Let $M_{i,j-1}$ be points in $D_{i,j-1}$ that are closer to c_h than any other center. That is, $M_{i,j-1}$ is the collection of $q \in D_{i,j-1}$ such that $\alpha(q) = h$. Then there is a directed edge from p to each point in $M_{i,j-1}$.

Before defining how to derive the fractional weights from G, let us take a detour to note that G is acyclic. The proof of the following lemma can be found in the appendix.

Lemma 64. *G is acyclic.*

Proof. Consider a directed edge $(p, q) \in E$, and c_i be the center in C that p is closest to, and $D_{i,j}$ the donut around c_i that contains p. Then, since $p \in D_{i,j}$ it must be the case that $\|p - c_i\|_2^2 > r_{i,j-1}$. Since $q \in B_{i,j-1}$ it must be the case that $\|q - c_i\|_2^2 \leq r_{i,j-1}$. Thus, $\|p - c_i\|_2^2 > \|q - c_i\|_2^2$. Therefore, the closest center to q must be closer to q than the closest center to p is to p. For this reason, as one travels along a directed path in G, although identify of the closest center can change, the distance to the closest center must be monotonically decreasing. Thus, G must be acyclic. □

We explain how to compute a fractional weight w_p^f for each point $p \in S$ using the network G. Initially, each w_p^f is set to 1. Then conceptually these weights flow toward the sinks in G, splitting evenly over all outgoing edges at each vertex. More formally, the following flow step is repeated until is no longer possible to do so:

Flow Step: Let $p \in S$ be an arbitrary point that currently has positive fractional weight and that has positive outdegree h in G. Then for each directed edge (p, q) in G increment w_q^f by w_p^f/h. Finally, set w_p^f to zero.

As the sinks in G are exactly the centers in C, the centers in C will be the only points that end up with positive fractional weight. Thus, we use w_i^f to refer to the resulting fractional weight on center $c_i \in C$.

6.4.2 Properties of the Fractional Weights

Let $f_{i,j}$ be the fraction of points that are closest to c_i among all centers in C in this donut $D_{i,j} = B_{i,j} - B_{i,j-1}$. We show in Lemma 65 and Lemma 67 that with high probability, either the estimated ratio is a good approximation of $f_{i,j}$, or the real ratio $f_{i,j}$ is very small.

We show in Lemma 69 that the maximum flow through any node is bounded by $1+\epsilon$ when N is big enough. This follows by induction because each point has $\Omega(k' \log N)$ neighbors

and every point can have in degree from one set of nodes per center. We further know every point that is not uncounted actually contributes to their centers' weight.

Lemma 65. *With high probability, either* $|f_{i,j} - f'_{i,j}| \leq \epsilon f_{i,j}$ *or* $f'_{i,j} \leq \frac{1}{2k'^2 \log N}$.

To prove Lemma 65, we use the following Chernoff Bound.

Lemma 66. *Consider Bernoulli trials* $X_i, \ldots, X_n$. *Let* $X = \sum_{i=1}^{n} X_i$ *and* $\mu = E[X]$. *Then, for any* $\lambda > 0$*:*

$$\mathbf{Pr}[X \geq \mu + \lambda] \leq \exp\left(-\frac{\lambda^2}{2\mu + \lambda}\right) \qquad \text{Upper Chernoff Bound}$$

$$\mathbf{Pr}[X \leq \mu - \lambda] \leq \exp\left(-\frac{\lambda^2}{3\mu}\right) \qquad \text{Lower Chernoff Bound}$$

Proof. Proof of Lemma 65: Fix any center $c_i \in C$ and $j \in [\log N]$. By applying the low Chernoff bound from Lemma 66 it is straight forward to conclude that τ is large then with high probability at least a third of the test points in each $T_{i,j}$ are in the donut $D_{i,j}$. That is, with high probability $s_{i,j} \geq \frac{\tau}{3\epsilon^2} k'^2 \log^2 N$. Therefore, let us consider a particular $T_{i,j}$ and condition $s_{i,j}$ having some fixed value that is at least $\frac{1}{3\epsilon^2} k'^2 \log^2 N$. As a result, $s_{i,j}$ is conditioned on being large.

Recall $t_{i,j} = \sum_{p \in W_{i,j}} (\mathbb{1}_{p \in T_{i,j}})(\mathbb{1}_{\alpha(p)=i})$, and the indicator random variables $\mathbb{1}_{p \in T_{i,j}}$ are Bernoulli trials. Further note by the definition of $\gamma_{p,i,j}$ it is the case that $E[t_{i,j}] = \sum_{p \in W_{i,j}} \gamma_{p,i,j} (\mathbb{1}_{\alpha(p)=i})$. Further note that as the sampling of test points is nearly uniform that $f_{i,j}(1-\delta)s_{i,j} \leq E[t_{i,j}] \leq f_{i,j}(1+\delta)s_{i,j}$. For notational convenience, let $\mu = E[t_{i,j}]$. We now break the proof into three cases, that cover the ways in which the statement of this lemma would not be true. For each case, we show with high probability the case does not occur.

Case 1: $f'_{i,j} \geq \frac{1}{2k'^2 \log N}$ **and** $f_{i,j} > \frac{1-\epsilon}{2k'^2 \log N}$ **and** $f'_{i,j} \geq (1+\epsilon) f_{i,j}$**.** We are going to prove that the probability of this case happening is very low. If we set $\lambda = \epsilon\mu$, then using Chernoff bound, we have

$$
\begin{aligned}
\mathbf{Pr}[t_{i,j} \geq (1+\epsilon)\mu] &\leq \exp\left(-\frac{(\epsilon\mu)^2}{2\mu+\epsilon\mu}\right) && \text{[Upper Chernoff Bound]} \\
&\leq \exp\left(-\frac{\epsilon^2(1-\delta)f_{i,j}s_{i,j}}{2+\epsilon}\right) && [\mu \geq (1-\delta)f_{i,j}s_{i,j}] \\
&\leq \exp\left(-\frac{\epsilon^2(1-\delta)(1-\epsilon)s_{i,j}}{3(2k'^2\log N)}\right) && [f_{i,j} > \frac{1-\epsilon}{2k'^2\log N}] \\
&\leq \exp\left(-\frac{\epsilon^2(1-\delta)(1-\epsilon)\tau k'^2\log^2 N}{3(2k'^2\log N)(3\epsilon^2)}\right) && [s_{i,j} \geq \frac{\tau}{3\epsilon^2}k'^2\log N] \\
&= \exp\left(-\frac{(1-\delta)(1-\epsilon)\tau\log N}{18}\right)
\end{aligned}
$$

Therefore, for $\delta \leq \epsilon/2 \leq 1/10$ and $\tau \geq 30$ this case cannot happen with high probability.

Case 2: $f'_{i,j} \geq \frac{1}{2k'^2\log N}$ and $f_{i,j} > \frac{1-\epsilon}{2k'^2\log N}$ and $f'_{i,j} < (1-\epsilon)f_{i,j}$. We can use Lower Chernoff Bound with $\lambda = \epsilon\mu$ to prove the probability of this event is very small.

$$
\begin{aligned}
\mathbf{Pr}[t_{i,j} \leq (1-\epsilon)\mu] &\leq \exp\left(-\frac{(\epsilon\mu)^2}{3\mu}\right) \\
&\leq \exp\left(-\frac{\epsilon^2(1-\delta)f_{i,j}s_{i,j}}{3}\right) && [\mu \geq (1-\delta)f_{i,j}s_{i,j}] \\
&\leq \exp\left(-\frac{\epsilon^2(1-\delta)(1-\epsilon)s_{i,j}}{3(2k'^2\log N)}\right) && [f_{i,j} > \frac{1-\epsilon}{2k'^2\log N}] \\
&\leq \exp\left(-\frac{\epsilon^2(1-\delta)(1-\epsilon)\tau k'^2\log^2 N}{3(2k'^2\log N)(3\epsilon^2)}\right) && [s_{i,j} \geq \frac{\tau}{3\epsilon^2}k'^2\log N] \\
&= \exp\left(-\frac{(1-\delta)(1-\epsilon)\tau\log N}{18}\right)
\end{aligned}
$$

Therefore, for $\delta \leq \epsilon/2 \leq 1/10$ and $\tau \geq 30$ this case cannot happen with high probability.

