From Raw Data to Processable Informative Data : Training Data Management for Big Data Analytics

GAO JINYANG

Bachelor of Science

Peking University, China

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2016

Declaration

I hereby declare that this thesis is my original work and it has been written by me in its entirety. I have duly acknowledged all the sources of information which have been used in the thesis.

This thesis has also not been submitted for any degree in any university previously.

Gao Jinyang
07 September 2016

ACKNOWLEDGEMENTS

This thesis would not have been accomplished without the support, advice and encouragement of so many people. I hereby thank them for their remarkable guidance and help.

First and foremost, my sincere gratitude to my supervisor, Professor Beng Chin Ooi, who has supported me throughout my study with his research insights, prospective thoughts and inspiriting moral. He teaches me to develop and expand my potential, and is always suggesting me to see things in a positive light. He also provides me lots of opportunities to broaden my horizon and develop new skills. Although very serious in research, Prof. Ooi is easy to get along with in life. He acts not only a supervisor, but also an elder friend that we grow and learn together. It has been my great honor to be his student. Inspired by him, I'm on the way of devoting myself to academic research, for self-fulfillment.

I would like to thank to Prof. H.V.Jagadish for his valuable insights and advice for my research. We met in 2013 when he took his sabbatical in Singapore. We have innumerable technical discussions about most of my thesis works. He taught me valuable research skills and insightful advice on the way of thinking.

I would like to thank to all my collaborators during my PhD study, especially Dr. Xuan Liu, Dr. Wei Lu, Dr. Ju Fan, Dr. Li Qian, Dr. Haixun Wang, Sheng Wang and Wei Wang for their assistance and support to my research works.

I would like to thank to all my labmates and ex-labmates: Dr Sai Wu, Dr Meihui Zhang, Dr Feng Li, Hao Zhang, Qian Lin, Chang Yao, Zhaojing

Luo, Kaiping Zheng, Xiangrui Cai, Wenfeng Wu et.al. for their assistance and support to my research and life.

I would like to thank my undergraduate friends Yang Yuan, Zhuo Peng, Tian Lan, Ziqian Hu and Xinran He for helping me developing skills from theory, engineering to learning and solving several sophisticated problems.

I would like to thank Prof. Roland Yap, Prof. Wee Sun Lee, and Prof. Chee-Yong Chan, and the external reviewer for their valuable comments from the beginning of my research proposal to the final thesis dissertation.

I would like to thank to my previous mentors Dr. Min Wang, Dr. Conglei Yao and Dr. Lijiang Chen for introducing me to the research world, and giving me continuing encouragement during my PhD study.

Thank you to my girlfriend Dr. Yanyan Shen, for all her love and support.

I owe my deepest gratitude to my parents for their endless love.

CONTENTS

ABSTRACT

Due to the surging volume of Big Data, data-driven approaches are playing an ever-increasing role in nowadays knowledge discoveries and decision makings. Though cheap raw data from various sources are produced everywhere, most of them cannot be directly used as training data and benefit analytics tasks. This is mainly because the size of raw data is usually too large to be directly processed, and the informative value in raw data is not as high as that collected from deliberately designed experiments. To fulfill the use of Big Data, there is an increasing need to establish an infrastructure for training data management, transforming raw data to processable informative data, by leveraging both human effort and computational resources.

In this thesis, we aim to develop effective and efficient solutions to transform the Big Data into a processable and informative form. Two challenging problems are discussed and addressed. The first challenge is to increase the information value in Big Data, mainly by acquiring extra supervised information from data annotation. We propose a preference quantified model to annotate complex tasks where the supervised information is difficult to be represent by simple labels, and adapt an active learning approach to reduce the cost of human efforts. To further reduce the cost of data annotation by using crowdsourcing, we develop a cost-sensitive method for crowdsourced data quality management. The second challenge is to squeeze and reorganize the data to a processable form without losing much information inside the original data, which typically includes representing, compressing, indexing and sampling the data to increase the computational efficiency. We propose a hashing

method to transform the training data into better compact representation, while preserving both internal information in each instances and external relations among those instances. Moreover, we index the data which are usually high-dimensional to support similarity queries based on the distance independent k-nearest neighbor measure. Finally, we study the effect of data sampling pattern on the efficiency of analytics model training, aiming to provide the most informative data in a processable size to the analytics model to speed up the model training procedure.

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Big Data Analytics

With the rapid growth of the Internet and production informatization, in everyday life, terabytes or even petabytes of data are generated from Internet users, collected from sensors or recorded in digital logs. In general, the quantity of data influences the quality of data-driven analytics. Decisions that are previously based on experiences or deliberately designed models, would be made based on the study of data itself. The prevalence of complex analytics models are growing at a fast rate, ranging from million-dimensional linear models to complex models like Deep Neural Networks or topic models. These models are playing increasingly important roles in various application domains such as healthcare, advertising, education and consulting. Through better analysis of the massive volumes of data that are becoming available, we increase the potential for making faster scientific advances, expediting enterprise development and conferring social benefits.

While the promise of Big Data is coming real, there is still an unnegligible gap between its potential and usability in practice. Big Data itself is not cheap to acquire and process. First, though cheap raw data from various sources are produced everywhere, without supervised information much of them are of no value. So as human cannot learn a new language without a dictionary, machine cannot decipher information in Big Data without labeled data as corpus.

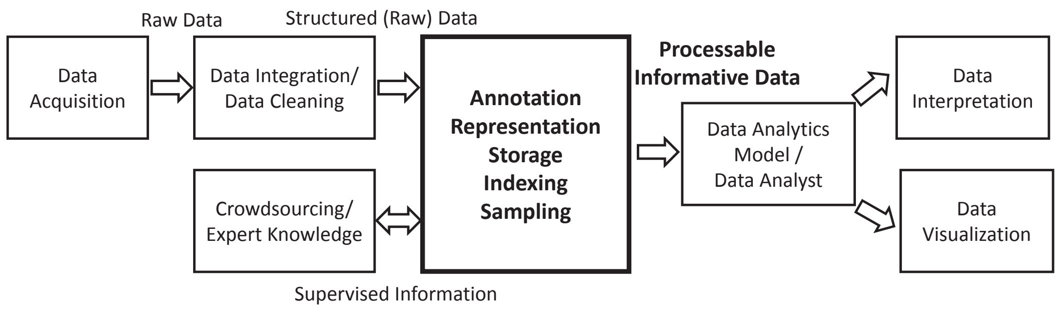


Figure 1.1: Thesis Focus: From Raw Data to Processable Informative Data

However, the acquisition of supervised information requires human annotation, resulting in a costlier exploitation of data. Second, Big Data also raises the computational complexity of data processing. Due to the increasing volume of Big Data, analytics models usually cannot afford to process all the data together. As a result, individual data instances are selected, located and processed at each time. Finding such needles from the Big Data haystack is imposing great challenges to the data processing technology. To summarize, simply developing more and more sophisticated analytics models is not sufficient to fully realize the potential of Big Data. Just as making bricks without straw is impossible, managing the training data properly is the prerequisite of the development of analytics models. Thereby, to support those analytics models at affordable cost, there is an increasing need to establish a cost-sensitive infrastructure for training data management, which prepares informative training data with human annotation and is able to quickly respond to various data access operations.

Figure 1.1 describes the pipeline of typical big data analytics. Though after the data integration and data cleaning stage we do acquire lots of cleaned structured data, such kind of data is still far from the expectation from analytics models, where the data should be informative and processable. For instances, the data may lack supervised information, live in inadequate format for analytics, need to be storage efficiently and build inter-instance relations, and valuable data instances may be hided by massive trite data. Therefore, we aim to conduct a systematic study of the training data management infrastructure for data transformation, from structured but still raw data to processable informative data, by leveraging affordable human effort and computational resources. The main focus of this thesis is to build a prototype of training data management system and address the main challenges during its realization.

1.2 Research Challenges in Training Data Management

Generally, to get processable informative data, there are two main tasks in the training data management. One is to increase the information value in the data, mainly by obtaining extra supervised information from data annotation. The other one is to compress and reorganize the data to a processable form without losing much information inside the original data, which typically includes representing, compressing, indexing and sampling the data to increase the computational efficiency. Among these two tasks, two costs are entailed, namely human effort and computational resources. The main research challenge is to reduce the cost to complete the training data management tasks to an affordable level.

1.2.1 Cost-sensitive Human Annotation

To reduce the cost involved in human annotation, voluminous research works have been conducted. In general, there are mainly two categories of methods. One is active learning, which aims to only annotate those important data instances and thereby the total number of annotated data required by the model to reach a certain performance level is significantly reduced. The other category of methods is crowdsourcing, which aims to reduce the cost per annotated data instance by getting the annotated data from the crowd rather than employed experts. These two categories of methods emphasize on different aspects, and are orthogonal to each other. In fact, the combination of them is a natural trend in the era of Big Data – using active learning to select the data to be annotated, and crowdsourcing the annotation task. However, in current status these two methods have limitations in real applications.

First, from the view of active learning, the supervised information in some complex Big Data analytics tasks may be hard to be quantified by human. Most easy annotating tasks can usually be well recognized by simply using machine efforts. The required tasks for human annotation are usually complex jobs such as describing a look or an emotion,, which often involve relative comparison or deep understanding. The results of such tasks usually can only be represented in the forms of ranking or relatively comparison among examples rather than a quantified label for every example. All these categories of supervised information can hardly be transformed to quantified labeled data and integrated to the existing active learning framework.

Second, the quality of crowdsourcing results is far from that of data labeled by domain experts. Naturally, human workers in crowdsourcing solve problems based on their knowledge, experience and perception, and may fail to give correct answers to complex problems. However, most analytics models are not robust to contaminated data, and hence directly using crowdsourced result is inadequate. It is still not clear how to detect and manage the quality of crowdsourced data and fuse them to get trustable results for further use or exploitation in Big Data analytics.

1.2.2 Computational Efficient Data Processing

There are also lots of works focusing on making the training data to be more processable, and hence reduce the computational cost during training data processing. The most common data structure in Big Data analytics is high-dimensional numerical data. Most other forms of data are transformed to high-dimensional numerical (or its simplest form – binary) data or its sparse representation for storage and model learning. Training data needs to be stored, indexed and automatically sampled to a processable form to support various data access operations from analytics model efficiently.

The storage of training data usually involves data representation, for the purpose of reducing data volume and selecting the most valuable features for analytics tasks. Hashing and compressing are two of the most popular schemes for data representation. Most existing methods focus on preserving internal information inside each high-dimensional data. However, there exist relations among data instances, which form a manifold space where the data instances reside. Such information is reflected in the similarity relations, and is valuable for data clustering, data sampling, casual discovery and various analytics models. Therefore, during data representation, not only the internal information inside each data instance, but also the external similarity relations among instances, are needed to be preserved.

The index of training data is also critical to support similarity search queries, where we are confronting the curse of dimensionality. Most of the access methods in the multi-dimensional space demonstrate poor performance when the number of dimensions is high. Locality Sensitive Hashing (LSH) and its variants, are generally believed to be the most effective search methods in high-dimensional spaces. However, LSH is designed to find points within a specified radius (i.e., to perform a radius search). In many cases, we are more interested in the relative similarity measure, i.e., the k-nearest neighbor relations, while the density of data at different locations may differ greatly and there is no specific fixed radius which can reflect such similarity relations. In short, there is still a gap between efficient k-NN search and the fixed radius search.

Moreover, most popular algorithms for model training are performed iteratively. Due to the humongous volume of the data, we typically can only process a fraction of the training data in each iteration because of the computational cost and the memory/cache usage. Therefore, sampling is also a critical operation that needs to be carefully designed in training data processing. The principle of sampling is to reflect the reality of the data without any bias, and expand the coverage and deliver more valuable samples. Currently, data are either uniformly sampled or sequentially accessed. However, it is still unknown how the data access pattern can affect model training.

1.3 Thesis Objectives and Contributions

To address all the challenges described in the previous subsection, it is important to conduct a comprehensive study on the cost-sensitive strategies of managing and transforming training data. In order to reduce the cost to support various analytics applications, there are five main issues to be addressed:

- Adapt the active learning based label selection methods to the scenario where supervised information is difficult to be directly quantified, and design interactive methods for more complex label format.
- Measure the quality of crowdsourced results, and design a cost-sensitive method to make the right trade-off between the quality of result and the expenses for human efforts.

- Transform the training data into better compact representation, while preserving both internal information in each instances and external relations among those instances.
- Index high-dimensional data to support similarity queries based on the distance independent k-nearest neighbor measure.
- Study the effect of data sampling pattern on the efficiency of analytics model training, and design suitable sampling operator to process the training data during model learning.

This thesis aims to solve the above five challenging issues in training data management. The main contributions of this thesis will be presented in the following subsections.

1.3.1 Preference Quantified Active Learning for Complex Human Annotation

The supervised information in some complex Big Data analytics tasks may be hard to be quantified via simply labeling. The typical required tasks for human annotation are usually complex jobs such as describing beautifulness or emotion, which may require relative comparison or deep understanding. The results collected from such tasks are usually in forms of rank or similarity comparison among examples rather than a quantified label for every example. All these categories of supervised information can hardly be transformed into quantified labeled data and integrated to the existing active learning framework.

We quantify these categories of information as a relative preference (i.e. a latent preference vector) learned by only pairwise comparisons on a set of entities with multiple attributes. We formalize the problem into two subproblems, namely preference estimation and comparison selection. We propose a novel approach to estimate the preference and introduce a binary search strategy to adaptively select the comparisons. We integrate these components into a system for inferring the preference using adaptive pairwise comparisons. The experiments on user preference demonstrate that our adaptive system significantly outperforms the naive random selection system on both real data and synthetic data.

1.3.2 Online Bayesian Model for Crowdsourcing Quality Management

Crowdsourcing has provided a means for solving many challenging problems by leveraging human intelligence. For example, applications such as image tagging, natural language processing, and semantic-based information retrieval can exploit crowd-based human computation to supplement existing computational algorithms. Since human workers in crowdsourcing solve problems based on their knowledge, experience, and perception, it is not clear which problems can be better solved by crowdsourcing than solving solely using traditional machine-based methods. In order to apply crowdsourcing as the data annotation methods and leverage crowdsourced results to generate more informative data, there is a great demand for measuring the quality of crowdsourced results, and designing a cost-sensitive method to make the right trade-off between the quality of result and the expenses for human efforts.

We design and implement a cost-sensitive method for crowdsourcing. We dynamically estimate the profit of the crowdsourcing job so that those questions with no future profit from crowdsourcing can be terminated. Two models are proposed to estimate the profit of crowdsourcing job, namely the linear value model and the generalized non-linear model. Based on these models, the expected profit of obtaining new answers for a specific question is computed based on the answers already received. A question is terminated in real time if the marginal expected profit of obtaining more answers is not positive. We extend the method to publish questions in a batch manner. We evaluate the effectiveness of our proposed method using two real world jobs on the Amazon Mechanical Turk (AMT) crowdsourcing platform. The experimental results show that our proposed method outperforms all the state-of-art methods.

1.3.3 Similarity Preserved Hashing for Data Representation

To reduce the data volume and selecting most important features, training data are usually represented by hash or quantization codes. Most existing methods focus on preserving internal information inside each high-dimensional data. However, the similarity relations between data instances also play an important role in lots of analytics tasks such as data clustering, data sampling, casual discovery and etc. Therefore, during data representation, similarity relations are expected to be preserved, and similarity search in the data representation should be accurate and efficient.

Locality Sensitive Hashing (LSH) has been widely accepted as an effective hash method for high-dimensional similarity search. However, data are typically not distributed uniformly over the space, and as a result, the buckets of LSH are unbalanced, causing the performance of LSH to degrade. We propose a new and efficient method called *Data Sensitive Hashing* (DSH) to address this drawback. DSH improves the hashing functions and hashing family by creating a set of hash codes via boosting. DSH leverages data distributions and is capable of directly preserving the nearest neighbor relations. We provide the theoretical guarantee of DSH, and demonstrate its efficiency experimentally.

1.3.4 Distance Independent Approach for Similarity Index

Many applications involve locating the k nearest neighbors (k-NN) of a given query point in a multi-dimensional space. The k-NN distance of different query points may differ greatly, especially in skewed datasets, which may lead to poor performance for LSH. Several learning-based methods are proposed to improve the performance of LSH by leveraging the knowledge of data distribution in the whole dataset. However, since the k-NN distance of a certain point is a *local* feature, the benefit of optimizing *global* hashing function is limited. In short, there is still a gap between efficient k-NN search and fixed radius search.

To close this gap, we propose a novel indexing scheme called *Selective Hashing*. In selective hashing, we aim to find the best index structure for each data point *locally*. The main idea is to create a disjoint set of indices with different granularities, and to store each point only in the most effective index. Intuitively, the index of each point is selected based on its local density, and the search range of query is automatically tuned based on its k-NN distance. Theoretically, we show that k-NN search using selective hashing can achieve the same quality as a fixed radius LSH search, using a radius equal to the distance of the $c_1 k_{th}$ nearest neighbor, with at most c_2 times overhead, where c_1 and c_2 are small constants. Selective hashing is also easy to build and update, and outperforms all the state-of-the-art algorithms.

1.3.5 Learning-value based Sampling for Effective Data Access

Recent years have witnessed amazing outcomes from “Big Models” trained using “Big Data”. Most model trainings are performed iteratively. Due to the huge volume of data, we usually can only afford to process a fraction of the training data in each iteration. Typically, the data are either uniformly sampled or sequentially accessed.

As in other data processing problems, data access pattern affects model training. Therefore, based on the stochastic gradient descent (SGD) framework, we propose a novel approach called ActiveSampler, where training data with more “learning value” to the model are sampled more frequently. The goal is to focus training effort on valuable instances near the classification boundaries, rather than evident cases, noisy data or outliers. We formalize the information gain of each training instance and develop a light-weight vectorized algorithm to accelerate the training process. Extensive experimental evaluations demonstrate that ActiveSampler can speed up the training procedure of SVM, feature selection and deep learning by 1.8-2.3x, for comparable training quality.

1.4 Synopsis

The remainder of this thesis is organized as follows. In Chapter 2, we first review existing techniques that are related to training data management. Then we present our approach about preference quantified active learning for complex human annotation task in Chapter 3, and online Bayesian model for crowdsourcing quality management task in Chapter 4. Chapter 5 and Chapter 6 discuss the data representation and similarity indexing. In Chapter 7, we study the effect of data access pattern to the efficiency of analytics model training. Finally, we conclude this thesis in Chapter 8.

CHAPTER 2

LITERATURE REVIEW

Various techniques have been proposed to address the challenges in training data management. In this chapter, we review the techniques that are closely related to this thesis. In particular, we first introduce the existing methods in cost-sensitive human annotation: data annotation for complex tasks and quality management for crowdsourcing are specific discussed under this theme. We then report the advances in computational efficient data processing: hashing methods to preserve similarity relations, indexing methods for similarity search and sampling methods for model training are specific discussed.

2.1 Data Annotation for Complex Tasks

The supervised information in some complex Big Data analytics tasks may be hard to be quantified via simply labeling. We quantify these categories of information as a relative preference (i.e. a latent preference vector) learned by only pairwise comparisons on a set of entities with multiple attributes. In this subsection, we review the literatures about annotating a preference vector.

Preference quantified data annotation is closely related to the general concept of active learning [91] in the machine learning community, where training points are actively selected by the training algorithm. The key idea of active learning is that learning algorithms can achieve greater accuracy with fewer training labels if it can choose the data from which it learns. The general solutions of active learning include reducing the uncertainty in training model [57], differentiating hypotheses which are consistent with the current learning set [94, 73, 103] (i.e. Query-By-Committee), maximizing the expected model change after receiving a new sample [93], minimizing the expectation [84] or variance [20] of the empirical loss, maximizing the information density among the whole query space [93] and etc.

Compare with active learning, preference ranking also asks comparisons as training instances. Most of the solutions in active learning area can be adapted into preference ranking. However, their training targets are different. Most active learning algorithms focus on detecting the exact border (i.e. hyperplane/hyper-surface) for classification purpose, while preference ranking aiming to find the most sensitive direction (i.e. a vector) for a user. In this thesis, we mainly focus on adapting the query-by-committee framework in active learning to solve the preference ranking problem. There is also a large amount of literature on *learning to rank*, which trains statistical models for ranking tasks [59, 58]. Recent approaches for learning to rank with pairwise comparisons include Active Ranking [45, 118, 29], RankNet [14], IR SVM [16] and LambdaRank [15]. Active Ranking also models the entities as vectors in $\mathbb{R}^d$, but the ranking is defined on the relative difference between these entities and a common reference point in $\mathbb{R}^d$. In contrast, preference quantified data annotation is learned by examining the comparison vector and pruning the possible preference hyperplane. In addition, active ranking only considers *ambiguous* comparisons and proposes an approach to learn the ranking from these comparisons. However, the work neither guarantees the quality of the selected comparison nor illustrates how to select them. Using active learning to annotate preference vector, we need to design specific algorithms to select the next comparison that is guaranteed to cut the remaining sphere as equally as possible.

Collaborative filtering [100] is also popular for learning preference, especially user preferences. Collaborative filtering assumes that similar users have similar preferences and similar items have similar ratings [100]. It can thus use other similar users' preferences to estimate a given user's preference, or use the ratings of the observed items to predict the unobserved ratings. Specifically, several algorithms were proposed to predict items preferred by individual users with a set of pairwise comparisons. Preference quantified data annotation assumes no prior knowledge from other users, as the preference here is only a general abstraction for those features that can be learned via comparisons. Therefore, the usage of collaborative filtering methods on preference quantified data annotation is limited.

There is also a recent trend for crowdsourced ranking [19, 116, 74, 101]. However, most of them focus on learning a single ranking function across the population, while preference quantified data annotation assumes different preferences among individuals. Yi et al. studies crowdsourced ranking where a different ranking list is learnt for an individual user [116]. However, it assumes no explicit feature representation of items with a small number of underlying intrinsic ranking functions. Preference quantified data annotation should also leverage crowdsourcing to complete comparison tasks, and the quality of crowdsourced data should be managed by crowdsourcing component in this thesis.

2.2 Quality Management for Crowdsourcing

Crowdsourcing has been widely used to solve challenging problems by human intelligence in comprehensive areas. In crowdsourcing systems, complex and difficult problems are partitioned to simple tasks. These tasks are assigned to several workers. The crowdsourcing system collects and integrates the answers from the workers as the results of the crowdsourcing jobs. Kitter et al. [52] studied the user behaviour in micro-task markets to show that user performs different behaviours.

Recently, crowdsourcing has been adopted and applied in several research areas such as database researches, machine learning and information retrieval. CrowdDB [26, 27], Qurk [68, 69] and TurkDB [78] designed three databases that are incorporated with crowdsourcing systems. These three databases allow queries to be partially answered through the AMT system. Selke et al. [90] expanded database schemas with additional attributes through querying the crowdsourcing systems. CrowdER [106] applied crowdsourcing to find the matching entities. CDAS [64] proposed a quality-sensitive answering model to manage the crowdsourcing tasks. In [64], a quality sensitive answering model for CDAS is proposed to manage the crowdsourcing tasks. In [82], Raykar et al. discussed the method of applying crowdsourcing in supervised learning without absolute golden standard. In [33], Guo et al. proposed a method to find the maximum element in a crowdsourcing database. Alonso et al. [2] developed a crowdsourcing based relevance evaluation method for information retrieval while Kazai et al. [50] proposed a crowdsourcing based book search evaluation method. Ipeirotis et al. [43] designed an approach to rank the workers by quality. Welinder et al. [112] proposed a crowdsourcing based online algorithm to find the ground truth. Crowdsourcing techniques has also been applied on other database based applications, such as graph search [79].

CrowdScreen [80] is designed to improve the accuracy and reduce the cost of binary choice problems in crowdsourcing systems by using a probabilistic method. Three facts limit the usability of CrowdScreen in real crowdsourcing data quality management: (1) the actual profit of the crowdsourcing job should be affected by three factors, namely the value of the questions, the risk of obtaining an incorrect answer and the cost of assigning questions to workers; the profit can be a linear or non-linear function given the accuracy of result; (2) different questions can different difficulties; the accuracy of each individual answer of a question is actually a random variable instead of a fixed value; (3) the quality management algorithm should be robust to the given crowdsourcing tasks and give accurate estimation of the quality of results in any cases. Therefore, there is a critical demand to build a quality management strategy for crowdsourcing to address the above issues.

2.3 Hashing Methods to Preserve Similarity Relations

There are lots of data representation [62, 85, 8, 76, 36] methods that focus on optimizing the hashing function to give a short but informative quantification for each data point. However, most existing methods only focus on preserving internal information inside each high-dimensional data. During data representation, similarity relations are expected to be preserved, and similarity search in the data representation should be accurate and efficient. Locality Sensitive Hashing [32, 21] (LSH) has been widely accepted as an effective hash method to preserve similarity relations in high-dimensional space. Here we review the LSH detailedly and analyze its insights and limitations.

LSH is an efficient approximate hashing method to preserve similarity relations in high-dimensional space. It is efficient and provides a rigorous quality guarantee for using the hash code to find similar points within a distance r. LSH leverages a family of functions where each function hashes the points in such a way that the possibility of collision is higher for similar points than for dissimilar points. Formally, an LSH family can be defined as follows [21]:

Definition 2.1. (LSH Family, $\mathcal{H}$) *A family $\mathcal{H} = \{h : R^d \rightarrow U\}$ of functions is called (r, cr, p_1, p_2)-sensitive if for any $p, q \in R^d$*

- *if $p \in B(q, r)$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \geq p_1$;*
- *if $p \notin B(q, cr)$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \leq p_2$;*

The family of hash functions are generated through the use of random projections – the intuition is that points that are nearby in the space will also be nearby in all projections. While two distance points can also be close in all projections, the possibility is extremely small if enough number of projections is taken. Usually, the difference between p_1 and p_2 is not enough to be used directly. To enlarge the difference, a concatenation of LSH functions is applied to generate a hash key for each point in R^d.

Definition 2.2. (Concatenation LSH Functions, $\mathcal{G}$) *A set of concatenation LSH functions $\mathcal{G} = \{g : R^d \rightarrow U^m\}$. Each $g_i \in \mathcal{G}$ consists of a sequence of m hash functions randomly extracted from $\mathcal{H}$. Formally,*

$$g_i(p) = (h_{i_1}(p), ..., h_{i_m}(p)),$$

where m is the number of hash functions in each concatenation and $h_{i_1}, ..., h_{i_m}$ are randomly selected from the LSH Family $\mathcal{H}$.

LSH applies these concatenation LSH functions to construct the hash tables. As a result,

- $\forall p \in B(q, r)$, $Pr(g(p) = g(q)) \geq p_1^m$
- $\forall p \in R^d n B(q, cr)$, $Pr(g(p) = g(q)) \leq p_2^m$

Further, the expected number of points in O that collide with q but are outside the ball $B(q, cr)$ is less than $p_2^m * |O|$. However, the recall for one hash table, p_1^m, is not large. To raise the overall recall, LSH typically applies l concatenation LSH functions and constructs l hash tables. Each concatenation LSH function randomly chooses m functions in $\mathcal{H}$. When the number of functions in $\mathcal{H}$ is large enough, the hash results of different concatenation functions can be regarded as independent. Thus, for any $o \in B(q, r)$, the possibility that o collides with q in at least one hash table is at least $1 - (1 - p_1^m)^l$, which is very close to 1. To summarize, LSH answers c-approximate r-NN problem as follows:

1. **Pre-processing:** LSH maintains l hash tables, and each hash table is attached with a concatenation hash function. Each hash table applies its concatenation hash function to hash the points in O, where each concatenation hash function consists of m hash functions randomly selected from the hashing family $\mathcal{H}$;

2. **Query processing:** Given a query object q, LSH finds c-approximate r-NN by examining points that collide with q in each of the l hash tables. In particular, the expected number of objects outside the $B(q, cr)$ is limited by $l * p_2^m * |O|$. This property guarantees its query efficiency.

While LSH has been shown to be very effective and with good theoretical guarantee in preserving similarity relations high-dimensional space, using LSH as the hashing methods to preserve K-NN relations, however, is not suitable. We should note that the similarity relation defined in LSH is related to a specified radius r, and an approximation allowance factor c. For a k-NN relation, the corresponding radius for different query points may vary by orders of magnitude. In such a case, LSH has to either (i) be run repeatedly with different values of r, cr,c^2r ... which leads to substantial increase in the query time and index storage cost, or (ii) use an ad-hoc r which leads to low quality guarantee. Further, LSH implicitly splits the whole space into lattices so that points in the same lattice cells are hashed into the same bucket. As a result, for real data distributions that are non-uniform, the hashing results are often unbalanced. We find that an unbalanced hashing leads to performance degradation of LSH. The reasons are two-fold. First, for the case that the number of points in one bucket is too small, LSH cannot find c-approximate r-NN and a further examination on a larger r is required; second, for the case that too many points collide in one bucket, LSH needs to examine each of these and so its performance suffers.

To preserve K-NN relations in the hashing codes, the requirements in the definition of LSH Family should be modified to reflect K-NN relations rather than a fixed radius. the definition of the hashing family and how it should be realized, remains a research gap.

2.4 Indexing Methods for Similarity Search

There is extensive work on high-dimensional indexes for k-NN queries, and there are surveys [18, 37] that provide good literature on the indexes. Access methods based on R-tree [34], K-d tree [9], and SR-tree [49] work nicely when the number of dimensions does not exceed 10 [110]. iDistance [117, 44] based on B^+-tree shows its superior performance when the number of dimensions does not exceed 30. However, they all suffer from the so-called curse of dimensionality and are eventually outperformed by a simple sequential scan when the number of dimensions is high. VA-file [110] designs an approximation-based indexing data structure and applies it to compress the input dataset. Based on this index, VA-file can answer k-NN queries by reducing the I/O cost to 1/8 to 1/4 time smaller than the input file. However, the computation cost of VA-file still grows linearly with the cardinality of the dataset since it is imperative to compute the distance between the approximation of each point and that of the query point. A more detailed survey about finding exact k-NN can be found in [18, 37]. To the best of our knowledge, the cost for existing approaches to finding exact k-NN grows at least linearly with the cardinality of the dataset when the number of dimensions is high.

Since we are concerned with a specific well known indexing mechanism, namely LSH, we focus on work related to it (the comparison among LSH-based indexes and other indexes, such as iDistance [117], can be found in [102]). Approximate k-NN provides approximate answers with acceptable error while constraining the growth of the cost sub-linearly with respect to the cardinality of the dataset. Locality sensitive hashing (LSH) methods [32, 21] are the best known approaches for approximate nearest neighbor search. The Bayesian LSH [86] improve the performance of LSH by using some novel query strategies. There are also lots of methods [66, 48, 77, 46] focusing on exploiting more buckets in a table to search far away k-NNs. However, radius is the decisive parameter in LSH, and those points far away from the radius can hardly be retrieved [21]. So far, there has been increasing attention focused on proposing data-sensitive hashing structures to solve k-NN search problems. The LSB-tree [102] leverages the intrinsic features of the B-tree to give a data-sensitive solution, with an approximation factor of only 2. In the design of hashing families, query-sensitive embeddings(QSE)[6, 5] and distance-based hashing(DBH)[7] have been proposed as the hash family for embedding and non-metric space, and [95, 55] were proposed for distance measure in the manifold. These works illustrate the potential for hashing family improvement. There are also other learning-based methods [62, 41, 72, 107, 36] aim to optimize the hashing functions. Although these methods are effective to catch most *global* distribution trends, they fail to catch specific *local* patterns such as dense groups or sparse hulls. Thus, there is no theoretical guarantee about their performance (neither efficiency or accuracy). There are also some data representation [85, 8, 76] methods that focus on optimizing the hashing function based on the data distribution. They significantly outperform LSH-based methods on precision. However, for the similarity search problem, we place much more emphasis on recall. This is because the recall affects the quality of the final result, while precision only affects performance by determining how many points have to be checked.

Partial indexing [98] in Postgres [99] considers the effectiveness of the index for each data object. In partial indexing, the index method chosen to be used is still based on the query. Thus, its starting point is to build a set of small but effective partial index to facilitate a wider range of query tasks. However, to use such idea to prune those indexes with inadequate granularities is not possible. This is because the k-NN query does not indicate what the search distance and there is no way to pick the most effective index based on query point. Therefore, to reduce the storage cost of a set of indexes for similarity search, the index to be accessed for a certain data object should depends on the data point itself, but not the query.

2.5 Sampling Methods for Model Training

Complex machine learning models, such as large-scale linear methods[96], feature selection [63] or deep learning [22], are widely adopted in Big Data analytics. Due to the huge size of both model and data, how to train these model

Algorithm	$f_{\mathbf{w}}(\mathbf{x})$	$L(f_{\mathbf{w}}(\mathbf{x}), y)$	$\rho_f(\mathbf{w})$
Linear Regression	$\mathbf{w^T x}$	$(y - f_{\mathbf{w}}(\mathbf{x}))^2$	0 or $\lambda\|\mathbf{w}\|_2^2$
Hinge-loss SVM	$\mathbf{w^T x}$	$\max(0, 1 - f_{\mathbf{w}}(\mathbf{x}) * y)$	0 or $\lambda\|\mathbf{w}\|_2^2$
Logistic Regression	$\mathbf{w^T x}$	$\log(1 + \exp(-f_{\mathbf{w}}(\mathbf{x}) * y))$	0 or $\lambda\|\mathbf{w}\|_2^2$
Feature Selection	$\mathbf{w^T x}$	$\log(1 + \exp(-f_{\mathbf{w}}(\mathbf{x}) * y))$	$\lambda\|\mathbf{w}\|_1$
Neural Network	complex	$\log(1 + \exp(-f_{\mathbf{w}}(\mathbf{x}) * y))$	0 or $\lambda\|\mathbf{w}\|_2^2$
PCA	$\mathbf{ww}^T\mathbf{x}$ - $\mathbf{x}$	$\|f_{\mathbf{w}}(\mathbf{x})\|_2^2$	0

Table 2.1: Examples of ERM Applications

efficiently is a challenging topic, and the solution requires efforts from learning, database, and system communities. Many optimizations have been proposed from a system's perspective for specific classes of models [119, 120, 56, 114, 22, 25]. Most of these algorithms (and many others) can fit into an Empirical Risk Minimization [104] (ERM) framework, for which we aim to develop a more general accelerator.

Empirical Risk Minimization (ERM) is a principle in the statistical learning theory which forms the basis for defining a family of analytics models. From the view of ERM, the central idea in machine learning is to learn a model and use it to approximate the data. The difference between the approximation and the real data is then measured by a loss function, which should be minimized by tuning the parameters of the model. Without loss of generality, in this work we formalize ERM from the supervised learning perspective, where each training instance is a pair $\langle \mathbf{x}, y \rangle$ consisting of content $\mathbf{x}$ and label y. For unsupervised problems, label y is a null term $\emptyset$, and data can be represented as $\langle \mathbf{x}, \emptyset \rangle$.

Definition 1 (Loss Function). *Given a data instance represented as $\langle \mathbf{x}, y \rangle$, and a model hypothesis $f_{\mathbf{w}}$ (i.e. a model f with parameter $\mathbf{w}$), the loss function $L(f_{\mathbf{w}}(\mathbf{x}), y)$ is a disagreement measure function between the model approximation (i.e. prediction) $f_{\mathbf{w}}(\mathbf{x})$ and the actual label y.*

After defining the measure of disagreement between the model output and the actual label, the ultimate goal is naturally to minimize the total disagreement by tuning the model parameters. This is called *Risk Minimization* [104], which is defined as follows:

Definition 2 (Risk Minimization). *Let $P_{\langle \mathbf{x}, y \rangle}$ be the distribution of data, the risk associated with model hypothesis $f_{\mathbf{w}}$ is defined as the expectation of the loss* *for the potential data distribution:*

$$R(f_{\mathbf{w}}) = E[L(f_{\mathbf{w}}(\mathbf{x}), y)] = \int L(f_{\mathbf{w}}(\mathbf{x}), y)\ \mathrm{d}P_{\langle \mathbf{x}, y \rangle} \tag{2.1}$$

The goal of learning algorithms is to find the parameter $\mathbf{w}$ *that minimizes the risk:*

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, R(f_{\mathbf{w}}) \tag{2.2}$$

However, in general, $R(f_{\mathbf{w}})$ cannot be directly minimized since the exact latent data distribution $P_{\langle \mathbf{x}, y \rangle}$ is unknown. Instead, the common way is to use the distribution of training data to approximate $P_{\langle \mathbf{x}, y \rangle}$. Therefore, the *Empirical Risk* [104] is used as the optimization target.

Definition 3 (Empirical Risk)**.** *The empirical risk is defined as the average of loss on the training set* D *with* n *instances.*

$$R_{emp}(f_{\mathbf{w}}) = \frac{1}{n} \sum_{i} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i) \tag{2.3}$$

To simplify the notation, we use $L(\mathbf{w})$ *to denote* $R_{emp}(f_{\mathbf{w}})$*.*

According to the VC-dimension theory [104], the difference between real risk and empirical risk may be large when the model hypothesis $f_{\mathbf{w}}$ is too complex while the size n of training data is not large enough. To prevent such over-fitting problem, the empirical risk is often regularized to penalize the complexity of model $f_{\mathbf{w}}$.

Definition 4 (ERM with Regularization)**.** *The empirical risk with regularization is defined as the average of loss function on the training set, plus a penalty regularization term* $\rho_f(\mathbf{w})$ *based on the complexity of the model* $f_{\mathbf{w}}$*.*

$$R_{reg-emp}(f_{\mathbf{w}}) = L(\mathbf{w}) + \rho_f(\mathbf{w}) \tag{2.4}$$

In ERM with regularization, the goal of learning algorithm is to minimize the empirical risk with regularization, i.e.,

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, R_{reg-emp}(f_{\mathbf{w}}) \tag{2.5}$$

We list some examples of analytics models from ERM family in Table 2.1, and show their connections. For most classification problems, we use log-logistic loss function $\log(1 + \exp(-f_{\mathbf{w}}(\mathbf{x}) * y))$ or other alternatives to approximate the exact 0-1 loss. For most applications, we use the l_2-norm of parameter $\lambda\|\mathbf{w}\|_2^2$ as the regularization function. This is actually a Gaussian prior over the parameter distribution from the Bayesian view. For feature selection methods such as Lasso [63], the l_1-norm regularization $\lambda\|\mathbf{w}\|_1$ is used to select those sparse features.

The optimization of the general ERM is widely studied in machine learning community [104]. Generally, there are two classes of methods: first-order algorithms such as gradient descent [11], and second-order algorithms such as Newton method [23]. Although second-order algorithms typically have a much faster convergence rate, they require the Hessian matrix [13] of parameters, making them not practical for large-scale models where the number of parameter is huge. For similar reasons, batch gradient methods [113] are very expensive for large training datasets. Therefore, stochastic methods [83] are the most favored algorithm in recent large-scale machine learning applications.

Stochastic Gradient Descent [83] (SGD) is one of the most popular stochastic optimization methods, and the theoretical results are well studied in [75]. However, [105] has shown that the variance in stochastic gradient is the key factor limiting the convergence rate of SGD. Consequently, many SGD variants such as SAG [88], SVRG [47], S3DG [71] have been developed to reduce the variance in the stochastic gradient. The convergence rate of these variants has been greatly improved in both theory and practice in terms of the number of iterations required to reach a certain accuracy. However, the optimization cost of these methods are not negligible, causing the training cost per iteration to increase substantially.

To optimize the ERM problem described in Equation 2.5, batch gradient descent method is used to iteratively alter the parameter towards the fastest direction to minimize the objective function. By defining the step size using the learning rate η, batch gradient descent method uses the following updating rule to optimize the parameter:

$$\begin{aligned} \mathbf{w}_{new} &= \mathbf{w} - \eta\nabla_{\mathbf{w}}(L(\mathbf{w}) + \rho_f(\mathbf{w})) \\ &= \mathbf{w} - \eta\nabla_{\mathbf{w}}\rho_f(\mathbf{w}) - \frac{\eta}{n}\sum_i \nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i) \end{aligned} \tag{2.6}$$

As can be observed from Equation 2.6, we need to evaluate the gradients

Notation	Meaning
$\langle \mathbf{x}, y \rangle$	training instance
$\mathbf{w}$	model parameters
$f_{\mathbf{w}}(\mathbf{x})$	model prediction for data $\mathbf{x}$
$L(f_{\mathbf{w}}(\mathbf{x}), y)$	loss on instance $\langle \mathbf{x}, y \rangle$
$L(\mathbf{w})$	empirical risk
$\rho_f(\mathbf{w})$	regularization term
$\nabla_{\mathbf{w}}$	gradient operator
$\nabla_{\mathbf{w}} L(\mathbf{w})$	batch gradient
$g_i(\mathbf{w})$	stochastic gradient
p_i	sampling probability for $\langle \mathbf{x_i}, y_i \rangle$

Table 2.2: Common Notations

$\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)$ for all training instances at each step, making the computation cost of $\nabla_{\mathbf{w}} L(\mathbf{w})$ extremely expensive. To avoid this cost, stochastic gradient methods use an inexact gradient which is estimated from random samples.

Definition 5 (Stochastic Gradient Descent). *In stochastic gradient descent, the true gradient $\nabla_{\mathbf{w}} L(\mathbf{w})$ is approximated by a stochastic gradient $g_i(\mathbf{w})$.*

$$\mathbf{w}_{new} = \mathbf{w} - \eta \nabla_{\mathbf{w}} \rho_f(\mathbf{w}) - \eta g_i(\mathbf{w}) \tag{2.7}$$

Taking $g_i(\mathbf{w})$ as a random variable, the expectation of $g_i(\mathbf{w})$ should equal to the gradient of $L(\mathbf{w})$, i.e.

$$E_i[g_i(\mathbf{w})] = \nabla_{\mathbf{w}} L(\mathbf{w}) \tag{2.8}$$

In the standard SGD algorithm, $g_i(\mathbf{w})$ is obtained by simply evaluating the gradient at a random single instance $\langle \mathbf{x_i}, y_i \rangle$:

$$g_i(\mathbf{w}) = \nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i) \tag{2.9}$$

where i is randomly drawn from $\{1, ..., n\}$, and the sampling probability p_i for each instance $\langle \mathbf{x_i}, y_i \rangle$ is $1/n$.

Intuitively, the requirement $E_i[g_i(\mathbf{w})] = \nabla_{\mathbf{w}} L(\mathbf{w})$ is to guarantee that the SGD algorithms will converge at the optimal point [75], as the expectation of update in SGD will be a zero vector at the point where batch gradient descent algorithm converges.

$$\begin{aligned} E_i[g_i(\mathbf{w})] &= \sum_i p_i g_i(\mathbf{w}) \\ &= \sum_i \frac{1}{n} \nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i) \\ &= \nabla_{\mathbf{w}} \frac{1}{n} \sum_i L(f_{\mathbf{w}}(\mathbf{x_i}), y_i) \\ &= \nabla_{\mathbf{w}} L(\mathbf{w}) \end{aligned} \tag{2.10}$$

Although the training cost per iteration for SGD is extremely light-weight compared to the batch gradient algorithm, the main drawback of SGD is that $g_i(\mathbf{w})$ is not the exact $\nabla_{\mathbf{w}} L(\mathbf{w})$. Therefore, the direction of the stochastic gradient $g_i(\mathbf{w})$ differs from the optimal direction $\nabla_{\mathbf{w}} L(\mathbf{w})$. This phenomenon causes SGD algorithm to be less efficient and take more iterations to converge or reach a certain accuracy. In essence, there is a trade-off between the number of iterations required to reach a certain accuracy and the computational cost per iteration. Therefore, the main goal of optimizing SGD algorithm is to making $g_i(\mathbf{w})$ close to $\nabla_{\mathbf{w}} L(\mathbf{w})$ at each iteration, while keeping the computation of $g_i(\mathbf{w})$ light-weight. Table 2.2 lists most of the important notations used throughout this chapter.