Case 3: $\boldsymbol{f'_{i,j} \geq \frac{1}{2k'^2\log N}}$ and $\boldsymbol{f_{i,j} \leq \frac{1-\epsilon}{2k'^2\log N}}$: Since $f'_{i,j} = \frac{t_{i,j}}{s_{i,j}}$, in this case:

$$t_{i,j} \geq \frac{s_{i,}}{2k'^2\log N} \tag{34}$$

Since $\mu \leq f_{i,j}(1+\delta)s_{i,j}$, in this case:

$$\mu \leq \frac{1-\epsilon}{2k'^2\log N}(1+\delta)s_{i,j} \tag{35}$$

Thus, subtracting line 34 from line 35 we conclude that:

$$t_{i,j} \geq \mu + \frac{(\epsilon-\delta+\epsilon\delta)s_{i,j}}{2k'^2\log N} \tag{36}$$

Let $\lambda = \frac{(\epsilon-\delta+\epsilon\delta)s_{i,j}}{2k'^2 \log N}$. We can conclude that

$$\begin{aligned}
\mathbf{Pr}[t_{i,j} \geq \mu + \lambda] &\leq \exp\left(-\frac{\lambda^2}{2\mu+\lambda}\right) && \text{Upper Chernoff Bound} \\
&\leq \exp\left(\frac{-\lambda^2}{\frac{1-\epsilon}{2k'^2 \log N}(1+\delta)s_{i,j} + \lambda}\right) && \text{Using line 35} \\
&= \exp\left(\frac{-\left(\frac{(\epsilon-\delta+\epsilon\delta)s_{i,j}}{2k'^2 \log N}\right)^2}{\frac{1-\epsilon}{2k'^2 \log N}(1+\delta)s_{i,j} + \frac{(\epsilon-\delta+\epsilon\delta)s_{i,j}}{2k'^2 \log N}}\right) \\
&= \exp\left(\frac{-\left(\frac{(\epsilon-\delta+\epsilon\delta)^2 s_{i,j}}{k'^2 \log N}\right)}{2(1-\epsilon)(1+\delta) + 2(\epsilon-\delta+\epsilon\delta)}\right) \\
&\leq \exp\left(\frac{-(\epsilon-\delta+\epsilon\delta)^2 s_{i,j}}{12k'^2 \log N}\right) \\
&= \exp\left(\frac{-(\epsilon-\delta+\epsilon\delta)^2 \tau \log N}{12\epsilon^2}\right) && \text{Substituting our lower bound on } s_{i,j}
\end{aligned}$$

Therefore, for $\delta \leq \epsilon/2 \leq 1/10$ and $\tau \geq 30$ this case cannot happen with high probability. □

The next case proves how large $f'_{i,j}$ is when we know that $f_{i,j}$ is large.

Lemma 67. *If $f_{i,j} > \frac{1+\epsilon}{2k'^2 \log N}$ then with high probability $f'_{i,j} \geq \frac{1}{2k'^2 \log N}$.*

Proof. We can prove that the probability of $f'_{i,j} < \frac{1}{2k'^2 \log N}$ and $f_{i,j} \geq \frac{1+\epsilon}{2k'^2 \log N}$ is small. Multiplying the conditions for this case by $s_{i,j}$ we can conclude that $t_{ij} < \frac{s_{i,j}}{2k'^2 \log N}$ and $\mu \geq (1-\delta)\frac{(1+\epsilon)s_{i,j}}{2k'^2 \log N}$. As a consequence, $t_{i,j} \leq \mu - \lambda$ where $\lambda = \frac{(\epsilon-\delta-\epsilon\delta)s_{i,j}}{2k'^2 \log N}$. Then we can conclude that:

$$\begin{aligned}
\mathbf{Pr}[t_{i,j} \leq \mu - \lambda] &\leq \exp\left(-\frac{\lambda^2}{3\mu}\right) && \text{[Lower Chernoff Bound]} \\
&= \exp\left(-\frac{\left(\frac{(\epsilon-\delta-\epsilon\delta)s_{i,j}}{2k'^2 \log N}\right)^2}{3\mu}\right) \\
&\leq \exp\left(-\frac{\left(\frac{(\epsilon-\delta-\epsilon\delta)s_{i,j}}{2k'^2 \log N}\right)^2}{3\frac{1-\epsilon}{2k'^2 \log N}(1+\delta)s_{i,j}}\right)
\end{aligned}$$

$$
\begin{aligned}
&= \exp\left(-\frac{\left(\frac{(\epsilon-\delta-\epsilon\delta)^2 s_{i,j}}{2k'^2 \log N}\right)}{3(1-\epsilon)(1+\delta)}\right) \\
&\leq \exp\left(\frac{-(\epsilon-\delta-\epsilon\delta)^2 s_{i,j}}{12k'^2 \log N}\right) && [\delta < \epsilon \leq 1] \\
&\leq \exp\left(\frac{-(\epsilon-\delta-\epsilon\delta)^2 (\frac{\tau}{3\epsilon^2} k'^2 \log^2 N)}{12k'^2 \log N}\right) && [\text{Using our lower bound on } s_{i,j}]
\end{aligned}
$$

Therefore, for $\delta \leq \epsilon/2 \leq 1/10$ and $\tau \geq 30$ this case cannot happen with high probability. □

We now seek to bound the fractional weights computed by the algorithm. Let $\Delta_i(p)$ denote the total weight received by a point $p \in S \setminus C$ from other nodes (including the initial weight one on p). Furthermore, let $\Delta_o(p)$ denote the total weight sent by p to all other nodes. Notice that in the flow step, $\Delta_o(p) = \Delta_i(p)$ for all p in $S \setminus C$.

Lemma 68. *Let $\Delta_i(p)$ denote the total weight received by a point $p \in S \setminus C$ from other nodes (including the initial weight one on p). Furthermore, let $\Delta_o(p)$ denote the total weight sent by p to all other nodes. With high probability, for all $q \in S$, $\Delta_i(q) \leq 1 + \frac{1+2\epsilon}{\log N} \max_{p:(p,q)\in E} \Delta_o(p)$.*

Proof. Fix the point q that redirects its weight (has outgoing arcs in G). Consider its direct predecessor $P(q) = \{p : (p,q) \in E\}$. Partition $P(q)$ as follows: $P(q) = \bigcup_{i=1,\ldots,k'} P_{c_i}(q)$, where $P_{c_i}(q)$ is the set of points that have flowed their weights into q, but c_i is actually their closest center in C. Observe the following. The point q can only belong to one donut around c_i. Due to this, $P_{c_i}(q)$ is either empty or contains a set of points in a single donut around c_i that redirect weight to q.

Fix $P_{c_i}(q)$ for some c_i. If this set is nonempty, suppose this set is in the j-th donut around c_i. Conditioned on the events stated in Lemmas 65 and 67, since the points in $P_{c_i}(q)$ are undersampled, we have $|P_{c_i}(q)| \leq \frac{(1+\epsilon)2^{j-1}}{2k'^2 \log N}$. Consider any $p \in P_{c_i}(q)$. Let β_i be the number of points that p charges its weight to (this is the same for all such points p). It is the case that β_i is at least $\frac{(1-\delta)2^{j-1}}{2k'}$ since p flows its weights to the points that are assigned to the center that has the most number of points assigned to it from c_i's $(j-1)$th donut.

Thus, q receives weight from $|P_{c_i}(q)| \leq \frac{(1+\epsilon)2^{j-1}}{2k'^2 \log N}$ points and each such point gives its weight to at least $\frac{(1-\delta)2^{j-1}}{2k'}$ points with equal split. The total weight that q receives from points in $P_{c_i}(q)$ is at most the following.

$$\begin{aligned}
&\frac{2k'}{(1-\delta)2^{j-1}} \sum_{p \in P_{c_i}(q)} \Delta_o(p) \\
&\leq \frac{2k'}{(1-\delta)2^{j-1}} \sum_{p \in P_{c_i}(q)} \max_{p \in P_{c_i}(q)} \Delta_o(p) \\
&\leq \frac{2k'}{(1-\delta)2^{j-1}} \cdot \frac{(1+\epsilon) \cdot 2^{j-1}}{2k'^2 \log N} \max_{p \in P_{c_i}(q)} \Delta_o(p) && [|P_{c_i}(q)| \leq \tfrac{(1+2\epsilon)2^{j-1}}{2k'^2 \log N}] \\
&\leq \frac{1+2\epsilon}{k' \log N} \max_{p \in P_{c_i}(q)} \Delta_o(p) && [\delta \leq \tfrac{\epsilon}{2} \leq \tfrac{1}{10}]
\end{aligned}$$

Switching the max to $\max_{p:(p,q)\in E} \Delta_o(p)$, summing over all centers $c_i \in C$ and adding the original unit weight on q gives the lemma.

□

The following crucial lemma bounds the maximum weight that a point can receive.

Lemma 69. *Fix η to be a constant smaller than $\frac{\log(N)}{10}$ and $\epsilon < 1$. Say that for all $q \in S \setminus C$ it is the case that $\Delta_o(q) = \eta \Delta_i(q)$. Then, with high probability for any $p \in S \setminus C$ it is the case that $\Delta_i(p) \leq 1 + \frac{2\eta}{\log N}$.*

Proof. We can easily prove this by induction on nodes. The lemma is true for all nodes that have no incoming edges in G. Now assume it is true for all nodes for which the longest path that reaches them in G has length $t-1$. Now we prove it for nodes for which the length of the longest path that reaches them in G is t. Fix such a node q. For any node p such that $(p,q) \in E$, by induction we have $\Delta_i(p) \leq 1 + \frac{2\eta}{\log N}$, so $\Delta_o(p) \leq 2(1 + \frac{2\eta}{\log N})$. By Lemma 68, $\Delta_i(q) \leq 1 + \frac{1+2\epsilon}{\log N} \max_{p:(p,q)\in E} \Delta_o(p) \leq 1 + \left(\frac{\eta(1+2\epsilon)}{\log N}\right)\left(1 + \frac{2\eta}{\log N}\right) = 1 + \frac{\eta}{\log N} + \frac{\eta}{\log N} \cdot \frac{2(1+2\epsilon)\eta + 2\epsilon}{\log N} \leq 1 + \frac{2\eta}{\log N}$. □

6.4.3 Comparing Alternative Weights to Fractional Weights

It only remains to bound the cost of mapping points to the centers they contribute weight to. This can be done by iteratively charging the total cost of reassigning each node to the flow. In particular, each point will only pass its weight to nodes that are closer to their center. We can charge the flow through each node to the assignment cost of that node to its closest center, and argue that the cumulative reassignment cost bounds the real fractional assignment cost. Furthermore, each node only has $1+\epsilon$ flow going through it. This will be sufficient to bound the overall cost in Lemma 71.

Lemma 70. *With high probability, for every center c_i, it is the case that the estimated weight w'_i computed by the weighting algorithm is $(1 \pm 2\epsilon)w_i^f$ where w_i^f is the fractional weight of i.*

Proof. Apply the union bound to Lemma 65 and 67 over all i and j.

Fix a center c_i. Consider all points that are closest to c_i and are not undersampled. Let w_i^s denote the number of these points. All incoming edges to c_i in G, are coming from these points; therefore based on Lemma 69, $w_i^s \leq w_i^f \leq w_i^s(1+\frac{2}{\log(N)})$. On the other hand, w'_i is $(1 \pm \epsilon)$ approximation of w_i^s. Therefore, $\frac{1-\epsilon}{1+\frac{2}{\log(N)}} w_i^f \leq w'_i \leq (1+\epsilon)w_i^f$. Assuming that $\log N$ is sufficiently larger than ϵ, the lemma follows. □

6.4.4 Comparing Fractional Weights to Optimal

Next, we bound the total cost of the fractional assignment defined by the flow. According to the graph G, any point $p \in S$ and $c_i \in C$, we let $\omega(p, c_i)$ be the fraction of weights that got transferred from p to c_i. Naturally, we have $\sum_{c_i \in C} \omega(p, c_i) = 1$ for any $p \in S$ and the fractional weights $w_i^f = \sum_{p \in S} \omega(p, c_i)$ for any $c_i \in C$.

Lemma 71. *Let ϕ_{opt} be the optimal k-means cost on the original set S. With high probability, it is the case that:*

$$\sum_{p \in S} \sum_{c_i \in C} \omega(p, c_i) \|p - c_i\|^2 \leq 160(1+\epsilon)\phi_{opt}$$

Proof. Let $\phi^* = \sum_{p \in S} \|p - c_{\alpha(p)}\|^2$. Consider any $p \in S$ and center c_i such that $\omega(p, c_i) > 0$. Let P be any path from p to c_i in G. If node p's only outgoing arc is to its closest center

$c_{\alpha(p)} = c_i$, then $P = p \to c_i$, we have $\sum_{c \in C} \omega(p,c)\|p-c\|^2 = \|p - c_{\alpha(p)}\|^2$. Otherwise assume $P = p \to q_1 \to q_2 \to \ldots \to q_\ell \to c_i$. Note that the closest center to q_ℓ is c_i. Let $\Delta(P)$ be the fraction of the original weight of 1 on p that is given to c_i along this path according to the flow of weights. As we observed in the proof of Lemma 64, we have $\|p - c_{\alpha(p)}\| > \|q_1 - c_{\alpha(p)}\| \geq \|q_1 - c_{\alpha(q_1)}\| > \|q_2 - c_{\alpha(q_1)}\| \geq \|q_2 - c_{\alpha(q_2)}\| > \ldots > \|q_\ell - c_{\alpha(q_\ell)}\|$. This follows because for any arc (u,v) in the graph, v is in a donut closer to $c_{\alpha(u)}$ than the donut u is in, and v is closer to $c_{\alpha(v)}$ than $c_{\alpha(u)}$.

We use the relaxed triangle inequality for squared ℓ_2 norms. For any three points x, y, z, we have $\|x-z\|^2 \leq 2(\|x-y\|^2 + \|y-z\|^2)$. Thus, we bound $\|p - c_i\|^2$ by

$$\begin{aligned}
\|p - c_i\|^2 &= \|p - c_{\alpha(p)} + c_{\alpha(p)} - q_1 + q_1 - c_i\|^2 \\
&\leq 2\|p - c_{\alpha(p)} + c_{\alpha(p)} - q_1\|^2 + 2\|q_1 - c_i\|^2 && \text{[relaxed triangle inequality]} \\
&\leq 2(\|p - c_{\alpha(p)}\| + \|c_{\alpha(p)} - q_1\|)^2 + 2\|q_1 - c_i\|^2 && \text{[triangle inequality]} \\
&\leq 8\|p - c_{\alpha(p)}\|^2 + 2\|q_1 - c_i\|^2 && [\|p - c_{\alpha(p)}\| \geq \|c_{\alpha(p)} - q_1\|].
\end{aligned}$$

Applying the prior steps to each q_i gives the following.

$$\|p - c_i\|^2 \leq 8(\|p - c_{\alpha(p)}\|^2 + \sum_{j=1}^{\ell} 2^j \|q_j - c_{\alpha(q_j)}\|^2)$$

Let $\mathcal{P}_q(j)$ be the set of all paths P that reach point q using j edges. If $j = 0$, it means P starts with point q. We seek to bound $\sum_{j=0}^{\infty} 2^j \sum_{P \in \mathcal{P}_q(j)} \Delta(P)\|q - c_{\alpha(q_j)}\|^2$. This will bound the charge on point q above over all path P that contains it.

Define a weight function $\Delta'(p)$ for each node $p \in S \setminus C$. This will be a new flow of weights like Δ, except now the weight increases at each node. In particular, give each node initially a weight of 1. Let $\Delta'_o(p)$ be the total weight leaving p. This will be evenly divided among the nodes that have outgoing edges from p. Define $\Delta'_i(p)$ to be the weight incoming to p from all other nodes plus one, the initial weight of p. Set $\Delta'_o(p)$ to be $2\Delta'_i(p)$, twice the incoming weight.