Active learning [92, 81] is originally proposed to select a set of labeled training data to maximize the accuracy of model. [65] uses the idea of weighted sampling to maximize the information gain of active learning. However, it is still not clear how the learning value of samples affects the training efficiency. Compare with active learning, in the training of model optimization, all training data are already labeled, and the selection should aim to maximize the learning speed of a passive learning model. Therefore, the optimization target should be the number of visit times rather than the number of visit points. SGD with weighted sampling is also recently studied in [121] which uses importance sampling for variance reduction.

Training data sampling is also related to feature selection [35]. Both methods focus on finding the most improtant part from the training data. However, feature selection algorithms are working on selecting those most important columns (i.e. features), while training data sampling is working on selecting those most important rows (i.e. data tuples).

CHAPTER 3

PREFERENCE QUANTIFIED ACTIVE COMPLEX HUMAN ANNOTATION

The supervised information in some complex Big Data analytics tasks may be hard to be quantified via simply labeling. The typical required tasks for human annotation are usually complex jobs such as describing beautifulness or emotion, which may require relative comparison or deep understanding. The results collected from such tasks are usually in forms of rank or similarity comparison among examples rather than a quantified label for every example. All these categories of supervised information can hardly be transformed into quantified labeled data and integrated to the existing active learning framework.

In this chapter, we quantify these categories of information as a relative preference (i.e. a latent preference vector) learned by only pairwise comparisons on a set of entities with multiple attributes. We formalize the problem into two subproblems, namely preference estimation and comparison selection. We propose an innovative approach to estimate the preference and introduce a binary search strategy to adaptively select the comparisons. We integrate these components into a system for inferring the preference using adaptive pairwise comparisons. The experiments on user preference demonstrate that our adaptive system significantly outperforms the naive random selection system on both real data and synthetic data.

3.1 Introduction

The supervised information in some complex Big Data analytics tasks may be hard to be given label by using human efforts. Most easy annotating tasks can usually be well recognized by simply using machine efforts. The required tasks for human annotation are usually complex jobs such as describing beautifulness or emotion, which often involve relative comparison or deep understanding. The results of such tasks usually can only be represented in the forms of ranking or relatively comparison among examples rather than a quantified label for every example. All these categories of supervised information can hardly be transformed to quantified labeled data and integrated to the existing active learning framework.

We quantify these categories of information as a relative preference (i.e. a latent preference function), which is an ideal abstraction for ranking or relative comparison. All ranking or relative comparison meanings, can now be transformed to the comparison of the value of each entity under the specific latent preference function. The task of annotation is to acquire such preference from users of the annotating system. Annotators (i.e. user of the annotating system) often have to choose entities of interest from a large multi-attribute set. While most systems allow users to specify selection conditions on individual attributes to reduce the size of the set, users may not always be good at tightly specifying such conditions. Furthermore, even within acceptable ranges of values for each individual attribute, there may still be too many entities for users to examine individually to choose the ones that they prefer.

In cognitive psychology, the Thurstone's Law of Comparative Judgement indicates that pairwise comparison is a more effective way to learn a preference function than directly choosing the set of preferred entities or deriving the overall ranking, In fact, there is considerable literature on learning and ranking by pairwise comparison in the machine learning community [59]. Recent works in crowdsourced ranking [19, 116] also leverage pairwise comparisons.

In this chapter, we study the problem of preference learning by pairwise comparison on structured entities. Specifically, given a set of entities with associated attributes, we choose a set of entity pairs and ask the user which entity is more preferable in each pair. Based on the feedback we estimate the overall user preference function. Once a system learns a preference function for a user, it can use it in many ways. For example, imagine a travel site that knows your preferences and automatically ranks hotels in your preferred order; or a shopping site that sends you an email when a promotion on a particular camera makes it likely to be your top choice. It is easy to come up with many such uses, and developing these is outside the scope of this chapter, which is focused on preference learning.

For such a preference learning scheme to be practical, it must meet three conditions. First, pairwise comparisons should be easy to answer. Second, one should be able to learn at least an approximation of the preference function with only a few comparisons. And third, the system must be able to pose such comparison questions at interactive speed. The first condition is immediately satisfied in most applications of interest: given two object instances, a user is quickly able to determine which one is preferred. The remaining two conditions are challenging, and addressing them constitutes the bulk of the technical work that follows.

To summarize, this chapter has the following contributions:

- We formalize the problem of preference learning by pairwise comparison on structured data, and define two subproblems, *preference estimation* and *comparison selection* (Section 3.2.).
- We study the problem of preference estimation based on pairwise comparison feedback and propose a solution minimizing the maximum of potential error. Then we find the optimization problem can be easily solved by typical SVM by doing reductions (Section 3.3).
- We develop a theoretical analysis of the comparison selection problem and introduce *adaptive comparison selection*, with effectiveness comparable to a theoretical optimum (Section 3.4).
- We implement a system integrating all these parts. We show that our system significantly outperforms the naïve system on both synthetic and real data, with either simulated or real user feedback. (Section 3.5).

3.2 Problem Formalization

We denote an entity by e and the set of all entities by $\mathbf{E}$. Without loss of generality, we define the preference to be a function $\mathcal{O} : \mathbf{E} \to \mathbb{R}$. Intuitively, we say e is *preferable* to e' if $\mathcal{O}(e) > \mathcal{O}(e')$, and vice versa. We assume no ties, and we assume transitivity. Given N input entities, our task is to learn the user preference function $\mathcal{O}$, which is not directly observable.

Definition 6 (Pairwise Comparison and Its Feedback)**.** *Given an entity set* $\mathbf{E}$ *and a preference* $\mathcal{O}$*, a* pairwise comparison *is an ordered pair* $\langle e, e' \rangle \in \mathbf{E} \times \mathbf{E}$*. Its* feedback *is a function* $\mathcal{C} : \mathbf{E} \times \mathbf{E} \to \{-1, 1\}$*, such that:*

$$\mathcal{C}(\langle e, e' \rangle) = \begin{cases} 1 & \textit{if } \mathcal{O}(e) > \mathcal{O}(e') \\ -1 & \textit{otherwise} \end{cases} \tag{3.1}$$

$\mathbf{D} \subseteq \mathbf{E} \times \mathbf{E}$ *is a set of pairwise comparisons, with feedback*

$$\mathcal{C}(\mathbf{D}) = \{\mathcal{C}(\langle e, e' \rangle) | \langle e, e' \rangle \in \mathbf{D}\} \tag{3.2}$$

Since $\mathcal{C}(\langle e, e' \rangle) = -\mathcal{C}(\langle e', e \rangle)$, we only need to consider one of the two. We define the *candidate comparisons* of $\mathbf{E}$ to be $\mathcal{D}(\mathbf{E}) = \{\langle e_i, e_j \rangle | e_i, e_j \in \mathbf{E}; i < j\}$.

Definition 7 (Preference Learning by Pairwise Comparison)**.** *Given a set of entities* $\mathbf{E}$*, an unobservable preference* $\mathcal{O}$*, choose a set of pairwise comparisons* $\mathbf{D} \subseteq \mathcal{D}(E)$*, and derive* $\mathcal{O}$ *based on* $\mathcal{C}(\mathbf{D})$*.*

While this problem definition is general, it can be impractical if the number of comparisons needed ($|\mathbf{D}|$) is large. Clearly, pairwise comparison of each pair is enough, requiring that the user labels $N(N-1)/2$ pairs. We can improve this quite a bit if we follow a standard sorting algorithm, in the worst case we need the user to label $N \cdot log(N)$ comparisons to derive the complete preference $\mathcal{O}$. Even in the best case, we need $N - 1$ comparisons to derive $\mathcal{O}$. Unfortunately, N can often be large in real scenarios. For instance, there may be hundreds of potential hotels and thousands of potential movies to consider. It is not reasonable to expect the user to label N pairs. In fact, one major motivation to learn a preference function is so that we can determine a user's preference for an item with only limited prior user input.

We model each entity as a structured tuple. Specifically, we assume the set of attributes is universal across the entity set, and denote it by A =

$\{\mathrm{A_1}, \mathrm{A_2}, ..., \mathrm{A_k}\}$. We use $e[\mathrm{A_i}]$ to denote the value on attribute $\mathrm{A_i}$ of entity e. For convenience, we further assume all these attribute values are numeric. (Categorical attributes can be mapped to numeric preference values, if need be). For a given e, we represent the entity by a k dimensional vector $\mathbf{e}$, where

$$\mathbf{e} = (e[A_1], e[A_2], ..., e[A_k])^T \tag{3.3}$$

Furthermore, for each e, we assume $\mathcal{O}(e)$ to be a linear combination of these attribute values. Specifically:

$$\mathcal{O}(e) = \mathbf{w} \cdot \mathbf{e} \tag{3.4}$$

where $\mathbf{w}$ is a k dimensional weight vector. Here we omit the constant offset since it has no impact on comparisons. Similarly, because the comparisons are independent of the magnitude of $\mathbf{w}$, we assume $\|\mathbf{w}\| = 1$. Note that, under these assumption, $\mathbf{w}$ uniquely characterizes $\mathcal{O}$. For this reason, we call $\mathbf{w}$ the *preference vector*.

In practice, we do not know that the preference function is linear. However, it is commonplace to build linear models as approximations of reality, and they often turn out to be good enough. If need be, individual attribute values can be scaled non-linearly, for example, through the use of a logarithm or some kernel function. Thus, linear regression is used widely, as are linear models in machine learning.

Having these assumptions, we now refine our preference learning problem.

Definition 8 (Preference Vector Learning by Pairwise Comparison)**.** *Given a set of entities* $\mathbf{E}$*, where each entity* e *is a* k*-dimensional vector, and a preference function* $\mathcal{O}(e) = \mathbf{w} \cdot \mathbf{e}$ *where* $\mathbf{w}$ *is a normalized* k*-dimensional vector not directly observable, select a set of pairwise comparisons* $\mathbf{D} \subseteq \mathcal{D}(\mathbf{E})$*, and estimate* $\mathbf{w}$ *based on* $\mathcal{C}(\mathbf{D})$*.*

Definition 8 comprises two subproblems: 1) select a subset of the candidate pairwise comparisons to ask the user for feedback and 2) estimate the preference vector $\mathbf{w}$ based on the feedback. We name the first subproblem **pairwise comparison selection** and the second **preference estimation**. We first discuss the preference estimation problem in Section 3.3, and then study the pairwise comparison selection problem in Section 3.4.

3.3 Preference Estimation

Suppose we have already selected a set of pairwise comparisons $\mathbf{D}$, in this section we focus on estimating the weight vector $\mathbf{w}$ based on the user feedback $\mathcal{C}(\mathbf{D})$. We first describe some intuitions about the preference estimation problem under the linear preference model, and then formalize the problem and propose an algorithm to estimate $\mathbf{w}$. We denote our *estimate* of $\mathbf{w}$ by $\hat{\mathbf{w}}$ and the *true value* of $\mathbf{w}$ by $\mathbf{w_0}$. To give a more intuitive geometric interpretation of this problem ,we begin with an assumption that the user makes no error in reporting pairwise comparisons. In other words, for every pair $\langle e, e' \rangle \in \mathbf{D}$, $\mathcal{C}(\langle e, e' \rangle) = 1$ if and only if $(\mathbf{e} - \mathbf{e}') \cdot \mathbf{w_0} > 0$. We will relax this assumption at the end of this section.

Recall that $\mathbf{w}$ is a k-dimensional vector and $\|\mathbf{w}\| = 1$, all $\mathbf{w}$ form a **hypersphere** $\mathbb{S}^k = \{\mathbf{w} | \|\mathbf{w}\| = 1\}$ in the k-dimensional space. Now, imagine we assign a pairwise comparison $\langle e, e' \rangle$ to the user. The user returns 1 if e is preferable to e', and -1 if e is less preferred than e'. In other words, we have:

$$\mathcal{C}(\langle e, e' \rangle) = \begin{cases} 1 & \text{if } \mathcal{O}(e) - \mathcal{O}(e') > 0 \\ -1 & \text{if } \mathcal{O}(e) - \mathcal{O}(e') < 0 \end{cases} \tag{3.5}$$

Which can be rewritten as[1]:

$$\mathcal{C}(\langle e, e' \rangle) \cdot (\mathcal{O}(e) - \mathcal{O}(e')) > 0 \tag{3.6}$$

By substituting Equation 3.4, we obtain:

$$\mathcal{C}(\langle e, e' \rangle) \cdot (\mathbf{e} - \mathbf{e}') \cdot \mathbf{w_0} > 0 \tag{3.7}$$

This implies the following geometric intuition. A pairwise comparison $\langle e, e' \rangle$ and its feedback $\mathcal{C}(\langle e, e' \rangle)$ uniquely identify a **hyperplane** with normal vector $\mathcal{C}(\langle e, e' \rangle) \cdot (\mathbf{e} - \mathbf{e}')$ in the k-dimensional vector space of $\mathbf{w}$. The hyperplane cuts $\mathbb{S}^k$ into two symmetric hemispheres, and $\mathbf{w_0}$ must be on the hemisphere $\{\mathbf{w} \mid \mathbf{w} \in \mathbb{S}^k, \mathcal{C}(\langle e, e' \rangle) \cdot (\mathbf{e} - \mathbf{e}') \cdot \mathbf{w} > 0\}$. In other words, each such hyperplane prunes half of $\mathbb{S}^k$ on which $\mathbf{w_0}$ could reside. For this reason, we call such a hyperplane the **pruning plane** and denote it by $\mathbb{P}_{\langle e, e' \rangle}$. Since the pruning is independent of the magnitude of the normal vector, we normalize it and define

[1]We use the dot operator to represent both scalar multiplication and vector inner product when the context is clear.

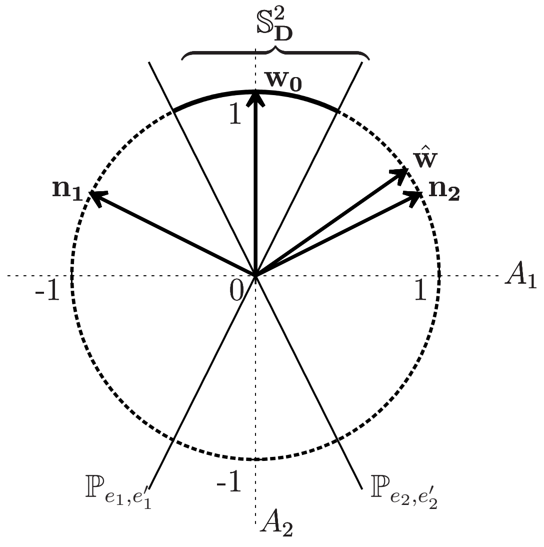


Figure 3.1: A Simple Preference Estimation in 2-D.

$\mathbf{n}_{\langle \mathbf{e},\mathbf{e}'\rangle} = \mathcal{C}(\langle e, e'\rangle) \cdot (\mathbf{e} - \mathbf{e}')/\|\mathbf{e} - \mathbf{e}'\|$. We further denote the hemisphere on the same side of $\mathbf{n}_{\langle \mathbf{e},\mathbf{e}'\rangle}$ by $\mathbb{S}^k_{\langle e,e'\rangle}$, and call it the **remaining hemisphere**. Formally, $\mathbb{S}^k_{\langle e,e'\rangle} = \{\mathbf{w}|\mathbf{n}_{\langle \mathbf{e},\mathbf{e}'\rangle} \cdot \mathbf{w} > 0, \mathbf{w} \in \mathbb{S}^k\}$.

Intuitively, the more pairwise comparisons feedback we obtain, the fewer possible values remain for $\mathbf{w}$. Specifically, given a set of pairwise comparisons $\mathbf{D}$, the remaining possible values for $\mathbf{w}$ can be characterized by the intersection of all the remaining hemispheres $\bigcup_{\langle e,e'\rangle \in \mathbf{D}} \mathbb{S}^k_{\langle e,e'\rangle}$, which forms a *spherical polygon* in k dimensions. We denote this spherical polygon by $\mathbb{S}^k_{\mathbf{D}}$ and simply call it the **remaining sphere**.

Under our error-free assumption, $\mathbf{w_0}$ must exist in the remaining sphere $\mathbb{S}^k_{\mathbf{D}}$. Therefore, we should estimate $\mathbf{w}$ within $\mathbb{S}^k_{\mathbf{D}}$. The following example illustrates the intuition in two dimensions.[2]

Example 1 (Simple 2-D Estimation)**.** *Suppose our entities have only two attributes, A_1 and A_2. Furthermore, we assume an extreme case, where the preference is only determined by A_2. In other words, $\mathbf{w_0} = (0, 1)$. Suppose we have two comparisons, $\langle e_1, e_1'\rangle$ and $\langle e_2, e_2'\rangle$, where $\mathbf{e_1} - \mathbf{e_1}' = (2, 1)$ and $\mathbf{e_2} - \mathbf{e_2}' = (-2, 1)$. Under the error-free assumption, $\mathcal{C}(\langle e_1, e_1'\rangle) = \mathcal{C}(\langle e_2, e_2'\rangle) = 1$. As a*

[2]Hereafter, we will omit the subscript $\langle e, e'\rangle$ of $\mathbf{n}_{\langle \mathbf{e},\mathbf{e}'\rangle}$ for readibility when the context is clear.

result, $\mathbf{n_1} = (2,1)/\sqrt{5}$ *and* $\mathbf{n_2} = (-2,1)/\sqrt{5}$*. In 2-dimensional space, the two pruning planes corresponding to* $\mathbf{n_1}$ *and* $\mathbf{n_2}$ *degrade to two lines shown in Figure 3.1. The remaining sphere* $\mathbb{S}^2_{\mathbf{D}}$ *degrades to the bold arc on the unit circle. In this case, a reasonable estimate of* $\hat{\mathbf{w}}$ *lies in the middle of* $\mathbf{n_1}$ *and* $\mathbf{n_2}$*, which overlaps* $\mathbf{w_0}$*.*

The above example indicates that the estimate in the "middle" of the normal vectors of all pruning planes may be a "good" estimate. In order to formalize this intuition, we need to first define such "goodness".

Let us first examine the simple case, where the user submits a feedback $\mathcal{C}(\langle e, e'\rangle)$ to a pairwise comparison $\langle e, e'\rangle$. To recall, this creates a pruning plane $\mathbb{P}_{\langle e,e'\rangle}$ with normal vector $\mathbf{n}_{\langle \mathbf{e},\mathbf{e}'\rangle} = \mathcal{C}(\langle e, e'\rangle) \cdot (\mathbf{e} - \mathbf{e}')$. On one extreme, if $\mathbf{w_0}$ is orthogonal to $\mathbb{P}$ (parallel to $\mathbf{n}$), we can tell this is a very *straightforward* comparison. Because, the difference between $\mathbf{e}$ and $\mathbf{e}'$ is maximized by $\mathbf{w_0}$. Thus the user should have little difficulty in answering the comparison. On the other extreme, if $\mathbf{w_0}$ is almost parallel to $\mathbb{P}$ (orthogonal to $\mathbf{n}$), the comparison would be very *difficult*. Because the difference between $\mathbf{e}$ and $\mathbf{e}'$ is only slightly reflected by the projection on $\mathbf{w_0}$. Intuitively, we prefer estimate $\mathbf{w}$ to be parallel to $\mathbf{n}$, since it tends to render the comparison more *confident* for the user. The notion of confidence is formalized below:

Definition 9 (Confidence)**.** *Given a pairwise comparison* $\langle e, e'\rangle$ *and an estimated weight vector* $\hat{\mathbf{w}}$*, the confidence of this estimate is* $\mathrm{C}^{\hat{\mathbf{w}}}_{\langle e,e'\rangle} = \mathbf{n}_{\langle \mathbf{e},\mathbf{e}'\rangle} \cdot \hat{\mathbf{w}}$*.*

Note that, the confidence must be positive according to the error-free assumption.

Informally, our goal is to maximize this confidence for all comparisons. If there is only one comparison, the estimate parallel to $\mathbf{n}$ is the optimal solution. In case of multiple comparisons, there are various ways to define the overall confidence. For example, we could define the overall confidence to be the sum of all individual confidences, which leads to the following problem definition:

Definition 10 (Preference Estimation by Total Confidence)**.** *Given a set of pairwise comparison* $\mathbf{D}$ *and the user feedback* $\mathcal{C}(\mathbf{D})$*, find a preference vector estimate* $\hat{\mathbf{w}}$*, such that* $\sum_{\langle e,e'\rangle \in \mathbf{D}} \mathrm{C}^{\hat{\mathbf{w}}}_{\langle e,e'\rangle}$ *is maximized.*

We note that $\hat{\mathbf{w}}$ may not always have $\mathrm{C}^{\hat{\mathbf{w}}}_{\langle e,e'\rangle} > 0$. For instance, in Example 1, if we continue to receive dozens of comparison feedbacks with normal vectors around $\mathbf{n_2}$, $\hat{\mathbf{w}}$ will be pulled to $\mathbf{n_2}$ and fall out of $\mathbb{S}^2_{\mathbf{D}}$, as shown in Figure 3.1.

We prefer a metric such that repetitive comparisons with same or similar normal vectors do *not* shift our final estimate. In the previous example, ideally, no matter how many comparisons overlap $\mathbf{n_1}$ or $\mathbf{n_2}$, a fair estimate of $\mathbf{w}$ should always point to the middle, the same as $\mathbf{w_0}$ in Figure 3.1.

In fact, if we have various additional comparisons with normal vectors between $\mathbf{n_1}$ and $\mathbf{n_2}$, they should contribute little to the estimate, because their ability to constrain $\mathbb{S}^2_{\mathbf{D}}$ is dominated by $\mathbf{n_1}$ and $\mathbf{n_2}$. In other words, the only comparisons that matter are those whose normal vectors are *most orthogonal* to $\mathbf{w_0}$. And according to our definition of confidence, these are exactly the comparisons the user is least confident about, which matches our intuition. In this case, our goal is to maximize those confidence values, which is formalized by the following revised problem statement:

Definition 11 (Preference Estimation by Maximal Least Confidence)**.** *Given a set of pairwise comparison* $\mathbf{D}$ *and the user feedback* $\mathcal{C}(\mathbf{D})$*, find a preference vector estimate* $\hat{\mathbf{w}}$*, such that* $\min_{\langle e,e'\rangle\in\mathbf{D}} \mathrm{C}^{\hat{\mathbf{w}}}_{\langle e,e'\rangle}$ *is maximized. Or formally, obtain:*

$$\begin{aligned} &\underset{\mathbf{w}}{\operatorname{argmax}} \min_{\langle e,e'\rangle\in\mathbf{D}} \mathrm{C}^{\mathbf{w}}_{\langle e,e'\rangle} \\ &\textit{subject to:} \quad \|\mathbf{w}\| = 1 \end{aligned} \tag{3.8}$$

Since the magnitude of $\mathbf{w}$ does not matter and only direction is the one we trying to solve, the above problem can be transformed into the following standard maximum margin optimization problem:

$$\begin{aligned} &\underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{w}\| \quad \text{subject to:} \\ &\forall\langle e,e'\rangle \in \mathbf{D}, \mathbf{n}_{\langle \mathbf{e},\mathbf{e'}\rangle} \cdot \mathbf{w} \geq 1 \end{aligned} \tag{3.9}$$

In this revised problem, the margin to be maximized is our minimal confidence value. Therefore, we can use a linear SVM to solve it efficiently. Specifically, for each comparison the user made, we create a positive example $(\mathbf{n}, 1)$ and a negative example $(-\mathbf{n}, -1)$ and call SVM to train all the support vectors. We then use the sum of these support vectors weighted on α as our estimate $\hat{\mathbf{w}}$.

In case the user feedback is error-prone, no $\mathbf{w}$ will yield a positive confidence margin. To solve this, we adopt the typical slack variant methods commonly used in SVM to enable error-tolerance. The problem is then revised as follows.

Definition 12 (Noise-tolerance Preference Estimation by Maximal Least Confidence)**.** *Given a set of pairwise comparison* $\mathbf{D}$ *and the user feedback* $\mathcal{C}(\mathbf{D})$*, find a preference vector estimate* $\hat{\mathbf{w}}$*, such that:*

$$\begin{aligned} &\operatorname*{argmin}_{\mathbf{w}} \|\mathbf{w}\| + C \sum \xi_{\langle e,e' \rangle} \quad \textit{subject to:} \\ &\forall \langle e,e' \rangle \in \mathbf{D}, \mathbf{n}_{\langle \mathbf{e},\mathbf{e'} \rangle} \cdot \mathbf{w} \geq 1 - \xi_{\langle e,e' \rangle} \end{aligned} \tag{3.10}$$

where C *is the slack variant indicates the tolerance level to user feedback error.*

3.4 Comparison Selection

So far we have discussed how to estimate the weight vector $\mathbf{w}$ given a set of pairwise comparisons $\mathbf{D}$. We now consider how these pairwise comparisons are selected. Admittedly, a randomly selected set of comparisons could still lead to a reasonably good estimate. The question is, can we do better?

To give a clear geometry interpretation, we keep the assumption that the user makes no error in answering the comparisons. Later, we will illustrate how errors are fixed when Noise-tolerance Preference Estimation are applied. To recall, each comparison $\langle e, e' \rangle$ with its feedback $\mathcal{C}(\langle e, e' \rangle)$ uniquely identifies a pruning plane $\mathbb{P}^k_{\langle e,e' \rangle}$, which prunes half of the k-dimensional sphere $\mathbb{S}^k$. These planes keep cutting and pruning, and finally restrict $\mathbf{w}$ to a spherical polygon $\mathbb{S}^k_{\mathbf{D}}$, which we call the remaining sphere.

Both $\mathbf{w_0}$ and our estimate $\hat{\mathbf{w}}$ lie on this remaining sphere. Obviously, the best estimate is the one that is close to $\mathbf{w_0}$. We define the **precision** of an estimate $\hat{\mathbf{w}}$ to be the inner product $\hat{\mathbf{w}} \cdot \mathbf{w_0}$. Intuitively, the higher the precision is, the better the estimate $\hat{\mathbf{w}}$ is. Given this definition, the worst precision we could have given $\mathbb{S}^k_{\mathbf{D}}$ and $\mathbf{w_0}$ is $\min_{\hat{\mathbf{w}} \in \mathbb{S}^k_{\mathbf{D}}} \hat{\mathbf{w}} \cdot \mathbf{w_0}$. Considering the fact that $\mathbf{w_0}$ can be anywhere inside $\mathbb{S}^k_{\mathbf{D}}$, the worst precision given only $\mathbb{S}^k_{\mathbf{D}}$ is $\min_{\mathbf{w_1}, \mathbf{w_2} \in \mathbb{S}^k_{\mathbf{D}}} \mathbf{w_1} \cdot \mathbf{w_2}$. We define the precision of $\mathbb{S}^k_{\mathbf{D}}$ by this worst precision and denote it by $\mathcal{P}(\mathbb{S}^k_{\mathbf{D}})$. We also define the angle between such $\mathbf{w_1}$ and $\mathbf{w_2}$ on which the worst precision is achieved to be the **diameter** of $\mathbb{S}^k_{\mathbf{D}}$ and denote it by $\mathcal{D}(\mathbb{S}^k_{\mathbf{D}})$. Since $\|\mathbf{w}\| = 1$, $\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}) = \cos(\mathcal{D}(\mathbb{S}^k_{\mathbf{D}}))$.

In other words, the diameter of the remaining sphere determines its precision. Intuitively, the more user comparisons we have, the smaller the remaining sphere is, and thus higher the precision we obtain. However, because each comparison places a user burden, the total number of cuts is limited in practice. As a result, we cannot indefinitely refine the remaining sphere to achieve arbitrarily high precision. Based on these observations, we formalize the comparison selection problem:

Definition 13 (Pairwise Comparison Selection)**.** *Given a set of entities* $\mathbf{E}$ *and an integer* M*, find a sequence of pairwise comparisons* $\mathbf{D} = \{\langle e_1, e'_1\rangle, ..., \langle e_M, e'_M\rangle\}$*, where* $\langle e_i, e'_i\rangle \in \mathcal{D}(\mathbf{E}), i \in 1..M$*, such that* $\mathcal{P}(\mathbb{S}^k_{\mathbf{D}})$ *is maximized.*

In the remainder of this section, we start with a naïve approach with random selection, and theoretically analyze it. We then propose a novel selection strategy that is theoretically optimal.

3.4.1 A Naïve Approach

To recall, each comparison identifies a pruning plane that cuts the k-sphere $\mathbb{S}^k$ across origin into two equal halves and prunes one of them according to the user feedback. Consequently, all the N comparisons will cut $\mathbb{S}^k$ into many small pieces, each of which is a spherical polygon. Under the error-free assumption, exactly one of these spherical polygons forms the remaining sphere $\mathbb{S}^k_{\mathbf{D}}$.

Here we analyze the best case average precision one can obtain after M randomly chosen cuts. Since $\mathbf{w_0}$ can fall into any of the remaining sphere, the best case is achieved when the sphere $\mathbb{S}^k$ is uniformly cut. Using $\mathcal{V}(\cdot)$ to denote volume, the best case average volume of the remaining sphere can be estimated as:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \frac{\mathcal{V}(\mathbb{S}^k)}{\mathcal{M}^k(M)} \tag{3.11}$$

where $\mathcal{M}^k(M)$ is the number of spherical polygons cut by M pruning planes in the k-dimensional space. According to the literature [38]:

$$\mathcal{M}^k(M) = 2 \cdot \left[\binom{M-1}{0} + \binom{M-1}{1} + ... + \binom{M-1}{k-1}\right] \tag{3.12}$$

Also, the volume of a k-sphere[3] with radius R is:

$$\mathrm{V}_k = 2 \cdot \frac{\pi^{k/2} R^{k-1}}{\Gamma(\frac{k}{2})} \tag{3.13}$$

Since $\mathbb{S}^k$ has radius one, its volume is:

$$\mathcal{V}(\mathbb{S}^k) = 2 \cdot \frac{\pi^{k/2}}{\Gamma(\frac{k}{2})} \tag{3.14}$$

By substituting Equation 3.14 into Equation 3.11, we have:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \frac{2}{\mathcal{M}^k(N)} \cdot \frac{\pi^{k/2}}{\Gamma(\frac{k}{2})} \tag{3.15}$$

On the other hand, given the remaining sphere with volume $\mathcal{V}(\mathbb{S}^k_{\mathbf{D}})$, the best precision is achieved when the remaining sphere is as evenly distributed as possible. When the remaining sphere is small enough, this can be approximated as a ball in the $k-1$ dimension. According to the literature, the volume of a k-ball with radius R is:

$$\mathrm{U}_k = \frac{\pi^{k/2} R^k}{\Gamma(1 + \frac{k}{2})} \tag{3.16}$$

Therefore, we have:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \mathrm{U}_{k-1} = \frac{\pi^{\frac{k-1}{2}} R^{k-1}}{\Gamma(\frac{k+1}{2})} \tag{3.17}$$

By substituting Equation 3.17 into Equation 3.15 and solving R, we have:

$$R = \left(\frac{2\sqrt{\pi}}{\mathcal{M}^k(M)} \cdot \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})} \right)^{\frac{1}{k-1}} \tag{3.18}$$

For a small remaining sphere $\mathbb{S}^k_{\mathbf{D}}$ shaped like a ball in the (k-1)-dimensional space, its diameter $\mathcal{D}(\mathbb{S}^k_{\mathbf{D}})$ can be estimated by $2R$. Therefore, its precision can be obtained by:

$$\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}) = \cos(2R) \tag{3.19}$$

[3] We define the k-sphere in the k-dimensional space instead of the $(k-1)$-dimensional space.

Substituting Equation 3.18 into Equation 3.19 results in:

$$\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}) = \cos\left(2 \cdot \left(\frac{2\sqrt{\pi}}{\mathcal{M}^k(M)} \cdot \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})}\right)^{\frac{1}{k-1}}\right) \tag{3.20}$$

As a final note for this section, since $\mathcal{M}^k(M)$ is upper bounded by $2 \cdot (M-1)^{k-1}$ for $M > 1$, we have:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) \geq \frac{\mathcal{V}(\mathbb{S}^k)}{2 \cdot (M-1)^{k-1}} = \frac{1}{(M-1)^{k-1}} \cdot \frac{\pi^{k/2}}{\Gamma(\frac{k}{2})} \tag{3.21}$$

for $M > 1$. That is to say, for any given dimension k, one cannot reduce the volume of the remaining sphere more than *polynomially* with respect to the number of comparisons using a random cutting strategy. And the volume is closely related to the precision of estimate. If we want to break this limit, a new cutting strategy has to be innovated.

3.4.2 Binary Cutting Strategy

In this section, we consider if we can perform any better than a random cutting strategy. Without loss of generality, we assume that given an arbitrary hyperplane, there always exists in the candidates a comparison whose pruning plane overlaps the given hyperplane. We will remove this assumption in Section 3.4.3.

If we have only one comparison to prepare, the problem is straightforward. Indeed, a random cut of the initial sphere $\mathbb{S}^k$ is the optimal choice. This is because, no matter what the user feedback is, the corresponding pruning plane will prune half of the sphere, or reduce half of the volume. However, if we continue to perform more cuts, their positions matter.

Example 2 (Cutting Strategy)**.** *We continue to illustrate the simple case in two-dimension, where $\mathbb{S}^2$ degrades to a unit circle, as shown in Figure 3.2. Suppose we first ask the user to compare entity e_1 and e_1'. This comparison identifies a pruning plane $\mathbb{P}_{\langle e_1, e_1' \rangle}$ with normal vector $\mathbf{e_1} - \mathbf{e_1}'$, as shown in Figure 3.2. We further assume the user feedback $\mathcal{C}(\langle e_1, e_1' \rangle)$ equals 1. Consequently, the plane prunes the bottom left half of the unit circle, leaving a remaining sphere highlighted by the bold arc $\widehat{QT}$.*

Now consider the next comparison. Assume there are two candidates: $\langle e_2, e_2' \rangle$ and $\langle e_3, e_3' \rangle$. The corresponding pruning planes with their normal vectors are

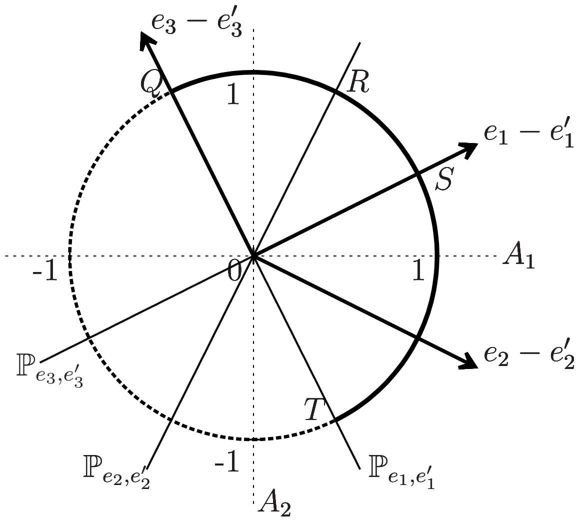


Figure 3.2: Binary Cutting Example in 2-Dimensions. (See Example 2.)

illustrated in Figure 3.2. If we choose $\langle e_2, e_2' \rangle$ as the next comparison, the pruning effect will depend on the user feedback. Specifically, if $\mathcal{C}(\langle e_2, e_2' \rangle) = 1$, the top-left side of $\mathbb{P}_{\langle e_2,e_2' \rangle}$ will be pruned, leaving arc $\widehat{RT}$. If $\mathcal{C}(\langle e_2, e_2' \rangle) = -1$, the bottom-right part of $\mathbb{P}_{\langle e_2,e_2' \rangle}$ will be pruned and arc $\widehat{QR}$ will remain.

This is not a very good choice, because in the worst case, only a small portion of the remaining sphere ($\widehat{QR}$) will be pruned. Overall, it will result in a worse average precision because our precision function is concave with respect to volume. In contrast, if we choose $\langle e_3, e_3' \rangle$ as the next comparison, regardless of the user feedback, half of the remaining sphere (either $\widehat{QS}$ or $\widehat{ST}$) is guaranteed to be pruned.

Example 2 elicits the following theory:

Theory 1 (Binary Cutting Strategy)**.** *In order to maximize the best case average precision, each comparison should cut the current remaining sphere into two equal halves.*

The intuition behind the theory is similar to the traditional binary search. We omit the proof due to space.

If we manage to choose a sequence of comparisons strictly satisfying the condition in Theorem 1, after M comparisons, the volume of the remaining sphere is:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \frac{\mathcal{V}(\mathbb{S}^k)}{2^M} \tag{3.22}$$

Compared with Equation 3.15 for random cut, this is much better because we can reduce the volume of the remaining sphere *exponentially* with respect to the number of comparisons.

By substituting Equation 3.14 into Equation 3.22, we obtain:

$$\mathcal{V}(\mathbb{S}^k_{\mathbf{D}}) = \frac{1}{2^{M-1}} \cdot \frac{\pi^{k/2}}{\Gamma(\frac{k}{2})} \tag{3.23}$$

Following analysis similar to Section 3.4.1, we can obtain the theoretical upper bound of the precision we can achieve after M comparison:

$$\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}) = \cos\left(2 \cdot \left(\frac{\sqrt{\pi}}{2^{M-1}} \cdot \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})}\right)^{\frac{1}{k-1}}\right) \tag{3.24}$$

By fixing $\mathcal{P}(\mathbb{S}^k_{\mathbf{D}})$ and solving M in the above equation, we obtain the *number of optimal cuts* to achieve a certain precision:

$$M_{optimal} = \frac{\log_2 \pi}{2} + \log_2 \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})} + (k-1) \cdot \log_2 \frac{2}{\arccos(\mathcal{P}(\mathbb{S}^k_{\mathbf{D}}))} \tag{3.25}$$

On the other hand, we have:

$$\lim_{k \to +\infty} \log_2 \frac{\Gamma(\frac{k+1}{2})}{\Gamma(\frac{k}{2})} = \frac{\psi^{\{0\}}(\frac{k}{2})}{2} = \frac{\log(\frac{k}{2})}{2} \tag{3.26}$$

Thus, the first term in Equation 3.25 is a constant, the second term converges to $(\log_2 k - 1)/2$, and the third term is the product of $k-1$ and To give an intuition, setting the precision to 0.95 results in an increase of 2.65 optimal cut needed per additional dimension.

3.4.3 Adaptive Comparison Selection

In the previous section we presented a theoretical analysis of the optimal comparison sequence. To recall, each time we desire the next comparison to cut the remaining sphere into two equal halves. Unfortunately, such an *optimal* comparison rarely exists in practice, because we have a limited number of comparison candidates. In this section, we instead examine the problem of finding *adaptively* the next comparison that cuts the remaining sphere as equally as possible. Moreover, we show how this strategy can deal with noise answers.

We start by following the two-dimensional scenario in Example 2. After the initial cut, the remaining sphere degrades to arc QT. Since the *center* of QT is S, one should select a comparison whose pruning plane is as close to the center S as possible. This is trivial in 2-d, because the pruning plane degrades to a line passing through the origin. Consequently, we only need to find the comparison whose decision plane (line) has the smallest angle with the line passing through the origin and the "center" of the remaining sphere (arc), which can be accomplished by a simple binary search. However, the following example demonstrates its complexity in higher dimensions.

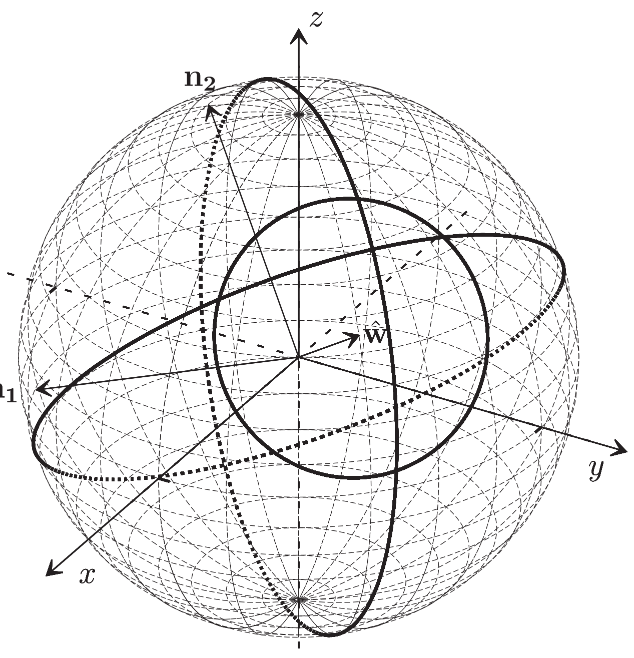


Figure 3.3: Two Pairwise Comparison Candidates in 3-Dimension

Example 3 (Comparison Selection in 3D). *For simplicity, assume the remaining sphere $\mathbb{S}^k_{\mathbf{D}}$ is uniformly distributed around center $\hat{\mathbf{w}}$ on the unit 3-sphere, as shown in Figure 3.3. Suppose we have two candidate comparisons: one with pruning plane identified by normal vector $\mathbf{n_1}$ and the other with pruning plane identified by $\mathbf{n_2}$. Intuitively, the plane with $\mathbf{n_1}$ is the better pick because it cuts* *the remaining sphere more evenly.*

This examples indicates the following two points. First, the problem is non-trivial in higher dimension, because there is no static ordering of the candidate comparisons. Indeed, the order of candidates are dependent on the current estimate $\hat{\mathbf{w}}$. Therefore, the simple binary search strategy in 2-d no longer applies. As a result, the naïve approach requires a scan of all the candidate comparisons, whose number is square of the total number of entities.

Second, the notion of "closeness" between the pruning plane and $\hat{\mathbf{w}}$ is no longer effective, because the pruning plane is a hyperplane in general. This leads us to think about the relationship between $\hat{\mathbf{w}}$ and the the normal vector $\mathbf{n}$, which is a better characteristic of the pruning plane. In fact, given a pruning plane with normal vector $\mathbf{n}$ and a remaining sphere with estimated center $\hat{\mathbf{w}}$, we notice that the more *orthogonal* $\mathbf{n}$ is with respect to $\hat{\mathbf{w}}$, the more evenly the plane cuts the remaining sphere. To quantify this, we have:

Definition 14 (Orthogonality). *Given two unit vectors* $\mathbf{u}$ *and* $\mathbf{v}$*, the orthogonality between* $\mathbf{u}$ *and* $\mathbf{v}$ *is* $1 - |\mathbf{u} \cdot \mathbf{v}|$. *Given a remaining sphere with center estimate* $\hat{\mathbf{w}}$ *and a candidate pruning plane with normal vector* $\mathbf{n}$*, we also say the orthogonality between the pruning plane and the remaining sphere is* $1 - |\mathbf{n} \cdot \hat{\mathbf{w}}|$.

Since both vectors are normalized, their orthogonality ranges from zero to one. It achieves its maximum when $\mathbf{n} \cdot \hat{\mathbf{w}}$ equals zero, or when the pruning plane passes through $\hat{\mathbf{w}}$. For simplicity, hereafter we will use the normal vector, the pruning plane and the corresponding comparison interchangeably, if the context is clear.

We also note, even a comparison with orthogonality one may not cut the remaining sphere into exactly equal halves. This is because the remaining sphere in practice has a very complex geometry. To distinguish such comparison from the optimal comparison we discussed in Section 3.4.2 , we call such comparison the *best next comparison* and have the following problem definition.