Lemma 69 implies that the maximum weight of any point p is $\Delta'_i(p) \leq 1 + \frac{4}{\log N}$. Further notice that for any q it is the case that $\Delta'_i(q) = \sum_{j=0}^{\infty} 2^j \sum_{P \in \mathcal{P}_q(j)} \Delta(P)$. Letting $\mathcal{P}(p, c_i)$ be the set of all paths that start at p to center c_i. Notice such paths correspond to how p's unit

weight goes to c_i. We have $\omega(p, c_i) = \sum_{P \in \mathcal{P}(p,c_i)} \Delta(P)$. Let $\mathcal{P}$ denote the set of all paths, $\ell(P)$ denote the length of path P (number of edges on P) , and let $P(j)$ denote the jth node on path P. Thus, we have the following.

$$
\begin{aligned}
& \sum_{p \in S} \sum_{c_i \in C} \omega(p, c_i) \|p - c_i\|^2 \\
= & \sum_{p \in S} \sum_{c_i \in C} \sum_{P \in \mathcal{P}(p,c_i)} \Delta(P) \|p - c_i\|^2 \\
\leq & \ 8 \sum_{p \in S} \sum_{c_i \in C} \sum_{P \in \mathcal{P}(p,c_i)} \Delta(P) (\sum_{j=0}^{\ell(p)-1} 2^j \|P(j) - c_{\alpha(P(j))}\|^2) \\
= & \ 8 \sum_{P \in \mathcal{P}} \Delta(P) (\sum_{j=0}^{\ell(p)-1} 2^j \|P(j) - c_{\alpha(P(j))}\|^2) \\
= & \ 8 \sum_{q \in S} \sum_{j=0}^{+\infty} \sum_{P \in \mathcal{P}_q(j)} 2^j \Delta(P) \|q - c_{\alpha(q)}\|^2 \\
= & \ 8 \sum_{q \in S} \Delta'_i(q) \|q - c_{\alpha(q)}\|^2 \\
\leq & \ \sum_{q \in S} 8(1 + \frac{4}{\log N}) \|q - c_{\alpha(q)}\|^2 = 8(1 + \frac{4}{\log N}) \phi^*
\end{aligned}
$$

Lemma 71 follows because if $k' \geq 1067k \log N$, $\phi^* \leq 20\phi_{opt}$ with high probability by Theorem 1 in [12]. □

Finally, we prove that finding any $O(1)$-approximation solution for optimal weighted k-means on the set (C, W') gives a constant approximation for optimal k-means for the original set S. Let $W^f = \{w_1^f, \ldots, w_{k'}^f\}$ be the fractional weights for centers in C. Let $\phi^*_{W^f}$ denote the optimal weighted k-means cost on (C, W^f), and $\phi^*_{W'}$ denote the optimal weighted k-means cost on (C, W'). We first prove that $\phi^*_{W^f} = O(1)\phi_{\mathrm{OPT}}$, where ϕ_{OPT} denote the optimal k-means cost on set S.

Lemma 72. *Let (C, W^f) be the set of points sampled and the weights collected by fractional assignment ω. With high probability, we have $\phi^*_{W^f} = O(1)\phi_{\mathrm{OPT}}$.*

Proof. Consider the cost of the fractional assignment we have designed. For $c_i \in C$, the weight is $w_i^f = \sum_{p \in S} \omega(p, c_i)$. Denote the k-means cost of ω by $\phi_\omega = \sum_{p \in S} \sum_{c \in C} \omega(p, c) \|p - c\|^2$. By Lemma 71, we have that $\phi_\omega \leq 160(1+\epsilon)\phi_{\text{OPT}}$.

Intuitively, in the following we show $\phi^*_{W^f}$ is close to ϕ_ω. As always, we let C_{OPT} denote the optimal centers for k-means on set S. For a set of points X with weights $Y : X \to \mathbb{R}^+$ and a set of centers Z, we let $\phi_{(X,Y)}(Z) = \sum_{x \in X} Y(x) \min_{z \in Z} \|x - z\|^2$ denote the cost of assigning the weighted points in X to their closest centers in Z. Note that $\phi^*_{W^f} \leq \phi_{(C,W^f)}(C_{\text{OPT}})$ since C_{OPT} is chosen with respect to S.

$$
\begin{aligned}
\phi^*_{W^f} &\leq \phi_{(C,W^f)}(C_{\text{OPT}}) \\
&= \sum_{c_i \in C} \Big(\sum_{p \in S} \omega(p, c_i)\Big) \min_{c \in C_{\text{OPT}}} \|c_i - c\|^2 && [w_i^f = \textstyle\sum_{p \in S} \omega(p, c_i)] \\
&= \sum_{c_i \in C} \sum_{p \in S} \min_{c \in C_{\text{OPT}}} \omega(p, c_i) \|c_i - c\|^2 \\
&\leq \sum_{c_i \in C} \sum_{p \in S} \min_{c \in C_{\text{OPT}}} \omega(p, c_i) \cdot 2(\|p - c_i\|^2 + \|p - c\|^2) && [\text{relaxed triangle inequality}] \\
&= 2\phi_\omega + 2\phi_{\text{OPT}} \leq 322(1+\epsilon)\phi_{\text{OPT}}
\end{aligned}
$$

□

Using the mentioned lemmas, we can prove the final approximation guarantee.

Proof of Theorem 63. Using Lemma 70, we know $w'_i = (1 \pm 2\epsilon) w_i^f$ for any center c_i. Let C'_k be k centers for (C, W') that is a γ-approximate for optimal weighted k-means. Let C^f_{OPT} be the *optimal* k centers for (C, W^f), and C'_{OPT} optimal for (C, W'). We have $\phi_{(C,W^f)}(C'_k) \leq (1 + 2\epsilon)\phi_{(C,W')}(C'_k)$ for the reason that the contribution of each point grows by at most $(1+2\epsilon)$ due to weight approximation. Using the same analysis, $\phi_{(C,W')}(C^f_{\text{OPT}}) \leq (1+2\epsilon)\phi^*_{W^f}$. Combining the two inequalities, we have

$$
\begin{aligned}
\phi_{(C,W^f)}(C'_k) &\leq (1+2\epsilon)^2 \phi_{(C,W')}(C'_k) \leq (1+2\epsilon)^2 \gamma \phi^*_{W'} \\
&\leq (1+2\epsilon)^2 \gamma \phi_{(C,W')}(C^f_{\text{OPT}}) && [\text{by optimality of } \phi^*_{W'}] \qquad (37) \\
&\leq (1+2\epsilon)^3 \gamma \phi^*_{W^f} \leq 322\gamma(1+2\epsilon)^4 \phi_{\text{OPT}} && [\text{using Lemma 72}]
\end{aligned}
$$

Let $\phi_S(C'_k) = \sum_{p\in S} \min_{c\in C'_k} \|p-c\|^2$. For every point $p \in S$, to bound its cost $\min_{c\in C'_k} \|p-c\|^2$, we use multiple relaxed triangle inequalities for every center $c_i \in C$, and take the weighted average of them using $\omega(p, c_i)$.

$$
\begin{aligned}
\phi_S(C'_k) &= \sum_{p\in S} \min_{c\in C'_k} \|p-c\|^2 \\
&= \sum_{p\in S} \sum_{c_i\in C} \omega(p,c_i) \min_{c\in C'_k} \|p-c\|^2 && [\textstyle\sum_{c_i\in C} \omega(p,c_i) = 1] \\
&\le \sum_{p\in S} \sum_{c_i\in C} \omega(p,c_i) \min_{c\in C'_k} 2(\|p-c_i\|^2 + \|c_i-c\|^2) && \text{[relaxed triangle inequality]} \\
&= 2\phi_\omega + 2\phi_{(C,W^f)}(C'_k) && [\textstyle\sum_{p\in S} \omega(p,c_i) = w_i^f] \\
&\le 2\phi_\omega + 2\cdot 322\gamma(1+2\epsilon)^4 \phi_{\mathrm{OPT}} && \text{[inequality (37)]} \\
&\le 2\cdot 160(1+\epsilon)\phi_{\mathrm{OPT}} + 2\cdot 322\gamma(1+2\epsilon)^4\phi_{\mathrm{OPT}} && \text{[Lemma 71]} \\
&= O(\gamma)\phi_{\mathrm{OPT}}
\end{aligned}
$$

□

7.0 Conclusion and Future Work

In summary, we designed relational algorithms for machine learning problems such as training logistic regression, training linear SVM, and k-means clustering. One common aspect of most of these problems is that the exact relational implementation of the conventional algorithms is hard, and as a result, we designed alternative approximation algorithms. In our journey to design the aforementioned relational algorithms, we also designed a relational approximation scheme for evaluating a class of SumProd queries under additive inequalities. This relational approximation scheme was used as a building block in designing the algorithms for training linear SVM and k-means clustering. The algorithms designed in this dissertation can be beneficial to commercially used databases, as companies such as RelationalAI and Google are developing databases capable of evaluating machine learning queries.

For linear SVM, we suggested two approaches which one of them can be applied to regularized instances and the other one is suitable for stable instance. For regularized instances, we extract a small subset of data points that we call coreset using uniform sampling and then change the regularizer coefficient and train the model on the uniformly sampled instance. We also show that the size of any other coreset is in the same ballpark as the uniform sampling. Moreover, the same method is applicable to regularized logistic regression. For stable instances of linear SVM, in which a small change of the data points does not change the hypothesis by a lot, we designed a variation of gradient descent algorithm that can be implemented relationally. However, neither of the algorithms is generalizable to all instances of linear SVM, and the problem of training those instances that are not stable and have a small regularizer coefficient remains open.