Definition 15 (Adaptive Best Next Comparison Selection). *Given an estimate* $\hat{\mathbf{w}}$*, find among all the remaining pairwise comparisons* $\mathcal{D}(\mathbf{E})\backslash\mathbf{D}$ *the best comparison, such that the orthogonality between the comparison and* $\hat{\mathbf{w}}$ *is maximized. Formally, obtain:*

$$\underset{\langle e,e' \rangle \in \mathcal{D}(\mathbf{E})\backslash\mathbf{D}}{\operatorname{argmin}} \left| \mathbf{n}_{\langle \mathbf{e},\mathbf{e}' \rangle} \cdot \hat{\mathbf{w}} \right| \tag{3.27}$$

This problem can be generalized as to find the vector that is most *orthogonal* to a given vector. This is very different from the traditional geographic queries which mostly focus on the nearest neighbors. To distinguish, we name this type of query **orthogonal query**.

Further, we give an intuition why this method will be noise-tolerance. Recall that the maximum margin optimization with slack variant relaxes the condition for those answers which is too hard to be satisfied. That is to say, when noise occurs, it will usually not used as the pruning plane since its condition is not satisfied. In the query selection phase, suppose one question is answered incorrectly, then the preference estimation will fall into another remaining sphere. However, as more and more questions are asked in the orthogonal plane of the estimation, the optimization function will choose to relax the condition of the noise comparison. Intuitively, the false plane are finally removed and the new estimation will live in the correct remaining sphere.

3.5 Experiment

In this section we describe our evaluation of our system. The most important question is whether the correct preference function was learned. We measure this correctness in terms of **precision** : the fraction of test (pairwise) comparisons correctly labeled by the system. We do so on both real and synthetic data. We also report results on additional experiments to evaluate the performance of the S-tree index structure.

As a baseline, we compare our **adaptive** algorithm against an algorithm that randomly chooses comparison pairs for training. This **random** algorithm still performs cuts in space and correctly computes the preference function based on observed preferences. Therefore, the superior results obtained by our algorithm can then all be attributed to its adaptive binary selection.

The algorithms were implemented in C++ referencing LIBSVM [17]. All experiments were run on an Unix machine with a 2.7 GHz Intel Core i7 processor and 8GB 1600 MHz DDR3 memory. The user study was built with a web user interface referencing the system via CGI.

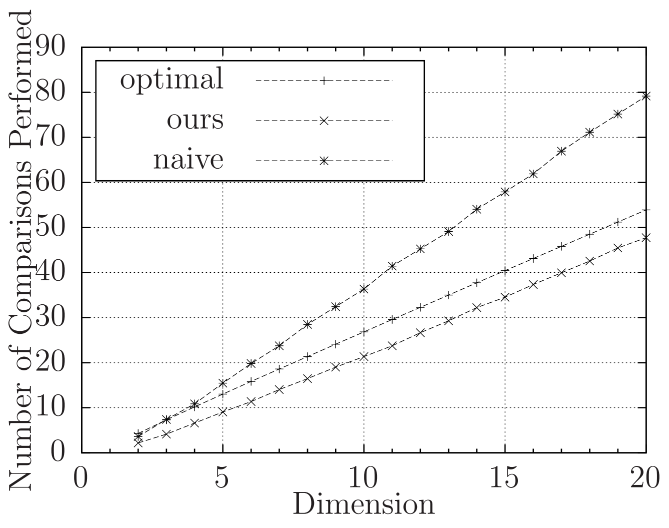


Figure 3.4: Number of Comparisons Required to Achieve a Target Precision of 0.95 for the Naïve System, the Theoretical Optimum, and Our System, with 100 Entities of 3 dimension

3.5.1 Experiment With Synthetic Data

The synthetic experiment is set up as follows. For each dimension, we randomly generated ten sets of one hundred entities. For each entity set, we randomly generated ten $\mathbf{w_0}$. And for each $\mathbf{w_0}$, we repeated the experiment ten times. We averaged the number of comparisons needed over all these variables. We randomly generated the source of truth $\mathbf{w_0}$, used Equation 3.1 to simulate the user feedback, estimated $\hat{\mathbf{w}}$ according to the discussion in Section 3.3, and calculated the precision as $\mathbf{w_0} \cdot \hat{\mathbf{w}}$.

We first measured the average number of pairwise comparisons needed to achieve a target precision of 0.95. The result is shown in Figure 3.4.

According to the result, our system dramatically outperforms the **naive** system, which randomly selects the next candidate comparison. On average, our system requires **70%** less comparisons to achieve the same precision. This is because, the random strategy does not guarantee the quality of the next cut. As the remaining sphere shrinks, it is increasingly hard for a random cut to pass through the remaining sphere. Even for the cuts that do pass through, they are likely to be imbalanced. In consequence, **random** requires a much larger number of total comparisons.

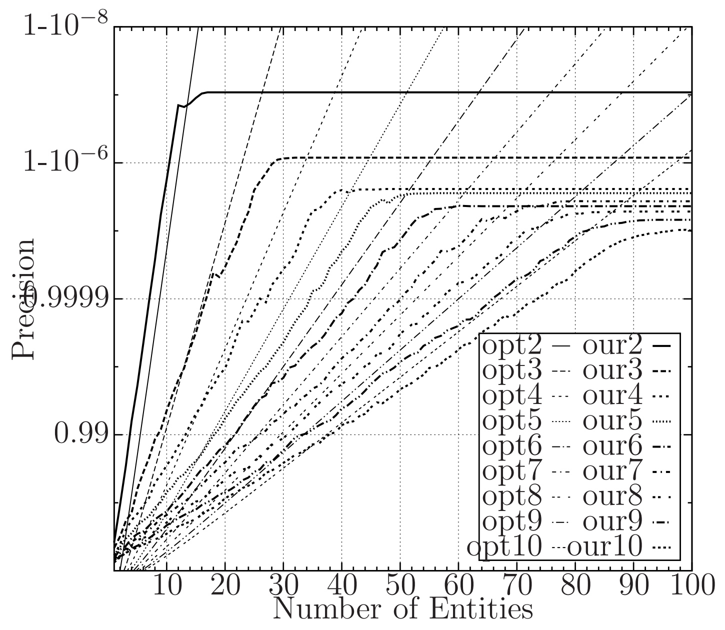


Figure 3.5: Precision Achieved by Our System and the Theoretical Optimum, w.r.t. the Number of Comparisons Performed and Dimension from 2 to 10

We also plot the theoretical optimum, assuming that perfect binary cutting is possible, as described in Section 3.4.2, and results in a number of comparisons according to Equation 3.25 and a worst case precision defined in Equation 3.24. This is an analytically computed worst case number. Our **adaptive** algorithm actually performs similar to and slightly better than this theoretical optimum. Note that this is not a contradiction. The theoretical optimum is defined with worst case precision, while in our system the precision is the observed average, which is better than the worst case.

To better understand the behavior beyond the target precision, we conducted a second experiment examining the precision change with respect the the number of comparisons performed. We kept the same experiment settings except we did not terminate the process when a target precision is reached. Instead, for each run, we performed one hundred comparisons, and recorded the average precision achieved after each comparison. We repeated the experiment for each dimension from 2 to 10. The result, as well as the theoretical optimal precision values (derived from Equation 3.24) are shown in Figure 3.5.

In general, the precision increases with the number of comparisons performed. It also decreases with dimension, because the higher the dimension is, the relatively larger radius the remaining sphere will have (see Equation 3.18).

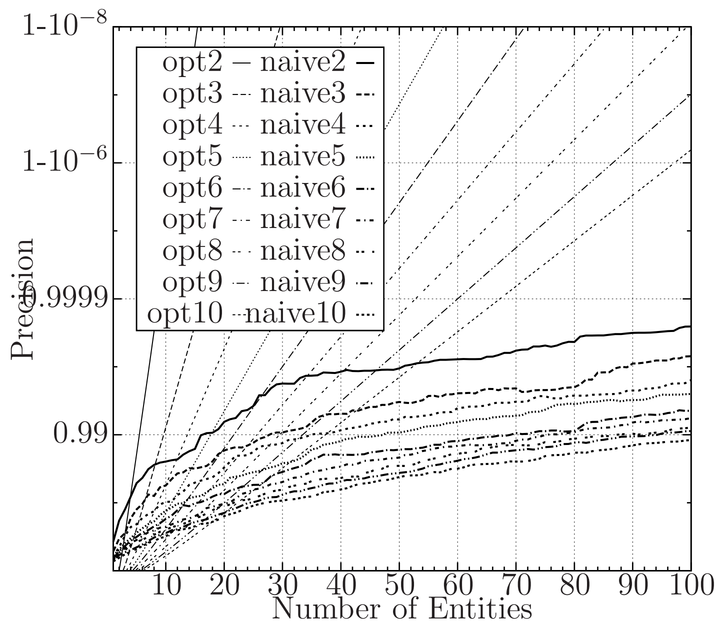


Figure 3.6: Precision Achieved by the Naïve System and the Theoretical Optimum, w.r.t. the Number of Comparisons Performed and Dimension from 2 to 10

We also notice that, below 1-10^{-5}, the precision obtained from our system is very close to the theoretical optimum. Our system outperforms the optimum for small number of comparisons because the approximation we used for the theoretical bound is only valid for larger number of comparisons.

For comparison purposes, we performed the same experiment on the naïve system with random selection and the result is shown in Figure 3.6. The result shows that the precision for the naïve system is significantly less than the optimum. By comparing this result with Figure 3.5, we see the error rate for our system $(1 - p)$ is **three orders of magnitude** less than the naïve system.

In Figure 3.5, we also notice that, our precision reaches an upper bound after a certain number of comparisons. This is due to the limited number of entities in practice. In theory, we can cut the remaining sphere infinitely, thus achieving arbitrarily high precision. Unfortunately, in practice, once we arrive at a remaining sphere such that no candidate comparison further cuts it, we cannot improve our estimate anymore. And the average precision we can obtain on such a *finest remaining sphere* is determined by the number of entities.

In practice, the precision is upper bounded by the distribution of the candi-

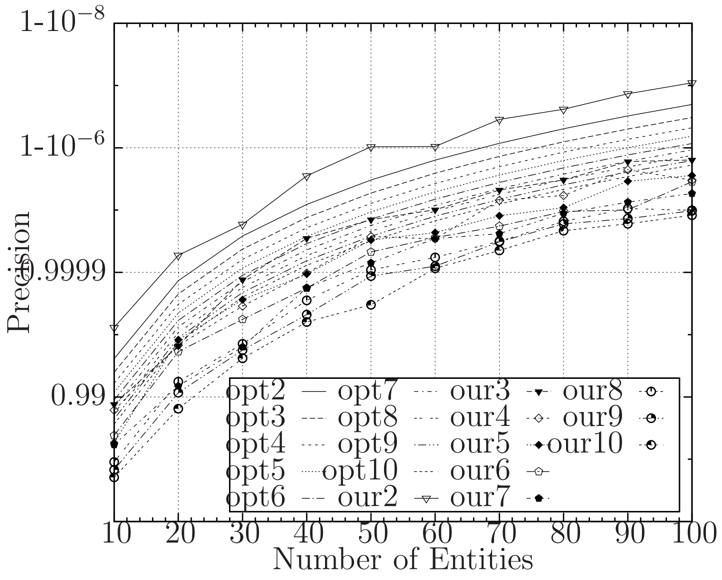


Figure 3.7: Practical Precision Upper Bound in Our System w.r.t. the Number of Entities, Compared with the Theoretical Values on Dimension 2 to 10

date comparisons. For instance, if at certain stage we derive a remaining sphere such that no candidate comparison further cuts it, we obviously cannot further improve our estimate.

We conducted a third experiment to provide more insights into this practical precision upper bound. We used the same experiment settings, except we continued to perform comparisons until our estimate did not change any more. (In fact, in these cases, the next best comparison we obtained from the orthogonal query must have been performed before.) For each dimension from 2 to 10 and the number of entities from 10 to 100, we recorded the average maximum precision achieved. The result is shown in Figure 3.7, together with the theoretical upper bound derived from Equation 3.20.

The result shows the best precision obtained by our system is not far from the theoretical upper bound (with one order of magnitude difference in error rate). We also note the precision upper bound increases with respect to the number of entities. This is because, the more entities we have, the more candidate comparisons we have, and the smaller the finest remaining spheres will be.

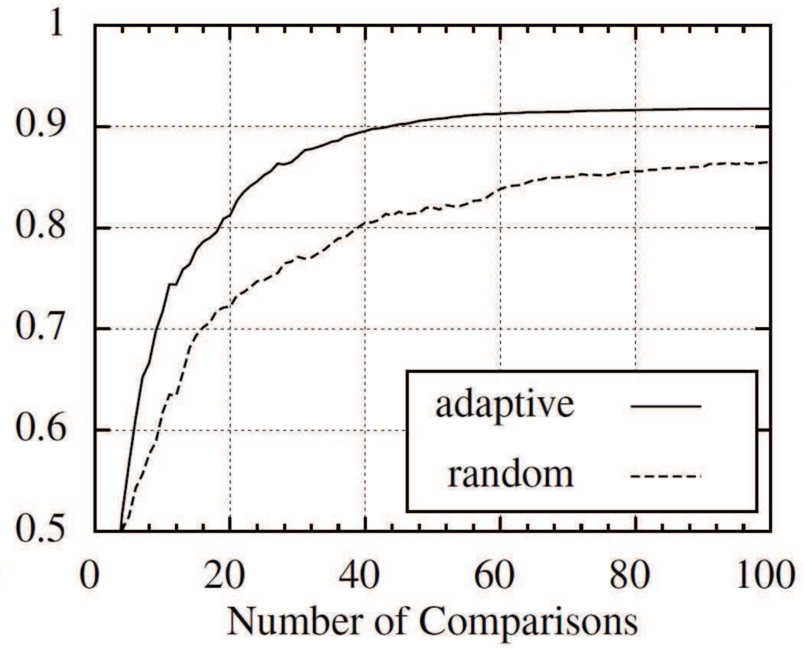


Figure 3.8: Precision Achieved by Our Adaptive System and the Naïve Random System, w.r.t. the Number of Simulated of Pairwise Comparisons on Yahoo Used Cars

3.5.2 Experiment with Real Data

To further demonstrate the improvement introduced by our adaptive comparison selection, we conducted a second experiment on real data. We used the Yahoo Used Car Dataset and randomly picked one thousand cars (one million comparisons). For each car, we represented it using nine dimensions including price, mileage, year etc., and normalized the data.

We measured the precision change with respect to the number of comparisons made, with the same settings as the previous experiment, expect that we replaced the synthetic data with the real used car dataset. The result is shown in Figure 3.8.

Overall, we observe a drop in precision for both our adaptive system and the naïve system, compared to the previous experiment with synthetic data. This is because, the real data cannot guarantee uniform distribution, for several reasons. For instance, the distributions of some attributes are quite skewed by themselves (e.g., price and mileage). Furthermore, certain categorical attributes are split, causing the comparison vectors to have very few values on certain dimensions.

Despite the overall drop, our adaptive system still significantly outperforms the naïve system. After 40 comparisons, the adaptive precision is 10% higher. We also notice that the adaptive precision is capped after around 60 comparisons, while the random system is catching up. Intuitively, this cap is introduced by the uneven distribution of real data. The random system will eventually reach the cap but with a much slower convergence rate.

3.5.3 User Study

Since real users may not have linear preference and may make mistakes in providing feedback, we conducted a user study to further analyze our system in real scenarios. We recruited ten subjects and asked them to complete an online survey. The survey consisted of one hundred pairwise comparisons of used cars. The cars are from the same set of one thousand cars used in the previous experiment.

We separated the one hundred comparisons into three buckets: An adaptive training bucket with thirty adaptively selected comparisons, a random training bucket with thirty randomly selected comparisons, and a testing bucket with forty random comparisons. We randomized the order of these buckets to counterbalance the learning effect.

We respectively trained the user preference on the adaptive training bucket and random training bucket, and tested the preference on the testing bucket. The result is illustrated in Figure 3.9.

According to the result, our adaptive system is slightly worse than the random system before 15 comparisons. This is because our adaptive algorithm is much more sensitive to user errors in early stages. After 20 comparisons, our adaptive system significantly outperforms the **random** system. Furthermore, after 20 comparisons, the precision obtained with real user responses (Figure 3.9) is almost as good as the precision with simulated responses (Figure 3.8). For example, at 30 comparisons, the precision with real user feedback is 83% and 75% for **adaptive** and **random** respectively, while it is 87% and 76% with simulated (ideal) feedback. This demonstrates that the linear preference assumption is reasonable and our system is error-tolerant in practice.

Finally, we conducted an experiment to determine the importance of computing individual preferences. For this purpose, we trained a baseline *general*

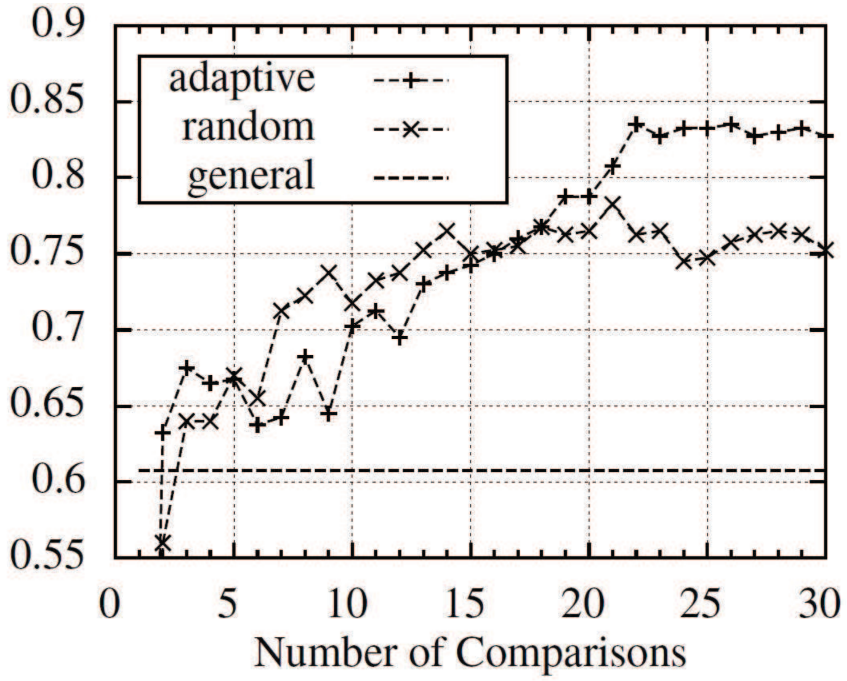


Figure 3.9: Precision Achieved by Our Adaptive System and the Naïve Random System, w.r.t. the Number of Comparisons Performed by Real Users on Yahoo Used Cars

preference from the union of all training data from all users, and tested this preference function on each user's individual testing bucket. The resulting precision averaged over users is also displayed in Figure 3.9. This precision is significantly lower than both the adaptive and the random precisions. This indicates that real users do exhibit very heterogenous preference in the used car scenario and there is not a universal preference that can satisfy all the users.

3.6 Summary

To address this need of supporting complex annotation problems which can not be directly labeled, we have developed a system for preference learning on structured entities from adaptively selected pairwise comparisons. Under a linear preference assumption, we formalized the preference learning problem and divided it into two subproblems, namely preference estimation and adaptive comparison selection.

For preference estimation, we derived the mathematical foundation of the problem and solved it using an innovative application of SVM on the normal vectors of the comparisons' pruning planes. For adaptive comparison selection, we theoretically analyzed a naïve random selection approach and an optimal binary selection strategy, and proposed our adaptive selection approach. We abstracted the adaptive selection into an innovative orthogonal query, and proposed a new type of high-dimensional index S-tree to answer it.

We described our algorithms for various parts and integrated them into a preference learning system. We further implemented the system and evaluated its effectiveness and performance. Our experiment with synthetic data showed that our system is able to achieve a high precision with just a few comparisons, coming close to the theoretical binary selection optimum. Both our experiment and user study with real data demonstrated that our system significantly outperforms the naïve random selection system. We also showed that our S-tree index can be built efficiently and serve orthogonal query in interactive speed with a reasonable entity size.

Finally, we experimentally demonstrated that our linear preference assumption is reasonable in practice, and furthermore that our method is able to tolerate naturally occurring user inconsistencies in pairwise preference reporting.

CHAPTER 4

ONLINE BAYESIAN MODEL FOR CROWDSOURCING QUALITY MANAGEMENT

Crowdsourcing has created a variety of opportunities for many challenging problems by leveraging human intelligence. For example, applications such as image tagging, natural language processing, and semantic-based information retrieval can exploit crowd-based human computation to supplement existing computational algorithms. Naturally, human workers in crowdsourcing solve problems based on their knowledge, experience, and perception. It is therefore not clear which problems can be better solved by crowdsourcing than solving solely using traditional machine-based methods. In order to apply crowdsourcing as the data annotation methods and leverage crowdsourced results to generate more informative data, there is a great demand for measuring the quality of crowdsourced results, and designing a cost-sensitive method to make the right trade-off between the quality of result and the expenses for human efforts.

In this chapter, we design and implement a cost-sensitive method for crowdsourcing. We online estimate the profit of the crowdsourcing job so that those questions with no future profit from crowdsourcing can be terminated. Two models are proposed to estimate the profit of crowdsourcing job, namely the linear value model and the generalized non-linear model. Based on these models, the expected profit of obtaining new answers for a specific question is computed based on the answers already received. A question is terminated in real time if the marginal expected profit of obtaining more answers is not positive. We extend the method to publish questions in a batch manner. We evaluate the effectiveness of our proposed method using two real world jobs on the AMT crowdsourcing platform. The experimental results show that our proposed method outperforms all the state-of-art methods.

4.1 Introduction

Crowdsourcing has attracted a great deal of interest as a platform for leveraging crowd-based human computation and intelligence. It is to some extent inspired by the vast amount of collaboration in Web 2.0 communities, where users share not only information, but also their knowledge and intelligence. Platforms such as, for example, the Amazon Mechanical Turk (AMT) [1], facilitate the refinement of data and reduction of noise by passing complex jobs to human workers. These complex jobs include image tagging, semantic-based information retrieval, and natural language processing, which are hard for computers, but relatively easy for human workers. Instead of designing sophisticated algorithms or spending a lot of money to consult experts, many of these jobs can be solved by human workers on a crowdsourcing platform at a much lower cost. Some successful crowdsourcing applications that appear recently include CrowdDB [27], CrowdSearch [115], and HumanGS [79].

Despite the success of crowdsourcing systems, employing crowdsourcing effectively remains challenging for three reasons. First, for most crowdsourcing jobs, we need to obtain multiple answers to guarantee their quality. Thus, we have to decide when to stop obtaining new results provided by human workers. Most existing work uses the accuracy or the cost as the optimization objective. These turn out to be too rigid in practice. However, the trade-off is not trivial. Typically crowdsourcing jobs may have different level of difficulty (e.g. homework of kids vs. research problems), risks (e.g. Flickr image tagging vs. cancer diagnosing), and profits (e.g. survey for personal interest vs. design for investment model). Therefore, it is important to have an online economic model that considers all these factors of crowdsourcing jobs.

Second, none of the current research focuses on whether a problem is suit-

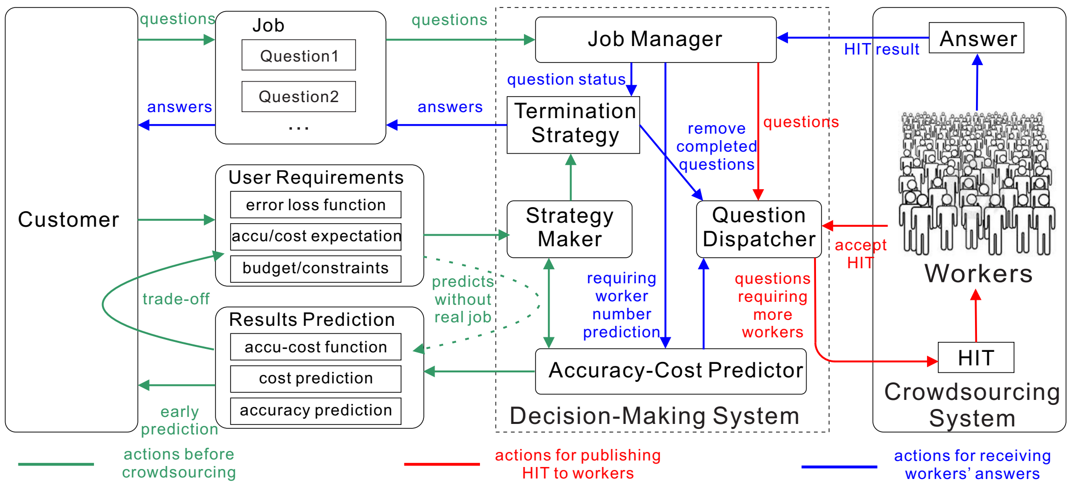


Figure 4.1: The Architecture of Cost Sensitive Decision Making System

able for crowdsourcing or not. Intuitively, crowdsourcing based techniques are more fitting for problems that require semantic processing. For example, sentiment analysis, image tagging, and information retrieval are good problems for crowdsourcing while large scale numerical analytics are better handled by machines.

Finally, the estimation of crowdsourcing quality is still primitive. Low quality answers may sharply reduce the quality of crowdsourcing, and introduce noises. To resolve the quality issue, several methods have been proposed, such as Crowdscreen [80] and CDAS [64]. Both proposals only consider the accuracy of the answers provided by the workers, without taking into account the difficulty of the tasks. However, it is very challenging to predict the difficulty of the tasks, and a robust algorithm that can handle problems of varied levels of difficulty and answers of varied levels of quality is needed.

In this chapter, we propose a novel online cost sensitive decision-making model to address the above three challenges. Our contributions include:

- We propose an online, cost sensitive decision-making model to analyze and decide whether to stop the question in its current status, given the value of the question, the risk of getting incorrect answers, and the cost of workers in the crowdsourcing system. To the best of our knowledge, our model is the first that provides an online quantitative profit analysis of crowdsourcing jobs. We further extend our algorithm to support online cost sensitive decision-making with constraints, such as limited budgets etc.

- An application or task may contain questions of diverse difficulty levels. We propose a model for measuring the difficulty of a question and We design and implement a robust algorithm that handles such questions.

- We propose a novel algorithm called Accuracy-Cost to perform the marginal analysis of the accuracy and the cost in crowdsourcing. The algorithm calculates the incremental pro- fit when the number of workers is increased.

- We conduct extensive experimental studies on two real data-sets obtained from the answers of workers in AMT to evaluate the effectiveness of our proposed method. The results show that our method obtains precise results while keeping the cost low. We also develop an automatic question dispatching method that assigns multiple questions in a HIT while each question can be terminated at any time.

The rest of this chapter is organized as follows. Section 4.2 gives an overview of our method. Section 4.3 explains the preliminaries. Section 4.4 presents our proposed linear model to get real time decisions and analyse the profit. Section 4.5 extends our linear model to the non-linear model, which demonstrates the relationship between cost and accuracy and gives our model widespread applicability. Section 4.6 discusses the experimental studies. Finally, we conclude in Section 4.7.

4.2 Overview

In this section, we give an overview of our cost sensitive decision-making method for crowdsourcing.

Figure 4.1 shows the architecture of our proposed method. The core part is the decision-making system (the dashed box), which consists of five components, namely the job manager, the question dispatcher, the accuracy-cost predictor, the strategy maker, and the termination strategy manager. The job manager passes the questions from the crowdsourcing customer to the question dispatcher. It keeps collecting the answers of the workers and reporting the updated question status to the termination strategy manager. After receiving the questions, the question dispatcher allocates these questions to several workers using a question dispatching algorithm (will be discussed in Subsection 4.5.2). The termination strategy manager determines whether we should stop getting more answers for a question and return the results to the customer based on the status of the question. These termination strategies are generated by the strategy maker according to user's requirements, namely the error loss function, the accuracy/cost expectation, the budget, or other constraints. The strategy maker employs a linear model we propose in Section 4.4. Note that the generated strategies are stored in the termination strategy manger so that our decision-making system is able to decide if we should terminate a question in real-time. The accuracy-cost predictor provides two major functionalities. First, it predicts the results as well as the accuracy/cost ratio. Second, it generates the termination strategies using a generalized non-linear model together with the strategy maker. The details of the non-linear model is discussed in Section 4.5.

In summary, our proposed decision-making system automatically dispatches questions to the workers. It terminates the questions and returns the answers in real-time according to the generated termination strategies. This decision-making system also provides an early prediction of the results before running a real crowdsourcing job in order to help the customer set proper user requirements. In the following sections, we present the idea and the method of implementing this decision making system.

4.3 Preliminaries

Before describing our method in Section 4.4 and 4.5, we discuss the required prior knowledge to understand the problem.

For the sake of brevity, in our model we assume the questions are two-choice problems. We can think of the two choices as binary values 0 and 1. However, we can easily extend our method to problems that have more than two choices.

Question status: We model the status of a question by a pair (m, l) that represents the numbers of the two different answers received from the workers. Since the values 0 and 1 are just two symbols to represent both choices, the 0s and 1s in the answers can be exchanged. As a result, the following two cases: (1) m 0s and l 1s; (2) m 1s and l 0s can be viewed as identical to represent the agreement of the workers on the two choices. Without loss of generality, in the remainder of this chapter, we use (m, l) $(m \geqslant l)$ to represent the above two cases.

Intuitively, the question status indicates the difficulty of the question. When m is far larger than l, most of the workers agree on one choice. It could be that this question is easy. On the other hand, when m is close to l, it is likely that the question is too difficult so that the workers are just making guesses.

Question run: A question run r is a sequence of question statuses as we get answers from workers for this question, i.e.,

$$r = \{(m_0, l_0), (m_1, l_1), \cdots, (m_n, l_n)\}$$

where $(m_0, l_0) = (0, 0)$ is the initial status. Every non-initial status (m_i, l_i) has exactly one more answer than its previous status (m_{i-1}, l_{i-1}), i.e., $(m_i, l_i) = (m_{i-1}, l_{i-1} + 1)$ or $(m_{i-1} + 1, l_{i-1})$. In the above question run r, the question is terminated at question status (m_n, l_n), after which it will not accept more answers. For example, $\{(0, 0), (1, 0), (2, 0), (2, 1)\}$ is a valid question run that stops after getting three answers but $\{(0, 0), (1, 0), (2, 1), (2, 0)\}$ is not a valid question run.

Accuracy of question answer: We use A_Q to denote the accuracy of an answer to question Q, i.e. the probability that a worker provides the correct answer. Most of existing works considering the accuracy of answers assume that the accuracy is a fixed value that can be computed from sampled answers, e.g. [64][80]. The major drawback of these models is that they do not take the difficulty of questions into consideration. Using these models, for a single worker, his answers to different questions would have the same quality. However, this observation contradicts the intuition that the answers of a hard question might be poor.

In this work, instead of modelling the accuracy as a single fixed value and employing sampling based methods to estimate this value, we represent the accuracy as a probability distribution. The probability distribution provides the ability of modelling answer quality based on the observed question status. We model the accuracy A_Q as a random variable and we estimate the value of A_Q by the observation of the question status (m, l). Specifically, we assume that A_Q obeys the Beta distribution:

Assumption 4.1. *Given a question Q, the probability density function of A_Q*

is:

$$f(A_Q = \mu) = \mu^{a-1}(1-\mu)^{b-1}/Beta(a,b)$$

We thus have $E[A_Q] = a/(a+b)$. a and b are parameters of the Beta distribution, representing the prior prediction of the random variable. The Beta distribution is actually a two-dimensional Dirichlet distribution [1]. We can replace the Beta distribution by the more general Dirichlet distribution for multiple choice questions.

If the prior distribution is $\mathrm{B}(\mu|1,1)$, i.e., the uniform distribution, then the estimation of A_Q is only determined by the observation of the question status. In other words, our method is a generalized form of both the traditional fixed accuracy model and the uniform distribution. It takes both the prior knowledge and the question status observation into consideration to adapt the estimated distribution of the difficulty of problems. Ideally the estimated distribution performs best when it is the same as the empirical distribution of problem difficulty. However, the empirical distribution is always difficult to obtain. Moreover, it is not feasible to compute Bayesian inference based on the empirical distribution. The Beta distribution is used to simplify the computation, as it is the conjugate prior distribution of Binomial and Bernoulli distribution. While on the other hand, Beta distribution with proper parameters fits the accuracy distribution well.

We have observed that our model is robust since using different prior prediction parameters a, b almost does not change the results. This is because our model is based on a probability distribution and it can be adaptively adjusted to be applied on questions with diversified difficulties. Moreover, we observed that the empirical distribution of A_Q in several real question sets, including the tweet sentiment analysis questions and common sense questions used in our experiments, is similar to the $\mathrm{B}(\mu|6,2)$ distribution. On the other hand, our experiments also show that when distribution changes, this estimation still works well. In Section 4.4, we explain the reason that the probability distribution based accuracy model outperforms the fixed value based accuracy models. In the experiment section, we use empirical data to support the above two observations.

Accuracy of question result: We use A_R to represent the accuracy of the result based on a majority voting on the current question status (m, l).

[1] http://en.wikipedia.org/wiki/Dirichlet_distribution

Q	a question in a crowdsourcing job
V_Q	the value of a crowdsourcing question Q
L_Q	the loss of getting a wrong answer for Q
C_Q	the cost of assigning a question to a worker
(m,l)	the status of the crowdsourcing question Q
A_R	the probability that the voting result of Q is correct
A_Q	the probability that a worker provides the correct answer to the question Q
$MI(m,l)$	the marginal income of the question accuracy A_R in status (m,l)
$P(m,l)$	the expected economic profit of the question in status (m,l)
$P_S(m,l)$	the expected economic profit of stopping the question in status (m,l)
$P_C(m,l)$	the expected economic profit of continuing the question in status (m,l)
$\mathrm{B}(\mu\|a,b)$	PDF of Beta distribution
$\Gamma(n)$	Gamma function
$Beta(a,b)$	Beta function

Table 4.1: Notations

We always choose the answer represented by the majority m. As a result, A_R is the probability that the majorities m choose the correct answer. We define $\langle x,y\rangle$ as the status having x correct answers and y incorrect answers. As the status (m,l) represents either m correct answers or m incorrect answers, $(m,l) = \langle m,l\rangle \cup \langle l,m\rangle$. By Bayesian analysis, the conditional probability, A_R given A_Q and observation (m,l) is:

$$A_R = \frac{Pr(\langle m,l\rangle)}{Pr(\langle m,l\rangle) + Pr(\langle l,m\rangle)} = \frac{{A_Q}^{m-l}}{{A_Q}^{m-l} + (1-A_Q)^{m-l}}$$

Note that A_R is also a random variable as A_Q is a random variable.

The notations used in the following sections of this chapter are listed in Table 4.1.

4.4 Linear Decision-Making

In this section, we introduce a linear model for online decision making. We first estimate the accuracy of the answers according to the question status and prior distribution. Based on the estimation of the answer accuracy, we obtain the marginal income and the profit of each status. This enables us to make decisions at each status according to the economic profit. In this chapter, we employ a dynamic programming algorithm to calculate the profit and generate the strategy for each question status. We prove that the time complexity of our strategy generating algorithm is $O(n^2)$ (n is the maximum possible number of answers for a single question, i.e. the search space), which is a significant improvement compared with existing methods such as CrowdScreen's $O(n^4)$ linear programming. In this section, we also discuss the techniques of extending our method to solve a more complex problem, namely linear model with constraints of accuracy and cost.

4.4.1 Linear model

For a question Q, the linear model has three variables: the question value V_Q, the error loss L_Q, and the question cost per worker C_Q. As discussed in Section 4.2, these values are preset by the crowdsourcing customer. V_Q is the value of this question given an answer (not necessarily a correct one). The error loss L_Q is the penalty of obtaining a wrong answer for Q. C_Q represents the cost of hiring a worker for Q. According to the definition of V_Q, L_Q and C_Q, we propose the following linear model:

Definition 4.1 (Profit of a question)**.** *Suppose a question Q ends after it receives k answers, and the result is ans. The profit P of Q is:*

(1) ans is correct: $P = V_Q - kC_Q$.

(2) ans is incorrect: $P = V_Q - L_Q - kC_Q$.

However, in practice we do not know whether the answer *ans* is correct or not. Thus, we compute the expected value $E[P]$ of the profit to estimate the average profit of question Q. The expected profit is computed by:

$$\begin{aligned} E[P] &= (V_Q - kC_Q)E[A_R] + (V_Q - L_Q - kC_Q)(1 - E[A_R]) \\ &= L_Q E[A_R] + V_Q - L_Q - kC_Q \end{aligned}$$

	Question Status / Prior Distribution	(1,0)	(3,3)	(4,0)	(8,2)	(100,100)	(101,100)	(110, 100)
A_Q	Fixed accuracy 0.75	0.75	0.75	0.75	0.75	0.75	0.75	0.75
	$\mathrm{B}(\mu\|6, 2)$	0.75	0.643	0.821	0.762	0.51	0.51	0.513
	Fixed accuracy 0.80	0.8	0.8	0.8	0.8	0.8	0.8	0.8
	$\mathrm{B}(\mu\|8, 2)$	0.8	0.687	0.853	0.79	0.514	0.514	0.52
A_R	Fixed accuracy 0.75	0.75	0.5	0.988	0.999	0.5	0.75	1
	$\mathrm{B}(\mu\|6, 2)$	0.75	0.5	0.962	0.953	0.5	0.51	0.591
	Fixed accuracy 0.80	0.8	0.5	0.996	1	0.5	0.8	1
	$\mathrm{B}(\mu\|8, 2)$	0.8	0.5	0.985	0.983	0.5	0.514	0.634

Table 4.2: Trends of A_Q and A_R

where A_R is the probability that *ans* is correct. Note that A_R is a random variable, we thus use the expected $E[P]$ with respect to all possible worlds of A_R.

We use two functions, namely the value function $f^V(E[A_R])$ and the cost function $f^C(E[A_R])$, to describe the expected profit function $E[P](E[A_R])$. $f^V(E[A_R])$ is the expected gain from the job, while $f^C(E[A_R])$ is the cost of crowdsourcing. In this model, obviously $f^V(E[A_R])$ (i.e. $L_Q E[A_R] + V_Q - L_Q$) is a linear function of accuracy $E[A_R]$. It is remarkable that the expected profit function is not a linear function for $E[A_R]$, since larger $E[A_R]$ requires more answers (larger number of k). We will discuss the non-linear $f^V(E[A_R])$ in next section for some special needs in multi-question situation.

In Subsection 4.4.2, we describe the method to estimate $E[A_R]$ based on the prior distribution $\mathrm{B}(\mu|a, b)$ of the difficulty A_Q and the observations on the final status (m, l) of Q.

We formally define the decision-making problem based on the linear model.

Definition 4.2. *Given V_Q, L_Q and C_Q of a question Q, find the decision-making algorithm to maximize $E[P]$ among all possible runs of Q on any answer sequence provided by the workers.*

We next show the method of deriving $E[A_Q]$, $E[A_R]$ etc. in Subsection 4.4.2 and we discuss the algorithm of finding the question run that maximizes $E[P]$ in Subsection 4.4.3.

4.4.2 Accuracy Estimation

Suppose the observation of the final status is (m, l), we can derive the $E[A_Q]$ based on the prior distribution $\mathrm{B}(\mu|a, b)$ using Bayesian Analysis. The posterior result is our estimation of $E[A_Q]$.

Theorem 4.1.

$$E[A_Q] = \frac{\Gamma(a+m+1)\Gamma(b+l)+\Gamma(a+l+1)\Gamma(b+m)}{(a+b+m+l)(\Gamma(a+m)\Gamma(b+l)+\Gamma(a+l)\Gamma(b+m))}$$

where $\Gamma(n)$ *is the Gamma function such that* $\Gamma(n) = (n-1)!$ *for any positive integer* n.

Proof. We prove the theorem using Bayesian Theorem. The conditional probability density function of the observation $O = (m, l)$ given the condition $A_Q = x$ ($x \sim \mathrm{B}(\mu|a, b)$) is

$$f_O(m, l|A_Q = x) = \binom{m+l}{m}(x^m(1-x)^l + x^l(1-x)^m)$$

Note that the probability density function contains two cases: (1) m correct answers, l incorrect answers and (2) m incorrect answers, l correct answers. Using Bayesian Theorem, we have:

$$f_{A_Q}(A_Q = x|m, l) = \frac{f_O(m, l|A_Q = x)f_{A_Q}(A_Q = x)}{f_O(m, l)}$$

$$= \frac{(x^m(1-x)^l + x^l(1-x)^m)x^{a-1}(1-x)^{b-1}}{\int_0^1 (x^m(1-x)^l + x^l(1-x)^m)x^{a-1}(1-x)^{b-1}dx}$$

$$= \frac{x^{a+m-1}(1-x)^{b+l-1} + x^{a+l-1}(1-x)^{b+m-1}}{\int_0^1 x^{a+m-1}(1-x)^{b+l-1} + x^{a+l-1}(1-x)^{b+m-1}dx}$$

Therefore, the expected value of A_Q is:

$$E[A_Q] = \int_0^1 x f_{A_Q}(A_Q = x|m, l)dx$$

$$= \int_0^1 \frac{x^{a+m}(1-x)^{b+l-1} + x^{a+l}(1-x)^{b+m-1}}{\int_0^1 (x^{a+m-1}(1-x)^{b+l-1} + x^{a+l-1}(1-x)^{b+m-1})dx}dx$$

$$= \frac{\int_0^1 x^{a+m}(1-x)^{b+l-1}dx + \int_0^1 x^{a+l}(1-x)^{b+m-1}dx}{\int_0^1 (x^{a+m-1}(1-x)^{b+l-1}dx + \int_0^1 x^{a+l-1}(1-x)^{b+m-1})dx}$$

$$= \frac{Beta(a+m+1, b+l) + Beta(a+l+1, b+m)}{Beta(a+m, b+l) + Beta(a+l, b+m)}$$

$$= \frac{\Gamma(a+m+1)\Gamma(b+l)+\Gamma(a+l+1)\Gamma(b+m)}{(a+b+m+l)(\Gamma(a+m)\Gamma(b+l)+\Gamma(a+l)\Gamma(b+m))}$$

□

Theorem 4.1 gives the average accuracy of the answers. Based on this theorem, we can further derive the expectation of the accuracy of the results

A_R using Bayesian Analysis. We have the following theorem:

Theorem 4.2. *Given observations on the final status (m, l) of a question, the expected accuracy of the results $E[A_R]$ is:*

$$E[A_R] = \frac{\Gamma(a+m)\Gamma(b+l)}{\Gamma(a+m)\Gamma(b+l) + \Gamma(a+l)\Gamma(b+m)}$$

Proof. Note that A_R is a function of A_Q. Thus, the expected value of A_R is:

$$\begin{aligned}
E[A_R] &= \int_0^1 A_R f_{A_Q}(A_Q = x|m,l)dx \\
&= \int_0^1 \frac{x^m(1-x)^l}{x^m(1-x)^l + x^l(1-x)^m} \\
&\quad \times \frac{x^{a+m-1}(1-x)^{b+l-1} + x^{a+l-1}(1-x)^{b+m-1}}{\int_0^1 x^{a+m-1}(1-x)^{b+l-1} + x^{a+l-1}(1-x)^{b+m-1}dx}dx \\
&= \frac{\int_0^1 x^{a+m-1}(1-x)^{b+l-1}dx}{\int_0^1 x^{a+m-1}(1-x)^{b+l-1} + x^{a+l-1}(1-x)^{b+m-1}dx} \\
&= \frac{Beta(a+m, b+l)}{Beta(a+m, b+l) + Beta(a+l, b+m)} \\
&= \frac{\Gamma(a+m)\Gamma(b+l)}{\Gamma(a+m)\Gamma(b+l) + \Gamma(a+l)\Gamma(b+m)}
\end{aligned}$$

□

Theorem 4.1 and 4.2 show that the expected values of A_Q and A_R only depend on the status (m, l) and the prior parameters a, b.

To illustrate Theorem 4.1 and 4.2, we show several examples of trends of A_Q and A_R on different question status given specified prior distribution of difficulty of questions in Table 4.2. We compare the two Beta distributions $\mathrm{B}(\mu|6, 2)$ and $\mathrm{B}(\mu|8, 2)$ with two fixed value based models with accuracy 0.75 and 0.8, which are the expected values of the two Beta distributions respectively. We obtain the following facts from Table 4.2: in status $(3, 3)$, the predicted value of A_Q of the $\mathrm{B}(\mu|8, 2)$ distribution model (0.687) is smaller than that of fixed value model (0.75); in status $(4, 0)$, the predicted value of A_Q of the $\mathrm{B}(\mu|6, 2)$ distribution model (0.821) is larger than that of fixed value model (0.8). These two facts show that the Beta distribution model provides a better prediction than the fixed value based models, since the predicted values of A_Q according to the fixed value based models are always the same no matter what the current question status is.

Moreover, the predicted values of different distribution models converge while the number of answers goes up. Therefore, the choice of prior parameters in the distribution model is not important, because the distribution model can adaptively adjust the predicted values according to the observation of question status Consider the status $(8, 2)$, the two fixed value models (0.75 and 0.8) provide very aggressive predictions on A_R, i.e. 0.999 and 1 respectively. However, in practice, 8 of 10 workers agree may not always guarantee the correctness of the results.

When the status is (100, 100), we intuitively believe A_Q is likely to be close to 0.5 rather than 0.75 or 0.8. Based on the status $(100, 100)$, we consider the case of accepting one more answer, i.e. question status $(101, 100)$. Obviously A_R should still be close to 0.5 as the votes of both choices are very close. However, the two fixed value based models provide the prediction of A_R as 0.75 and 0.8 respectively, which are the same as the predicted values in question status $(1, 0)$. This example shows that the Beta distribution model outperforms the fixed value based models by considering the question status.

According to Theorem 4.1 and 4.2, we can predict the possible future answers based on $E[A_Q]$, $E[A_R]$ and the current status (m, l). We define two transitive probabilities:

(1) $Pr(+1|m, l)$: probability from status (m, l) to $(m + 1, l)$.

(2) $Pr(-1|m, l)$: probability from status (m, l) to $(m, l + 1)$.

Intuitively $Pr(+1|m, l)$ and $Pr(-1|m, l)$ represent the probability that the next answer is the same value as the value voted by m and l answers respectively. Obviously $Pr(+1|m, l) + Pr(-1|m, l) = 1$. We have the following theorem:

Theorem 4.3.

$$\begin{aligned} & E[Pr(+1|m, l)] \\ = & \frac{\Gamma(a+m+1)\Gamma(b+l) + \Gamma(b+m+1)\Gamma(a+l)}{(a+b+m+l)(\Gamma(a+m)\Gamma(b+l) + \Gamma(a+l)\Gamma(b+m))} \end{aligned}$$

The proof is analogous to Theorem 4.1. Based on these above theorems, we can quantitatively derive the decision-making strategies.

4.4.3 Decision Making

We consider the marginal income of accuracy at each status (m, l). Then we derive the profit according to the marginal income of accuracy. We define the marginal income of accuracy at status (m, l) as the expected increase of the $E[A_R]$ after obtaining a new answer. We denote the marginal income of accuracy at status (m, l) as $MI(m, l)$.

Definition 4.3.

$$MI(m,l) \quad = \quad E'[A_R|m,l] - E[A_R|m,l]$$

where $E'[A_R|m,l]$ *is the expected accuracy of the results after getting one more answer, i.e.* $E'[A_R|m,l] =$

if m¿l: $E[E[A_R|m+1,l]Pr(+1|m,l) + E[A_R|m,l+1]Pr(-1|m,l)]$*, i.e.* $E[A_R|m+1,l]E[Pr(+1|m,l)] + E[A_R|m,l+1]E[Pr(-1|m,l)]$

if m=l: $E[E[A_R|m+1,l]Pr(+1|m,l) + E[A_R|l+1,m]Pr(-1|m,l)]$*, i.e.* $E[A_R|m+1,l]E[Pr(+1|m,l)] + E[A_R|l+1,m]E[Pr(-1|m,l)]$

By simplifying the equations in Definition 4.3, we have the following theorem:

Theorem 4.4. *The marginal income of accuracy* $MI(m,l)$ *satisfies:*

(1) $MI(m,l) = 0$ *when* $m > l$.

(2) $MI(m,m) = \dfrac{a-b}{2(a+b+2m)}$.

Proof. We prove the theorem by calculating $MI(m,l)$. When $m > l$, getting an extra answer cannot change the result of the majority voting. Therefore, the marginal income of accuracy is 0. As a result, $MI(m,l) = 0$ when $m > l$.

When $m = l$,

$$
\begin{aligned}
& MI(m,m) = E[A_R|(m+1,m)] - E[A_R|(m,m)] \\
= \; & \frac{\Gamma(a+m+1)\Gamma(b+m)}{\Gamma(a+m+1)\Gamma(b+m) + \Gamma(a+m)\Gamma(b+m+1)} \\
& - \frac{\Gamma(a+m)\Gamma(b+m)}{\Gamma(a+m)\Gamma(b+m) + \Gamma(a+m)\Gamma(b+m)} \\
= \; & \frac{(a+m)\Gamma(a+m)\Gamma(b+m)}{((a+m)+(b+m))\Gamma(a+m)\Gamma(b+m)} - \frac{1}{2} \\
= \; & \frac{a+m}{a+b+2m} - \frac{1}{2} \\
= \; & \frac{a-b}{2(a+b+2m)}
\end{aligned}
$$

□

Note that $a > b$ is guaranteed by the prior distribution. Therefore, $MI(m,m)$ decreases while m increases.

Based on the marginal income of the accuracy, we can further derive the profit $P(m,l)$ of a question in status (m,l). The profit is defined as the maximum economic profit at status (m,l) given the linear model in Subsection 4.4.1. We introduce two more profit functions before formally defining $P(m,l)$, namely the profit $P_S(m,l)$ of stopping the question at status (m,l) and the profit $P_C(m,l)$ of continuing the question at status (m,l). Since a question has only two choices (stopping and continuing) at any status, we therefore formally define the profit at status (m,l) as:

Definition 4.4.

$$P(m,l) = \max\{P_S(m,l), P_C(m,l)\}$$

Intuitively, we should stop the question when $P_S(m,l) >= P_C(m,l)$ because we cannot benefit from this question any more by waiting for more answers.

Now we consider the two profit functions $P_S(m,l)$ and $P_C(m,l)$. Given the accuracy $E[A_R]$, we know that the result is incorrect with probability $1-E[A_R]$. Therefore, according to the linear model, the profit $P_S(m,l)$ satisfies:

$$P_S(m,l) = V_Q - (1 - E[A_R])L_Q - (m+l)C_Q$$

When we continue a question, we will pay one more C_Q and get a new answer from a new worker. Meanwhile, the status is also changed to a new status. It is

Algorithm 1: Generate Linear Strategy Algorithm

```
Input: Parameter $a, b$ of prior distribution, Error Loss $L_Q$, question cost $C_Q$
Output: Strategy $\mathcal{S}$ for each status $(m, l)$
1  $M \leftarrow \lceil \frac{1}{2}(\frac{L_Q(a-b)}{6C_Q} - (a+b)) \rceil$;
2  $\mathcal{S}.M \leftarrow M$;
3  for $i$ from $M$ to $0$ do
4      $\mathcal{S}.(M, i) \leftarrow$ stopping;
5      $\mathcal{S}.P_S(M, i) \leftarrow V_Q - (1 - E[A_R|M, i])L_Q - (M+i)C_Q$;
6      $\mathcal{S}.P(M, i) \leftarrow \mathcal{S}.P_S(M, i)$;
7  for $i$ from $M-1$ to $0$ do
8      for $j$ from $i$ to $0$ do
9          $\mathcal{S}.P_S(i, j) \leftarrow V_Q - (1 - E[A_R|i, j])L_Q - (i+j)C_Q$;
10         $\mathcal{S}.P_C(i, j) \leftarrow$
           $E[Pr(+1|i, j)]\mathcal{S}.P(i+1, j) + E[Pr(-1|i, j)]\mathcal{S}.P(i, j+1) - C_Q$;
11         $\mathcal{S}.P(i, j) \leftarrow \max\{\mathcal{S}.P_S(i, j), \mathcal{S}.P_C(i, j)\}$;
12         if $\mathcal{S}.P_S(i, j) >= \mathcal{S}.P_C(i, j)$ then
13             $\mathcal{S}.(i, j) \leftarrow$ stopping;
14         else
15             $\mathcal{S}.(i, j) \leftarrow$ continuing;
16 return $\mathcal{S}$
```

easy to find that the next status is $(m+1, l)$ with probability $Pr(+1|m, l)$ and $(m, l+1)$ with probability $Pr(-1|m, l)$. Thus, the profit $P_C(m, l)$ is recursively defined as following:

$$\begin{aligned} P_C(m, l) &= E[Pr(+1|m, l)]P(m+1, l) \\ &\quad + E[Pr(-1|m, l)]P(m, l+1) - C_Q \end{aligned}$$

Given the recursive definition, it is difficult to calculate them directly. Now we discuss the condition to guarantee the strategy of one status (m, l) stopping, i.e.

$$P_S(m, l) \geqslant P_C(m, l)$$

m that satisfies $MI(m, m)L_Q < C_Q$ is obvious a lower bound, as we will not get a positive profit in each future step. Moreover, we have found a looser lower bound of m. We have the following theorem:

Theorem 4.5 (Termination Theorem). *A sufficient condition of* $P_S(m, l) \geqslant$

$P_C(m,l)$ is

$$m \geqslant \frac{1}{2}(\frac{L_Q(a-b)}{6C_Q} - (a+b))$$

and $l < m$.

Proof. We define $P'(m,l) = P(m,l) - P_S(m,l)$. Based on this definition, we can rewrite the Definition 4.4 by

$$\begin{aligned} P'(m,l) &= \max\{0, MI(m,l)L_Q + E[Pr(+1|m,l)]P'(m+1,l) \\ &+ E[Pr(-1|m,l)]P'(m,l+1) - C_Q\} \end{aligned}$$

We have the observations $P'(m,l) \geqslant P'(m,l-1)$ and $P'(m,l) \geqslant P'(m+1,l)$ for all $m > l$. For a large enough number m, we have $MI(m,m) < C_Q/L_Q$ since $\lim_{m\to+\infty} MI(m,m) = 0$. Based on $MI(m,m)L_Q < C_Q$, there exist some m such that $P'(m+1,m) = 0$. We can prove the following Lemma:

Lemma 4.1. *$P'(m,m-1) = 0$ if $P'(m+1,m) = 0$ and $m \geqslant \frac{1}{2}(\frac{L_Q(a-b)}{6C_Q} - (a+b))$*

Based on Lemma 4.1, we therefore state that $P'(m,m) = 0$ when $m \geqslant \frac{1}{2}(\frac{L_Q(a-b)}{6C_Q} - (a+b))$. ☐

Proof of Lemma 4.1. *We prove the lemma by contradiction. If $P'(m,m-1) > 0$, we have the following inequality: $MI(m,m-1)L_Q + E[P(+1|m,m-1)]P'(m+1,m-1) + E[P(-1|m,m-1)]P'(m,m) - C_Q > 0$ where $MI(m,m-1) = 0$. Meanwhile, we have*

$$P'(m+1,m-1) <= P'(m+1,m)$$

As a result, $P'(m+1,m-1) = 0$. Obviously, $E[P(-1|m,m-1)] < 1/2$. Therefore, we have $P'(m,m) > 2C_Q$.

Note that

$$P'(m,m) = MI(m,m)L_Q + P'(m+1,m) - C_Q$$

According to the fact that $P'(m+1,m) = 0$, we have

$$P'(m,m) = MI(m,m)L_Q - C_Q$$

Therefore,

$$MI(m,m)L_Q - C_Q > 2C_Q$$

Moreover, $MI(m,m) = (a-b)/(2(a+b+2m))$. As a result, we have $m < \frac{1}{2}(\frac{L_Q(a-b)}{6C_Q} - (a+b))$, which contradicts with $m \geqslant \frac{1}{2}(\frac{L_Q(a-b)}{6C_Q} - (a+b))$.

We have designed the linear model based algorithm (Algorithm 1) to generate the strategy deciding whether the question should stop at each status (m,l). The search space bound M is the upper-bound of m in all possible *continuing* status (m,l) in Theorem 4.5:

$$M = \lceil \frac{1}{2}(\frac{L_Q(a-b)}{6C_Q} - (a+b)) \rceil \tag{4.1}$$

We apply dynamic programming to iteratively compute the decision of the generated strategy $\mathcal{S}$ from the upper-bound back to 0. For each status (m,l), we compute $P_S(m,l)$ and $P_C(m,l)$ based on $P(m+1,l)$ and $P(m,l+1)$. In strategy $\mathcal{S}$, the decision is made for each status (m,l) by comparing $P_S(m,l)$ and $P_C(m,l)$. As a result, the question run that maximizes $P(m,l)$ is found by stopping the question at the first status (m^*,l^*) such that $P_S(m^*,l^*) \geqslant P_C(m^*,l^*)$. Algorithm 1 can be applied to find the question run that maximize $P(m,l)$ for all input 0-1 sequences. Obviously, the time complexity of Algorithm 1 is $O(M^2)$ where M is computed in Equation 4.1. Note that Algorithm pre-computes all possible decisions offline and it only takes $O(M)$ time to make decisions online by querying $S.(m,l)$ in $O(1)$ time for each status (m,l).

4.4.4 Model with Constraints

In this subsection, we discuss the decision-making problem with constraints on the accuracy and cost of the result of the question. Suppose the constraints are represented by $cost \leqslant Budget$. We solve this problem by simply adapting Algorithm 1. We mark the status (m,l) as stopping when $(m+l+1)C_Q > Budget$. This adapted algorithm is outlined in Algorithm 2. We can extend algorithm 2 to support other constraints such as accuracy etc.

In Algorithm 2, the upper bound of search space M is computed in line 1. Line 3-6 pre-computes the strategy for status (M,i). All statuses (M,i) are set to be *stopping*. Line 7-20 the strategy of each status (i,j) is computed. When status (i,j) does not satisfy the constraints, it is set to be *stopping* in line 10-12. Otherwise, $P_S(i,j)$ and $P_C(i,j)$ are computed respectively in line 14-15. The decision of status (i,j) is decided by comparing $P_S(i,j)$ with $P_C(i,j)$ in line

Algorithm 2: Generate Linear Strategy Algorithm with Constraints

```
Input: Parameter $a, b$ of prior distribution, Error Loss $L_Q$, question cost $C_Q$,
       Budget constraint $\theta$
Output: Strategy $\mathcal{S}$ for each status $(m, l)$
1  $M \leftarrow \lceil \frac{1}{2}(\frac{L_Q(a-b)}{6C_Q} - (a+b)) \rceil$;
2  $\mathcal{S}.M \leftarrow M$;
3  for $i$ from $M$ to $0$ do
4      $\mathcal{S}.(M, i) \leftarrow$ stopping;
5      $\mathcal{S}.P_S(M, i) \leftarrow V_Q - (1 - E[A_R|M, i])L_Q - (M + i)C_Q$;
6      $\mathcal{S}.P(M, i) \leftarrow \mathcal{S}.P_S(M, i)$;
7  for $i$ from $M - 1$ to $0$ do
8      for $j$ from $i$ to $0$ do
9          if $(i + j + 1)C_Q > \theta$ then
10             $\mathcal{S}.P_S(i, j) \leftarrow V_Q - (1 - E[A_R|i, j])L_Q - (i + j)C_Q$;
11             $\mathcal{S}.P(i, j) \leftarrow \mathcal{S}.P_S(i, j)$;
12             $\mathcal{S}.(i, j) \leftarrow$ stopping;
13         else
14             $\mathcal{S}.P_S(i, j) \leftarrow V_Q - (1 - E[A_R|i, j])L_Q - (i + j)C_Q$;
15             $\mathcal{S}.P_C(i, j) \leftarrow$
               $E[Pr(+1|i, j)]\mathcal{S}.P(i + 1, j) + E[Pr(-1|i, j)]\mathcal{S}.P(i, j + 1) - C_Q$;
16             $\mathcal{S}.P(i, j) \leftarrow \max\{\mathcal{S}.P_S(i, j), \mathcal{S}.P_C(i, j)\}$;
17             if $\mathcal{S}.P_S(i, j) >= \mathcal{S}.P_C(i, j)$ then
18                 $\mathcal{S}.(i, j) \leftarrow$ stopping;
19             else
20                 $\mathcal{S}.(i, j) \leftarrow$ continuing;
21 return $\mathcal{S}$
```

17-20. The computed strategy $\mathcal{S}$ is returned as the result of Algorithm 2 (line 21).

4.5 Non-linear Decision-Making

In this section, we discuss a non-linear model based approach to predict the relationship between the accuracy and the cost of the question without prior knowing L_Q. We have presented the decision making algorithm in Section 4.4 for a single question, where the job value is a linear function of the accuracy. However, considering the case of making decisions on a batch of questions, the value function might be more complicated than a linear function. For example,

Algorithm 3: Accuracy-Cost Algorithm

```
Input: Parameter $a, b$ of prior distribution, question cost $C_Q$
Output: List of tuples of accuracy, cost and error loss $\{\langle accu, cost, L_Q\rangle\}$
1  result $\leftarrow \emptyset$;
2  $L_Q \leftarrow 1000C_Q$;
3  while $L_Q > 0$ do
4      $\mathcal{S} \leftarrow$ GenerateLinearStrategy($a, b, L_Q, C_Q$);
5      accu $\leftarrow$ ComputeAccuracy($\mathcal{S}$);
6      cost $\leftarrow$ ComputeCost($\mathcal{S}$);
7      result $\leftarrow$ result $\cup\langle$accu[0][0], cost[0][0], $L_Q\rangle$;
8      $max \leftarrow 0$;
9      foreach Non-stop status $(m, l)$ in $\mathcal{S}.status$ do
10         $c \leftarrow$ cost$[m][l]$;
11         $l_Q \leftarrow cL_Q/(P_C(m,l) - P_S(m,l) + c)$;
12         if $l_Q > max$ then $max \leftarrow l_Q$ ;
13         ;
14     $L_Q \leftarrow max$;
15 return result;
```

we list three non-linear functions in Figure 4.2, representing the cases that we have constraints on the quality of the data and we use the information entropy to measure the informativeness of the data. Therefore, the value functions are not linear. Moreover, the non-linear model based method is also driven by situations where the customers are not able to estimate the error loss function L_Q of some problems. The method we have discussed in Section 4.4 cannot be simply applied to make decisions for the question in the above situations. Since the quality of data can be well estimated by the accuracy when the number of questions is large enough, we propose Algorithm 3 to find the relationship between accuracy and cost (shown in the bottom of Figure 4.2) in Subsection 4.5.1. Algorithm 3 calculates the difference between the value function and the cost function as the profit function (as illustrated on the right hand side of Figure 4.2). Thus, the maximum point of the profit function (star point in Figure 4.2) is the trade-off point to maximize the profit. We extend our method to solve the problem of decision making for multiple questions in Subsection 4.5.2.

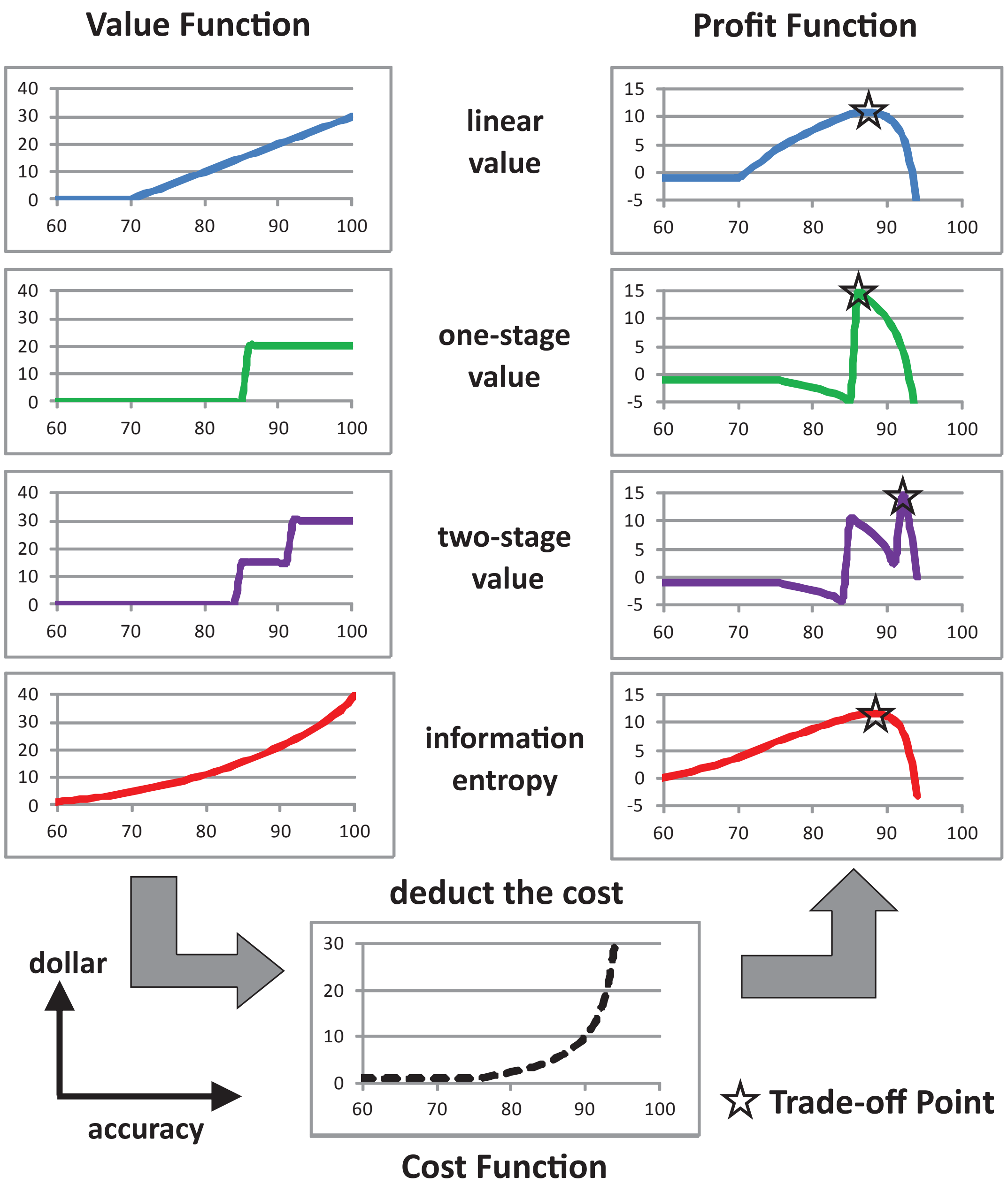


Figure 4.2: Examples of non-linear decision making

4.5.1 Accuracy-Cost Relationship

To obtain the cost function of accuracy, we propose the non-linear model based algorithm (Algorithm 3) by iteratively applying Algorithm 1. According to the Algorithm 1, we know that given fixed a, b, L_Q/C_Q, we have a determined strategy to find the question run that maximizes the profit.

The basic idea of this non-linear model based algorithm is to:

Step 1. initialize L_Q to be a sufficiently large value (i.e. $1000C_Q$) and compute the accuracy and cost based on L_Q.

Step 2. iteratively adjust the value of L_Q by reducing L_Q such that the decision of only one status (m, l) changes from *continuing* to *stopping*.

Step 3. compute accuracy and cost and go back to Step 2.

This method is to enumerate the accuracy and cost pairs by gradually reducing the value of L_Q. We show the detailed method in Algorithm 3. In Algorithm 3, we record the expected accuracy and cost for each status (m, l) using the strategy generated from L_Q. Algorithm 3 iteratively reduces L_Q by selecting the maximum l_Q that can change exactly one *continuing* status to *stopping*. l_Q is computed in line 11. The correctness is guaranteed by the following lemma:

Lemma 4.2. *Given a continuing status (m, l), we reduce L_Q to be $cost[m][l]$ $L_Q/(P_C(m, l) - P_S(m, l) + cost[m][l])$. If the stopping/continuing status of all status (m', l') other than (m, l) are not changing, we have $P_C(m, l) = P_S(m, l)$.*

Proof of Lemma 4.2. *By the definition of the two profits, i.e. $P_S(m, l)$ and $P_C(m, l)$:*

$$\begin{aligned} P_S(m, l) &= V_Q - L_Q + E[A_R|m, l]L_Q - (m + l)C_Q \\ P_C(m, l) &= V_Q - L_Q + accu[m][l]L_Q - (m + l)C_Q \\ &\quad -cost[m][l] \end{aligned}$$

We therefore have

$$P_S(m, l) - P_C(m, l) = L_Q\Delta accu - cost[m][l]$$

where $\Delta accu = accu[m][l] - E[A_R|m, l]$. We assume that $l_Q\Delta accu - cost[m][l] = 0$. Thus,

$$l_Q = \frac{cost[m][l]}{\Delta accu} = \frac{cost[m][l]L_Q}{P_C(m, l) - P_S(m, l) + cost[m][l]} \qquad \square$$

The strategy is generated according to the newly updated L_Q. We employ dynamic programming method to compute the accuracy and cost in Algorithm 4 and 5. These two algorithms compute the accuracy and cost respectively for each status (m, l) from (M, M) (*stopping*) back to $(0, 0)$.

We formally define the non-linear value function problem as:

Algorithm 4: Compute Accuracy

```
Input: Strategy $\mathcal{S}$
Output: Expected accuracy accu of the problem in every status $(m, l)$ using
        Strategy $\mathcal{S}$
1 $M \leftarrow \mathcal{S}.M$;
2 for $i$ from $M$ to $0$ do
3     for $j$ from $i$ to $0$ do
4         if $(i, j)$ is stopped then
5             accu$[i][j] \leftarrow E[A_R|i, j]$;
6         else
7             accu$[i][j] \leftarrow Pr(+1|i, j)$accu$[i + 1][j] + Pr(-1|i, j)$accu$[i][j - 1]$;
8 return accu
```

Algorithm 5: Compute Cost

```
Input: Strategy $\mathcal{S}$
Output: Expected accuracy cost of the problem in every status $(m, l)$ using
        Strategy $\mathcal{S}$
1 $M \leftarrow \mathcal{S}.M$;
2 for $i$ from $M$ to $0$ do
3     for $j$ from $i$ to $0$ do
4         if $(i, j)$ is stopped then
5             cost$[i][j] \leftarrow (i + j)C_Q$;
6         else
7             cost$[i][j] \leftarrow Pr(+1|i, j)$cost$[i + 1][j] + Pr(-1|i, j)$cost$[i][j - 1] + C_Q$;
8 return cost
```

Definition 4.5. *Given the functions of the value of question quality $f^V(accu)$ and cost $f^C(accu)$ with respect to the accuracy of the result, find the strategy to maximize $f^P(accu) = f^V(accu) - f^C(accu)$ on each possible question run.*

Algorithm 3 provides the accuracy-cost relationship $f^C(accu)$, which makes it possible to find a trade-off point when user requirements $f^V(accu)$ is clear. Moreover, this can be used to give customer the accuracy and cost predictions early and help the user to choose suitable requirements like $f^V(accu)$ or special point of $\langle accu, cost, L_Q\rangle$.

The solution of those non-linear $f^V(accu)$ requirements can be done as follows:

1. Compute $result = \{\langle accu, cost, L_Q\rangle\}$ by calling Algorithm 3.

2. Find $\max_{accu \in result}\{f^V(accu) - f^C(accu)\}$ by computing the difference between the non-linear value and cost for each *accu* in the *result*. Generate the strategy using the corresponding L_Q of *accu* as parameter of Algorithm 2.

Notice that in general case the $f^P(accu)$ is not convex (e.g. the profit function for stage value). As a result, we have to calculate every point for computing $f^P(accu)$.

This method can also be applied to solve the problem with constraints on the expectation of accuracy or cost. We only need to modify the non-linear functions $f^V(accu)$ to present the constraints (e.g.stage value function gives a strict constraint of accuracy).

4.5.2 Decision-Making for Multiple Questions

We discuss the question dispatching algorithm (Algorithm 6) in this subsection. This algorithm aims to build a question dispatcher that assigns HITs containing a batch of questions to the workers. Batching questions in a HIT is an effective approach to reduce the average cost of each question. However, our proposed algorithm only generates strategies for a single question. Questions may be included multiple times in different HITs and their decisions are made in real-time. The numbers of required workers are different among the questions. Therefore, we need to design an algorithm to dynamicly batch non-stopped questions in a HIT. The question dispatching algorithm is designed in order to reduce the rate of already stopped questions in the HITs.

The key idea of our question dispatching algorithm is to manage all the questions to be finished at almost the same time. We store all the unfinished questions in a question pool and maintain the number of questions included in the HITs. Attributed to the fact that the randomness of receiving order of the answers, it is difficult to design a deterministic method to find the best assignment of questions. Instead, we maintain the expected number of asking questions from current status such that these numbers are synchronously decreased. As a result, we put questions with the largest expected number of asking questions into a HIT.

We outline the function of dispatcher and updater in Algorithm 6. For the question, the number of expected asking questions C_i at status $(Q_i.m, Q_i.l)$ is retrieved from $cost[Q_i.m][Q_i.l]$ computed by Algorithm 5. Meanwhile, there

Algorithm 6: Question Dispatching Algorithm

```
1  Dispatcher:
2  Initialize min-heap heap to be empty;
3  foreach non-stop question $Q_i$ do
4      $C_i \leftarrow cost[0][0]$;
5      $P_i \leftarrow 0$;
6      $E_i \leftarrow C_i - P_i$;
7      if heap.size()¡k then
8          heap.push($Q_i$);
9  while not all questions are stopped and a new worker comes do
10     foreach question $Q_i$ in heap do
11         Assign $Q_i$ to the HIT; $P_i \leftarrow P_i + 1$;
12         $E_i \leftarrow C_i - P_i$;
13     foreach non-stop question $Q_i$ in heap do
14         Maintain heap using $E_i$;
15
16 Updater:
17 while any question $Q_i$ gets into a stopping status $(Q_i.m, Q_i.l)$ do
18     Mark $Q_i$ as a stopped question.
19 while any question $Q_i$ gets a new answer to the status $(Q_i.m, Q_i.l)$ do
20     $P_i \leftarrow P_i - 1$;
21     $C_i \leftarrow cost[Q_i.m][Q_i.l]$;
22     $E_i \leftarrow C_i - P_i$;
23     Maintain heap using $E_i$;
```

are some HITs on the worker's hand. We record the number of HITs that are posted but have not yet received answers of each question Q_i as P_i. We use $E_i = C_i - P_i$ as the estimation of expected asking questions. When a HIT is posted or its answer is received, the expectations of effected questions are updated and the questions with the largest expected numbers are maintained with a min heap.

4.6 Experimental Studies

This section will discuss the experiment results of our methods. To evaluate the performance of our proposed methods, we have conducted extensive experiments on two real-world datasets on the AMT. Besides, various robustness tests and theoretical results are studied using synthetic datasets. We show that our

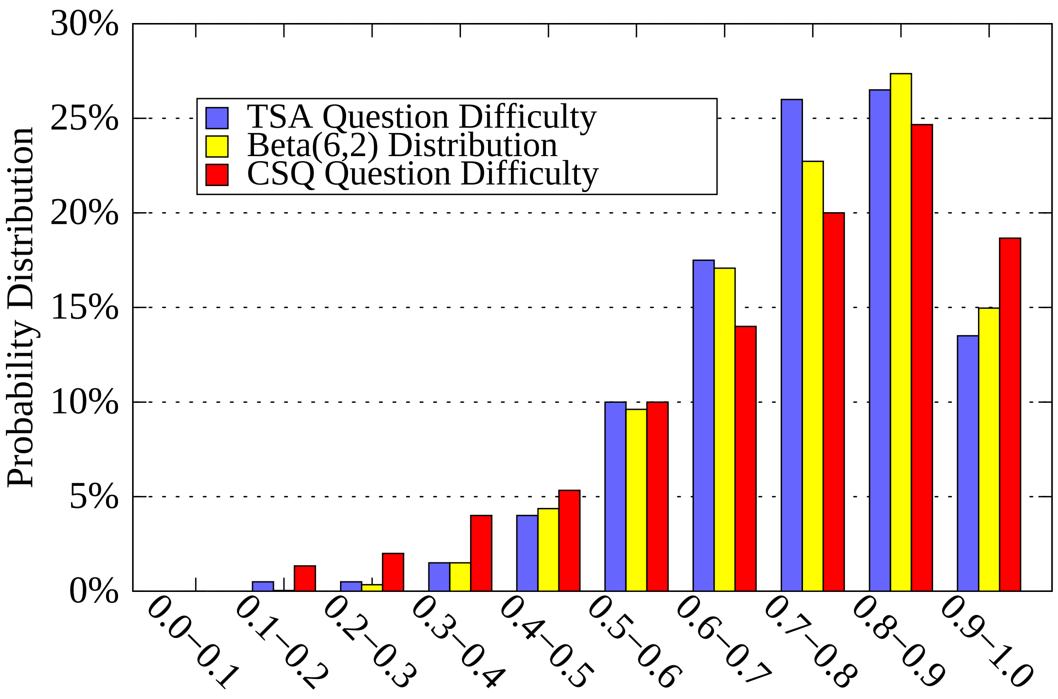


Figure 4.3: Distribution of the difficulty of problems vs. B($\mu|6, 2$) distribution

proposed method: (1) works better than any other existing method in terms of both the accuracy and the cost; (2) is robust to handle questions with diversified difficulty distribution and unexpected data quality; (3) is scalable when the maximum number of workers is increased; (4) works well on various kinds of crowdsourcing questions.

4.6.1 Experiment Setup

We use the following two datasets for our performance studies, namely tweet sentiment analysis (*TSA*) dataset and common sense question (*CSQ*) dataset. Humans are good at comprehension and perform well on problems requiring background knowledge. These two datasets focus on the two main advantages of crowdsourcing respectively.

TSA dataset: A real-world tweets dataset containing 400 comments of 20 movies is crawled from Twitter. We generate a sentiment analysis question (positive, negative) for each of the comments as a candidate crowdsourcing task. We assign each of the 400 questions to up to 50 workers using the question dispatching algorithm (Algorithm 6). We repeat the question dispatching

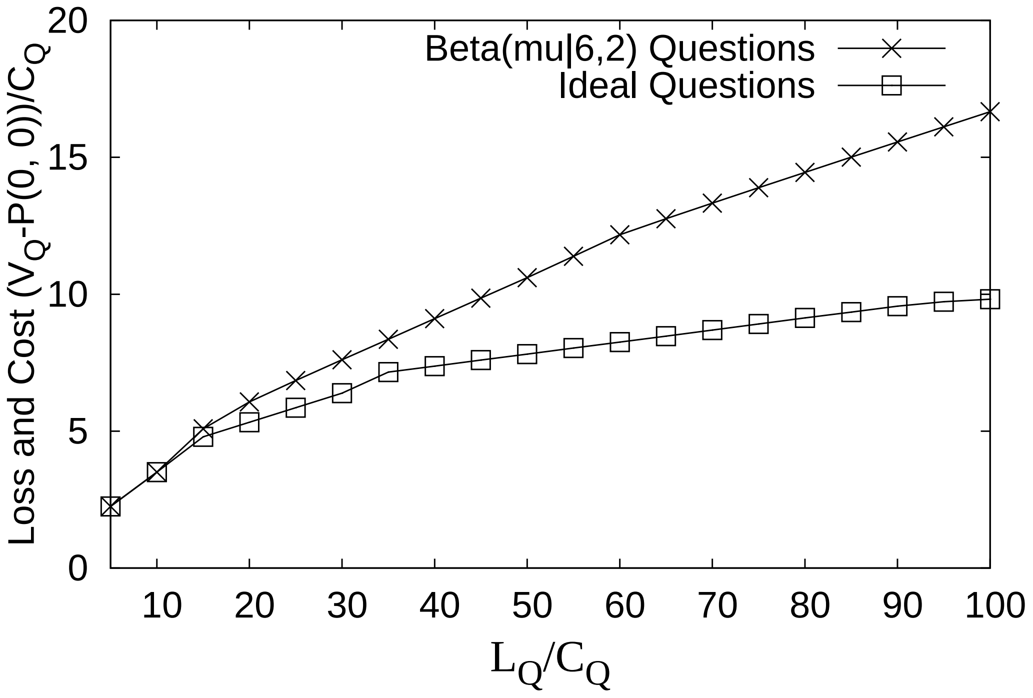


Figure 4.4: Lower bound of V_Q

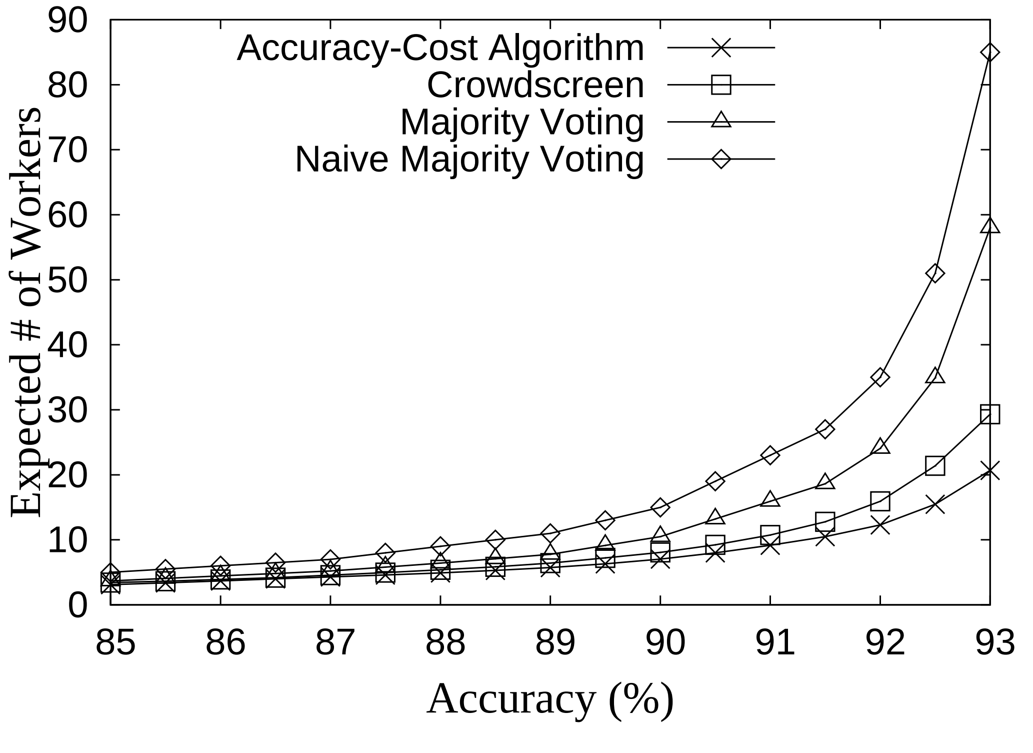


Figure 4.5: Expected number of workers given required accuracy

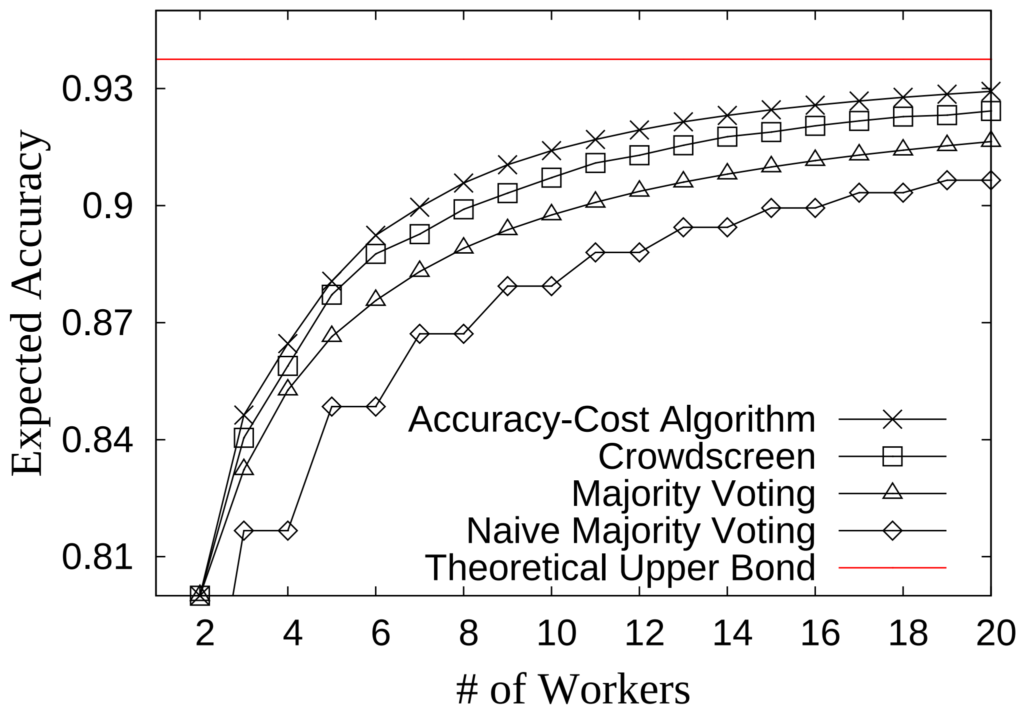


Figure 4.6: Expected accuracy given the number of workers

algorithm 10 times to get 10 different question runs for each question. In total, we get 200,000 answers from the workers as the *TSA* dataset.

CSQ dataset: We crawl 400 common sense problems from the Internet as the candidate crowdsourcing tasks. We also assign each of the 400 questions to up to 50 workers by repeatedly using Algorithm 6 10 times. In total, we also get 200,000 answers from the workers as the *CSQ* dataset.