In addition to linear SVM, we provided a relational approximation algorithm for k-means clustering. Our algorithm samples a small subset of the points using a relational implementation of k-means++ sampling that has a polynomial expected time complexity, and then weights these small sets of points using a randomized relational algorithm. We proved that with high probability this weighted subset of points is a constant approximation

coreset of the original dataset. Prior to this work, there has been no k-means clustering algorithm with polynomial time complexity that can work on relational data.

Besides the positive results, we also proved some hardness results which can broaden our understanding of the limitations of the relational algorithms. We proved some fundamental problems such as counting the number of points in an acyclic join satisfying two linear inequalities is NP-hard to approximate. For linear SVM, we used a similar technique to prove that calculating the gradient of linear SVM is $\#P$-hard when the data is relational, and for k-means clustering, we showed the hardness result for finding the center of mass of the points assigned to a center. These hardness results are suggesting that it is unlikely for us to be able to relationally implement conventional algorithms such as gradient descent and Lloyd's algorithm. Prior to this work, most of the hardness results emphasize on the hardness of evaluating queries on cyclic joins; we believe the techniques used for proving the hardness results in this dissertation provide the foundation for a new way of separating the complexity of the aggregation queries and the structure of the join.

This study provides a springboard for relational approximation algorithms with an emphasize on machine learning applications. In this dissertation, we proved theoretical guarantees for the proposed relational machine learning algorithms; however, more research should investigate the limitation of these algorithms on real datasets using experiments. Future research may also focus on other implementation aspects of our proposed algorithms such as making them cache efficient or suitable for distributed computing. Moreover, there are numerous machine learning problems for which there are no efficient relational algorithms, and there are many other combinatorial use cases with geometric input data that may benefit from a relational algorithm. In the following, we explain what questions can be asked for future work and we explain a few machine learning problems that can benefit from a relational algorithm.

7.1 Experimental Future Work

There are two aspects of the algorithms introduced in this dissertation that can be compared experimentally with other available systems: the runtime and the approximation error. In particular, the algorithms for FAQ-AI, linear SVM, and k-means can be experimentally compared with naive solutions.

In order to achieve the best runtime in practice, there are multiple aspects that one should consider, such as the cache efficiency of the algorithm and many special guarantees that some join queries provide. For example, in practice, many datasets have enough functional dependencies that can prevent the exponential growth of the size of the join. As a result, one may consider a looser definition for relational algorithms that can be used for different machine learning problems and perform well on those datasets.

For the same reason, many of the approximation guarantees that were provided in this dissertation might be pessimistic in practice and the algorithms may achieve better approximation error in practice. For example, if functional dependencies provide a one to one correspondence between the rows of the design matrix and the rows of an input table T_i, then our algorithm for FAQ-AI can produce an exact solution if we group the result by T_i. This phenomenon can be also seen in some of the prior works such as Rk-means [38], in which the approximation error in their experiments is much lower than their theoretical approximation guarantee.

Another implementation aspect that should be addressed is handling the categorical features and working efficiently with compression algorithms. Often data stored in databases have categorical features and they are stored in a compressed format. However, in all algorithms in this dissertation, we have assumed that the categorical columns are already encoded as numerical values. This can potentially be problematic for datasets with lots of categorical data as it is shown in [6]. One may see categorical data as a sparse representation of their one hot encoding. Therefore, it can be more efficient to handle the categorical data and computations related to them using sparse computation and adapt our relational algorithms accordingly. Furthermore, for compressed data, the algorithm can be more memory efficient if it can return the aggregation results of the compressed data without decompressing them first.

7.2 Boosting Decision Trees

Boosting is often used with limited decision trees and it may provide great accuracy for tabular data. The motivation for boosting is combining the output of multiple weak regressors and it can utilize limited depth decision tree as a weak classifier and train multiple of them on the data. In the following, we explain the decision tree model, the greedy algorithm for training a decision tree, and AdaBoost algorithm which is one of the famous boosting algorithms. Then we explain the open problem of training a relational decision tree.

Decision Trees: Decision trees are a famous model for classification. A decision tree is a binary tree with a fixed depth. Each internal node v has a dimension d_v and a threshold α_v assigned to it, and the leaves have a prediction label. To predict the label of a data point x using a decision tree, the algorithm starts from the root r and checks if $x_{d_r} \leq \alpha_r$. If that condition is true, the algorithm recursively repeats the same procedure for the left child of r and otherwise it goes to the right child of r. At the end, after reaching to a leaf node, the algorithm returns the prediction value of that leaf as the prediction for the label of x.

A famous algorithm for training decision trees is the greedy algorithm. The input to the greedy algorithm is a set of d dimensional points $X = \{x_1, \ldots, x_n\}$ with associated labels $Y = \{y_1, \ldots, y_n\}$ and a depth L. The algorithm for training the decision tree is a recursive algorithm that starts from the root to assign a dimension and a threshold to the root, and then it divides the data points using the assigned dimension. The algorithm passes the points that are less than the threshold to the left branch and the rest to the right branch and trains each subtree independently using the same recursive algorithm and the data points assigned to them. To find the best dimension and threshold, the greedy algorithm considers all possible dimensions of $i\{1, \ldots, d\}$, and for every choice of the dimension it considers all possible thresholds of that dimension. It chooses the combination that minimizes the impurity of the points assigned to the left branch and the right branch. The leaves of the tree will have a prediction label instead of a dimension and threshold. The prediction label of a leaf is the label with the most number of data points among the points assigned to that leaf.

More formally, given a dimension i, and a threshold α, let X_L be $\{x \,|\, x \in X \text{ and } x_i \leq \alpha\}$ and Y_L be the label of these points. Furthermore, let $X_R = X \setminus X_L$ with the labels Y_R. Then the algorithm chooses the combination (i, α) that minimizes

$$|X_L| I(Y_L) + |X_R| I(Y_R)$$

where $I(Y)$ is the impurity of the labels. Let Z be the set of all possible labels, and given a collection of data points' labels Y, let $k(Y)$ be the majority label in Y and $n_i(Y)$ be the number of occurance of some label i in Y. Then, some common choices of measuring the impurity are the following functions [47]:

- **Misclasification error:** $\frac{1}{|Y|} \sum_{i \in Z : i \neq k(Y)} n_i(Y)$
- **Gini index:** $\frac{1}{|Y|^2} \sum_{i \neq j} n_i(Y) n_j(Y)$
- **Cross-entropy or deviance:** $-\sum_{i \in Z} \frac{n_i(Y))}{|Y|} \log(\frac{n_i(Y)}{|Y|})$

AdaBoost: One of the famous boosting algorithms is AdaBoost. It utilizes a series of weak classifiers and weight them in order to get a stronger classifier with a better accuracy. The weak classifiers should be able to generate a better than 50 percent accuracy on the weighted set of data points with binary labels. AdaBoost starts with uniform weight of $\frac{1}{|X|}$ on data points. It then trains the first weak classifier and after assigning the weight α_1 to the classifier itself, it changes the weight of the data points based on the prediction of this classifier and trains the second classifier on the data points using the new weights, and it repeats the same process. At the end, the prediction will be the weighted majority of the predictions of the weak classifiers [46].

More formally, let w_j^i be the weight of point (x_j, y_j) after training the i-th classifier, note that $w_j^0 = \frac{1}{|X|}$ for all j. Let $f_i(x_j)$ be the prediction of the classifier i on x_j, and let a_i be the weighted error of the i-th classifier on the data points using the previous weights; that is

$$a_i = \sum_{j \,:\, f(x_j) \neq y_j} w_j^{i-1}.$$

Then, the algorithm performs the following steps for $i = 1, \ldots, T$:

A. Train the classifier f_i using the weights $w_1^{i-1}, \ldots, w_n^{i-1}$.

B. Set $\alpha_i = \frac{1}{2} \log(\frac{1-a_i}{a_i})$.

C. Set $\beta_i = 2\sqrt{a_i(1-a_i)}$.

D. For all points j that $f_i(x_j) = y_j$, set $w_j^i = \frac{w_j^{i-1} e^{-\alpha_i}}{\beta_i}$.

E. For all points j that $f_i(x_j) \neq y_j$, set $w_j^i = \frac{w_j^{i-1} e^{\alpha_i}}{\beta_i}$.

Then the final prediction for a point is $F(x_j) = \text{sign}(\sum_i \alpha_i f_i(x_j))$, and we have the following theorem on the accuracy of F on the original dataset.