Figure 4.3 compares the distribution of the difficulty of the questions with $\mathrm{B}(\mu|6,2)$ distribution. The average accuracy of results A_R is tested on more than 200 answers. This figure indicates that $\mathrm{B}(\mu|6,2)$ distribution can be well used to model the distribution of the difficulty of the Tweet sentiment analysis questions. Moreover, the results also show that about 6.8% of questions have an average A_R smaller than 0.5, which fits the prediction well. The predicted value of the proportion is 6.25% based on $\mathrm{B}(\mu|6,2)$ distribution. The accuracy of these questions become even worse when more workers answer them. As a result, the overall accuracy of results on all questions can only achieve 93%-94% rather than very close to 100%. We also observe that the questions in both datasets are a bit more difficult than the expectation based on the prior

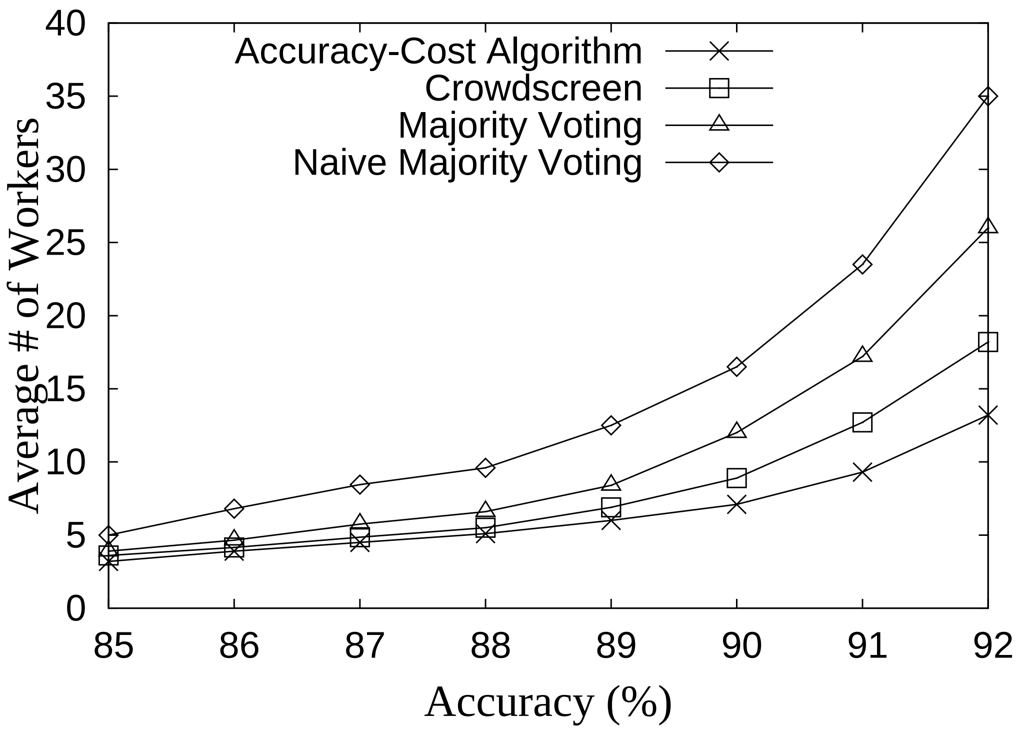


Figure 4.7: Empirical number of workers given required accuracy on TSA dataset

B(μ|6, 2) distribution.

To compare the performance of proposed algorithm with other existing algorithms, we implement four algorithms, i.e. Accuracy-Cost (Algorithm 3), Crowdscreen [80], Majority Voting and Naive Majority Voting Algorithm. In the majority voting algorithm, when the number of providers of a value is more than a half of the maximum number of workers, the online majority voting algorithm outputs this value and stops, whereas in the offline naive majority voting algorithm, all the values from all workers are collected and output the majority results.

4.6.2 Problem-Crowdsourcing Fitness

In this subsection we give the analysis on whether a job is appropriate for crowdsourcing or not. We use C_Q as the unit. The results in Figure 4.4 show that the lower bound of V_Q to get benefit from crowdsourcing job in various L_Q. The loss and cost $V_Q - P(0,0)$ means the total cost of solving a question by crowdsourcing (notice that $P(0,0)$ contains V_Q in it, this measure only contains

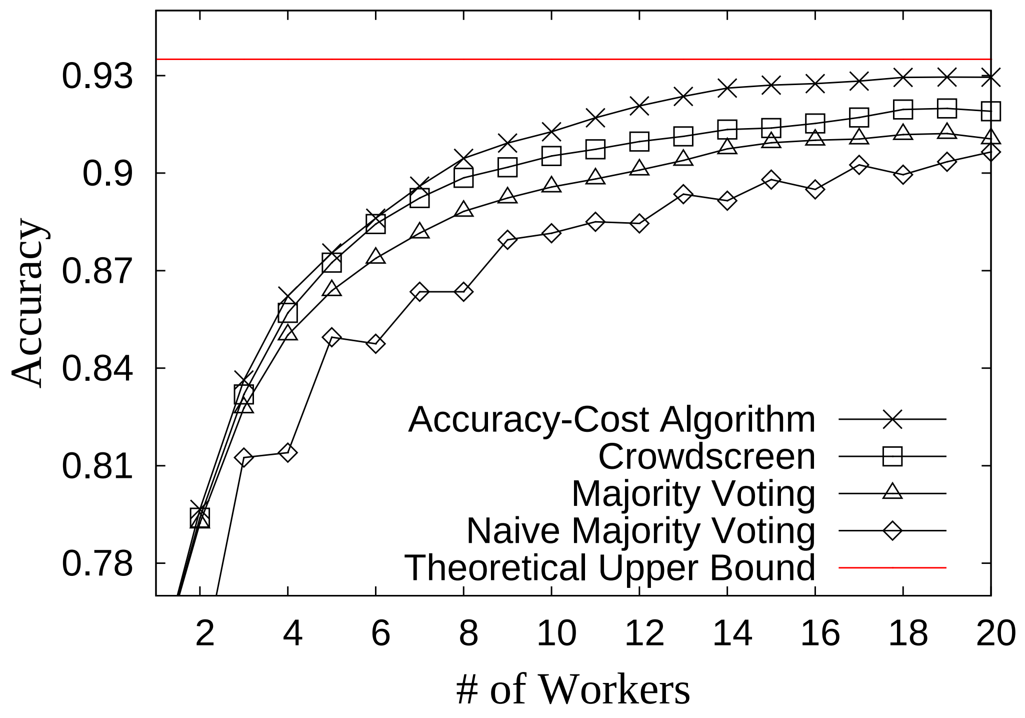


Figure 4.8: Empirical accuracy given the number of workers on TSA dataset

the error loss expectation and questions cost and is irrelevant to V_Q). For those jobs where L_Q is too high or V_Q is not enough, crowdsourcing is not a cost-effective way. We compare the loss and cost on two set of questions, namely the questions obeying $\mathrm{B}(\mu|6,2)$ distribution and the questions having the ideal same $A_Q = 0.75$.

4.6.3 Performance of Accuracy-Cost Algorithm

We test the performance of our Accuracy-Cost Algorithm by investigating the accuracy-cost relationship on both datasets *TSA* and *CSQ*. The results of the number of workers given required accuracy and the accuracy given the number of workers are showed here. We compare the four algorithms, namely Accuracy-Cost, CrowdScreen, Majority Voting and Naive Majority Voting. All four algorithms make the trade-off according to their strategy.

Figure 4.5 and Figure 4.6 present the theoretical predictions of the number of workers given a required accuracy and the accuracy given the number of workers respectively based on the $\mathrm{B}(\mu|6,2)$ distribution model.

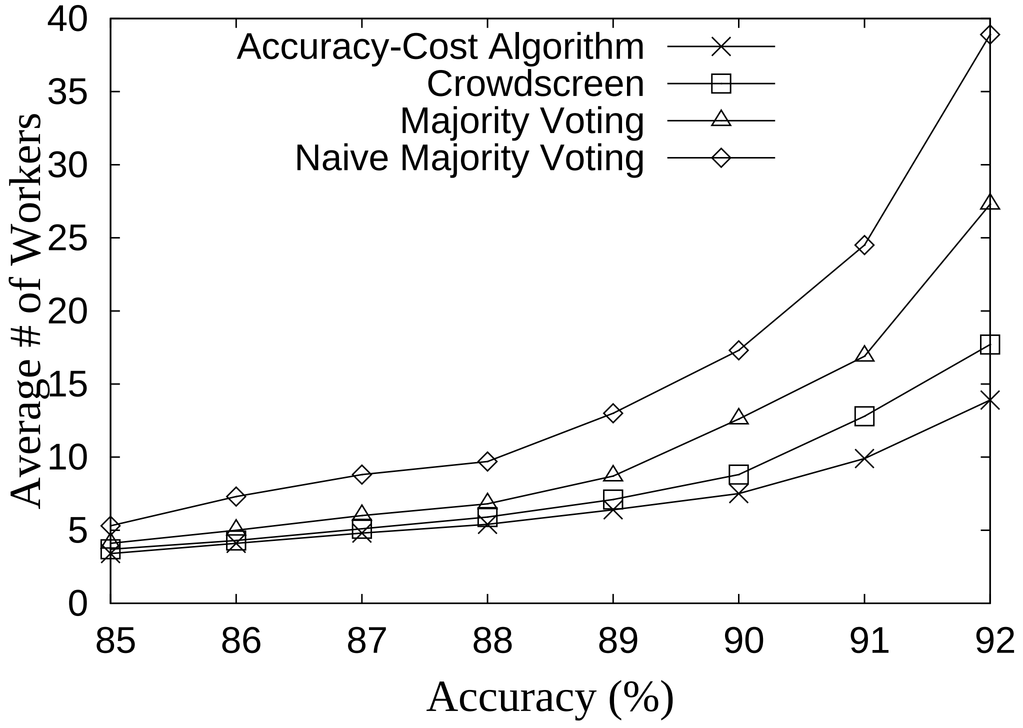


Figure 4.9: Empirical number of workers given required accuracy on CSQ dataset

Experimental results on the *TSA* dataset: Figure 4.7 shows the empirical number of workers hired in the crowdsourcing system. In Figure 4.7, the results show that our Accuracy-Cost Algorithm needs the smallest number of workers and Naive Majority Voting needs the largest number. Meanwhile, comparing Figure 4.7 with Figure 4.5, the empirical results validate the theoretical predictions. Figure 4.8 shows the empirical accuracy of the crowdsourcing tasks on the *TSA* dataset. The results in Figure 4.8 show that our Accuracy-Cost Algorithm has the highest accuracy while Naive Majority Voting has the lowest. The empirical results also fit the theoretical predictions well by comparing Figure 4.8 with Figure 4.6. Our Accuracy-Cost Algorithm outperforms other algorithms because our Accuracy-Cost Algorithm computes the maximum accuracy for each possible cost and the smallest cost for each possible accuracy (in Algorithm 3).

Experimental results on the *CSQ* dataset: Figure 4.9 shows the empirical number of workers hired in the crowdsourcing system. We observe similar trends in Figure 4.9, i.e. the number of workers of Accuracy-Cost Algorithm is

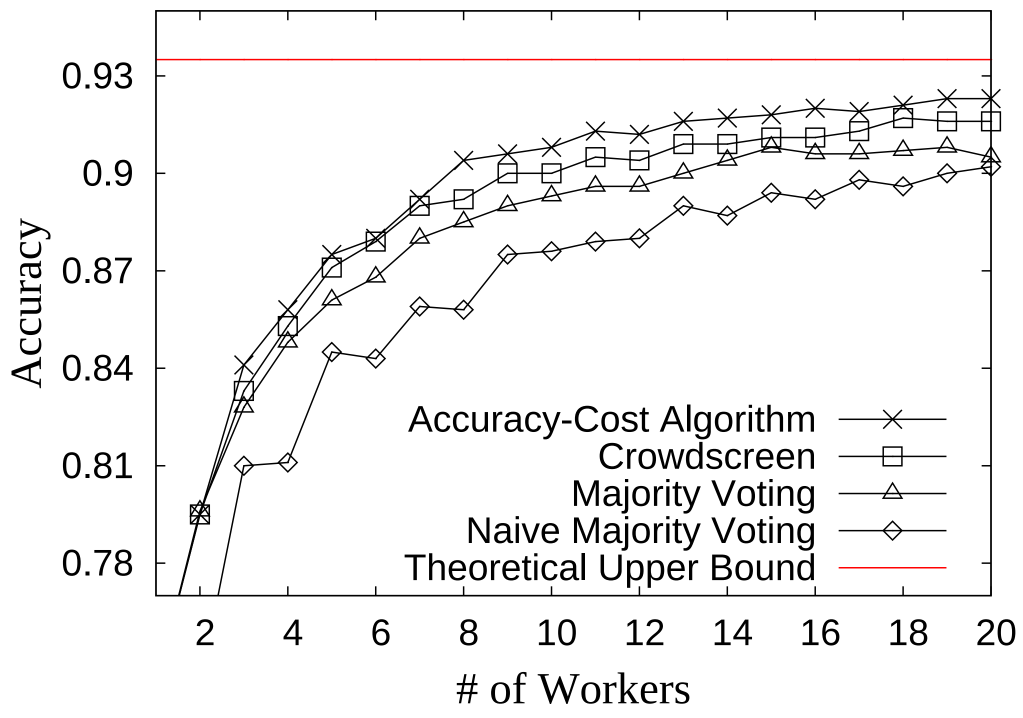


Figure 4.10: Empirical accuracy given the number of workers on CSQ dataset

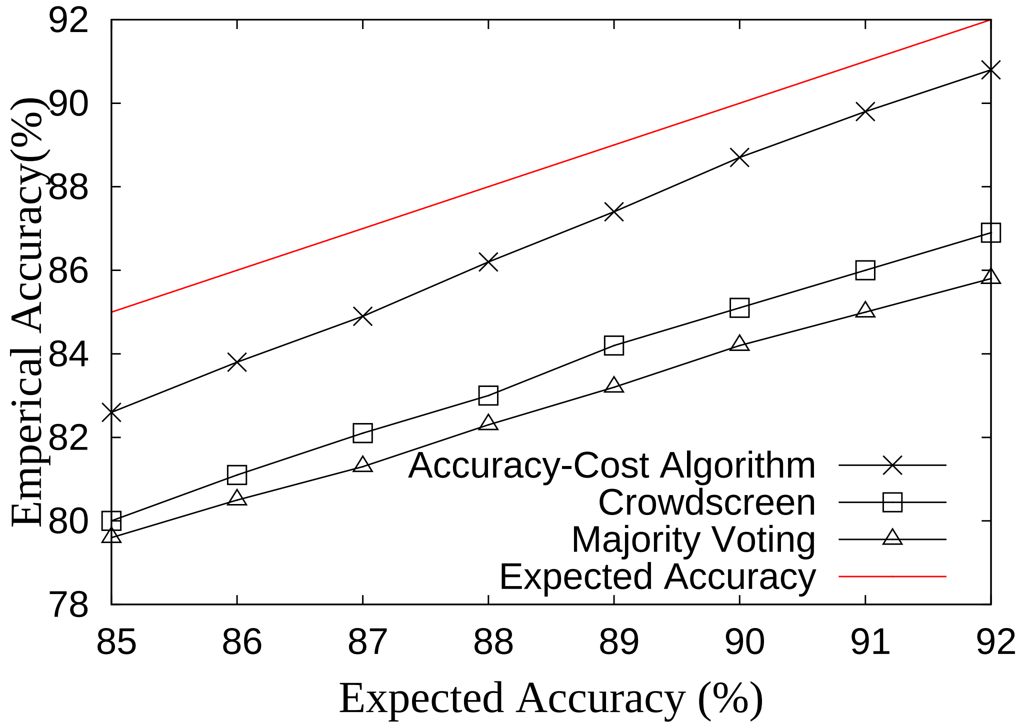


Figure 4.11: Accuracy of the algorithms on hard questions

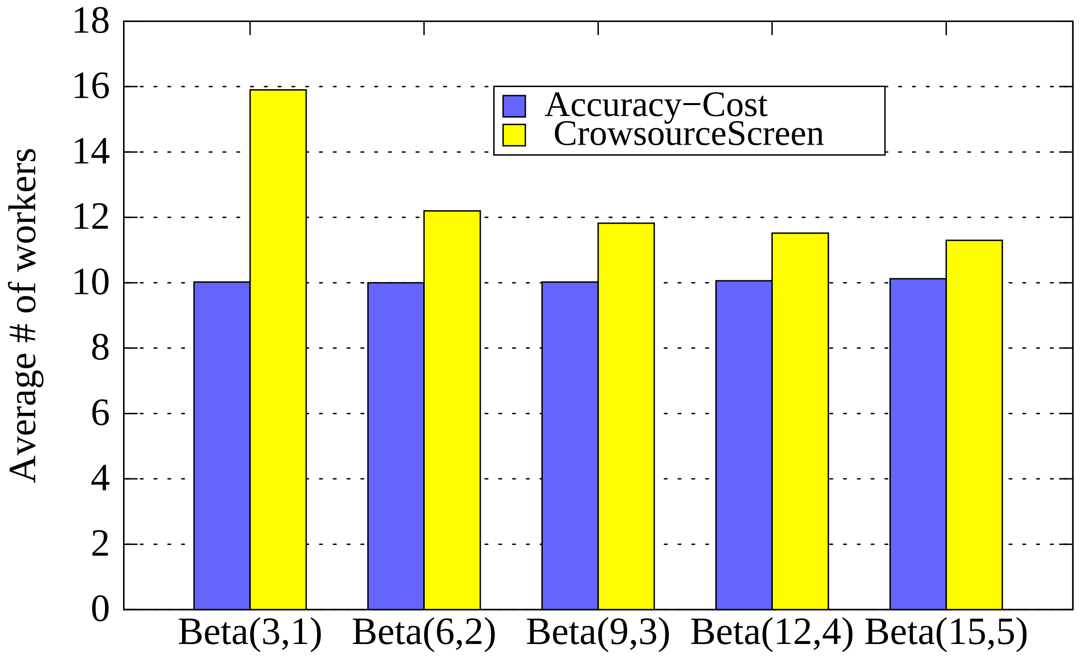


Figure 4.12: Vary the variance of difficulty of questions

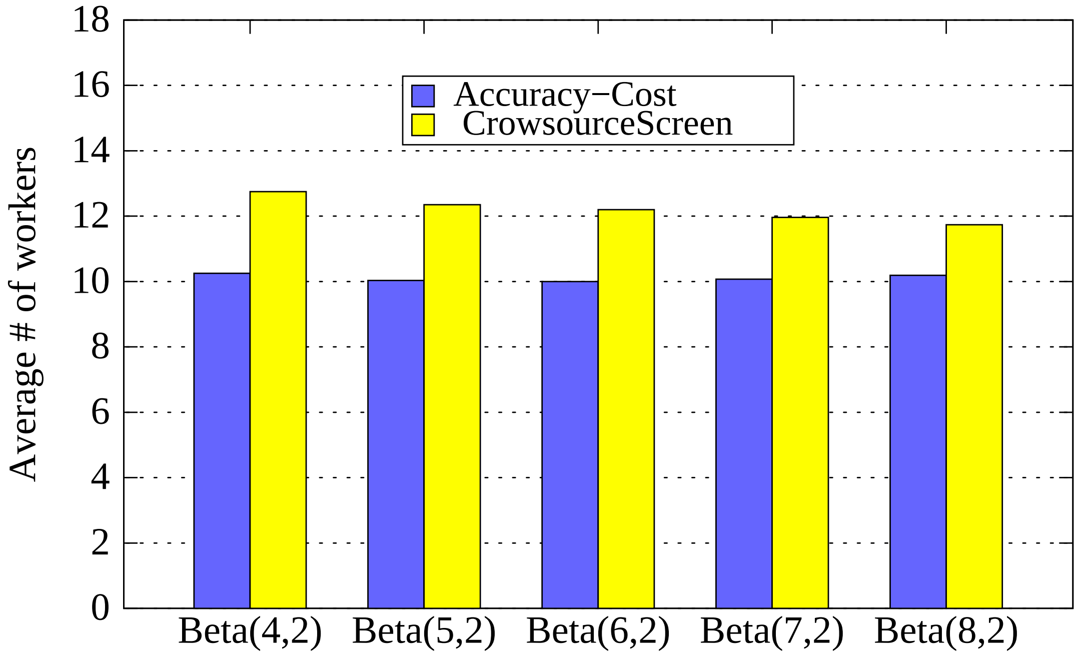


Figure 4.13: Vary the expectation of difficulty of questions

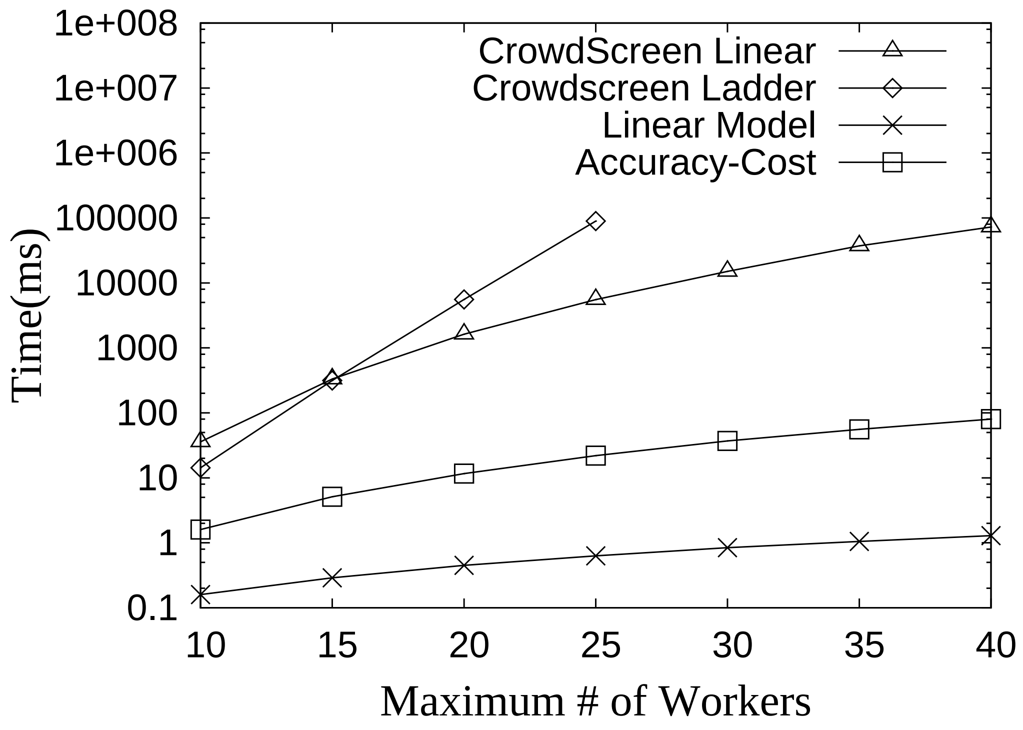


Figure 4.14: Strategy Generating Time

the smallest. The empirical results in Figure 4.9 also fit the theoretical prediction of the results in Figure 4.5 well. Figure 4.10 shows the empirical accuracy of the crowdsourcing tasks on the *CSQ* dataset. The results in Figure 4.10 are similar to the results in Figure 4.6.

The experimental results on these two datasets show that our method can be applied on various crowdsourcing questions and yield good accuracy while requiring the least number of workers.

4.6.4 Robustness

We study the robustness of our algorithm by varying the distribution of question difficulty. We vary variance and expectation of the difficulty distribution of questions such that the distribution is different from our prior $\mathrm{B}(\mu|6, 2)$ distribution. The robustness of obtaining satisfied results on unexpected hard questions (or low quality users) is another key requirement, since low data quality without expectation is usually unacceptable.

Figure 4.11 demonstrates the empirical accuracy of our Accuracy-Cost Algorithm, CrowdScreen and Majority Voting working on unexpected low quality answers with an average 65% accuracy. Note that these four algorithms expect the difficulty of questions to be 75% based on the prior $\mathrm{B}(\mu|6,2)$ distribution. Figure 4.11 shows that our algorithm still produces results with accuracy very close to the accuracy required by the customer while the other two algorithms fail to obtain results with high accuracy. This phenomenon is due to the fact that our algorithm models the probability of a worker providing an answer as a random variable. This property provides the ability to detect the decrease of accuracy and guarantee high quality results by asking more questions automatically. We can see that with the growth of the number of workers(i.e. the increase of accuracy requirement), our method obtains more answers and produces more precise estimation, which results in stronger resistance.

Figure 4.12 and Figure 4.13 show the performance when we vary the difficulty distribution of questions. In both experiments algorithm with $\mathrm{B}(\mu|6,2)$ distribution and CrowdScreen algorithm are compared to the algorithm using the exact prior distribution (our algorithm outputs the best strategy when data exactly match the prior distribution). In our experiments, we use the accuracy that the exact prior distribution algorithm using 10 workers can achieve as the accuracy requirements. We report the number of needed workers to achieve the accuracy requirements for both algorithms. Figure 4.12 reports the results of varying the variance of the distribution and Figure 4.13 reports the results of varying the expectation of the distribution. The results in these two figures indicate that our Accuracy-Cost algorithm needs almost the same number (less than 2% increment) of workers as the exact prior distribution algorithm when we vary the difficulty distribution of the questions, while Crowdscreen algorithm needs more workers when the variance of the distribution is increased.

Instead of using a fixed value, this random variable based probability model is more robust.This implies that our algorithm can by applied to solve unexpected hard questions or working on data sources with unpredictable quality in real crowdsourcing applications.

4.6.5 Scalability and Question Dispatching

In this subsection, we discuss the scalability of our algorithm when the number of workers and the number of questions increases.

# of questions in a HIT	400	200	100	50	20	10
Effective questions rate using question dispatcher	48.20%	88.40%	94.60%	98.40%	99.50%	99.80%
Effective questions rate by randomly assigning questions	48.20%	77.50%	87.30%	93.60%	96.70%	98.40%
Average HIT finish time (s)	2850	1472	765	393	189	113
Average cost per question (0.01 USD)	0.396	0.409	0.425	0.437	0.526	0.627
Average cost per effective question (0.01 USD)	0.822	0.463	**0.449**	**0.444**	0.529	0.628

Table 4.3: Performance of Question dispatching algorithm (2 USD per hour per worker on average)

We first report the results on the running time of our algorithms with respect to the number of workers in Figure 4.14. We compare both our linear and non-linear algorithms with the CrowdScreen's linear and ladder algorithms. Note that the time reported is the offline strategy generating time. The results show that our algorithms are scalable when the number of workers is increased. The linear strategy generating algorithm takes 1.29 milliseconds and non-linear algorithm takes 80.06 milliseconds when there are 40 workers. The results also show that the two CrowdScreen algorithms need to take a lot more time to generate the strategy.

We report the performance of the question dispatching algorithm in Table 4.3. In this experiment, the maximum number of questions in a HIT is varied from 400 to 10. The measurements contain the average percentage of valid questions in each HIT, average finishing time, average cost per question and average cost per effective question. The valid question refers to the not yet stopped questions in the HIT. We compare the average percentage of valid questions of our question dispatching algorithm to that of randomly assigning questions. The results show that our question dispatching algorithm assigns up to 10.9% and on average 4.53% more valid questions in each HIT than randomly assigning algorithm.

4.7 Summary

Crowdsourcing has attracted a great deal of interest in solving challenging problems by integrating human intelligence with algorithms. However, the system cannot be applied with unreliable data quality. Moreover, it is even harder to decide whether a problem is suitable to be solved by crowdsourcing. In this chapter, we propose an online cost sensitive decision making method with novel data quality estimation. To show the effectiveness and efficiency of our method, we conduct extensive experiments over two real datasets on the

Amazon Mechanical Turk. The experimental results show that our proposed method achieves a better accuracy-cost performance than all the existing methods. Moreover, our method is both scalable and robust such that it outputs reliable answers with diversified crowdsourcing data quality.

CHAPTER 5

SIMILARITY PRESERVED HASHING REPRESENTATION

To reduce the data volume and selecting most important features, training data are usually represented by hashing or quantization codes. Most existing methods are focusing on preserving internal information inside each high-dimensional data. However, the similarity relations between data instances also play an important role in lots of analytics tasks such as data clustering, data sampling, casual discovery and etc. Therefore, during data representation, similarity relations are expected to be preserved, and similarity search in the data representation should be accurate and efficient. Locality Sensitive Hashing (LSH) has been widely accepted as an effective hash method for high-dimensional similarity search. However, data are typically not distributed uniformly over the space, and as a result, the buckets of LSH are unbalanced, causing the performance of LSH to degrade.

In this chapter, we propose a new and efficient method called *Data Sensitive Hashing* (DSH) to address this drawback. DSH improves the hashing functions and hashing family by creating a set of hash codes via boosting. DSH leverages data distributions and is capable of directly preserving the nearest neighbor relations. We provide the theoretical guarantee of DSH, and demonstrate its efficiency experimentally.

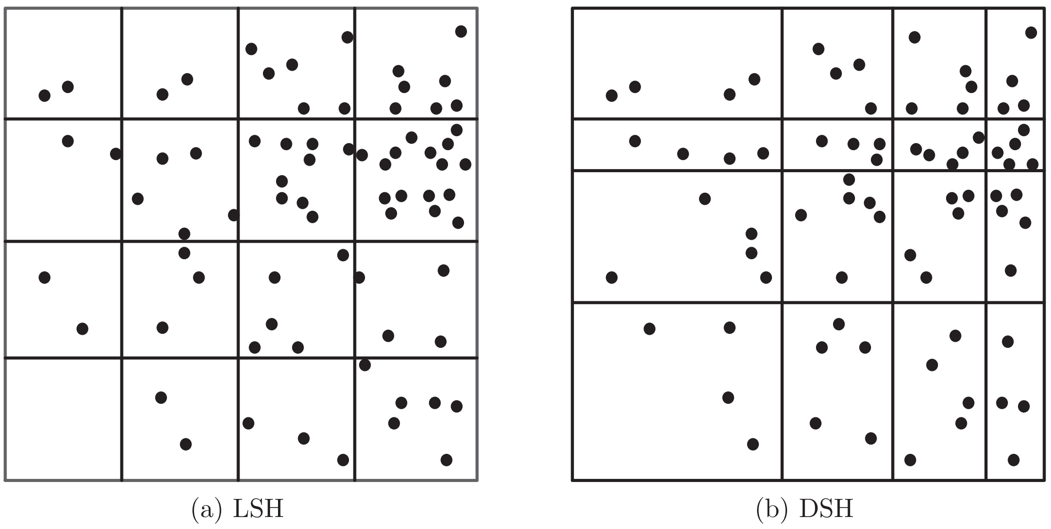

(a) LSH (b) DSH

Figure 5.1: Motivation

5.1 Introduction

In a wide range of applications, data can be represented as points in a multi-dimensional space. For example, feature vectors are often used to represent multi-media data such as images and music. Similarly, a company may use a number of attributes for profiling each customer, or for each product. Each attribute, or each element of a feature vector, can be considered as a dimension in a multi-dimensional space in which each object is a point. Similarity search consequently is transformed to finding points in this multi-dimensional space that are close to a given query point. In many applications, we may be interested in data points that are within some distance of a query point. However, we may not have a meaningful way of setting a distance threshold, and instead we seek the k-nearest points (e.g. the results displayed in the first page of search engine). Therefore, it is essential to support efficient k-nearest neighbor (k-NN) search.

Access methods in the multi-dimensional space, including the k-NN problem, have been studied extensively. However, most of them suffer from the so-called curse of dimensionality and demonstrate poor performance when the number of dimensions is high [110]. One technique that shows promise in high-dimensional spaces is locality sensitive hashing (LSH) [32]. LSH relaxes the k-NN problem to a *c-approximate k-NN problem* that aims to find k points within distance $c \times R$ where R is the maximum distance between the query point and its k-nearest neighbors. In essence, LSH solves k-NN problem by first obtaining a set of *similar points* to q and then extracting the k-nearest points from these similar points. For this purpose, LSH designs a novel scheme to hash the points so that the possibility of collision is much higher for similar points than for dissimilar points. Similar objects are then identified by examining a certain number of points that collide with the query point. However, defining similar points is challenging, and as such LSH simply adopts a constant value r to define two points as similar if their distance is no greater than r. In essence, LSH searches for near neighbors within a radius r. For k-NN search, LSH performs well in a uniform data distribution setting in which a good-quality r can be derived. When the data are skewed, the distance between k-NN pairs can vary greatly, using a consistent r to define all similar points is inadequate. It is for this reason that LSH performance suffers. Figure 5.1a shows an example of the hashing results of LSH. On the indexing level, we see that the hashing is unbalanced, where some buckets are empty while a few buckets contain too many points. From the k-NN perspective, as the distance between k-nearest neighbor pairs can vary greatly, LSH solutions either have to maintain a huge set of indexes for different values r, or use one (or a few fixed) r which may lead to arbitrary bad results. More discussion will be given in Section 5.2.

Recently, there has been increasing attention focused on designing new indexes [102] or query strategies [66, 30] to alleviate the limitations of LSH. However, to the best of our knowledge, all of them are still based on locality-sensitive hashing families, i.e. random projections, which cannot adapt based on data distributions. Real data distributions are often non-uniform and frequently very skewed. Consequently, these methods still suffer from poor hashing effectiveness. In this chapter, instead of trying to enhance LSH with the use of more efficient structures or processing strategies, we seek to address a more fundamental question: are there methods that can provide consistently effective hashing results regardless of the data distribution. To this end, we propose the concept of data-sensitive hashing (DSH) as an efficient mechanism for addressing the k-NN problem.

DSH directly deals with the k-NN problem instead of finding neighbors within a distance r. Hence, DSH avoids the issue of selecting the distance r

that is required in LSH. DSH designs a novel scheme to facilitate the retrieval of a set of k-nearest neighbors with respect to a query point. Figure 5.1b shows the intuition of DSH. Since each bucket has a similar number of objects, its radius is adapted to the distance of k-NN. The basic idea of DSH is to hash the points such that the possibility of collision is much higher for k-NN pairs(unidirectional or bidirectional) than for non-k-NN pairs. It follows the same procedure as LSH for similarity search, the key difference being within the corresponding hashing families. Compared with random projections which have a large probability to hash the objects within a distance r together, our hashing family needs to have a large probability to hash the k-NN pairs together.

To find such hashing families, we learn from the data. Since we expect each k-NN pair to collide in most of the hashing functions, the requirement for hashing family can be represented as a strong classifier. Obviously, a single hashing function cannot protect all the k-NN pairs while separating all the non-k-NN pairs. Using spectral techniques, we can do an optimization on preserving the k-NN pairs while cutting the others. It is well known that adaptive boosting [28] is an efficient algorithm to generate a strong classifier using a set of weak classifiers. Therefore, we treat the hashing family as the strong classifier, and each hashing function in the family as a weak classifier. We adopt the adaptive boosting technique to ensure that the k-NN relations are theoretically protected by most of the hashing functions.

We summarize the contributions of the chapter as follows.

- We propose a novel access method called Data Sensitive Hashing (DSH) to answer the k-nearest neighbor queries. Compared with LSH, DSH is able to capture the data distribution more effectively. Equipped with the distribution knowledge, DSH is able to deal with the k-NN problem in a more direct and efficient manner. DSH focuses on the hashing family, and thus most LSH extensions designed to enhance LSH are orthogonal to our proposal, and can be applied to DSH easily. (Section 5.3).

- We design an efficient algorithm to generate the data-sensitive hashing family – the key challenge for DSH. The algorithm combines adaptive boosting and spectral techniques, resulting in a good theoretical guarantee. The indexing time is also comparable with LSH. (Section 5.4).

- We experimentally verify the effectiveness and efficiency of our proposed

DSH using three real datasets. Compared with LSH, our hashing results are much more balanced. Consequently, DSH is three times more efficient than LSH in query response time and index size in order to achieve the same quality of search results. (Section 5.5).

5.2 Preliminaries

In this section, we provide the problem definition followed by a brief introduction and analysis of LSH. In the end, we review related work.

5.2.1 Problem Definition

In this chapter, we consider data objects represented as points in a d-dimensional vector space R^d. Let $\|p, q\|$ be the distance measure between two points p and q in R^d. The k-nearest neighbor(k-NN) problem is defined as follows:

Definition 5.1. (k-nearest neighbor problem) *Given a set O of n data points, a point $q \in R^d$ and an integer k, we aim to find the k data points that are nearest to q in O. We denote this answer set by $NN(q, k)$. Formally, $|NN(q, k)| = k$, $\forall o \in NN(q, k)$, $o' \in OnNN(q, k)$, $\|q, o\| \leq \|q, o'\|$.*

Typically, finding exact k-NN potentially results in a sequential scan of the entire dataset O, and its cost grows linearly with the cardinality of O. For this reason, approximate k-nearest neighbor problem is accepted as a compromise since the cost of the solution grows sub-linearly with the cardinality of O while the quality is within an acceptable level. We define the approximate k-nearest neighbor problem as follows:

Definition 5.2. (δ-recall k-NN problem) *Given a set O of n data points, a point $q \in R^d$ and an integer k, the δ-recall k-NN problem aims to find a set of k points $\delta NN(q, k)$, such that $|\delta NN(q, k)| = k$, $|\delta NN(q, k) \cap NN(q, k)| \geq \delta \times k$.*

The definition provides a lower bound of the recall that at least $k \times \delta$ points of exact k-NN are returned. δ-recall k-NN problem is indeed compatible with c-approximate k-NN problem discussed in Section 5.1. In c-approximate k-NN problem, it provides the upper bound of the distance for the points to be returned. To achieve a similar upper bound in the δ-recall k-NN problem, we can relax k to a larger k' ($k' = k/\delta$), and select k points from $\delta NN(q, k')$ with the smallest distances. As a result, we can guarantee that the farthest distance of these k points to q is within R' where R' is the maximum distance between q and the k'th-NN.

5.2.2 Locality Sensitive Hashing

Locality Sensitive Hashing is an efficient approximate algorithm for high dimensional similarity search. It is efficient and provides a rigorous quality guarantee for finding similar points within a distance r, i.e. the r-NN problem.

Definition 5.3. (r-NN problem) *Given a set O of n data points, a point $q \in R^d$ and a distance r, we aim to find the data points that are within a distance r to q in O. We denote the sphere centered at point q by $B(q, r)$. Formally, $B(q, r) = \{p | p \in R^d, \|p, q\| \leq r\}$. The r-NN problem aims to find $\{o | o \in O$ and $o \in B(q, r)\}$.*

LSH leverages a family of functions where each function hashes the points in such a way that the possibility of collision is higher for similar points than for dissimilar points. Formally, an LSH family can be defined as follows [21]:

Definition 5.4. (LSH Family, $\mathcal{H}$) *A family $\mathcal{H} = \{h : R^d \to U\}$ of functions is called (r, cr, p_1, p_2)-sensitive if for any $p, q \in R^d$*

- *if $p \in B(q, r)$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \geq p_1$;*
- *if $p \notin B(q, cr)$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \leq p_2$;*

The family of hash functions are generated through the use of random projections – the intuition is that points that are nearby in the space will also be nearby in all projections. While two distance points can also be close in all projections, the possibility is extremely small if enough number of projections is taken. Usually, the difference between p_1 and p_2 is not enough to be used directly. To enlarge the difference, a concatenation of LSH functions is applied to generate a hash key for each point in R^d.

Definition 5.5. (Concatenation LSH Functions, $\mathcal{G}$) *A set of concatenation LSH functions $\mathcal{G} = \{g : R^d \to U^m\}$. Each $g_i \in \mathcal{G}$ consists of a sequence of m hash functions randomly extracted from $\mathcal{H}$. Formally,*

$$g_i(p) = (h_{i_1}(p), ..., h_{i_m}(p)),$$

where m is the number of hash functions in each concatenation and $h_{i_1}, ..., h_{i_m}$ are randomly selected from the LSH Family $\mathcal{H}$.

LSH applies these concatenation LSH functions to construct the hash tables. As a result,

- $\forall p \in B(q,r)$, $Pr(g(p) = g(q)) \geq p_1^m$
- $\forall p \in R^d n B(q,cr)$, $Pr(g(p) = g(q)) \leq p_2^m$

Further, the expected number of points in O that collide with q but are outside the ball $B(q,cr)$ is less than $p_2^m * |O|$. However, the recall for one hash table, p_1^m, is not large. To raise the overall recall, LSH typically applies l concatenation LSH functions and constructs l hash tables. Each concatenation LSH function randomly chooses m functions in $\mathcal{H}$. When the number of functions in $\mathcal{H}$ is large enough, the hash results of different concatenation functions can be regarded as independent. Thus, for any $o \in B(q,r)$, the possibility that o collides with q in at least one hash table is at least $1 - (1 - p_1^m)^l$, which is very close to 1. To summarize, LSH answers c-approximate r-NN problem as follows:

1. **Pre-processing:** LSH maintains l hash tables, and each hash table is attached with a concatenation hash function. Each hash table applies its concatenation hash function to hash the points in O, where each concatenation hash function consists of m hash functions randomly selected from the hashing family $\mathcal{H}$;
2. **Query processing:** Given a query object q, LSH finds c-approximate r-NN by examining points that collide with q in each of the l hash tables. In particular, the expected number of objects outside the $B(q,cr)$ is limited by $l * p_2^m * |O|$. This property guarantees its query efficiency.

While LSH has been shown to be very effective in finding nearby points in high-dimensional space, we should note that this is accomplished with respect to a specified radius r, and an approximation allowance factor c. For a k-NN problem, the corresponding radius for different query points may vary by orders of magnitude. In such a case, LSH has to either (i) be run repeatedly with different values of r, cr,c^2r ... which leads to substantial increase in the query time and index storage cost, or (ii) use an ad-hoc r which leads to low quality guarantee.

Further, LSH implicitly splits the whole space into lattices so that points in the same lattice cells are hashed into the same bucket. As a result, for real data distributions that are non-uniform, the hashing results are often unbalanced. We find that an unbalanced hashing leads to performance degradation of LSH. The reasons are two-fold. First, for the case that the number of points in one bucket is too small, LSH cannot find c-approximate r-NN and a further examination on a larger r is required; second, for the case that too many points collide in one bucket, LSH needs to examine each of these and so its performance suffers.

5.3 Data Sensitive Hashing

In this section, we introduce the basic concepts and principles of Data Sensitive Hashing (DSH), and propose two variants of DSH hashing families, DSH-basic and DSH-relaxed. The associated algorithms are presented in Section 5.4.

Intuitively, the problem we wish to solve is to directly search for the k-NN, whereas LSH is designed to find the r-NN. Therefore, we seek to define a new hashing family DSH. We begin with a DSH hashing family definition that migrates all the properties from LSH hashing family, called the DSH-utopia family. Here, the basic binary hashing $R^d \rightarrow \{0,1\}$ is employed for DSH.

Definition 5.6. (DSH-utopia Family) *A family $\mathcal{H} = \{h : R^d \rightarrow \{0,1\}\}$ is called (k, ck, p_1, p_2)-sensitive if for any $q \in R^d$ and $o \in O$*

- *if $o \in NN(q,k)$, then $Pr_{\mathcal{H}}(h(o) = h(q)) \geq p_1$;*
- *if $o \notin NN(q,ck)$, then $Pr_{\mathcal{H}}(h(o) = h(q)) \leq p_2$;*

Unfortunately, while the above definition allows us to express what we desire, it has no known efficient implementation. Therefore, we fall back on the intrinsic properties that affect the effectiveness of a hashing-based approach. We define the following properties that should be satisfied in our DSH concatenation functions.

Definition 5.7. (Effectiveness for k-NN) *For two probability values P_a and P_b, s.t. $P_a \gg P_b$,*

- **(Recall)** *for any query point* $q \in R^d$*, and* $o \in NN(q,k)$*,* $\forall g_i \in \mathcal{G}$*,* $Pr(g_i(q) = g_i(o)) \geq P_a$*;*

- **(Efficiency)** *for any query point* $q \in R^d$*,* $\forall g_i \in \mathcal{G}$*,* $|\{o | o \in OnNN(q, ck), g_i(q) = g_i(o)\}| \leq P_b \times |O|$*.*

This definition provides two conditions to qualify a DSH concatenation function. The first condition provides a lower bound of recall and the second condition gives an upper bound (i.e., $ck + P_b \times |O|$) of the number of points in each bucket. By applying multiple hash tables to raise the recall, high quality results can be achieved, and the algorithm is still efficient. In addition, the results of different hash tables should be independent. If each concatenation function randomly chooses some functions from a large enough hashing family $\mathcal{H}$, this condition is always satisfied. The following theorem shows the relation between the quality and efficiency under such concatenation functions.

Theorem 5.1. *If the concatenation functions satisfy the properties in Definition 5.7, to achieve an expected recall* δ*, the number of points to be checked is at most :*

$$\left\lceil \frac{\log(1-\delta)}{\log(1-P_a)} \right\rceil \times (ck + P_b \times |O|)$$

Proof. Considering that l hash tables are used, we have:

$$(1-\delta) = (1-P_a)^l$$

To achieve an expected recall δ, the number of hash tables required is:

$$l = \left\lceil \frac{\log(1-\delta)}{\log(1-P_a)} \right\rceil$$

On the other hand, the number of points in each bucket is at most $ck + P_b \times |O|$. Thus, the number of points to be checked is at most:

$$\left\lceil \frac{\log(1-\delta)}{\log(1-P_a)} \right\rceil \times (ck + P_b \times |O|) \qquad \square$$

The quality of results is decided by δ, while c is only an efficiency factor. c and P_b decide the total number of points to be checked, and there is a trade-off between them. Obviously, the pairs with a longer distance are easier to be hashed to different values, and choosing a higher c often results in a smaller P_b.

To devise an effective hashing-based approach, according to the above guiding principles in Definition 7, there is no need that each **individual hash function** $h \in \mathcal{H}$ has a large probability to hash a pair correctly. Instead, we only need to design a **hash family** $\mathcal{H}$ such that most of the hash functions hash data correctly. As a concatenation LSH function is obtained by randomly picking m hash functions from $\mathcal{H}$, such a family can produce a good set of effective $g_i \in \mathcal{G}$ that satisfies the properties of recall and efficiency. Intuitively, based on following definition, we may design a set of hash functions each of which complements the others.

Definition 5.8. (DSH-basic Family) *A family $\mathcal{H} = \{h : R^d \rightarrow \{0,1\}\}$ is called (k, ck, p_1, p_2)-sensitive if for any query point $q \in R^d$ and $o \in O$*

- *if $o \in NN(q,k)$, then $\frac{|\{h|h(o)=h(q), h\in\mathcal{H}\}|}{|\mathcal{H}|} \geq p_1$;*
- *if $o \notin NN(q,ck)$, then $\frac{|\{h|h(o)=h(q), h\in\mathcal{H}\}|}{|\mathcal{H}|} \leq p_2$;*

Theorem 5.2. *The concatenation functions generated using DSH-basic family are effective, i.e. they have the properties given in Definition 5.7.*

Proof. Using m randomly chosen hash functions in the DSH-basic family, setting $P_a = {p_1}^m$ and $P_b = {p_2}^m$, we have:
Recall: $\forall q \in R^d, o \in NN(q,k), g_i \in \mathcal{G}$,

$$
\begin{aligned}
& Pr(g_i(q) = g_i(o)) \\
& = \prod_{j=1}^{m} Pr(h_{i_j}(q) = h_{i_j}(o)) \\
& = \{\frac{|\{h|h(o) = h(q), h \in \mathcal{H}\}|}{|\mathcal{H}|}\}^m \\
& \geq {p_1}^m
\end{aligned}
$$

Efficiency: Likewise, $\forall q \in R^d, o \in OnNN(q,ck), g_i \in \mathcal{G}$,

$Pr(g_i(q) = g_i(o)) \leq {p_2}^m$. Thus,

$\sum_{o\in OnNN(q,ck)} Pr(g_i(q) = g_i(o)) \leq |O| \times {p_2}^m$ □

We further note that the purpose of the second condition in the **Effectiveness** is to prevent too many false hits. In the DSH-basic family, this is accomplished by keeping the probability of inclusion low for each non-ck-NN. However, we do not really care what the probability of inclusion is for any **individual non-answer point**: all we wish to do is to limit the **total number of false positives**. Accordingly, we can relax the second condition:

Definition 5.9. (DSH-relaxed Family) *A family $\mathcal{H} = \{h : R^d \to \{0,1\}\}$ is called (k, ck, p_1, p_2)-sensitive if for any query point $q \in R^d$*

- *for all $o \in NN(q,k)$, $\frac{|\{h|h(o)=h(q),h\in\mathcal{H}\}|}{|\mathcal{H}|} \geq p_1$;*
- $\sum_{o \notin NN(q,ck)} (\frac{|\{h|h(o)=h(q),h\in\mathcal{H}\}|}{|\mathcal{H}|})^m \leq {p_2}^m * |O|$;

Similar to Theorem 5.2, the concatenation functions generated using the DSH-relaxed family still have the **Effectiveness** properties given in Definition 5.7.

5.4 DSH Family Generation

After having specified the properties of data sensitive hash functions, we now show how we can find such functions. For LSH, there is a straightforward geometric interpretation, so random projection turns out to work well. For DSH, there is no such easy geometric interpretation. Instead, the hash function must adapt based on the data distribution. We borrow and adapt machine learning techniques for this purpose. In particular, we first learn good atomic hash functions, and then use boosting to get even better results with an ensemble of hash functions, as we describe in this section.

5.4.1 Overview

Given an query-object pair $\langle q_i, o_j \rangle$, let

$$\varphi_h(\langle q_i, o_j \rangle) = (h(q_i) - h(o_j))^2. \tag{5.1}$$

In particular, if q_i collides with o_j according to hash function h, then $\varphi_h(\langle q_i, o_j \rangle)$ equals to 0; otherwise, it becomes 1. For DSH-basic family, based on Definition

8, the requirements for each hash function can be rewritten as:
For every pair q_i and o_j,

$$\begin{cases} \sum_{h\in\mathcal{H}} \varphi_h(\langle q_i, o_j\rangle) \leq (1-p_1)|\mathcal{H}|, \text{if } o_j \text{ is a } k\text{-NN of } q_i; \\ \sum_{h\in\mathcal{H}} \varphi_h(\langle q_i, o_j\rangle) \geq (1-p_2)|\mathcal{H}|, \text{if } o_j \text{ is a non-}ck\text{-NN of } q_i; \end{cases} \tag{5.2}$$

Through this expression, we observe that the hashing family is actually required to be a strong classifier for the pairs, i.e. if we combine all the hash function in $\mathcal{H}$ together and use Equation 5.2 to classify the pairs, every k-NN pair and non-ck-NN pair can be well classified. [We can also obtain similar requirements for DSH-relaxed family. The difference here is that DSH-relaxed only requires all the k-NN pairs to be well classified. For those non-ck-NN pairs, DSH-relaxed limits the total number of false-positives for each query q.]

Adaptive boosting [28] is an efficient meta-algorithm to generate a strong classifier. Given an algorithm that can generate a weak classifier, adaptive boosting can generate a set of weak classifiers by tweaking the weight of training instances, such that their combination constitutes a strong classifier. In our problem, we do not really combine them to generate a strong classifier. Instead, each weak classifier is used as a hash function h in the family $\mathcal{H}$, while the family $\mathcal{H}$ has the above desirable properties. Therefore, we have to adapt standard adaptive boosting.

We divide the problem into two parts: (1) computing the optimal weak classifier, and (2) applying appropriate adaptive boosting algorithm to tune the input weights of the weak classifier. In both parts, we use a weight matrix W to represent the k-NN and non-ck-NN relations. For the reason that non-ck-NN relations are much more than k-NN relations, we only sample parts of them to reduce the computation cost, and make the number of samples comparable with the number of k-NN relations. Thus, W is a sparse matrix, as it only contains some sampled query-object pairs ($O(k)$ non-zero elements per query, and $O(qk)$ non-zero elements in total). Using $i \in [1, m]$ to denote those queries, and $j \in [1, n]$ to denote those data points, we have:

$$W_{ij} = \begin{cases} 1, \text{if } o_j \text{ is a } k\text{-NN of } q_i \text{ ;} \\ -1, \text{if } o_j \text{ is a sampled non-}ck\text{-NN of } q_i \text{ ;} \\ 0, \text{else.} \end{cases} \tag{5.3}$$

Each hash function h hashes the points. On the other hand, we regard it as a weak classifier φ_h for the pairs. Therefore, we expect it to optimize the resultant pairs based on the weight matrix. The weight of the matrix will be tuned by the adaptive boosting procedure. And the optimization for each function is defined as follows.

Figure 5.2 outlines how we generate DSH family. Given some sampled queries based on the query distribution, we get a series of query-object pairs. The k-NN pairs are expected to collide more in DSH family and vice versa. We represent this optimization target in a weight matrix form. And based on the weight matrix, we compute the optimal hash function and put it into hashing family. We then hash the points based on the hash function and check if it performs well on each pair. After that, we run the adaptive boosting procedure: for the pairs that are hashed correctly (k-NN pairs collide and vice versa), we reduce their weight in the matrix; for the pairs that are hashed mistakenly, we increase their weight in the matrix. Then we get a new weight matrix and compute a new optimal hash function. As shown in [28], after repeating this procedure multiple times, the weight of every pair becomes smaller than the original one, and a smaller weight means the pair has a larger probability to be hashed correctly.

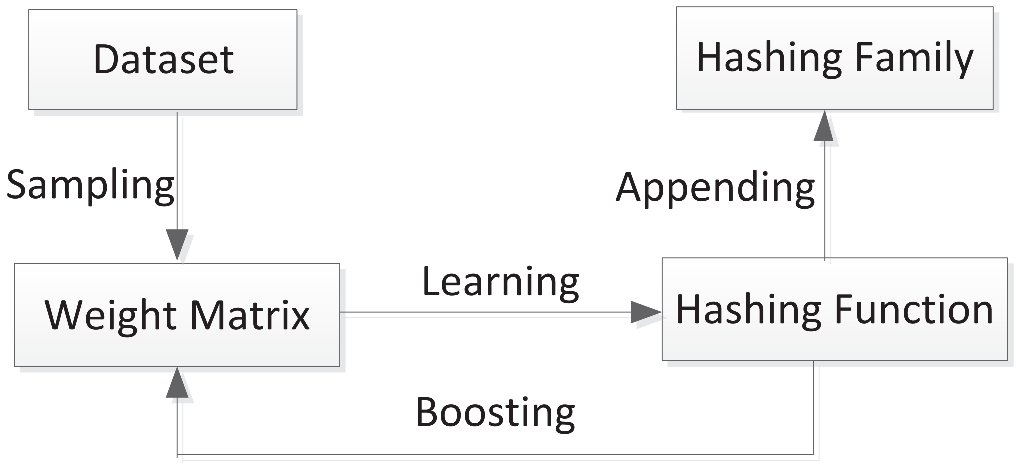


Figure 5.2: Overview of DSH Family Generation

The formal optimization problem set up requires specification of the data points, the query points, and the parameter k for k-NN. However, the principle of DSH is to leverage the distribution knowledge, but not tune the hash function based on specific points. We only seek to obtain the properties of DSH family for sampled queries based on the query distributions. Every sampled query should achieve high recall and be efficient. When the query distribution is hard to estimate, we can still use the uniform query distribution to cover all the potential queries. In the experimental section we show that the results obtained are not very sensitive to the query distribution or to the value of k. However, they do depend strongly on the data distribution.

Definition 5.10. (Hash Function Optimization)

Given the dataset O, the sampled queries Q and the weight matrix W, we aim to find the best hash function such that

$$\arg\min_{h} \sum_{ij} \varphi_h(\langle q_i, o_j \rangle) W_{ij} \tag{5.4}$$

s.t. $q_i \in Q, o_j \in O$, $\forall p \in R^d, h(p) \in \{0, 1\}$.

5.4.2 Single Hash Function Optimization

We would like to solve the optimization problem defined in Definition 5.10. However, finding the exact optimal hash function is an NP-hard problem.

Theorem 5.3. *Hash Function Optimization (HFO) is an NP-hard problem.*

Proof. We can reduce a well-known NP-hard problem, Minimum Graph Bisection (MGB) [4] to the hash function optimization (HFO) problem. To make this reduction, we construct a mapping in the following way: let the $n \times n$ weight matrix in MGB be W and the largest weight in W be w_{max}. For each vertex v_i in MGB, we create o_i and q_i in HFO. We begin with the initial weight matrix W in MGB. Then, we ensure that o_i and q_i must be assigned to the same value by setting $W'_{ii} = n^4 \times w_{max}$. Finally, we set $W'_{ij} = W_{ij} - n^2 \times w_{max}$. The intuition here is that if a large penalty is added for every pair that is not separated, then HFO will absolutely choose a balanced partition(the minimal penalty) while doing the optimization in MGB.

□

To tackle this problem efficiently, we consider the hash function in a linear form, i.e. separating the space by a hyperplane. This has been widely adopted, and shown to be effective [76]. Moreover, it has been shown in [87] that linear classifiers fit well with boosting algorithms.

Algorithm 7: Hash Function Optimization

```
Input: Weight matrix $W_{q\times n}$, data point matrix $X_{d\times n}$, query point
       matrix $Q_{d\times q}$.
Output: Hash Function $h$.
1 Compute the eigenvector $\mathbf{a}$ with the minimal eigenvalue in Theorem 5.4 ;
2 $h \leftarrow \mathbf{a}^T X$;
3 Generate $h'$ based on Equation 5.10;
4 return $h'$
```

That is, $h(o_j)$,$h(q_i)$ can be represented as $\mathbf{a}^T X_j$,$\mathbf{a}^T Q_i$, where $\mathbf{a}$ is the projection vector and X_j,Q_i is the d dimension vector presentation of o_j, q_i(suppose it has been regularized to $\overline{X_j} = \mathbf{0}$).

However, the range of $\mathbf{a}^T X_j$ is R instead of $\{0, 1\}$. The mean of $h(o_j)$ is always 0, as $\overline{h(o_j)} = \overline{\mathbf{a}^T X_j} = \mathbf{a}^T \overline{X_j} = 0$. To protect the result from being affected by the scaling of $\mathbf{a}$, we give a constraint $\mathbf{a}^T X X^T \mathbf{a} = 1$ to fix the variance of $h(o_j)$. We transform the problem of hash function generation as follows:

$$\arg\min_{\mathbf{a}} \sum_{ij} (\mathbf{a}^T Q_i - \mathbf{a}^T X_j)^2 W_{ij} \tag{5.5}$$

subject to $\mathbf{a}^T X X^T \mathbf{a} = 1$.

Theorem 5.4. *The vector in Equation 5.5 is equal to the minimal general eigenvector of*

$$((XD - QW)X^T + (QD' - XW^T)Q^T)\mathbf{a} = \lambda X X^T \mathbf{a} \tag{5.6}$$

where $D_{n\times n}$ and $D'_{q\times q}$ are two diagonal matrices defined as follows:

$$D_{jj} = \sum_i W_{ij} \qquad D'_{ii} = \sum_j W_{ij} \tag{5.7}$$

Proof.

$$\begin{aligned}
&\sum_{ij} (\mathbf{a}^T Q_i - \mathbf{a}^T X_j)^2 W_{ij} \\
&= \sum_j \mathbf{a}^T X_j D_{jj} X_j^T \mathbf{a} - 2\sum_{ij} \mathbf{a}^T Q_i W_{ij} X_j^T \mathbf{a} + \sum_i \mathbf{a}^T Q_i D'_{ii} Q_i^T \mathbf{a} \\
&= \mathbf{a}^T ((XD - QW)X^T + (QD' - XW^T)Q^T)\mathbf{a}
\end{aligned}$$

Therefore, the vector in Equation 5.5 is equal to

$$\arg\min_{\mathbf{a}} \mathbf{a}^T((XD - QW)X^T + (QD' - XW^T)Q^T)\mathbf{a} \tag{5.8}$$

subject to $\mathbf{a}^T XX^T \mathbf{a} = 1$. All the two matrices are symmetric. Therefore, $\mathbf{a}$ is equal to the eigenvector with the minimal eigenvalue in the following generalized eigenvector problem:

$$((XD - QW)X^T + (QD' - XW^T)Q^T)\mathbf{a} = \lambda XX^T \mathbf{a} \tag{5.9}$$

$\square$

The computational complexity of solving the eigenvector problem is $O(nd^2 + qkd + d^3)$, where n is the number of objects plus sampled queries, d is the number of dimensions and q is the number of sampled queries. The main cost here is to compute $((XD - QW)X^T + (QD' - XW^T)Q^T)$. Here D only has n non-zero values, D' has q non-zero values, and W has $O(qk)$ non-zero values. And for the general eigenvector problem, the computation cost is $O(d^3)$. Noted that the size of dataset is nd, our algorithm runs in time that is at most **linear** with the size of dataset.

After computing the hashing function $h : R^d \to R$, we convert it to the desired binary hash function $h' : R^d \to \{0, 1\}$ as:

$$h'(o) = \begin{cases} 0, \text{if } h(o) \leq 0 \\ 1, \text{else} \end{cases} \tag{5.10}$$

5.4.3 Adaptive Boosting for DSH-basic

Adaptive boosting [28] is a meta-algorithm for building a strong classifier using a set of weak classifiers, and can be used in conjunction with many other learning algorithms. Theoretical results that adaptive boosting always obtains a strong classifier can be found in [28]. Here we only show the intuitions. The weight for each instance depends on its results (well classified or not) on the weak classifiers, and thus represents if it is well classified using their combination. If its weight is smaller than the original, then it is well classified using the linear combination. On the other hand, in each step, the weak classifier performs better than random so that the weight of well classified instances is larger than the weight of misclassified instances. Therefore, the total weight of all instances is always reduced by the weak classifier. As the weight is tuned exponentially, the total weight also reduces exponentially. After logarithmic number of steps, the total weight is reduced to be less than the original weight for a single instance, and by then, every instance has a weight less than its initial weight and is therefore well classified.

For our specific problem, not only a strong classifier is needed, but also the difference between p_1 and p_2 is expected to be large. Obviously we cannot expect the boosting algorithm to achieve $p_1 = 1$ and $p_2 = 0$. However, the boosting algorithm optimizes to enlarge the difference [87]. Therefore, we choose some appropriate p_1 and p_2 and iterate based on the following weight updating function in which α is a weight tuning parameter of boosting:

For k-NN pairs, we have:

$$W_{ij}^{(t+1)} = W_{ij}^{(t)} * \alpha^{(p_1 - 1 + \varphi_{h_t}(\langle q_i, o_j \rangle))} \tag{5.11}$$

For non-ck-NN pairs, we have:

$$W_{ij}^{(t+1)} = W_{ij}^{(t)} * \alpha^{-(p_2 - 1 + \varphi_{h_t}(\langle q_i, o_j \rangle))} \tag{5.12}$$

Algorithm 8 describes our boosting procedure in which $itrCounter$ is the number of hash functions that we aim to obtain. We next prove that if every instance has a weight smaller than its initial, then $\mathcal{H}$ conforms to DSH-basic family defined in Definition 5.8 for the sampled data when the updating function is based on Equation 5.11 and 5.12.

Theorem 5.5. *$\forall \langle q_i, o_j \rangle$, if $|w_{ij}^{(t)}| \le |w_{ij}^{(0)}|$, $\mathcal{H}$ conforms to Definition 5.8 for all sampled pairs.*

Proof. Let T_{ij} be $|\{h | h(q_i) = h(o_j), h \in \mathcal{H}\}|$, and S_{ij} be $|\{h | h(q_i) \neq h(o_j), h \in \mathcal{H}\}|$. Obviously, $S_{ij} = |\mathcal{H}| - T_{ij}$. Hence, for every k-NN pair $\langle q_i, o_j \rangle$,

$$\begin{aligned}
&w_{ij}^{(0)} \geq w_{ij}^{(t)} \\
\Longrightarrow &w_{ij}^{(0)} \geq w_{ij}^{(0)} \times \alpha^{(p_1-1)|\mathcal{H}|+S_{ij}} \\
\Longrightarrow &\alpha^{(p_1-1)|\mathcal{H}|+S_{ij}} \leq 1 \\
\Longrightarrow &(p_1-1)|\mathcal{H}| + S_{ij} \leq 0 \\
\Longrightarrow &p_1|\mathcal{H}| - T_{ij} \leq 0 \\
\Longrightarrow &\frac{|\{h|h(o_i) = h(o_j), h \in \mathcal{H}\}|}{|\mathcal{H}|} \geq p_1
\end{aligned}$$

For every non-*ck*-NN pair $\langle q_i, o_j \rangle$, it is analogous to prove that $\frac{|\{h|h(q_i)=h(o_j),h\in\mathcal{H}\}|}{|\mathcal{H}|} \leq p_2$. $\square$

5.4.4 Adaptive Boosting for DSH-relaxed

Now we give the adaptive boosting solution for DSH-relaxed. The procedure is similar to that for DSH-basic except one major difference, i.e., we only limit the total number of points in each bucket. Therefore, for every query, the non-*ck*-NN relations now construct one big instance. For ease of presentation, we assume that all the pairs are sampled and the size of the current H is large enough. However, as only a small number of pairs are sampled and the algorithm begins with empty H, some trivial regularization should be applied in the implementation. We use the expected number of points that collide with q_i to measure the efficiency of query q_i, which is defined as:

$$Collision_i = \sum_j (1 - \frac{\sum_{h\in\mathcal{H}}(\varphi_h(\langle q_i, o_j\rangle)}{|\mathcal{H}|})^m \tag{5.13}$$

Note that if the collision rate bound is p_2, then the upper bound of the false-positives is $n * {p_2}^m$. Thus, $(\frac{Collision_i}{|O|})^{\frac{1}{m}}$ is the equivalent collision rate to get the same upper bound of the false positives. After being regularized, the weight for each query is:

$$weight_i^{(t)} = \alpha^{t*(p_2-(\frac{Collision_i}{|O|})^{\frac{1}{m}})} \tag{5.14}$$

For *k*-NN pairs, we tune their weight as usual. On the other hand, we tune the weight for each query q_i based on $Collision_i$. However, our hash function

method	recall	query time	error ratio	# hash tables
LSH		18.6	1.012	100
DSH-basic	0.94	12.8	1.012	80
DSH-relaxed		6.2	1.013	30
LSH		12.1	1.024	50
DSH-basic	0.9	6.8	1.027	32
DSH-relaxed		3.8	1.025	14
LSH		8.2	1.036	25
DSH-basic	0.86	4.4	1.036	16
DSH-relaxed		2.4	1.038	8

(a) Forest

method	recall	query time	error ratio	# hash tables
LSH		40.8	1.004	90
DSH-basic	0.96	28.6	1.004	72
DSH-relaxed		16.0	1.004	36
LSH		26.0	1.008	40
DSH-basic	0.92	17.2	1.009	36
DSH-relaxed		9.8	1.010	16
LSH		19.2	1.013	25
DSH-basic	0.88	12.0	1.014	20
DSH-relaxed		8.0	1.014	10

(b) Flickr

method	recall	query time	error ratio	# hash tables
LSH		/	/	/
DSH-basic	0.95	25.6	1.002	64
DSH-relaxed		15.6	1.002	32
LSH		38.8	1.006	75
DSH-basic	0.91	14.4	1.006	28
DSH-relaxed		9.4	1.006	16
LSH		24.8	1.018	35
DSH-basic	0.87	11.6	1.017	20
DSH-relaxed		7.6	1.021	10

(c) DBPedia

Table 5.1: Overall comparative study: DSH-relaxed is the most efficient in terms of query time and space usage

Algorithm 8: Adaptive Boosting Procedure

```
1 initialize$t \leftarrow 0$;   $\mathcal{H} \leftarrow \emptyset$ ;
2 initialize $W^{(0)} = W$ given in Equation 5.3;
3 while $t < itrCounter$ —— $\exists w_{ij}^{(t)}, w_{ij}^{(0)}, |w_{ij}^{(t)}| > |w_{ij}^{(0)}|$ do
4     compute $h^{(t)}$ based on Algorithm 7 using $W^{(t)}$ as input;
5     $\mathcal{H} \leftarrow \mathcal{H} \cup \{h^{(t)}\}$;
6     updating $W^{(t+1)}$;
7 return $\mathcal{H}$;
```

optimization algorithm is based on the weight for every query-object pair. For this reason, we have to assign the weight of each query q_i to its related pairs. Here, we still expect the hash function to minimize the total weight of training instances. Thus, we study the "gradient" vector of $Collision_i$, i.e. how much it will be changed when a new hash function hashes each point $o_1 ... o_n$ together with q_i or not. We then adjust the weight for each related pair based on its "partial derivative". Formally, the "partial derivative" is defined as:

$$\Delta Collision_{ij} = \frac{m}{|\mathcal{H}|} \times (1 - \frac{\sum_{h \in \mathcal{H}}(\varphi_h(\langle q_i, o_j \rangle))}{|\mathcal{H}|})^{m-1} \tag{5.15}$$

Intuitively, if an object has a larger probability to collide with the query, it will have a larger weight. For those objects that are almost impossible to collide, there is little benefit to separate them with the query in the new hash function, and the weight for them is close to zero.

Now we define the weight updating function for those non-ck-NN pairs:

$$W_{ij}^{(t)} = weight_i^{(t)} \Delta Collision_{ij} \tag{5.16}$$

The weight updating function for k-NN pairs remains the same.

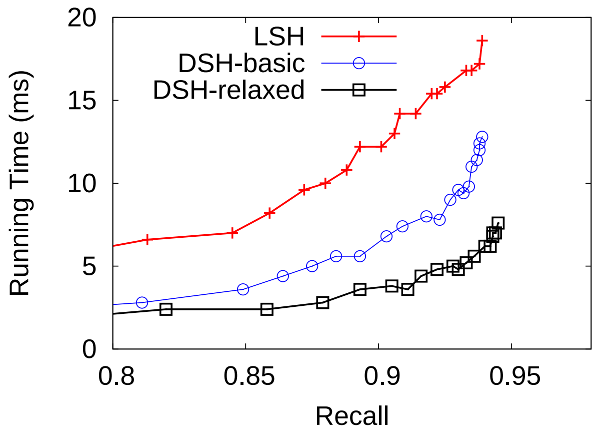
5.4.5 Difference with other learning based methods

So far, all the discussion is based on the similarity search prospective. There are many other learning based hashing or embedding methods such as [111, 6, 5, 85, 8, 76]. Now we will discuss our difference with them. Compared with these methods, DSH is distinctive in the optimization target. Traditional learning based methods mostly focus on the high precision end, and evaluate their performance using PR-value, F_1-measure or MAP. However, this may not be appropriate for our similarity search problem. As we have a validation step, the precision itself is not a crucial measure. We aim to achieve a high recall with acceptable candidate size. For example, to solve a 20-NN search problem in 1M points, we expect the algorithm to return 2000 points that contain all the positives, but not to return 20 points that contain 12 of them. We can validate those 2000 points one by one and obtain the exact answer. However, for the 20 points set, we cannot expand it to obtain a higher recall. Although the algorithm that outputs 2000 points has a 1% precision, the check rate is only 0.2%. For this reason, we focus on the relatively lower precision area, and optimize our algorithm to yield a higher recall. Distinguished with traditional methods, DSH does not pay much to filter every false positive, but to ensure every true answer is in the result, while most other methods pay equal emphasis to these two targets.

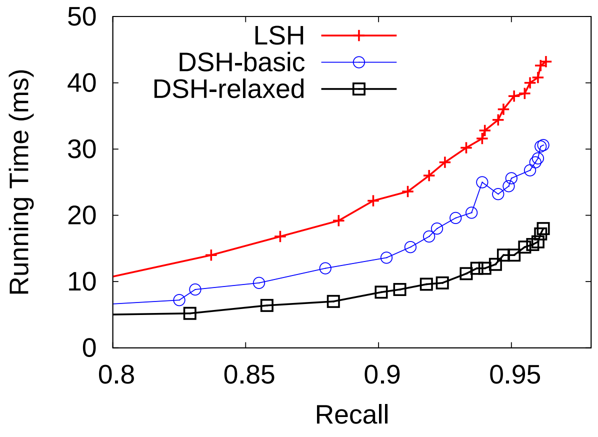
Following this guideline, in the design of DSH-relaxed, we provide zero tolerance to the false negatives, and a much higher tolerance to the false positives. For the false negatives, we use boosting techniques to ensure all the positives are well classified. And for the false positives, we only give a very weak limitation on the total numbers, and do our best for each individual. By using this heterogeneous solution, our optimization target is realized.

5.4.6 Incremental Maintenance

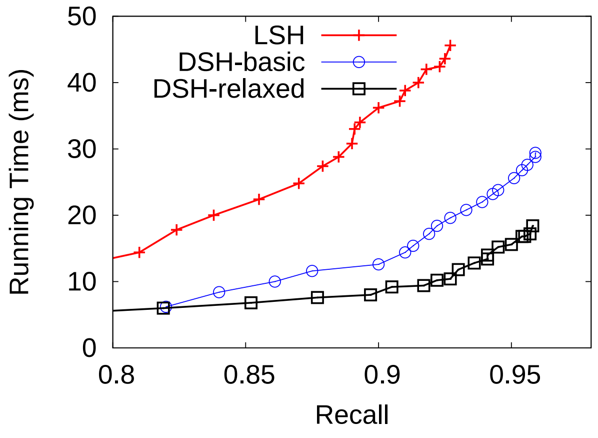
DSH is sensitive to data distribution, unlike LSH. Indeed, that is the whole point. This raises the question of what to do if the data distribution changes over time in a dynamic setting. Fortunately, incremental update of DSH hashing family is straightforward: simply replace any hash function that has a negative effect on the weight of boosting by a newly computed one since the boosting procedure aims at reducing the total weight of the weight matrix. The cost of such updates can be limited by changing at most one hash function each time. In practice, it turns out to be enough to make one such incremental update after 1-5% of the data has changed.



(a) Forest



(b) Flickr



(c) DBPedia

Figure 5.3: Average running time executed using different access methods to achieve a certain query quality: DSH-relaxed achieves the highest recall in the same running time, followed by DSH-basic and LSH.

5.5 Experiments

5.5.1 Experimental Setup

We evaluate the performance using three real datasets: scientific data, images and text. Properties of these three datasets are summarized in Table 5.2 and described in detail below:

- **Forest**[1] dataset is provided by the US Forest Service. We observe that some attributes are strongly correlated since they describe related information. For example, there are some attributes describing Hillshade Index at different times of a day. For this reason, Forest dataset is ideal to study the performance of indexing methods with respect to **data skew**;
- **Flickr**[2] is an image hosting website in which members can upload their images. Over this dataset, we do similarity search based on the feature space of images instead of pixel space. Following common practice, we use the PCA technique to pre-process the images and keep 80 feature dimensions. In contrast to the Forest dataset, the PCA feature space is **orthogonal**;
- **DBPedia**[3] is a project aiming to extract structured content from the Wikipedia database. We utilize LDA to pre-process the dataset, and keep 150 topic dimensions. The data points after employing LDA topic model transformation lie in the probability distribution space, and hence we use KL-divergence as the distance measure since it is widely used in this space. We use DBPedia dataset to study the effectiveness of DSH when a **non-Euclidean distance** measure is used.

We evaluate the following methods in the experiments:

- **LSH.** E2LSH[21] is a recent state-of-the-art implementation for LSH.
- **DSH-basic.** Our method proposed in Section 5.4.3.
- **DSH-relaxed.** Our method proposed in Section 5.4.4.

[1]http://archive.ics.uci.edu/ml/datasets/Covertype
[2]http://www.flickr.com/
[3]http://wiki.dbpedia.org

Dataset	# Objects	# Dimension	Property
Forest	580K	54	skewness
Flickr	1M	80	orthogonal
DBPedia	1M	150	non-Euclidean

Table 5.2: Dataset properties

- **Query-Sensitive Embedding**[6] is one efficient embedding hashing method that also applies boosting to guarantee the performance of similarity search.
- **Spectral Hashing**[111] is a state-of-the-art learning based hashing methods.

We randomly remove 1000 points from each dataset and use them as query points in our performance study. The ground-truth for each query point is obtained by a linear scan of the entire dataset. The sampled queries used in DSH are generated via random selection that is independent of the query set. Unless otherwise specified, the sample rate is 0.5%, k is set to 20 and c is set to 5 throughout the experiments. For all algorithms, each hash table contains 2000 buckets(i.e. m is set to 11). To avoid duplicating the points, entries in the table are in form of point IDs instead of the points. For this reason, the storage cost can be reduced by about 100 times. Besides, all these methods keep the entire indexes as well as the original dataset in main memory.

We evaluate the performance in terms of the following three aspects:

- **Query Quality** is measured by *recall* and *error ratio*. Let $\widehat{NN(q,k)}[i]$ (resp. $NN(q,k)[i]$) be the point in $\widehat{NN(q,k)}$ (resp. $NN(q,k)$) with the ith smallest distance to q. The *error ratio* [66] is to measure how close are the distances between points in $\widehat{NN(q,k)}$ and points in $NN(q,k)$:

$$error\ ratio = \frac{1}{|Q| \times k} \sum_{q \in Q} \sum_{i=1}^{k} \frac{|\widehat{NN(q,k)}[i], q|}{|NN(q,k)[i], q|} \tag{5.17}$$

- **Query Efficiency** is measured by the *running time* to answer k-NN queries. The *precision* in those learning based methods can also be viewed as a measurement of efficiency, since $\frac{recall \times k}{precision}$ is the number of points to be checked.

- **Space Requirement** is measured by the *number of hash tables being used*. The space cost for one hash table is about the same for each method, and is a good surrogate for index size.

5.5.2 Comparison with LSH

In this study, we compare our proposed methods, namely DSH-basic and DSH-relaxed, with LSH. We choose the best width parameter W for LSH, and the parameters for DSH tuned as discussed in Section 5.5.5.

Table 5.1 summarizes the average results over all the datasets. In each table, we report the query time, the error ratio and the number of hash tables required per query to achieve three different query quality levels (in terms of recall). First, we observe that DSH-relaxed is significantly more space and time efficient than DSH-basic and LSH. It is worth mentioning that for the DBPedia dataset, LSH cannot find approximate k-nearest neighbors to achieve recall 0.94 within acceptable time. Further, we note that all three methods provide almost the same error ratio across all the datasets and each error ratio is close to 1, in which the exact k-NN are identified.

Typically, a larger δ will result in a higher query cost. We therefore study the relationship between recall δ and query performance in detail for all three access methods. Figure 5.3 summarizes the results. First, we observe that the query time increases super-linearly to achieve a higher recall. Second, both DSH-relaxed and DSH-basic perform better than LSH by a wide margin and there is an obvious trend of increasing time to achieve a higher recall. Finally, we observe that DSH-relaxed and DSH-basic outperform LSH by a wider margin over the DBPedia dataset than the other two datasets. The reason will be provided when we discuss the relationship between the recall and the space usage.

Regarding the comparative methods, the query performance basically depends on the number of candidates to be probed in the same buckets that collide with the query points. Hence, we report the distribution of bucket size of hash tables and discuss how this affects the query performance. Figure 5.4 shows the distribution of bucket size of hash tables. The points are distributed more unevenly in hash tables of LSH than those of DSH-basic and DSH-relaxed. In particular, for hash tables in LSH, the top 1% buckets take 21%, 13%, 13%

Query		**Dense**	**Sparse**	**Original**
LSH	query time	581	42	130
	recall	0.99	0.803	0.916
DSH-basic	query time	161	58	98
	recall	0.978	0.893	0.932
DSH-relaxed	query time	150	59	90
	recall	0.984	0.922	0.962

Table 5.3: Query time and recall for various distributions

points of the Forest, Flickr and DBPedia datasets, respectively. Obviously, the query performance suffers when the query points are hashed to these buckets. Typically, the query distribution follows the data distribution, and in this case, the query points are often hashed to the buckets with a large number of points. Consequently, the number of candidates in LSH is larger on average than that in DSH methods, and incurs a higher query cost.

We proceed to study the relationship between the recall and the space usage, and report the results in Figure 5.5. To achieve a higher recall, a larger number of hash tables is typically required. In particular, DSH-relaxed outperforms DSH-basic and LSH by a wide margin. The trends are fairly similar to Figure 5.3. However, there exists a slight difference between them. We note that using similar number of hash tables, LSH incurs about 25% longer query time than DSH methods. The reasons are two-fold. First, due to the unbalanced hashing in LSH, the number of points to be checked in each table is about 3 times larger than that in DSH. Second, there exist many more duplicates from different tables in LSH than those in DSH where each point will be checked at most once. As a result, DSH methods still benefit from its balanced hash tables.

In addition, DSH-relaxed and DSH-basic outperform LSH by a wider margin on the DBPedia dataset than the other two datasets. Due to the difference between Euclidean distance and KL-divergence, the recall of LSH is limited at 94%. Although DSH methods are not specially designed for KL-divergence measurement, they achieve a recall of 96%, which can be improved further by using a larger number of hash tables.

We also investigate the performance of each method under different query distributions. Besides the *original* queries that are randomly extracted from

Dataset	**Forest**		**Flickr**		**DBPedia**	
Size	1×	10×	1×	10×	1×	10×
LSH	57	502	92	875	105	1001
DSH-basic	37	331	148	1350	84	798
DSH-relaxed	32	286	109	980	77	719

Table 5.4: Indexing time (s)

each dataset, we generate two additional query sets for each dataset and each set consists of 100 queries. Queries in the first set (labeled as *dense*) are randomly selected and the distances between each query and its k-NNs are small. Queries in the second set (labeled as *sparse*) are randomly selected as well and the distances between each query and its k-NNs are large. The results (dense, sparse and the original) of using $l = 40$ on Flickr dataset are shown in Table 5.3. Due to the space constraint, in what follows, we only report the results on the Flickr dataset whenever the results on the other two datasets exhibit similar trend. From the result we can see that the performance for DSH is generally consistent in all cases. However, the recall of LSH drops greatly in the sparse area, and the query time increases significantly in the dense area.

5.5.3 Scalability

Table 5.4 summarizes the indexing time for the comparative methods. To test the scalability of our indexing method, we also conduct an experiment where all the data points are duplicated for 10 times. Notice that both DSH and LSH treat those duplicated points as new points, thus indicating the expected performance with a larger data set. First we can see that both LSH and DSH take 9-10 times longer in the 10 times larger dataset, suggesting that the indexing time grows linearly with size of data set for all methods.

We also note that neither DSH methods nor LSH takes the minimum indexing time across all three datasets, and the indexing time over different datasets for all methods varies slightly. Specifically, DSH-relaxed and DSH-basic are more efficient than LSH on Forest and DBPedia datasets while we get a reverse result on Flickr dataset. Note that in all comparative methods, the indexing time consists two components: (1) the time to obtain the hash functions; (2) the cost of hashing points to the hash tables. LSH uses the random projection to

Precision	0.1	0.03	0.01	0.003	0.001
C/N(%)	0.01	0.03	0.1	0.3	1
LSH	0.551	0.704	0.8	0.852	0.916
DSH-basic	0.684	**0.816**	0.869	0.904	0.932
DSH-relaxed	0.652	0.812	**0.894**	**0.936**	**0.962**
[111]	**0.738**	0.808	0.864	0.888	0.919
[6]	0.677	0.794	0.864	0.897	0.925

Table 5.5: Precision-recall comparison with learning methods

generate hash functions and hence it performs much better than DSH-relaxed and DSH-basic in obtaining the hash functions. However, DSH-relaxed and DSH-basic require fewer hash tables, and incur less cost to hash points. By considering both factors, the overall indexing time for DSH is comparable with LSH.

5.5.4 Comparison with Learning Based Methods

We now compare DSH with other learning based methods. As discussed in Section 5.4.5, the optimization target of DSH is different from that of other learning based methods. Table 5.5 gives the comparison with query-sensitive embedding and spectral hashing. In this table, we use the precision-recall tradeoff to show the basic difference illustrated in Section 5.4.5. We give each method equal space limitation $l = 40$, and evaluate their precision and recall when m is varied.

DSH-relaxed performs worse than the others in the high precision end, as it allows too many false positives into the results. But as a payback, it wins the highest recall in the relative lower precision area. On the other hand, spectral hashing performs the best in the high precision area. However, in the low precision area, its performance is the worst (similar with the performance of LSH) since it does not make any guarantee for each individual and some positive objects are totally lost. DSH-basic and Query Sensitive Hashing lie in the middle, as they both enforce a strict constraint on both false positives and false negatives. And DSH-basic outperforms query-sensitive hashing about 1% for the reason that it preserves the collision probability for each k-NN point, while query-sensitive hashing only guarantees that the probability is larger than those non-k-NNs. When the precision is 1%, the check rate(candidate size/datasize) is only 0.1%. Thus, we focus more on the performance in the relatively lower precision area and DSH-relaxed outperforms other algorithms.

For embedding or data representation applications, there is a need to compute the similarity in the transformation space. Thus, the precision itself is very important. However, to answer the similarity search, the validation is conducted in the original space. For this reason, all those false-positives can be pruned. From this experiment we see that the most suitable algorithms for these two scenarios are different.

5.5.5 Parameter Studies

In this study, we investigate the parameters that potentially affect the performance of DSH, and suggest guidelines for parameter setting.

We first study the effect of sample rate by varying from 0.1%(1K queries) to 5%(50K). We evaluate the recall property P_a DSH can achieve when the efficiency property is guaranteed by is $P_b = 0.005$. Figure 5.6a shows the results. By enlarging the sample size, a more accurate model can be trained to capture the distribution of the whole dataset. The value in the training set shows that the upper bond of P_a is less than 0.282. On the other hand, the real performance becomes very close to that value when sample rate is larger than 0.5%(5K). Therefore, we set the sample rate to 0.5%. Also, for each query, we only acquire 10 k-NN relations and 10 non-ck-NN relations. The recall needed here is only 50%. Thus, we can process the sampling process efficiently by some approximate search algorithm. Figure 5.6b shows the indexing time when different sample size are used.

We next study the effect of k by varying k from 5 to 80. Figure 5.6c shows the results. In general, the recall drops slightly when k increases. It turns out to be increasingly insensitive to k by using a larger number of hash tables.

We also study the relationship between c and query efficiency, and report the query time in Figure 5.7a using different k. In general, the optimal c that achieves the minimal query time differs when k varies. However, we can see that for every k, the value ck that achieves the minimal query time always falls in the range from 100 to 200. As shown in Theorem 5.1, the efficiency is decided by both $P_b \times N$ and ck. Thus, the best ck value depends on the bucket size. In

Method	**# table**		**Query time**	
	original	multi-probe	original	multi-probe
LSH	90	9	40.8	44.2
DSH-basic	72	7	28.6	29.4
DSH-relaxed	36	4	16	17.2

Table 5.6: Augmentation with multi-probe proposed for LSH

our experiments, the bucket size is about 500 points, and the ck that achieves the best query performance lies in the range from 100 to 200.

Besides, there is a need to generate a DSH family for various k. Thus, we study the performance using different k in both the learning and query phase. Using $l = 40$, the result in Figure 5.7b shows a larger k in the learning phase performs well over various query k in the query phase. For DSH, using $k = 80$ only introduces a cost of checking 80 points for each bucket at most. Meanwhile, when a larger k is applied in the learning phase, the query quality performance for those smaller k will also be guaranteed. Thus, we can use a larger k to learn DSH in the various query k scenario.

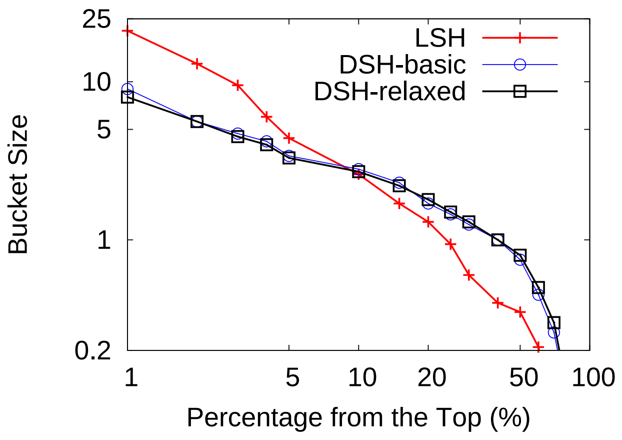
Finally, we study the space-efficiency trade-off of DSH. Typically, to achieve the same level of recall, using larger m and l will reduce the query time, while leading to additional storage cost. Thus, we study the relationship between query time and storage cost when the recall requirement is fixed to 0.96, and the results are shown in Figure 5.7c. We can observe that by increasing the number of hash tables, the query time drops and the performance gain becomes smaller.