Theorem 73 ([46]). *Let $\gamma_i = 1 - a_i$, then the training error of the final hypothesis F on the dataset is at most* $\exp(-2\sum_i \gamma_i^2)$.

Relational boosting decision trees: It is possible to have a relational algorithm for training a decision tree using SumProd queries. That is because given a set of axis-parallel constraints, as we saw in Chapter 6, we can count the number of rows satisfying those constraints grouped by an input table. However, it is not clear how to train boosted regression trees without having an exponential dependence on the number of trees. The hardest part is measuring the impurity of a node using the weights obtained by the previous trees. Note that, unlike the conventional setting, in a relational setting we cannot store the weight of the points because it will take the time proportionate to the size of the design matrix.

An open question is whether it is possible to have a relational boosting decision tree algorithm using AdaBoost. If not, the next question would be if it is possible to have some other boosting algorithm with a similar guarantee.

7.3 Euclidean K-Centers Problem

The input to the euclidean K-Centers problem is a set of d dimensional points $X = \{x_1, \ldots, x_n\}$ and the output is a set C of K points, called the centers, such that C minimizes the following objective

$$F(C) = \max_{x_i \in X} \min_{c_i \in C} \|c_i - x_i\|_2$$

Note that if we assign each point in X to its closest center in C, then $F(C)$ is the furthest distance that any point has to its center. In the relational context, the set of points X is the design matrix that can be obtained by joining m input tables.

It can be shown that Euclidean K-centers problem is NP-Hard to solve or approximate up to any constant less than 1.82 [44]. Also it can be shown K-centers problem in general metric space is NP-Hard to approximate up to any constant less than 2 by a gap reduction from dominant set [60]. The common approximation algorithm for k-center is the following greedy algorithm that will produce a 2-approximate solution:

- Pick an arbitrary point from X as c_1
- Repeatedly pick the j-th center as the point $c_j = \text{argmax}_{x \in X} \min_{i \in [k-1]} \|c_i - x\|_2$

The bad news is that the following theorem shows that finding the point that has the furthest distance to its center is NP-Hard. However, finding the furthest point up to any constant factor is also sufficient for a constant approximation algorithm for K-centers.

Theorem 74. *Given an acyclic join J with d columns, and two d dimensional centers c_1, c_2, it is NP-hard to find the point x in J that maximizes* $\min_{i \in 1,2} \|c_i - x\|_2$.

Proof. We can prove this theorem by a reduction from the partition problem which is a famous NP-Complete problem. In the partition problem, the input is a set of weights $w_1, \ldots, w_m$ and the output is true if we can divide the weights into two disjoint subsets such that their summation is the same.

We perform the reduction by creating m tables such that table T_i has one column c_i and two rows with values 0 and $\sqrt(w_i)$. And, we use the following centers: $c_1 = (0, \ldots, 0)$ and $c_2 = (\sqrt(w_1), \ldots, \sqrt(w_m))$. The claim is if $\min_{i \in 1,2} \|c_i - x\|_2 = \sqrt(\sum_i w_i^2/2)$, then the output of the partition problem is true. To see the claim, note that the design matrix has 2^m rows, and each row has a subset of square roots of the weights. That means the squared distance of a row to c_1 represents the total weight of a subset of weights and its squared distance to c_2 is the total weight of the complement of that subset. Therefore, if there is any row with distance $\sqrt(\sum_i w_i^2/2)$ to one of the centers, its distance to the other center is also the same and the answer to the partition problem is true. □

7.4 DBSCAN Algorithm

Density-based spacial clustering of applications with noise (DBSCAN) is a famous clustering algorithm which is usually used when the clusters have long and skinny shapes. Unlike k-means or k-centers, DBSCAN is not a problem and it is rather an algorithm. The algorithm assumes a distance function and performs the clustering based on that. The input to the algorithm is a set of points $X = \{x_1, \ldots, x_n\}$, parameters ϵ and M. The following is the abstract description of the algorithm [94]:

A. Find all the *core points*. A core point is a point which has M other points within ϵ distance of it.
B. Assign all the core points that are in ϵ distance of each other to the same cluster.
C. Find all *border points* and assign it to the same cluster as the closest core point to it. A border point is a point that has a core point in ϵ distance of it.
D. Remove all the *noise points*. A noise point is a point which does not have any core point within its ϵ distance.

The algorithm only needs to return the core points; then, using core points, it is possible to find the assigned cluster of any given point. The runtime of the algorithm depends on how fast one can perform neighbourhood queries (counting the number of points within some distance of a given point) and the complexity of the distance function. If the neighborhood query is done by a linear scan of the data and the distance function is Euclidean distance, then the runtime of the algorithm is $O(N^2 d)$ when the data is explicitly present. There are data structures that can improve the runtime in practice, such as R-Trees [53], k-d-trees [16], and locality sensitive hashing [62, 49]. However, there are no good theoretical guarantees for these data structures, and it is proven that any data structure that can return the neighborhood query in time less than $\Omega(n^{1/3})$ needs $\Omega(n^{4/3})$ construction time [48] unless Hopcroft problem can be solved in $o(n^{4/3})$.

While there are relational algorithms for clustering algorithms such as K-means, there is no relational algorithm for DBSCAN. Therefore, an interesting research question would be whether there is an efficient implementation of DBSCAN that accepts relational input.

Appendix Pseudocodes for Section 6.2.

Algorithm 1 Algorithm For Creating Axis-Parallel Hyperrectangles

procedure CONSTRUCT BOXES(C_{i-1})

Input: Current centers $C_{i-1} = \{c_1, \ldots, c_{i-1}\}$

Output: $\mathcal{B}_i$, a set of boxes and their centers

$\mathcal{B}_i \leftarrow \emptyset$

$\mathcal{G}_i \leftarrow \{(b_j^*, c_j) \mid b_j^*$ is a unit size hyper-cube around $c_j, j \in [i-1]\}$ ▷ We assume there is no intersection between the boxes in G initially, up to scaling

while $|\mathcal{G}_i| > 1$ **do**

Double all boxes in $\mathcal{G}_i$.

$\mathcal{G}_i' = \emptyset$ ▷ Keeps the boxes created in this iteration of doubling

while $\exists (b_1, y_1), (b_2, y_2) \in \mathcal{G}_i$ that intersect with each other **do**

$b \leftarrow$ the smallest box in Euclidean space containing both b_1 and b_2.

$\mathcal{G}_i \leftarrow (\mathcal{G}_i \setminus \{(b_1, y_1), (b_2, y_2)\}) \cup \{(b, y_1)\}$

$\mathcal{G}_i' \leftarrow (\mathcal{G}_i' \cup \{(b, y_1)\}$

if $(b_1, y_1) \notin \mathcal{G}_i'$ **then** ▷ Check if box b_1 hasn't been merged with other boxes in the current round

$b_1' \leftarrow$ halved b_1, add (b_1', y_1) to $\mathcal{B}_i$

if $(b_2, y_2) \notin \mathcal{G}_i'$ **then** ▷ Check if box b_2 hasn't been merged with other boxes in the current round

$b_2' \leftarrow$ halved b_2, add (b_2', y_2) to $\mathcal{B}_i$

There is only one box and its representative remaining in $\mathcal{G}_i$, replace this box with the whole Euclidean space.

$\mathcal{B}_i \leftarrow \mathcal{B}_i \cup \mathcal{G}_i$.

Return $\mathcal{B}_i$.