5.5.6 Extensibility

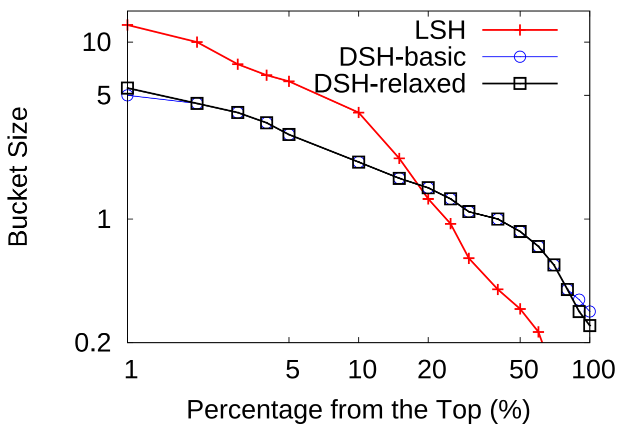
DSH is orthogonal to most of the research works that use LSH family as a foundation and improve on LSH. Therefore, the results from most of these works can be applied to DSH as an alternative to LSH. As an example, we adapt Multi-probe LSH [66] into DSH. The results to obtain 0.96 recall are given in Table 5.6. We can see that the combination further reduces the number of tables needed while requiring almost the same query time.

5.6 Summary

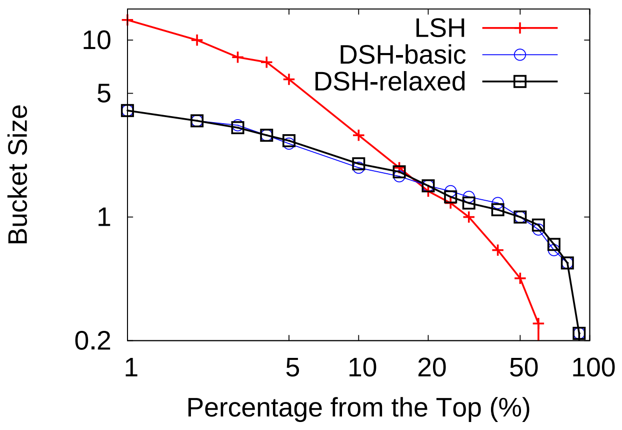
In this chapter, we propose an efficient hashing method called *Data Sensitive Hashing* (DSH) for high-dimensional approximate k-NN search problem. The hashing family is directly designed for preserving k-NN relation. By leveraging knowledge from the sampled data, DSH balances its buckets even in non-uniform data distributions, and aims to preserve most k-NN relations in its hashing. We conducted extensive experiments using three real datasets, and the results confirm the efficiency and robustness of the DSH. The DSH-relaxed variant is the most efficient, which requires only 1/3 of the hash tables and query time of the LSH. Furthermore, existing LSH extensions can easily be applied to the DSH.



(a) Forest

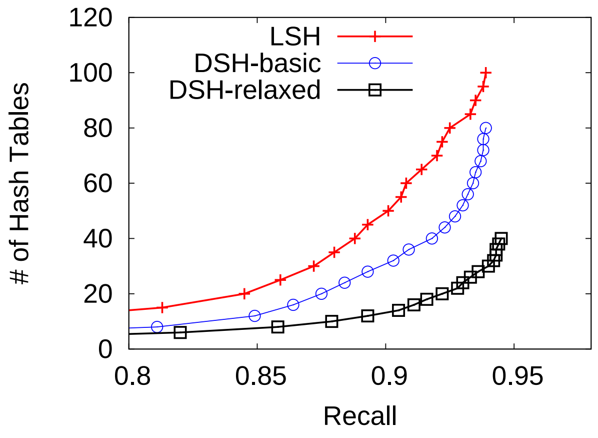


(b) Flickr

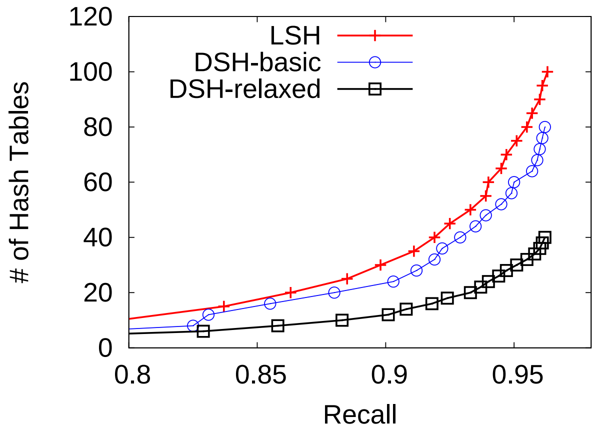


(c) DBPedia

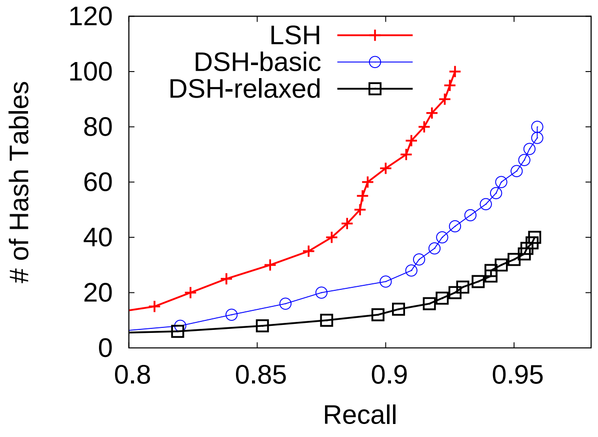
Figure 5.4: Distribution of Bucket Size



(a) Forest

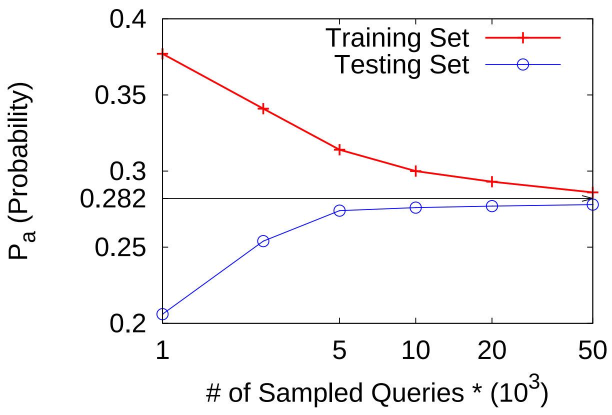


(b) Flickr

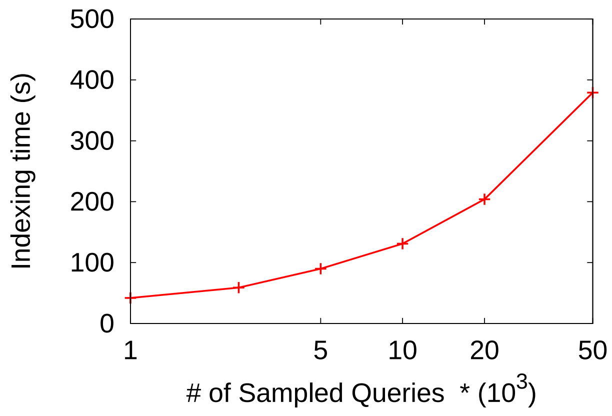


(c) DBPedia

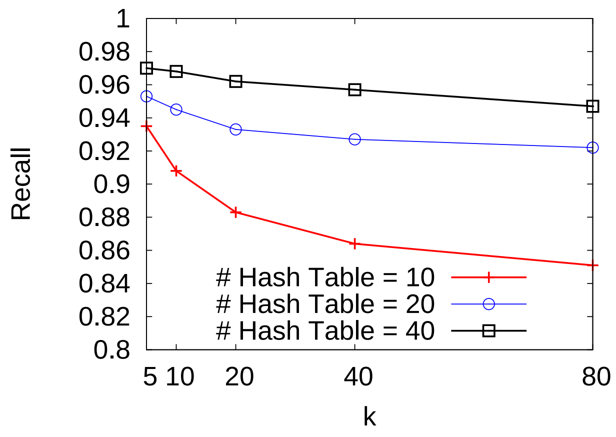
Figure 5.5: Number of hash tables that are required by different access methods to achieve a certain query quality: DSH-relaxed achieves the highest recall by using the same number of hash tables, followed by DSH-basic and LSH.



(a) Sampling size

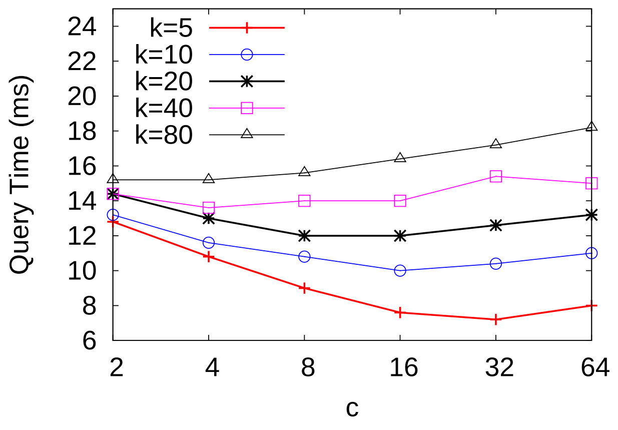


(b) Indexing time

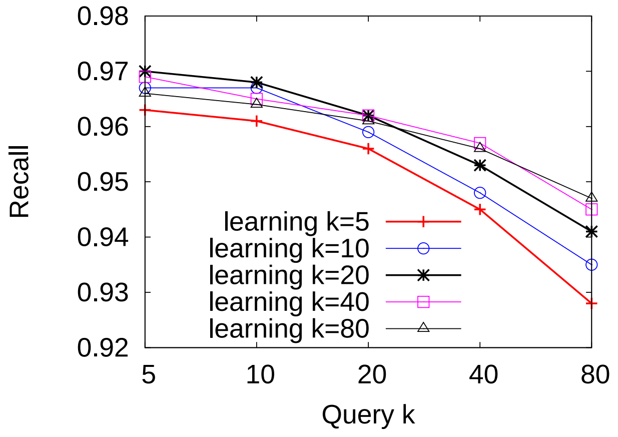


(c) Effect of k

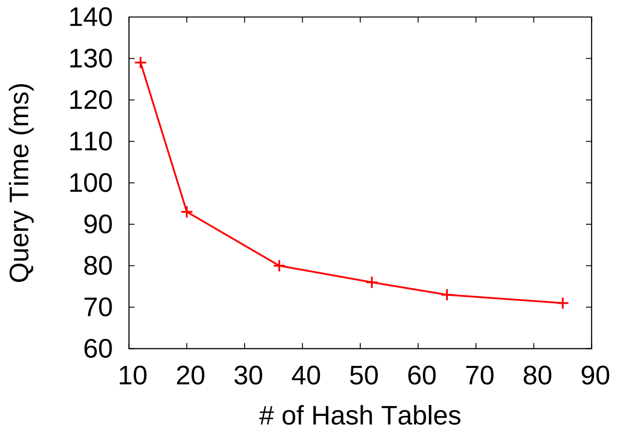
Figure 5.6: Effect of parameters in DSH – Part1



(a) Effect of c



(b) Effect of learning k



(c) Space efficiency trade-off

Figure 5.7: Effect of parameters in DSH – Part2

CHAPTER 6

DISTANCE INDEPENDENT APPROACH FOR SIMILARITY INDEX

Many applications involve locating the k nearest neighbors (k-NN) of a given query point in a multi-dimensional space. Locality Sensitive Hashing (LSH) and its variants, are generally believed to be the most effective search methods in high-dimensional spaces. However, LSH is designed to find points within a specified radius (i.e., to perform a radius search). In many applications the k-NN distance of different query points may differ greatly, especially in skewed datasets, which may lead to poor performance for LSH. Several learning-based methods are proposed to improve the performance of LSH by leveraging the knowledge of data distribution in the whole dataset. However, since the k-NN distance of a certain point is a *local* feature, the benefit of optimizing *global* hashing function is limited. In short, there is still a gap between efficient k-NN search and fixed radius search.

To close this gap, we propose a novel indexing scheme called *Selective Hashing*. In selective hashing, we aim to find the best index structure for each data point *locally*. The main idea is to create a disjoint set of indices with different granularities, and to store each point only in the most effective index. Intuitively, the index of each point is selected based on its local density, and the search range of query is automatically tuned based on its k-NN distance. Theoretically, we show that k-NN search using selective hashing can achieve the same quality as a fixed radius LSH search, using a radius equal to the distance of the $c_1 k_{th}$ nearest neighbor, with at most c_2 times overhead, where c_1 and c_2 are small constants. Selective hashing is also easy to build and update, and outperforms all the state-of-the-art algorithms.

6.1 Introduction

Efficient k-nearest neighbor (k-NN) search is essential for a wide range of applications in the areas such as information retrieval, data mining and machine learning. Objects are frequently characterized as feature vectors, and represented as points in a multi-dimensional space. Access methods in multi-dimensional space, including the k-NN problem, have been studied extensively. However, most of them suffer from the so-called curse of dimensionality and demonstrate poor performance when the number of dimensions is high [110]. One technique that shows promise in high-dimensional spaces is locality sensitive hashing (LSH) [32]. LSH is designed for finding points within a fixed radius of the query point, i.e., radius search. For a k-NN problem (e.g., the first page results of search engine), the corresponding radii for different query points may vary by orders of magnitude, depending on how densely the region around the query point is populated. To consider a human analogy: in a crowded city, we may be able to find the k-NN even within our own building, whereas in the middle of the desert, we may have to go many miles even to find the k-NN. In such a case, LSH has to either (i) build index for different radii $R, cR, c^2R, \dots$, which increases the query time and index storage cost, or (ii) use an ad-hoc radius, which voids any quality guarantee.

Recently, there has been increasing interest in designing learning-based hashing functions [5, 111, 31] to alleviate the limitations of LSH. If the data distributions only have *global* density patterns, good choices of hashing functions may be possible. However, it is not possible to construct global hashing functions capturing diverse *local* patterns. Actually, k-NN distance (i.e., the desired search range) depends on the local density [108] around the query, e.g., a local feature such as a dense group or sparse hull. Since every global function is used for all the data points, when it is learnt to adjust to one local pattern, it will also cause lots of side-effects for other areas. In such cases, no global choice can be good.

We propose the concept of *Selective Hashing* to give a locally optimized index option for every data point. It is a meta-algorithm for k-NN search which works on the top of radius search algorithms such as LSH. Our main innovation is to create multiple LSH indices with different granularities (i.e., radii). Then, every data object is only stored in one selected index, with certain granularity that is especially effective for k-NN searches near it. Using this idea, our method achieves good quality for k-NN search regardless what the distance is, while avoiding having to maintain a huge set of complete indices for various distance radii (and additional query cost to visit multiple indices).

Within the framework of selective hashing, we further study what kind of granularity is most effective for a data point. A main insight here is that selecting the granularity has the equal effect of setting a *guard range* for every data object, i.e., when a query point falls into that range, this data object will be checked. Although k-NN is not a symmetric relation, the inverse relation can be satisfied with high confidence for only a small constant overhead. For example, a 20-NN of a point will have 99% probability to be a reverse 100-NN of that point as well. Thus, for every data object, we estimate its density, decide the *guard range* with certain confidence, and select the index with corresponding granularity.

Comparing k-NN search with radius search, the only difference is that k-NN search does not provide the radius in the query. Thus, it is a strictly harder problem than the radius search where the radius equals to the distance of k_{th}-NN. As LSH has been refined as a technique, it is now near-optimal for high-dimensional radius search, and the gap between the current solution [3] and the lower bound [70] is almost closed. However, no such good results exist for the k-NN problem and there remains a gap between efficient k-NN search and fixed radius search. Theoretically, we show that k-NN search using selective hashing can achieve the same recall as fixed $c_1 k_{th}$-NN radius LSH search with at most c_2 times overhead, where c_1 and c_2 are small constants. If the density does not change dramatically from one object to its nearest neighbors, we can show that c_1 is less than 5 and c_2 is less than 1.3. Experimentally, the observed overheads is under a factor of two in all the cases.

We propose an efficient algorithm to build the selective hashing index, as well as to handle updates. Unlike most learning-based approaches, selective hashing only needs to maintain the local density for each area, and re-select the radius when necessary. The amortized update cost is independent of the data size, and only depends linearly on the number of indices. We also consider write-intensive applications, for which we propose a lazy updating strategy to re-select the radius when the check-rate (i.e., the proportion of data to be accessed) for a certain object is high in the past queries.

We summarize the contributions of this chapter as follows.

• We propose a novel meta-method for indexing, called *Selective Hashing*, which combines multiple index instances together and is especially suitable for k-NN queries. Compared with previous works, selective hashing is the first to optimize the index structure *locally* for every data point. Further, it provides good quality results regardless of the distance of k-NN and does not incur extra storage cost in consuming multiple indices (Section 6.3).

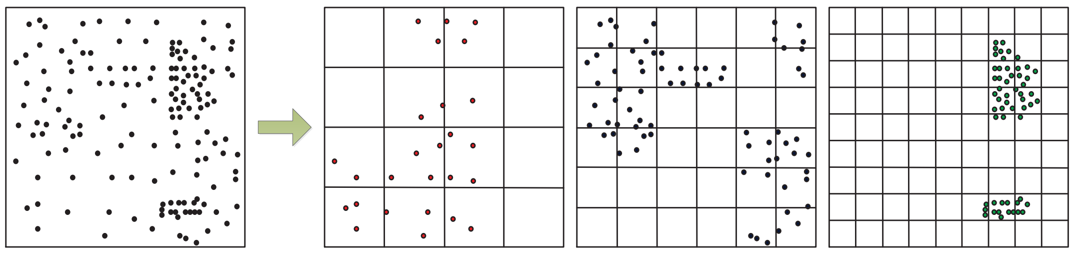
• We develop an effective index-selection scheme under the selective hashing framework. Our scheme guarantees high recall while achieving check-rate complexity comparable to a fixed radius search. In addition, this index is easy to construct and maintain with a cost that is comparable to the conventional LSH (Section 6.4).

• Experimental results show that selective hashing is effective for k-NN queries: producing high recall results while incurring low query time (i.e., low check-rate), compared to several leading alternative strategies. It is also small in size and fast to update incrementally (Section 6.4.4).

6.2 Preliminaries

In this chapter, we consider data objects represented as points in a d-dimensional metric space R^d. We use $\|p, q\|$ to denote the distance between two points p and q. Given a dataset, a point $q \in R^d$ and an integer k, the k-NN of the query q is the k data points that are nearest to q in the dataset. In the high-dimensional case, for the reason that the cost of exact k-NN problem usually grows linearly with the size of dataset, approximate k-nearest neighbor problem is accepted as a compromise.

Locality Sensitive Hashing [32] (LSH) is an efficient approximate algorithm for high-dimensional similarity search. It is efficient and provides a rigorous


(a) Indexing

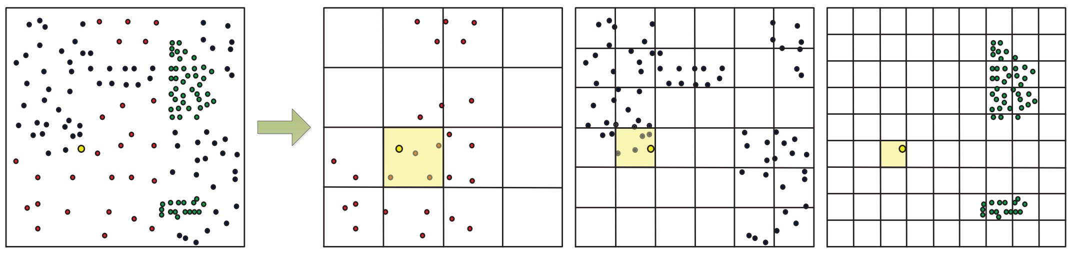

(b) Querying

Figure 6.1: Overview Intuition of Selective Hashing.

quality guarantee for finding similar points within a radius r. We denote the sphere centered at point q by $B(q, r)$. Formally, $B(q, r) = \{p | p \in R^d, \|p, q\| \leq r\}$. The general idea of LSH is that objects that are within a given distance will be hashed to the same value with high probability, where the set of hash functions called LSH family:

Definition 6.1. (LSH Family, $\mathcal{H}$) *A family $\mathcal{H} = \{h : R^d \rightarrow U\}$ of functions is called (r, cr, p_1, p_2)-sensitive if $\forall p, q \in R^d$:*

- *if $p \in B(q, r)$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \geq p_1$;*
- *if $p \notin B(q, cr)$, then $Pr_{\mathcal{H}}(h(p) = h(q)) \leq p_2$;*

LSH family for Euclidean distance is accomplished by random Gaussian projection. There are also various versions of LSH family for Jaccard similarity, cosine distance, Hamming distance etc.

LSH usually applies a concatenation of m hashing functions randomly selected from the hashing family to build the hash table(reducing check-rate), and builds l independent hash tables and takes the union of results (raising recall). m and l are parameters of LSH and the best value is related to the data size. In querying phase, all points collide with the query point in any of the hash tables will be considered as candidates. The exact distances between the query and all the candidates will be validated. For k-NN query, the top-k in candidates will be returned and for radius query, those satisfy the radius threshold will be returned. Consider a group of LSH tables as a whole, it provides the following properties:

Lemma 6.1. *Given a group of LSH tables denoted as $\mathcal{G}$, $\forall p, q \in R^d$, $\mathcal{G}(p,q)$ is true iff p and q collide in at least one of the hash tables. Then:*

- *if $p \in B(q,r)$, then $Pr(\mathcal{G}(p,q)) \geq P_a$;*
- *if $p \notin B(q,cr)$, then $Pr(\mathcal{G}(p,q)) \leq P_b$;*
- *$P_a = 1-(1-p_1^m)^l$, $P_b = 1-(1-p_2^m)^l$, $P_b \ll P_a$.*

In subsequent sections, we will consider a group of LSH hash tables $\mathcal{G}$ as a black-boxed index, offering a radius search with some recall guarantee. Taking p_1, p_2 and m as inner parameters optimized based on a certain approximation ratio c, and tuning l to adjust the recall requirement, we have:

Lemma 6.2. *To get a recall $P_a = 1-\delta$, the check-rate P_b for points outside $B(q,cr)$ for $\mathcal{G}$ is bounded by $\log\delta * \phi$, where ϕ is an inner property of $\mathcal{G}$.*

Proof. $\delta = (1-p_1^m)^l$

$$\begin{aligned}
\Rightarrow & l = \log\delta / \log(1-p_1^m) \\
\Rightarrow & P_b = 1-(1-p_2^m)^l \\
\Rightarrow & P_b \leq l * p_2^m \\
\Rightarrow & P_b \leq \log\delta * p_2^m / \log(1-p_1^m)
\end{aligned}$$

Here, $\phi = p_2^m / \log(1-p_1^m)$ is an inner property of $\mathcal{G}$. □

The number of tables (i.e., index size) is also proportional to $\log\delta$ (it is also relevant to the data size in a theoretically optimized solution). Note that achieving a high recall typically does not lead to a significant increase in query complexity (or the number of tables), as it only affects the term $\log\delta$.

To simplify the analysis here we use Euclidean Selective Hashing and Euclidean LSH (i.e., E2LSH) as a running example throughout the rest of discussion. The Euclidean requirement is not compulsory, since we only need a black-boxed radius search index that can offer arbitrary high recall as required. In our subsequent analysis, we also only need the properties of a metric space. Given a radius search algorithm for any metric space with such recall and check-rate guarantee, selective hashing can support k-NN search for such space by using it as a black box.

The theoretical bound of Euclidean LSH can be presented in two ways depending on the problem requirement. If the checking procedure is terminated when sufficient number of points in $B(q, cr)$ is found, then it returns a c-approximate r-NN solution, and its query time is $O(dn^{1/c})$. If all the buckets are fully checked and only those points within $B(q, r)$ are returned, then the results are exact r-NN with a constant recall δ. Usually there is no theoretical bound on the check rate for such an exact solution, since LSH cannot distinguish the points in $B(q, cr)$ - $B(q, r)$ from those in $B(q, r)$. However, in practice, the worst case seldom happens. In this thesis, we only focus on the overhead using LSH to solve k-NN problem, and the main result can be achieved by using both versions of theoretical analysis. Since most LSH implementations, such as E2LSH [21], choose to do exact search, and recall is the most common measure for the k-NN problem, we adopt the exact search LSH as the base algorithm for the rest of our analysis.

6.3 Selective Hashing Framework

Selective hashing is a meta-method for indexing. It combines multiple index instances together, e.g., black-boxed LSHs, and produces a more efficient index without extra storage or query cost. Considering that we have multiple index choices for data objects, a traditional approach usually includes all the objects in every index. At query time, there is only one index or a few indices to be consulted. In contrast, selective hashing builds multiple indices, with each object in the dataset being placed in only one index. At query time, all the indices have to be consulted to locate the objects of interest. Thus, selective hashing works in a "select one, query all" manner. It is easy to see that this scheme can provide correct and complete results: each object exists in exactly one of those indices, each of which will be accessed for any single query. The check rate of each object only depends on the index in which it is stored. Each index only contains a disjoint fraction of objects, and hence selective hashing will not bring in any additional cost (especially for hash-based methods with $O(1)$ lookup cost). Moreover, there is almost no extra storage overhead as well. In contrast, traditional methods usually need to build multiple complete index instances with all the data objects stored, which may lead to a huge index size and limit the possibility of combining more indices.

We create multiple indices with different radii $R, cR, c^2R, \ldots,$ and adopt selective hashing on the top of these indices. Figure 6.1 gives an example of our workflow. In the indexing phase, for each data object, we select one index unit to use. The index selection is based on the local density around that data point. Finally, we will have a group of indices, and each data object is stored in only the most suitable one. In Figure 6.1a we see how each data point is mapped to one of three indices (with c=1.5). Data points in dense regions are stored in the right (small granularity) index, while data points in sparse regions are stored in the left (large granularity) index. In the querying phase, we push down the query to all the indices and collect the results. Then we aggregate the results and return the top k. In Figure 6.1b all the indices are used to retrieve the nearest neighbors of the given query. The search range in each index is indicated by the shaded cell. The nearest neighbors are found in the left two indices, with no nearby neighbors at all in the right index. In the dense area, most of the data points tend to be in indices with small granularities. When a query falls into such areas, although indices with large granularity are also checked, the data points are only stored and accessed in indices with small granularities. Thus, the actual search range is small accordingly to avoid unnecessary checking but large enough to get accurate results. In the sparse area, however, the query needs a larger search range. Data points are stored in indices with larger granularities to ensure recall. In short, the search range for a query is automatically tuned so that only the potential k-NN candidates will be checked.

From the perspective of data objects (or points), we can also see that every data object has chosen its best local structure for k-NN search. For a k-NN search problem, the best local structure for a query is to only store k-NN while leaving other data objects outside. Note that only query-object relations are considered and queries are pushed to all the indices, so similar objects can choose different radius and be placed in different indices. Thus, the radius selected by each object is determined independently. Although all the hash functions are defined globally, only those with suitable radius are used for each object. Essentially, we use the global hash tables to offer locally optimized index structure. While there are learning-based methods, such as DSH [31], that attempt to optimize hash functions based on the data, they are much harder to tune, since every hash function affects all objects. Now the only problem is that we can only select the index for each data point, not each query. Notice that data p is a k-NN of query q equals with q is a reverse k-NN of p. We need to choose the radius that is similar to the range that contains all the reverse k-NN (i.e., potential queries) of the data point. k-NN is not a symmetric relation, however, we will build a connection between them in the next section.

6.4 Algorithm and Analysis

6.4.1 Density Estimation

As mentioned before, our radius selection is based on the local density near each object. Many methods have been proposed for density estimation on multi-dimensional data [89, 108]. We shall first define density to serve the need in radius selection, in a manner inspired by these previous methods. Then we propose a simple density estimation algorithm based on collisions in LSH tables.

Density Definition

Definition 6.2. (Data Density Observation $D_o(p, r)$)
We define the density observation of one point p within radius r as the number of points in the area $B(p, r)$. For a given dataset O, the density observation can be represented as $D_o(p, r) = |\{o|o \in O, o \in B(p, r)\}|$.

This definition makes all the following analysis independent of the properties of space: all we need is a distance measure. Note also that LSH is considered as a black box with only radius r and recall δ as parameters. Henceforth, we only need to consider the k-NN problem in a metric space.

Further, we consider the generation model behind density observation. Points appearing in a certain area can be formally defined as a spatial Point Process [51]. Although the density of different areas may be different, by looking into two disjoint areas, the counts of points are independent of each other but only depend on their own densities. Thus, this point process can be described as a spatial Poisson process [51] where the number of points in a certain area (e.g., density observation) follows Poisson distribution. We define the expectation of $D_o(p, r)$ as *actual data density* $D_a(p, r)$.

Definition 6.3. (Actual Data Density $D_a(p,r)$)
We consider data objects that are generated from spatial Poisson process. The number of points in a certain area $B(p,r)$ is a random variable following Poisson process, and its expectation value is defined as the actual density $D_a(p,r)$. $D_o(p,r)$ is the observation of $Poisson(D_a(p,r))$.

The above two definitions emphasize two different aspects of the density concept respectively. From the original physical perspective, density is described as the mass divided by a unit volume, which is an observation. And from the statistical view point, density refers to the probability density function (PDF) of a certain random variable, which is the hidden parameter behind the observation. Therefore, we use two terms, density observation D_o and actual density D_a, to eliminate this ambiguity. For a Poisson distribution $F_{Poisson}(x;\lambda)$, when λ is more than 10, its cumulative distribution function (CDF) can be well approximated by the CDF of Gaussian distribution [51] [1]:

$$CDF_{Poisson(\lambda)}(x) \approx CDF_{Gaussian(\mu=\lambda;\sigma^2=\lambda)}(x) = \Phi(\frac{x-\lambda}{\sqrt{\lambda}})$$

In most of the density estimation works [89, 108], there is one common but indispensable assumption: regardless of the randomness of observation, the actual density should be a continuous and smooth enough function over its domain space. Only with this assumption, can we recover the actual density from a set of isolated points. In our work, we also make a similar assumption about the continuity of actual density — λ-continuity for actual data density. It assumes that for two points within a k-NN distance, the actual density changes at most λ times.

Definition 6.4. (λ-Continuity for Actual Data Density)
Given a dataset O, the λ-continuity for data density is satisfied iff: $\forall p$, let R be its k_{th}-NN distance, i.e., $D_a(p,R) = k$, then $\forall o \in B(p,R)$ and $r \leq R$, $D_a(o,r) \leq \lambda D_a(p,r)$.

λ-continuity for data density assumes that the actual density should not change dramatically (i.e., λ times) among the nearest neighbor pairs. For most

[1]In this chapter, the main bulk of analysis are from the perspective of the CDF of Gaussian. Note that we can also analyze from the CDF of Poisson and similar results can be derived. However, the CDF form of standard Gaussian Φ is more intuitive and commonly used.

typical real datasets, even with high skew, λ is no more than 1.5-3. It is interesting that even when outlier clusters exist, λ will still be relatively low (e.g., 3), since the actual density is much smoother than the observation.

Observation Estimation

After having provided a formal definition of density, we are now ready to discuss the estimation algorithm. Specifically, we aim to get the observation $D_o(p,r)$. Indeed, the exact value of $D_o(p,r)$ can be directly counted by enumerating the whole dataset. However, the computation cost is usually not affordable — the complexity of computing the observations for all N data points is $O(N^2)$. In the following analysis, we are only concerned with the lower and upper bounds of such observations. It is therefore sufficient for us to estimate these bounds for $D_o(p,r)$ effectively.

A natural idea here is that the estimation can be directly obtained from LSH tables via collision counting. Collisions indicate how many points are within the radius. Combining the counts from a group of tables, the estimation can be quite accurate.

Theorem 6.1 (Lower Bound for $D_o(p,r)$)**.** *Given an LSH index $\mathcal{G}$ with radius r/c and properties P_a and P_b described in Lemma 6.1, using $\#Collision(p,r)$ to denote the number of points collides with p (i.e., $\mathcal{G}(p,q)$ is true), then with a probability of ζ, $D_o(p,r)$ will be large than $LB(p,r)$, where:*

$$LB(p,r) = \#Collision(p,r) - P_b * N - \Phi^{-1}(1-\zeta) * \sqrt{P_b * N}$$

Proof. Recall that from an LSH group with a radius $r' = r/c$ and approximate ratio c: for all collided points of p, the expected number of points outside $B(p,cr')$ (i.e., $B(p,r)$) is at most $P_b * N$. Besides, the collision of one data point is independent with the others. Thus, the count of such collisions follows a Poisson distribution with a λ (i.e., expectation) less than $P_b * N$. Using Gaussian to approximate Poisson, it follows $Gaussian(P_b{*}N, P_b{*}N)$. The PDF of Gaussian (or Poisson) drops exponentially in the tail (3 standard variances or more), we can compute an upper bound $UB(p,r)$ for such a random variable with confidence $1 - \zeta$:

$$\begin{aligned}
& x \sim Poisson(P_b * N) \\
& \Pr(x < UB(p, r)) = 1 - \zeta \\
\Rightarrow & UB(p, r) = {CDF_{Poisson(P_b * N)}}^{-1}(1 - \zeta) \\
\Rightarrow & UB(p, r) = P_b * N + \Phi^{-1}(1 - \zeta) * \sqrt{P_b * N}
\end{aligned}$$

Φ is the CDF of the standard Gaussian. The collisions consist of two parts: one part is from points outside $B(p, r)$ which has an upper bound derived above, the other part is from points within $B(p, r)$. Thus, we have the lower bound for $D_o(p, r)$ denoted as $LB(p, r)$, where $LB(p, r) = \#Collision(p, r) - UB(p, r)$. □

Although this bound is not tight, it tells us that we can obtain a similar estimation in a range that is c times larger since $D_o(p, r/c) * P_a$ points will appear in $\#Collision(p, r)$. Note that in the radius selection, the step size of choosing the next radius is also c times. Therefore at the worst case we select the next larger radius than the most suitable one. In real applications, this estimation works well that the lower bound is about a half of the actual value.

The upper bound for $D_o(p, r)$ can be derived analogously. In what follows, we will directly use the value $D_o(p, r)$ without considering how it is estimated (or simply counted). Giving a more accurate (or even faster) estimation is not the main focus of this chapter and we deal it only briefly here.

6.4.2 Radius Selection

Armed with the power of λ-continuity for data density and density observation of data points, we are ready to solve the core radius selection problem. We first present the selection algorithm that satisfies the quality guarantee, and then analyze the cost of query.

Selection Algorithm

The key problem of selective hashing is to choose an index with suitable radius for each data point. A main insight here is that by selecting a radius for a data point, we are in fact setting a *guard range* for this point, i.e., when a query point falls into that range, this point will be checked. We use the term *target* *queries* of data p to denote those potential queries which contain p in their k-NN results. Obviously, k-NN is not a symmetric relation and we cannot guarantee that all target queries will fall into the guard range (unless we set the guard range to be the full space). However, the confidence of target queries falling into the guard range could be arbitrary high by the following detailed analysis. Thus, to ensure a recall of $1-\delta$, we allocate the allowed error rate δ into two parts: (1) $\delta/3$ for cases where LSH fails to get k-NN when target queries fall into guard range; and (2) $2\delta/3$ for cases where the target queries fall outside the guard range. The split parameter (1/3, 2/3) here is optional, since we only aim to build a linear connection between radius search and k-NN search while the constant does not matter at the theoretical level.

To match the first part, we will build our LSH table groups using $\delta' = \delta/3$. Then we consider how to find the guard range (i.e., radius to be selected) so that the overall probability of target queries falling outside the guard range of their k-NN is bounded by $2\delta/3$.

Theorem 6.2 (Radius Selection)**.** *Given a dataset under λ-continuity assumption, and using ϕ to denote $\Phi^{-1}(1-\delta/3)$, let $\mathcal{B}_k = \lambda k' + \Phi^{-1}(1-\delta/3)\sqrt{\lambda k'}$, where $k' = k+\phi(\sqrt{\phi^2+4k}+\phi)$. Given a data point o, the confidence*[2] *of target queries falling into r where $D_o(o,r) \geq \mathcal{B}_k$ is larger than or equal to $1-2\delta/3$.*

Proof. For a k-NN query q, we denote the distance between the query and its k_{th}-NN point as r', and we have $D_o(q,r') = k$. As $D_o(q,r')$ is an observation of $Poisson(D_a(q,r'))$, we know that with a confidence of $1-\delta/3$:

$$\begin{aligned} &D_a(q,r') - \phi\sqrt{D_a(q,r')} \leq k \\ \Rightarrow &D_a(q,r) \leq k + \phi(\sqrt{\phi^2+4k}+\phi) \end{aligned}$$

We denote this upper bound of confidence interval as k'.

Now we consider k-NN of q and we want to determine the guard range. First we analyze $D_o(o,r')$ for $o \in B(q,r')$ (i.e., o is a k-NN of q). According to λ-density continuity, we have $D_a(o,r') \leq \lambda D_a(q,r')$. Besides, we also know $D_o(o,r') \sim Poisson(D_a(o,r'))$. Thus with probability $1-\delta/3$ (splitting the

[2]There is a slight difference between probability and confidence, since the parameter being estimated is not a random variable but a fixed value. The confidence $1-\delta$ here means that with probability $1-\delta$, the confidence interval based on the observation will contain the actual parameter.

error $2\delta/3$ again into two halves):

$$\begin{aligned} D_o(o, r') &\leq D_a(o, r') + \phi\sqrt{D_a(o, r')} \\ &\leq \lambda D_a(q, r') + \phi\sqrt{\lambda D_a(q, r')} \end{aligned}$$

Since $D_a(q, r') \leq k'$ with confidence $1-\delta/3$, the following statement is satisfied with the probability larger than $1 - 2\delta/3$:

$$D_o(o, r') \leq \lambda k' + \phi\sqrt{\lambda k'}$$

We denote this upper bound of confidence interval for $D_o(o, r')$ as $\mathcal{B}_k$. Up to now, we know that with confidence larger than $1 - 2\delta/3$, $D_o(o, r)$ will not exceed $\mathcal{B}_k$. Therefore, if we set the guard range of the data object to be r, where $D_o(o, r) \geq \mathcal{B}_k$, with confidence larger than $1 - 2\delta/3$ we have $r' \leq r$, which means the query falls into its guard range. □

If this guard range assignment strategy is applied to all the data points, for every <query, k-NN> pair, the guard range of k-NN will cover the query with a confidence larger than $1 - 2\delta/3$. Combined with the failure probability of LSH which is smaller than $1 - \delta/3$, the overall recall is larger than $1 - \delta$.

In summary, for each data point o, we can select the radius r from candidate set $\{R, cR, c^2R, \dots\}$ where r is the smallest value that satisfies $D_o(o, r) \geq \mathcal{B}_k$. Then for any query, the probability its k-NN to be returned is larger than $1-\delta$. The algorithm for the overall radius selection and index process is described in Algorithm 9.

Here we give a complexity analysis for the indexing cost of Selective Hashing compared with E2LSH. E2LSH involves 3 steps: 1) Get the inner products between data points and projection vectors in hashing family (using F to denote the size of hashing family). $O(FDN)$. 2) Get the integer codes based on the product, bias and projection width. $O(FN)$. 3) Compute the buckets for points based on universal hashing. Insert the points into their corresponding buckets. $O(MLN)$. Hence, the complexity of E2LSH is $O(FDN + MLN)$. Selective Hashing involves 3 major steps: 1) step 1 in E2LSH. $O(FDN)$. 2) For H radii: follow step 2 and 3 in E2LSH to build LSH, $O(MLN)$; collision counting, $O(LN)$; radius selection by comparing $LB(p, r)$ and threshold B_k, $O(N)$. $O(HMLN)$ in total. 3) follow step 2 and 3 in E2LSH to insert points into the selective hashing index. $O(MLN)$. Hence, the complexity of Selective

Algorithm 9: INDEX

```
  Input: Dataset: List⟨Point⟩ $O$
  /* smallest radius is $R$, increment ratio is $c$, and number of
     indices is $SH.size$                                              */
  Output: Index: List⟨LSH⟩ $SH$
1 foreach $p$ in $O$ do
      /* observation estimation                                        */
2     find $r$ where $D_o(p, r) = \mathcal{B}_k$;
      /* insert into only one index.                                   */
3     integer $i = \lfloor \log_c (r/R) \rfloor$;
4     if $i < 0$ then $i = 0$;
5     ;
6     if $i \geq SH.size$ then $i = SH.size - 1$;
7     ;
8     $SH[i].insert(p)$;
9 return $SH$
```

Hashing is $O(FDN + HMLN)$. Typically, M, L and H are about 20, D is more than 100, F is around 100. Since HML is usually smaller than FD, step 1 is the dominant cost in Selective Hashing, which is consumed in all the hashing indexes. That is why Selective Hashing does not involve H times of cost compared with E2LSH.

Cost Analysis

We first introduce an optimal solution for k-NN using radius search, which cannot be achieved in reality. After that, we discuss the overhead of selective hashing compared to the optimal one.

Using radius search to solve a k-NN problem, the best result it can achieve is to foresee distances of k-NN and to index/query with the k_{th}-NN distance as the radius. Obviously, the k-NN distances of different queries differ greatly, and for different queries the whole dataset needs to be indexed with different radii. Thus, this approach cannot be realized in practice. However, it represents the best we can achieve with the idea of radius search algorithm for k-NN search. Its cost is just the same as conducting a simple radius search. We define this optimal solution as *optimal k-NN*. When the radius search algorithm is LSH, we call it *optimal LSH*.

Now we analyze the overhead of selective hashing compared to the optimal k-NN (e.g., optimal LSH). We start by comparing the number of returned points in a radius, and the additional cost of checking outside points will be discussed later. For optimal k-NN, the number of points in the radius is exactly k, since the radius is set as k_{th}-NN distance. For selective hashing, some more points will also be checked. The exact number equals to the count of data points whose guard ranges cover the query. According to Section 6.4.2, we know that all the reverse $\mathcal{B}_k$-NN points have a guard range which covers the query point. Thus, all the reverse $\mathcal{B}_k$-NN of this query need to be checked.

Unfortunately, we cannot tell how many reverse $\mathcal{B}_k$-NN a given query has. Intuitively, the average should be $\mathcal{B}_k$, but the actual number depends on the specific query. An adversary could deliberately choose points with large number of reverse $\mathcal{B}_k$-NN. However, if the query distribution is expected to be "well-behaved", this is an upper bound for the expectation number. Similar to data distribution previously described, we define the query density as Q_a, and assume λ'-continuity for query density, which limits the rate of change for Q_a/D_a.

Definition 6.5. (λ'-Continuity for Query Density)
Using $\mathcal{S}$ to denote the whole space, D_a to denote the data distribution, and Q_a to denote the query distribution, where D_a and Q_a are the density function over the whole space, giving some area A, the expectation of the number of data points can be expressed as the integration of D_a on A: $\int_{p\in A} D_a(p)\,\mathrm{d}p$. [3] *And it is analogous for Q_a. Using $D_a^{-1}(p,k)$ to denote the inverse function for $D_a(p,r)$, which returns a radius r such that $D_a(p,r)=k$, the λ'-continuity for query density is satisfied iff:*
$\forall p \in \mathcal{S}$, $\forall o \in B(p, D_a^{-1}(p,\mathcal{B}_k))$, $Q_a(o)/D_a(o) \le \lambda Q_a(p)/D_a(p)$.

Note that we do not require the query distribution to follow the data distribution, but the ratio should be a continuous function. Under this assumption, we have the following theorem.