Algorithm 2 Algorithm for sampling the next center

procedure KMeans++Sample($C_{i-1}, T_1, \ldots, T_m$)
  Let $p(b)$ be the box that is the parent of b in the tree structure of all boxes in $\mathcal{B}_i$.
  $c_i \leftarrow \emptyset$
  $\mathcal{B}_i \leftarrow$ Construct Boxes(C_{i-1})
  Let (b_0, y_0) be the tuple where b_0 is the entire Euclidean space in $\mathcal{B}_i$.
  while $c_i = \emptyset$ **do**
    for $1 \leq \ell \leq m$ **do** ▷ Sample one row from each table.
      Let H be a vector having an entry H_r for each $r \in T_\ell$.
      $J' \leftarrow r_1 \bowtie \ldots \bowtie r_{\ell-1} \bowtie J$. ▷ Focus on only the rows in J that uses all previously sampled rows from $T_1, \ldots, T_{\ell-1}$ in the concatenation.
      $\forall r \in T_\ell$ evaluate $H_r \leftarrow \sum_{x \in r \bowtie J' \cap b_0} \|x - y_0\|_2^2$
      for $(b, y) \in \mathcal{B}_i \setminus \{(b_0, y_0)\}$ **do**
        Let $(b', y') \in \mathcal{B}_i$ be the tuple where $b' = p(b)$.
        $\forall r \in T_\ell$ use SumProd query to evaluate two values: $\sum_{x \in r \bowtie J' \cap b} \|x - y\|_2^2$ and $\sum_{x \in r \bowtie J' \cap b} \|x - y'\|_2^2$.
        $H_r \leftarrow H_r - \sum_{x \in r \bowtie J' \cap b} \|x - y'\|_2^2 + \sum_{x \in r \bowtie J' \cap b} \|x - y\|_2^2$
      Sample a row $r_\ell \in T_\ell$ with probability proportional to H_r.
    $x \leftarrow r_1 \bowtie \cdots \bowtie r_m$.
    Let (b^*, y^*) be the tuple where b^* is the smallest box in $\mathcal{B}_i$ containing x.
    $c_i \leftarrow x$ with probability $\frac{\min_{c \in C_{i-1}} \|x-c\|_2^2}{\|x-y^*\|_2^2}$. ▷ Rejection sampling.
  return c_i.

Bibliography

[1] Kaggle machine learning and data science survey. `https://www.kaggle.com/kaggle/kaggle-survey-2018`, 2018.

[2] Mahmoud Abo Khamis, Ryan R Curtin, Benjamin Moseley, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. On functional aggregate queries with additive inequalities. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 414–431. ACM, 2019.

[3] Mahmoud Abo-Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. Instance optimal join size estimation. *arXiv preprint arXiv:2012.08083*, 2020.

[4] Mahmoud Abo-Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. Approximate aggregate queries under additive inequalities. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 85–99. SIAM, 2021.

[5] Mahmoud Abo-Khamis, Sungjin Im, Benjamin Moseley, Kirk Pruhs, and Alireza Samadian. A relational gradient descent algorithm for support vector machine training. In *Symposium on Algorithmic Principles of Computer Systems (APOCS)*, pages 100–113. SIAM, 2021.

[6] Mahmoud Abo Khamis, Hung Q Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Ac/dc: in-database learning thunderstruck. In *Second Workshop on Data Management for End-To-End Machine Learning*, page 8. ACM, 2018.

[7] Mahmoud Abo Khamis, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. In-database learning with sparse tensors. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 325–340, 2018.

[8] Mahmoud Abo Khamis, Hung Q. Ngo, Dan Olteanu, and Dan Suciu. Boolean tensor decomposition for conjunctive queries with negation. In *ICDT*, pages 21:1–21:19, 2019.

[9] Mahmoud Abo Khamis, Hung Q. Ngo, and Atri Rudra. Faq: Questions asked frequently. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '16, pages 13–28, 2016.

[10] Margareta Ackerman and Shai Ben-David. Clusterability: A theoretical study. In *Artificial intelligence and statistics*, pages 1–8, 2009.

[11] Isolde Adler. *Width functions for hypertree decompositions*. PhD thesis, Citeseer, 2006.

[12] Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive sampling for k-means clustering. In *International Conference on Approximation Algorithms for Combinatorial Optimization Problems*, pages 15–28. 2009.

[13] Srinivas M Aji and Robert J McEliece. The generalized distributive law. *IEEE transactions on Information Theory*, 46(2):325–343, 2000.

[14] Haris Angelidakis, Konstantin Makarychev, and Yury Makarychev. Algorithms for stable and perturbation-resilient problems. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 438–451, 2017.

[15] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 1027–1035, 2007.

[16] Sunil Arya and David M Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA*, volume 93, pages 271–280, 1993.

[17] Albert Atserias, Martin Grohe, and Dániel Marx. Size bounds and query plans for relational joins. In *2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 739–748. IEEE, 2008.

[18] Pranjal Awasthi, Avrim Blum, and Or Sheffet. Center-based clustering under perturbation stability. *Information Processing Letters*, 112(1-2):49–54, 2012.

[19] Olivier Bachem, Mario Lucic, and Andreas Krause. Scalable k -means clustering via lightweight coresets. In *ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2018.

[20] Mihai Badoiu and Kenneth L. Clarkson. Smaller core-sets for balls. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 801–802, 2003.

[21] Mihai Badoiu, Sariel Har-Peled, and Piotr Indyk. Approximate clustering via core-sets. In *Proceedings on 34th Annual ACM Symposium on Theory of Computing, May 19-21, 2002, Montréal, Québec, Canada*, pages 250–257, 2002.

[22] Maria-Florina Balcan, Avrim Blum, and Anupam Gupta. Clustering under approximation stability. *Journal of the ACM (JACM)*, 60(2):1–34, 2013.

[23] Maria Florina Balcan and Yingyu Liang. Clustering under perturbation resilience. *SIAM Journal on Computing*, 45(1):102–155, 2016.

[24] Nikhil Bansal and Anupam Gupta. Potential-function proofs for first-order methods. *CoRR*, abs/1712.04581, 2017.

[25] Jinbo Bi and Tong Zhang. Support vector classification with input data uncertainty. In *Advances in neural information processing systems*, pages 161–168, 2005.

[26] Yonatan Bilu and Nathan Linial. Are stable instances easy? *Combinatorics, Probability and Computing*, 21(5):643–660, 2012.

[27] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[28] Vladimir Braverman, Dan Feldman, and Harry Lang. New frameworks for offline and streaming coreset constructions. *CoRR*, abs/1612.00889, 2016.

[29] Mihai Bădoiu and Kenneth L. Clarkson. Optimal core-sets for balls. *Comput. Geom. Theory Appl.*, 40(1):14–22, May 2008.

[30] Peter Buhlmann and Sara van de Geer. *Statistics for High-Dimensional Data: Methods, Theory and Applications*. Springer Publishing Company, 2011.

[31] Andriy Burkov. *The Hundred Page Machine Learning Book*.

[32] Andriy Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[33] Iván Cantador, Peter Brusilovsky, and Tsvi Kuflik. Second workshop on information heterogeneity and fusion in recommender systems (hetrec2011). In *Proceedings of the fifth ACM conference on Recommender systems*, pages 387–388, 2011.

[34] George Casella, Christian P Robert, Martin T Wells, et al. Generalized accept-reject sampling schemes. In *A Festschrift for Herman Rubin*, pages 342–347. Institute of Mathematical Statistics, 2004.

[35] Yu Chen and Ke Yi. Random sampling and size estimation over cyclic joins. In *23rd International Conference on Database Theory (ICDT 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[36] Zhaoyue Cheng and Nick Koudas. Nonlinear models over normalized data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1574–1577. IEEE, 2019.

[37] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. 2009.

[38] Ryan R. Curtin, Benjamin Moseley, Hung Q. Ngo, XuanLong Nguyen, Dan Olteanu, and Maximilian Schleich. Rk-means: Fast clustering for relational data. *CoRR*, abs/1910.04939, 2019.

[39] Amit Daniely, Nati Linial, and Michael Saks. Clustering is difficult only when it does not matter. *arXiv preprint arXiv:1205.4891*, 2012.

[40] Anirban Dasgupta, Petros Drineas, Boulos Harb, Ravi Kumar, and Michael W. Mahoney. Sampling algorithms and coresets for ℓ_p regression. *SIAM J. Comput.*, 38(5):2060–2078, 2009.

[41] Rina Dechter. Bucket elimination: A unifying framework for probabilistic inference. In *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*, 1996.

[42] Martin Dyer. Approximate counting by dynamic programming. In *ACM Symposium on Theory of Computing*, pages 693–699, 2003.

[43] Tarek Elgamal, Shangyu Luo, Matthias Boehm, Alexandre V Evfimievski, Shirish Tatikonda, Berthold Reinwald, and Prithviraj Sen. Spoof: Sum-product optimization and operator fusion for large-scale machine learning. In *CIDR*, 2017.

[44] Tomás Feder and Daniel Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444, 1988.

[45] Dan Feldman and Michael Langberg. A unified framework for approximating and clustering data. In *ACM Symposium on the Theory of Computing*, 2011.

[46] Yoav Freund, Robert Schapire, and Naoki Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[47] Jerome Friedman, Trevor Hastie, Robert Tibshirani, et al. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[48] Junhao Gan and Yufei Tao. Dbscan revisited: Mis-claim, un-fixability, and approximation. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*, pages 519–530, 2015.

[49] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *Vldb*, volume 99, pages 518–529, 1999.

[50] Martin Grohe. The structure of tractable constraint satisfaction problems. In *International Symposium on Mathematical Foundations of Computer Science*, pages 58–72. Springer, 2006.

[51] Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[52] Maya Gupta. Machine learning crash course.

[53] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, 1984.

[54] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *ACM-SIAM Symposium on Discrete Algorithms*, 2004.

[55] Sariel Har-Peled. *Geometric Approximation Algorithms*. American Mathematical Society, 2011.

[56] Sariel Har-Peled, Dan Roth, and Dav Zimak. Maximum margin coresets for active and noise tolerant learning. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 836–841, 2007.

[57] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *Acm transactions on interactive intelligent systems (tiis)*, 5(4):1–19, 2015.

[58] Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2(3-4):157–325, 2016.

[59] Joe Hellerstein, Christopher Ré, Florian Schoppmann, Daisy Zhe Wang, Eugene Fratkin, Aleksander Gorajek, Kee Siong Ng, Caleb Welton, Xixuan Feng, Kun Li, et al. The madlib analytics library or mad skills, the sql. *arXiv preprint arXiv:1208.4165*, 2012.

[60] Dorit S Hochbaum and David B Shmoys. A unified approach to approximation algorithms for bottleneck problems. *Journal of the ACM (JACM)*, 33(3):533–550, 1986.

[61] Jonathan H. Huggins, Trevor Campbell, and Tamara Broderick. Coresets for scalable bayesian logistic regression. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 4087–4095, 2016.

[62] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

[63] Rajesh Jayaram, Alireza Samadian, David Woodruff, and Peng Ye. In-database regression in input sparsity time. In *International Conference on Machine Learning*, pages 4797–4806. PMLR, 2021.

[64] Anthony Klug. On conjunctive queries containing inequalities. *J. ACM*, 35(1):146–160, January 1988.

[65] J. Kohlas and N. Wilson. Semiring induced valuation algebras: Exact and approximate local computation algorithms. *Artif. Intell.*, 172(11):1360–1399, 2008.

[66] Paraschos Koutris, Tova Milo, Sudeepa Roy, and Dan Suciu. Answering conjunctive queries with inequalities. *Theory of Computing Systems*, 61(1):2–30, Jul 2017.

[67] Amit Kumar and Ravindran Kannan. Clustering with spectral norm and the k-means algorithm. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, pages 299–308. IEEE, 2010.

[68] Arun Kumar, Mona Jalal, Boqun Yan, Jeffrey Naughton, and Jignesh M Patel. Demonstration of santoku: optimizing machine learning over normalized data. *Proceedings of the VLDB Endowment*, 8(12):1864–1867, 2015.

[69] Arun Kumar, Jeffrey Naughton, and Jignesh M. Patel. Learning generalized linear models over normalized data. In *ACM SIGMOD International Conference on Management of Data*, pages 1969–1984, 2015.

[70] Arun Kumar, Jeffrey Naughton, Jignesh M. Patel, and Xiaojin Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *International Conference on Management of Data*, pages 19–34, 2016.

[71] Michael Langberg, Shi Li, Sai Vikneshwar Mani Jayaraman, and Atri Rudra. Topology dependent bounds for faqs. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 432–449. ACM, 2019.

[72] Side Li, Lingjiao Chen, and Arun Kumar. Enabling and optimizing non-linear feature interactions in factorized linear algebra. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1571–1588, 2019.

[73] Shangyu Luo, Zekai J Gao, Michael Gubanov, Luis L Perez, and Christopher Jermaine. Scalable linear algebra on a relational database system. *IEEE Transactions on Knowledge and Data Engineering*, 31(7):1224–1238, 2018.

[74] Konstantin Makarychev, Yury Makarychev, and Aravindan Vijayaraghavan. Bilu–linial stable instances of max cut and minimum multiway cut. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 890–906. SIAM, 2014.

[75] Nantia Makrynioti, Ruy Ley-Wild, and Vasilis Vassalos. sql4ml a declarative end-to-end workflow for machine learning. *arXiv preprint arXiv:1907.12415*, 2019.

[76] Nantia Makrynioti and Vasilis Vassalos. Declarative data analytics: a survey. *IEEE Transactions on Knowledge and Data Engineering*, 2019.

[77] Gustavo Malkomes, Matt J. Kusner, Wenlin Chen, Kilian Q. Weinberger, and Benjamin Moseley. Fast distributed k-center clustering with outliers on massive data. In *NeuralPS*, 2015.

[78] Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. *Journal of the ACM (JACM)*, 60(6):42, 2013.

[79] Adam Meyerson, Liadan O'Callaghan, and Serge A. Plotkin. A k-median algorithm with running time independent of data size. *Machine Learning*, 56(1-3):61–87, 2004.

[80] Vahab S. Mirrokni and Morteza Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *ACM Symposium on the Theory of Computing*, 2015.

[81] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[82] Benjamin Moseley, Kirk Pruhs, Alireza Samadian, and Yuyan Wang. Relational algorithms for k-means clustering. *arXiv preprint arXiv:2008.00358*, 2021.

[83] Alexander Munteanu and Chris Schwiegelshohn. Coresets-methods and history: A theoreticians design pattern for approximation and streaming algorithms. *KI - Künstliche Intelligenz*, 32(1):37–53, Feb 2018.

[84] Alexander Munteanu, Chris Schwiegelshohn, Christian Sohler, and David P. Woodruff. On coresets for logistic regression. In *NeurIPS*, 2018.

[85] Sahand N. Negahban, Pradeep Ravikumar, Martin J. Wainwright, and Bin Yu. A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers. In *Neural Information Processing Systems*, pages 1348–1356, 2009.

[86] Hung Q. Ngo, Ely Porat, Christopher Ré, and Atri Rudra. Worst-case optimal join algorithms: [extended abstract]. In *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 37–48, 2012.

[87] Liadan O'Callaghan, Adam Meyerson, Rajeev Motwani, Nina Mishra, and Sudipto Guha. Streaming-data algorithms for high-quality clustering. In *International Conference on Data Engineering*, pages 685–694, 2002.

[88] Rafail Ostrovsky, Yuval Rabani, Leonard J Schulman, and Chaitanya Swamy. The effectiveness of lloyd-type methods for the k-means problem. *Journal of the ACM (JACM)*, 59(6):1–22, 2013.

[89] Steffen Rendle. Scaling factorization machines to relational data. In *Proceedings of the VLDB Endowment*, volume 6, pages 337–348. VLDB Endowment, 2013.

[90] Tim Roughgarden. Beyond worst-case analysis. *Commununications of the ACM*, 62(3):88–96, 2019.

[91] Alireza Samadian, Kirk Pruhs, Benjamin Moseley, Sungjin Im, and Ryan Curtin. Unconditional coresets for regularized loss minimization. In *International Conference on Artificial Intelligence and Statistics*, pages 482–492. PMLR, 2020.

[92] Maximilian Schleich, Dan Olteanu, Mahmoud Abo Khamis, Hung Q Ngo, and Xuan-Long Nguyen. A layered aggregate engine for analytics workloads. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1642–1659, 2019.

[93] Maximilian Schleich, Dan Olteanu, and Radu Ciucanu. Learning linear regression models over factorized joins. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, pages 3–18. ACM, 2016.

[94] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: why and how you should (still) use dbscan. *ACM Transactions on Database Systems (TODS)*, 42(3):1–21, 2017.

[95] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, New York, NY, USA, 2014.

[96] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[97] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011.

[98] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. Mit Press, 2012.

[99] Elad Tolochinsky and Dan Feldman. Coresets for monotonic functions with applications to deep learning. *CoRR*, abs/1802.07382, 2018.

[100] Ivor W. Tsang, James T. Kwok, and Pak-Ming Cheung. Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6:363–392, 2005.

[101] Vladimir N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.

[102] Keyu Yang, Yunjun Gao, Lei Liang, Bin Yao, Shiting Wen, and Gang Chen. Towards factorized svm with gaussian kernels over normalized data. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1453–1464. IEEE, 2020.

[103] Zhuoyue Zhao, Robert Christensen, Feifei Li, Xiao Hu, and Ke Yi. Random sampling over joins revisited. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1525–1539, 2018.