Theorem 6.3 (Expectation of reverse $\mathcal{B}_k$-NN)**.** *When the query distribution follows λ'-continuity for query density, the expectation of reverse $\mathcal{B}_k$-NN is at most $\lambda'\mathcal{B}_k$ per query.*

Proof. For any point $p \in \mathcal{S}$ (e.g., R^d for a multi-dimensional space), its guard range is $B(p, D_a^{-1}(p,k))$, and its expectation number of reverse $\mathcal{B}_k$-NN can be

[3]This definition is also consistent with the previous one $D_a(p,r)$, which can be represented as $\int_{p\in B(p,r)} D_a(p)\,\mathrm{d}p$.

defined as $E(p)$:

$$E(p) = \int_{q \in B(p, D_a^{-1}(p,k))} Q_a(q) \, \mathrm{d}q$$

Using $t(p)$ to denote the ratio $Q_a(p)/D_a(p)$, due to the λ'-continuity for query density, we have $Q_a(q) \leq \lambda' t(p) D_a(q)$ for any $q \in B(p, D_a^{-1}(p,k))$. On the other hand, $D_a(p, D_a^{-1}(p, \mathcal{B}_k))$ is just $\mathcal{B}_k$. Consequently, we have an upper bound for $E(p)$:

$$\begin{aligned} E(p) &\leq \int_{q \in B(p, D_a^{-1}(p,k))} \lambda' t(p) D_a(q) \, \mathrm{d}q \\ &= \lambda' t(p) \int_{q \in B(p, D_a^{-1}(p,k))} D_a(q) \, \mathrm{d}q \\ &= \lambda' \mathcal{B}_k t(p) \end{aligned}$$

Thus, the expectation of <object, reverse $\mathcal{B}_k$-NN query> pairs for the whole space is $\int_{p \in \mathcal{S}} D_a(p) E(p) \, \mathrm{d}p$. And on the other hand, we know the total number of query is the integration of Q_a on $\mathcal{S}$, i.e., $\int_{p \in \mathcal{S}} Q_a(p) \, \mathrm{d}p$. Therefore, the upper bound of the expectation of reverse $\mathcal{B}_k$-NN per query is:

$$\begin{aligned} \frac{\int_{p \in \mathcal{S}} D_a(p) E(p) \, \mathrm{d}p}{\int_{p \in \mathcal{S}} Q_a(p) \, \mathrm{d}p} &\leq \frac{\int_{p \in \mathcal{S}} D_a(p) \lambda' \mathcal{B}_k t(p) \, \mathrm{d}p}{\int_{p \in \mathcal{S}} Q_a(p) \, \mathrm{d}p} \\ &= \lambda' \mathcal{B}_k \frac{\int_{p \in \mathcal{S}} D_a(p) \frac{Q_a(p)}{D_a(p)} \, \mathrm{d}p}{\int_{p \in \mathcal{S}} Q_a(p) \, \mathrm{d}p} \\ &= \lambda' \mathcal{B}_k \end{aligned}$$

□

Based on the above analysis, the search range of selective hashing is smaller than the distance of $\lambda' \mathcal{B}_k$-NN of the query. Thus, the check rate is smaller than the cost of searching for $\lambda' \mathcal{B}_{kth}$-NN distance in radius search. In general, the cost of searching $\lambda' \mathcal{B}_k$-NN will be smaller than $\lambda' \mathcal{B}_k / k$ times the cost of searching k-NN, since we can always search k objects each time and repeat $\lambda' \mathcal{B}_k / k$ times, in the absence of ties. However, there exist some query cases that other nearest neighbor points are much more difficult to find than the k-NN (e.g., k points collide with the query, while all other points in the dataset have almost the same distance to the query point).

Recall that the LSH in selective hashing requires a recall of $1 - \delta/3$, and it will lead to additional $(\log \delta - \log 3)/\log \delta$ times overhead. When k increases, $\lambda' \mathcal{B}_k / k$ decreases, and when δ increases, $(\log \delta - \log 3)/\log \delta$ also decreases.

Algorithm 10: QUERY

```
  Input: Query: Point $q$, Index: List⟨LSH⟩ $SH$
  Output: KNN result: List⟨Point⟩ $KNN_{current}$
1 $KNN_{current} = \{\}$;
2 List⟨Point⟩ $pts$;
  /* search indices begining from the smallest radius one */
3 for $i$ = 0 to $SH$.size-1 do
4     pts = $SH[i]$.search($q$);
5     foreach $p$ in $pts$ do
6         KNN.checkAndUpdate($KNN_{current}$, $p$);
      /* density-based pruning */
7     $dis = \max_{p \in KNN_{current}} \|p, q\|$;
8     if $dis < \mu \cdot SH[i].radius$ then break;
9 return $KNN_{current}$
```

Thus, $\lambda' \mathcal{B}_k / k$ and $(\log \delta - \log 3)/\log \delta$ can be bounded by some constants.

Theorem 6.4 (Effectiveness of Selective Hashing).
Selective hashing achieves the same quality for k-NN search as the fixed radius LSH search, with radius equals the distance to the $c_1 k_{th}$-NN, with as most c_2 times overhead, where $c_1 = \lambda' \mathcal{B}_k / k$ and $c_2 = (\log \delta - \log 3)/\log \delta$. c_1 and c_2 are bounded by constant.

Theorem 6.4 is just a theoretical bound with several relaxations made during its derivation. For typical datasets and query requirements, c_1 is bounded by 4-5 and c_2 is around 1.3. The real performance is much better as we will see in the experimental study.

6.4.3 Density-based Pruning

In the query scheme discussed above, the query is pushed to all the index instances. In this subsection, we discuss how to further reduce the check rate by not checking some index instances while still obtaining correct results.

We observe that indices with larger radii have the highest check rate. Intuitively, the guard ranges for points in those indices are very large, potentially including many irrelevant areas (e.g., a small dense area nearby). In our experiments on real datasets, we found that points in the top 20% indices with the largest radius typically have 2-5x the average check rate. Thus, we would like to avoid checking those indices if at all possible.

In selective hashing, usually none of the indices can be skipped, as every point only exists in one index. However, if a query has already found enough points within a very small radius, then all its k-NN should fall in a high density area and therefore would not have been placed in a large radius index. Thus, we can search the applicable indices from the smallest radius to the largest radius one by one, and stop searching when all the remaining indices will not contain any potential k-NN. The following theorem gives a proof for the correctness of this pruning.

Theorem 6.5 (Correctness of density-based pruning)**.** *For a query q, if $D_o(q,r) \geq k$ and $D_o(q,r') \geq \mathcal{B}_k$, all the indices with a radius larger than $c(r+r')$ will not contain k-NN of q.*

Proof. Let $kNN(q)$ to be the k-NN points of a query q. Since $D_o(q,r) \geq k$, $\forall o \in kNN(q)$, $||o,q|| \leq r$. Due to the triangle inequality, $\forall p \in B(q,r')$, $||o,p|| \leq r+r'$. For this reason, we have: $D_o(o,r+r') \geq D_o(q,r') \geq \mathcal{B}_k$.

On the other hand, the indexing algorithm will select the smallest radius r that satisfies $D_o(o,r) \geq \mathcal{B}_k$. And the radius increases c times each time. Thus, there will be a radius $R \in [r+r', c(r+r'))$ used in our selective hashing index. $\forall o \in kNN(q)$, o should be indexed in a radius equal to or smaller than $R < c(r+r')$. □

Using this pruning, r is the bound of k-NN and is naturally updated all the time. We may build another heap to maintain the $\mathcal{B}_{kth}$-NN distance r', or simply use a ratio μr to approximate $c(r+r')$ for better performance. Algorithm 10 outlines our querying strategy.

6.4.4 Incremental Maintenance

Efficient update is an essential requirement for dynamic index structures. Selective hashing can support efficient updates as well. Inserting or removing an individual point from a hash table is straightforward. The challenge is that updates can change the density of points in some regions, and therefore impact the choices made in constructing the index.

In the framework of selective hashing, the update process not only has to consider the insertion or deletion for the point to be processed, but also the impact on nearby points. For example, if there are many insertions in a sparse

method		OPT	SH	MLSH	SLSH	LLSH	DSH	ISO
query time(ms)	recall = 0.99	5.5	8.8	27.9	/	/	/	/
	recall = 0.96	4.2	5.5	19.5	/	52.3	14.5	28.5
	recall = 0.90	3.1	3.7	14.5	11.4	21.5	7.9	6.9
check rate(%)	recall = 0.99	0.61	0.97	2.33	/	/	/	/
	recall = 0.96	0.47	0.61	1.62	/	5.82	1.45	3.15
	recall = 0.90	0.37	0.40	1.23	1.26	2.35	0.88	0.79
index size(M)	recall = 0.99		283	3485	/	/	/	/
	recall = 0.96		170	1765	/	405	349	520
	recall = 0.90		105	827	264	124	152	145

(a) Flickr1M

method		OPT	SH	MLSH	SLSH	LLSH	DSH	ISO
query time(ms)	recall = 0.99	5.3	9.8	26.3	/	/	34.8	/
	recall = 0.96	4.1	5.9	20.2	/	47.6	13.7	21.5
	recall = 0.90	3.1	3.9	15.1	13.1	24.4	6.8	5.75
check rate(%)	recall = 0.99	0.57	1.08	2.99	/	/	3.62	/
	recall = 0.96	0.44	0.66	2.31	/	5.28	1.43	2.24
	recall = 0.90	0.33	0.42	1.71	1.46	2.71	0.71	0.64
index size(M)	recall = 0.99		148	1770	/	/	347	/
	recall = 0.96		101	832	/	207	137	160
	recall = 0.90		66	441	113	70	90	80

(b) MillionSong

Table 6.1: Overall Comparison

area, then the area becomes denser and all the points in this area should be moved to a smaller radius in order to reduce the check rate. Therefore, we need to make sure that the radius selection is based on the correct density observation. Thus, we need to maintain the density information for each data point, and re-select the radius when necessary.

First, we observe that the radius selection for each point is performed independently. That is, the choice of radius depends only on the existence of nearby points and not on the radius choice for these points. Therefore, when an update is applied, we only need to re-compute the density observation of points that may be affected. The number of affected points depends on the applied algorithm that estimates the density observation. For example, if it is estimated based on collisions in LSH, all the points that collide with the updated point should update their count of collisions and move to a new radius R where R is the smallest one satisfying $LB(p, R) \geq \mathcal{B}_k$. The number of points being affected is around $\mathcal{B}_k + P_b * N$ and such cost will not be high in most cases.

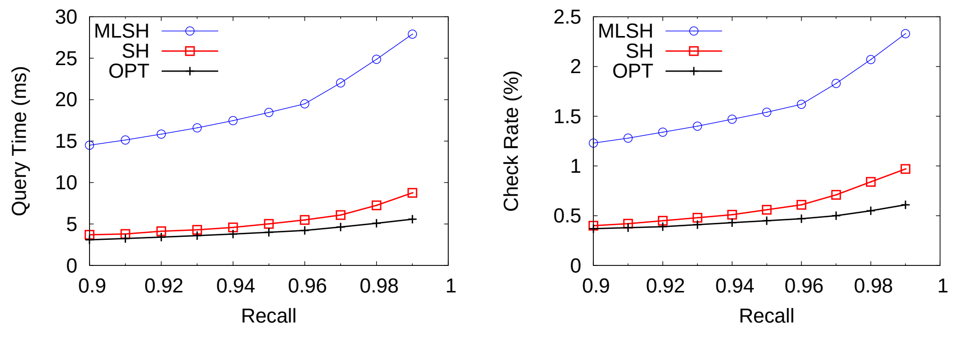
In write-intensive applications such as social networks and sensory observations, data points arrive as a stream. Most updates are new insertions and sub-optimal query performance may be acceptable in return for faster updates. To address this scenario, we propose a lazy updating strategy to postpone the radius re-selection to the moment when the check rate for an object becomes unacceptably high. For new insertions, the radius is only reduced and never increased. Thus, using the original radius will only affect the check rate, while the recall is still guaranteed. Instead of monitoring the density observation, lazy updating maintains the check rate information for each point, re-computes its density observation and re-selects its radius when its check rate is higher than some threshold. When a new point comes, we simply insert it into selective hashing index. Such updating strategy slight increases the query time while achieving an update speed as fast as the naive LSH.

6.5 Experiments

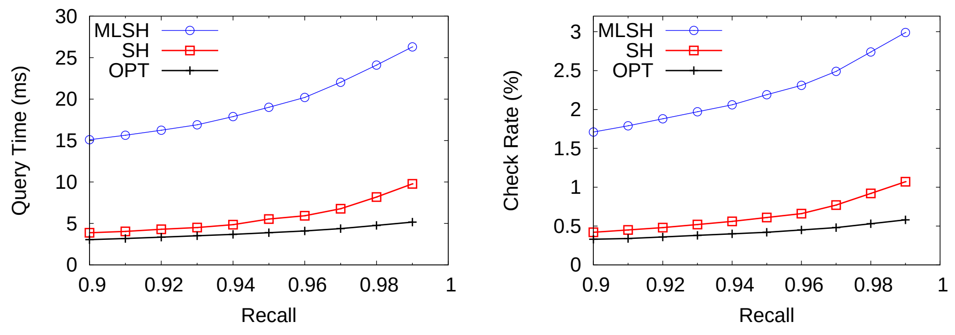
6.5.1 Experimental Setup

Datasets. We evaluate the performance using two real datasets: an image dataset *Flickr1M* and an audio recording dataset *MillionSong*, both of them represented as high-dimensional feature vectors. Flickr1M [97] contains extracted features of 1 million images from Flickr. Each image is represented as a 3,857-dimensional feature vector comprising a concatenation of SIFT feature, color histogram, etc. MillionSong [10] is a collection of audio features and meta data for a million contemporary popular music tracks. We split the audio into sections such that each contains 50 continuous small voice segments. Each segment is described using 12 MFCC features. We get 2 million sections and each section is a 600-dimensional vector. For whitening, normalization and dimension reduction purposes, following common practice, we use PCA to pre-process these two datasets and keep 200 and 100 dimensions respectively.

Methods and implementations. We have implemented the following meth-



(a) Flickr1M



(b) MillionSong

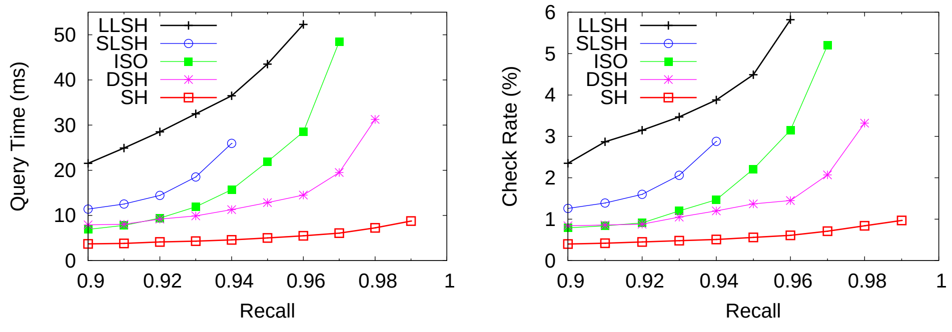
Figure 6.2: Comparison of Methods with Strict Query Guarantee

ods for comparison.

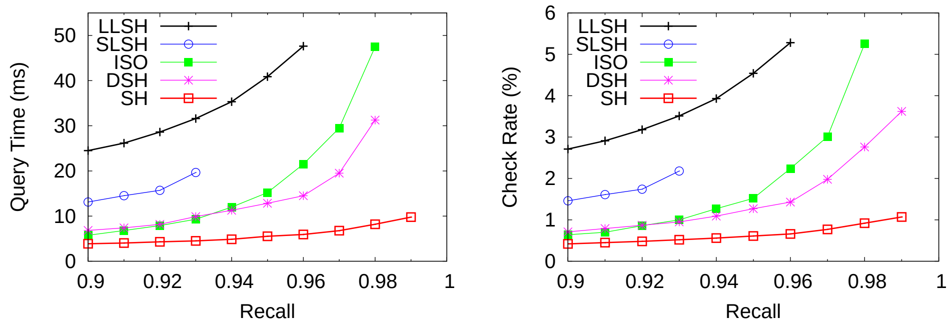
• *Selective hashing* **(SH)**. Our algorithm presented in Section 6.4. The default setting is with density-based pruning and normal updating.

• *Fixed radius LSH.* E2LSH [21] is the most common LSH implementation for exact Euclidean search. For each dataset, one small radius is selected for a more efficient solution *small radius LSH* **(SLSH)**, and one larger radius is selected for a more accurate solution *large radius LSH* **(LLSH)**. In general, the small radius one is more effective when recall is low. However, its recall cannot be further improved as there are many k-NN outside its radius. On the other hand, the larger radius one can offer higher recall, but is more likely to have too many collisions and hence requires a higher check rate.

• *Multi-radius LSH* **(MLSH)**. We build $|\mathcal{G}|$ E2LSHs using radii $\{R, cR, c^2R, \ldots\}$. When a query is issued, we search the indices from the smallest radius E2LSH to those with larger radii. When there are at least k points in current radius R, all the k-NN are within radius R; the recall guarantee holds and search procedure is stopped. Its index size is $|\mathcal{G}|$ times that of the fixed radius LSH.



(a) Flickr1M



(b) MillionSong

Figure 6.3: Comparison of Ad-hoc and Learning-based Methods

MLSH provides a strict theoretical guarantee for the recall.

• *Optimal LSH* **(OPT)**. We have presented an *Optimal LSH* in Section 9. It gives a lower bound for the cost of k-NN search. We test its performance using multi-radius LSH, while only searching the LSH which has the same radius as the distance of k_{th}-NN (pre-calculated and given as input).

• *IsoHashing* **(ISO)**. ISO [53] is a learning-based hashing method trying to give equal variance for all dimensions where quantizers are applied.

• *Data sensitive hashing* **(DSH)**. DSH [31] is a learning-based high-dimensional data-sensitive hashing which is directly designed for k-NN search and especially targeted to give high recall guarantee.

For all these hash-based indices, we follow the typical index and query strategy used in E2LSH which is described in Section 6.2. For DSH and ISO, all the hashing functions are used to build the hashing family. In table construction, universal hashing is applied to all the hash tables and the total number of buckets per table is 9973 (the closest prime to 10000). For SH, MLSH and OPT, the number of radii $|\mathcal{G}|$ is set to 20, and the approximation ratio c is set to 1.2. The largest radius can be 30x larger than the smallest one. Finally, the memory is large enough for all the methods to keep the original dataset and all the indices in memory. Parameters for all methods are optimized to offer the best performance. All the experiments were performed on a Ubuntu system equipped with one Intel 3.4GHz processor, 16GB of memory.

Measurements. We evaluate the performance using the following measures: *Recall* is used to measure the quality of query result. *Check Rate* and *Query Time* are used to measure the efficiency of algorithm. Precision or MAP is not used as measurement since all the algorithms are used to generate a set of candidates. The exact distances are calculated for the candidates, and the top-k are used as k-NN result. As an example, to solve a 20-NN search problem in 1M points, we expect the algorithm to return 2000 points that contain all the k-NN (1% precision 100% recall), but not to return 20 points that contain 12 of them (60% precision & recall). We can validate those 2000 points and obtain the exact answer. However, we cannot expand the 12 points to obtain a higher recall. For this reason, we only focus on the query performance under high recall (90% - 99%) requirements. We also compare *Index Size*, *Construction Time* and *Update Cost*. 1000 points are randomly removed from each dataset and used as query points. Unless specified otherwise, k is set to 20 except in the experiment comparing the performance of different methods under various k.

6.5.2 Query Performance

Query performance is the principal criterion for indexing. Table 6.1 summarizes the performance of the methods that are required to reach a specified recall on the two datasets. Only SH and MLSH (and the unrealizable OPT) can reach 99% recall on both datasets, and can be considered as having strict quality guarantee. Although DSH ensures the recall of the results on its training sets and performs much better than the other methods (ISO, SLSH and LLSH), there are still more than 1% missing results. In what follows, we first present the comparison results for the methods with strict quality guarantee, and then show the comparison results for ad-hoc and learning-based methods which are faster but achieve lower quality. After that, we study how different values of k will affect the performance for those methods, and also the effect of density-based pruning on SH.

Methods with strict quality guarantee. We first present comparison result for SH, MLSH and OPT. Figure 6.2 shows their performance on check rate and query time to reach a certain recall. The trends in two datasets are similar. Furthermore, the trends of check rate are consistent with those of query time since most query time is spent in computing the exact distances for the candidates. Specifically, all the algorithms take about 9ms to scan 1% of data (about 8MB data). From the figure, we can see that SH almost reaches the lower bound — OPT, when the recall is around 90%. When recall increases, δ becomes smaller and $\Phi(1-\delta/3)$ will be slightly larger. For this reason, SH uses 1.5x-2x time than OPT when recall is 99%. However, the effect of δ is bounded, since Gaussian CDF decays fast to zero in its tail. Compared to MLSH, both query time and check rate of SH are only about one third. The query time of MLSH is about 5x larger than OPT. From the result we can see, even if multiple indices for different radii have been built, the overhead of attempting a suitable radius is still very high. If the distance of k-NN varies W times, the overhead of query time can only be bounded by $O(\log_c W)$, and index size is also $O(\log_c W)$ times the fixed radius one. Moreover, SH also benefits from density-based pruning, while MLSH does not.

Ad-hoc and learning-based methods. From Table 6.1, it is clear that the index size of MLSH is huge and may not be acceptable in lots of real scenarios. Here we give a comparison for the methods using only one group of hash tables, including SLSH, LLSH, ISO, DSH and SH. Figure 6.3 shows their check rate and query time as a function of recall. SH performs more than 1.75x faster than DSH. This is because although DSH also aims to offer a better structure to maintain the k-NN relations, it is based on the optimization of global hash functions, whereas SH offers a more fine-grained local optimization for every data point. LLSH and SLSH give the worst performance, due to the fact that they do not have any optimization on k-NN at all. Since SLSH cannot provide high recall, say 96%, we shall use LLSH as the representative of the fixed radius LSH in subsequent comparisons.

Sensitivity to k. In most real-life applications (e.g., displaying the first page of results, k-NN joins, k-NN classifier), k is typically pre-defined throughout all the queries. However, it is important for an index to be insensitive to the value of k so that we only need to maintain one k-NN index to serve most potential

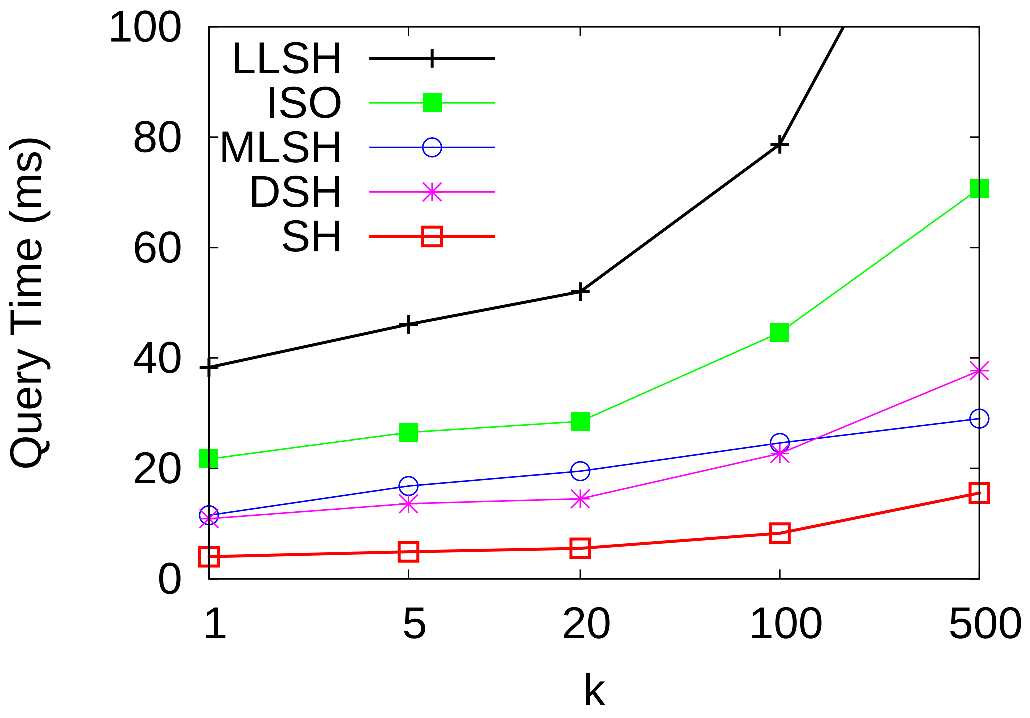
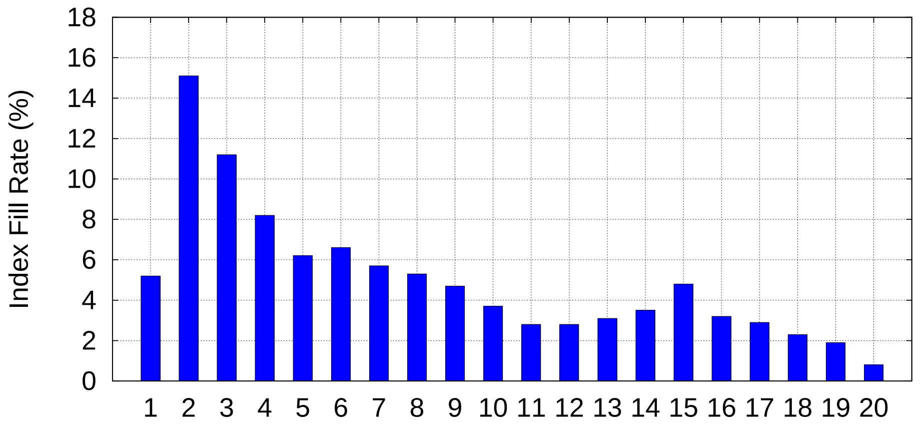


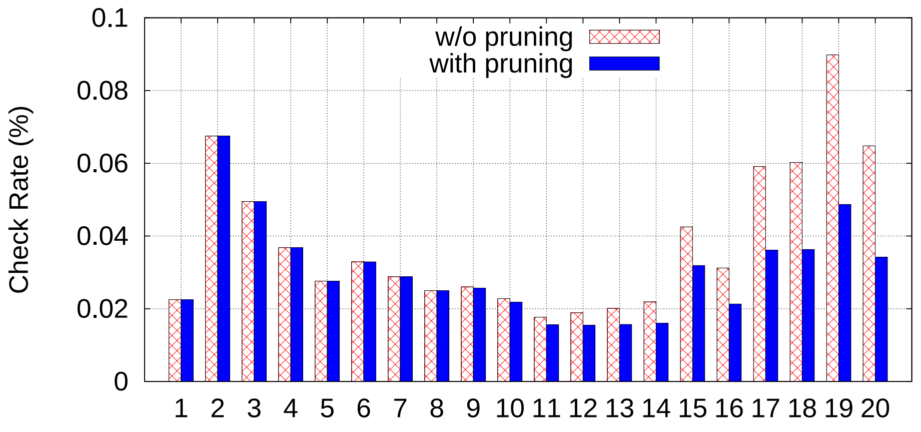
Figure 6.4: Sensitivity to k

applications. Here we study the query performance when k of the query varies while keeping $k = 20$ in index construction. Figure 6.4 shows the query time for achieving 96% recall for queries with different values of k. Obviously, the query with a larger k is harder to compute and needs longer query time. MLSH is the only method that does not use k for index construction and hence is not sensitive to k. For LSH-based methods SH and LLSH, their relative query time performance compared with MLSH increase around 25% for $k = 1$, 20% for $k = 100$ and 90% for $k = 500$. This increase can be viewed as the overhead due to using an incorrect k during index construction. For all the cases, SH is 2-5x faster than all the other methods. This is because the segment width of LSH may be effective for an interval of radius like $[r, 1.2r]$, with the performance dropping significantly outside the interval [21], while the distances of 100_{th}-NN (or 1-NN) and 20_{th}-NN usually do not have a significant difference. Therefore a single index (e.g., $k = 20$) can serve a moderate range of k values (e.g., 1 to 100), which serves the need of most applications and searches.

Density-based pruning. Here we study the effect of density-based pruning



(a) Data Distribution among Indices



(b) Pruning Effect on Indices

Figure 6.5: Density-based Pruning

on selective hashing. The distribution of data among different radii is more balanced in the Flickr1M dataset and therefore we use it to show how the check rate is changed. The distribution of data among all the 20 indices is shown in Figure 6.5a, where the first bar represents the index with the smallest radius. Figure 6.5b reports the check rate to reach 96% recall before and after the density-based pruning is adopted. Although the proportion of points in 4 indices with largest radii are only 10%, they contribute about 40% of the check rate. By using the density pruning strategy, their check rate is reduced by almost a half.

	Flickr1M			**MillionSong**		
Methods	#table	1x	10x	#table	1x	10x
SH	15	112	1020	17	86	800
MLSH	11	205	1922	10	143	1346
LLSH	40	112	1098	44	85	833
DSH	34	135	1215	26	122	1188
ISO	30	120	1140	24	90	880

Table 6.2: Index Construction Time(s)

6.5.3 Index Construction

The index size of each method has already been described in Table 6.1, and SH requires the least storage space to reach a given recall. Compared with MLSH, the index size of SH is only 7% - 10%. Compared with the other methods, the hash tables of SH are also the most space efficient. It will be fairer to compare the index size, construction time and update cost of each method to reach a given recall. For this reason, the cost of the fixed radius LSH can be higher than the other methods where fewer tables are constructed.

Efficient construction is also an essential requirement for indexing, since periodical index reconstruction is commonly adopted as a way to improve the query efficiency. Table 6.2 summarizes the index construction time for all the methods. We also conduct a scalability test where all the data points are duplicated 10 times. All the methods take about 9x-10x longer time in the 10x dataset, suggesting that the construction time is linear with respect to the size of the dataset. SH and LLSH incur the least time to construct the tables, followed by ISO and DSH. The construction time typically consists of four components: training hashing functions (or estimating density for SH); computing the inner product between data points and projection vectors; getting the key for each hash table; and inserting points into hash tables. Most approaches have the same inner product computation step and have the same cost if their hashing families have the same size. The major differences of construction time happen in the last two components, i.e., table construction. For SH, its hash tables are very efficient, thus fewer tables are built and the cost of insertions is therefore lower.

	Flickr1M		**MillionSong**	
Methods	insertion	throughput	insertion	throughput
SH	196	2.14	301	2.79
SH-Lazy	97	4.33	165	5.09
MLSH	236	1.78	329	2.55
LLSH	146	2.88	221	3.80
DSH	310	1.36	448	1.86

Table 6.3: Update Cost

Ratio	**Method**	Update(ms)	Query(ms)	Total(ms)
1:1	SH	3.0	97.7	100.7
	SH-Lazy	1.7	104.2	105.9
1:10	SH	30.2	97.7	127.9
	SH-Lazy	16.8	112.8	129.6
1:100	SH	301.6	97.7	399.3
	SH-Lazy	166.3	119.1	285.4
1:1000	SH	3015.3	97.7	3113.0
	SH-Lazy	1650.7	126.3	1777.0

Table 6.4: Lazy Updating

6.5.4 Update

We also evaluate the update cost, measured by the average processing time per thousand insertions, and the throughput, measured by the amount (MB) of data processed per second. Table 6.3 summarizes the results. Lazy updating version of SH, named SH-Lazy, is the fastest, reaching a throughput of 5MB per second which is usually sufficient to handle sensor data. This is because the update cost per table for lazy updating is very close to LLSH, while fewer tables are maintained to get the same quality of results. The update for DSH is relatively slow since it involves re-training a fraction of hashing functions. ISO does not support update, limiting its applicability as an indexing structure for dynamic databases.

We also study the effect of lazy updating in different scenarios. Table 6.4 shows the performance under different update/query ratios. In this experiment, the query and insertion of points are processed at the same time. Half of the points are selected as original points in the dataset and the other half are used as points for updating. Thus, the average density changes 2x during update when all the update points are processed. For lazy updating, the query performance degrades about 5% to 30%, whereas the update cost is reduced by half. Interestingly, query performance converges to the density monitoring approach when more queries are issued. This is because more queries may lead to more frequent updates, and therefore the estimation is more accurate. In general, when the update/query ratio is larger than 100, lazy updating is definitely a better choice.

6.6 Summary

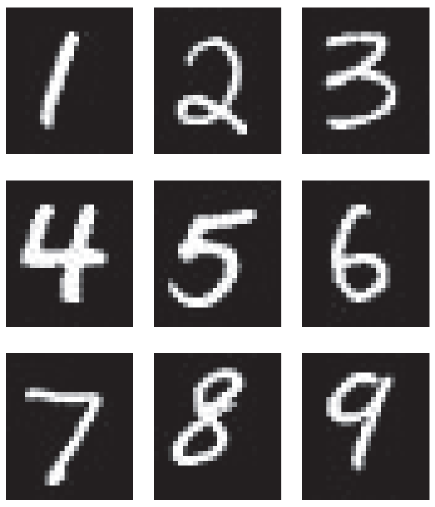
The LSH family of algorithms is considered to be very good at similarity search in high-dimensional space. However, it has been designed to address the problem of fixed radius search, rather than the k-NN problem. In this chapter, we introduced Selective Hashing (SH) as a meta-technique for k-NN search, constructed on top of any fixed radius search techniques, such as LSH. The key innovation in SH is the ability of choosing the index independently for each point, and then consulting multiple disjoint indexes for each query. The effectiveness of the proposed technique was established theoretically through careful analysis and demonstrated experimentally through performance studies using real datasets.

CHAPTER 7

LEARNING-VALUE BASED SAMPLING FOR EFFECTIVE DATA ACCESS

Recent years have witnessed amazing outcomes from "Big Models" trained by "Big Data". Most popular algorithms for model training are performed iteratively. Due to the surging volumes of data, we usually can only afford to process a fraction of the training data in each iteration. Typically, the data are either uniformly sampled or sequentially accessed.

We study how the data access pattern can affect model training. Based on the stochastic gradient descent (SGD) framework, we propose a novel approach called ActiveSampler, where training data with more "learning value" to the model are sampled more frequently. The goal is to focus training effort on valuable instances near the classification boundaries, rather than evident cases, noisy data or outliers. We formalize the information gain of each training instance and develop a light-weight vectorized algorithm to accelerate the training process. Extensive experimental evaluations demonstrate that ActiveSampler can speed up the training procedure of SVM, feature selection and deep learning by 1.8-2.3x, for comparable training quality.


(a) Easy

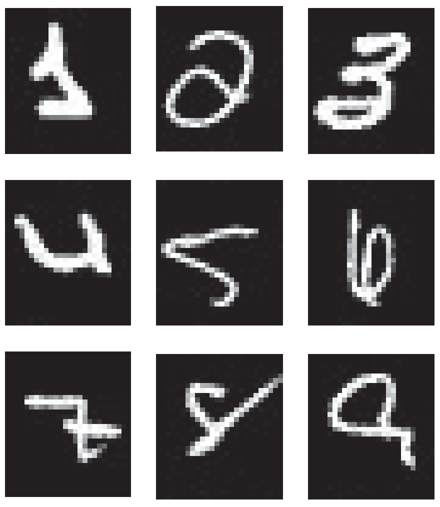

(b) Hard

Figure 7.1: Information Gain from Training Data

7.1 Introduction

We live in the era of ever-increasing size and complexity of "Big Data". To understand the data and decipher the information that truly counts, many advanced large-scale machine learning models have been devised, from million-dimension linear models (e.g. Logistic Regression [40], Support Vector Machine [96], feature selection [63, 119], Principal Component Analysis [42]) to complex models like Deep Neural Networks [22] or topic models [39]. While these models have demonstrated value for a wide spectrum of applications [67, 63, 54], their complexity causes the training cost to increase dramatically with the surging volume of data. This difficulty with scale severely affects the viability of many advanced models on industry-scale applications. Consequently, accelerating the training procedure of those "Big Models" on "Big Data" has attracted a great deal of interest.

Model training can usually be formulated as minimizing a specific objective function based on a set of data observations, i.e. the Empirical Risk Minimization [104] (ERM) problem. Even though batch gradient descent [113, 13, 11] has been widely used for decades, evaluating the full gradient over all the training samples is extremely expensive in the prevailing scale of millions of training samples. To reduce the computational cost at every iteration, Stochastic Gradient Descent (SGD) [83, 12] optimizes the objective function based on a single random sample at each iteration. Thereby, the computation cost per iteration is reduced greatly, but now many more iterations are required to reach a certain degree of accuracy or finally converge [75]. This is because the stochastic gradient used in each iteration is highly sensitive to the specific random sample chosen. Although the expectation of stochastic gradient is exactly the full gradient, the large variance causes the direction of stochastic gradient to deviate from that of the full gradient, which is the optimal direction to minimize the objective function. Some samples may even direct the model to the opposite of the correct direction. It is a trade-off between the quality of gradient and the computational cost per iteration. A commonly used scheme called Mini-batch SGD [61] is developed for this purpose: by averaging the gradient from a mini-batch of samples, the variance of gradient is significantly reduced, at the cost of some increased computation per iteration.

In this chapter, we seek to further optimize the stochastic gradient by not merely averaging gradients from more random samples but rather improving the quality of data samples. To this end, we propose a light-weight SGD accelerator called ActiveSampler inspired by active learning [92, 81]. In active learning, training data are selected to maximize the "learning value". For example, to train a classification model, training data points near class boundaries are more valuable than points in the interior of a class. We adapt the idea of active learning to the SGD optimization context by choosing samples from not a uniform distribution over the training data but rather a biased distribution from which we expect to learn more.

Figure 7.1 gives an example of how different samples can affect the training efficiency. The left side contains some images of written digits that are very easy to classify. For these "easy-to-classify" images, most models can classify them correctly even after a handful of steps. In subsequent thousands of iterations or even more, these easy-to-classify images will be sampled and correctly classified with high confidence, contributing almost zero gradient to the model. Consequently, the training time consumed by those easy-to-classify images is largely wasted. In contrast, the right side of Figure 7.1 contains some images that are hard to classify. By putting more effort on those "hard-to-classify" images, the accuracy of model may improve at a faster rate.

Based on the above intuition, we define the information gain from each sample for the model training, and aim to maximize it. We find that to maximize the information gain in each iteration, the sample frequency for each training sample should be proportional to the expectation of its gradient magnitude. Notwithstanding the sampler itself is biased, we show that the original objective function based on uniform weight can still be correctly minimized by re-weighting the gradient of each sample. Our algorithm shares lots of common features with importance sampling [121], which provides minimal variance of stochastic gradient, but the sampling frequency in ActiveSampler is independent of the label of each data instance. Therefore, ActiveSampler may be more robust to contaminated data instances. The net result is a system that requires far fewer iterations for model convergence.

Although optimization methods can reduce the number of iterations, it should also be noted that they may introduce additional computation cost in each iteration. There have been a great amount of research works [88, 47, 71] focusing on accelerating SGD. Theoretically, these methods have significantly faster convergence rate. While Momentum and AdaGrad [24] methods have shown their effectiveness and have been integrated into practical SGD solver, most methods are far from being used in practice due to their significant additional computation cost. For example, the cost of SVRG per iteration [47] is at least three times the cost of standard SGD per iteration. As noted in [71], mini-batch SGD still dominates in most cases, due to its light-weight computation and good vectorization.

To make ActiveSampler as efficient as possible in practice, in the actual implementation, we use existing knowledge to approximate the information that needs additional computation cost, and the sampling distribution is decided by the gradient magnitude in previous iterations and an entropy function. To evaluating the gradient magnitude for each sample, an effective scheme is applied to avoid the explicit calculation of the gradient of each sample. This scheme makes it possible that the computation for multiple samples can still be efficiently vectorized. Moreover, the computation cost is only $O(m + l)$ for a $m \times l$ parameter matrix. Therefore, for ActiveSampler, the computation overhead introduced in each iteration is light-weight, considering its significant contribution in reducing the number of total iterations.

To summarize, the contributions of this chapter are the following.

- We propose a general SGD accelerator, called ActiveSampler, where more informative training data is sampled more frequently for model training.

- We develop a light-weight and fully vectorized algorithm for ActiveSampler, making its computation cost in each iteration comparable to the naïve mini-batch SGD.

7.2 ActiveSampler

7.2.1 Overview

In this chapter, we revisit the SGD algorithm from a brand new angle – the information gain of the model at each iteration. Specifically, we regard the SGD algorithm as an active learning procedure which sequentially takes samples from the dataset and learn the model. Naturally, training model using samples with more information would facilitate faster improvement of the model. We term our weighted sampling strategy as ActiveSampler. The intuition here is that a large number of training samples are not helpful for refining the model (or at least not helpful at a certain training stage), including data that are too evident to predict, data that are noisy, and data that have just been visited. By simply skipping these samples, we can save a significant amount of training time. In contrast, the samples that are close to the border of class may be very helpful to refine the model (or even define the model in some cases such as SVM). This idea is very similar to active learning but with a major difference – the objective of active learning is to reduce the number of training samples, while the objective of ActiveSampler is to reduce the number of training iterations.

To exploit information gain as a means to speed up SGD training, three issues have to be addressed: (1) define what is the information gain for model training from each training sample; (2) adapt the ActiveSampler to the SGD framework and study how information gain can help speeding up SGD; (3) design a light-weight implementation of ActiveSampler that can be applied to real systems. We shall address these three issues in the following subsections.

7.2.2 Information Gain

We define the information gain following the basic intuition from the training of typical soft margin classifiers. In soft margin classifiers, instead of giving a single label as the prediction, the classifier outputs a probabilistic distribution over latent labels. Many recently proposed classification algorithms (e.g. Lasso, Neural Network and Soft SVM) are typically trained as soft margin classifiers, since the optimization over a continuous loss functional space is more efficient than the discrete optimization, which is a NP-Hard problem.

Definition 16 (Soft Margin Classifier)**.** *Given a predictor* $f_{\mathbf{w}}$*, the soft probabilistic classifier is defined by using log logistic function as the loss function, i.e.*

$$L(f_{\mathbf{w}}(\mathbf{x}), y) = \log(1 + \exp(-f_{\mathbf{w}}(\mathbf{x}) * y)) \tag{7.1}$$

Obviously, all algorithms in Table 2.1 using the logistic loss function are soft margin classifiers. The rationale is that the logistic function is used to transfer the prediction $f_{\mathbf{w}} \in R$ *to a classification probability, i.e.*

$$Pr(y|f_{\mathbf{w}}(\mathbf{x})) = 1/(1 + \exp(-f_{\mathbf{w}}(\mathbf{x}) * y)) \tag{7.2}$$

Then the log-likelihood $\log Pr(y|f_{\mathbf{w}}(\mathbf{x}))$ *is maximized, i.e. the loss is* $\log(1 + \exp(-f_{\mathbf{w}}(\mathbf{x}) * y))$.

Now, let us analyze the possible factors that may affect the information gain of a model from each training sample. First, from the view of active learning, the information gain by revealing a label which can be easily predicted is very limited. This is also true for the SGD algorithm – visiting a sample that the model can always classify correctly is not helpful, as the loss cannot be further reduced if the loss is already close to 0. A model can only learn from samples which are uncertain in prediction. Figure 7.2 shows an example to illustrate why uncertainty helps the model to improve: the images on the left are very easy to predict, and therefore $Pr(y|f_{\mathbf{w}}(\mathbf{x}))$ is close to 1. As a result, the loss $L(f_{\mathbf{w}}(\mathbf{x}), y)$ for each of those images is nearly zero and has little room for further optimization. In contrast, the images on the right is much harder to predict. By selecting them as samples for optimization purpose, the loss $L(f_{\mathbf{w}}(\mathbf{x}), y)$ on those samples could be significantly reduced and the average performance of

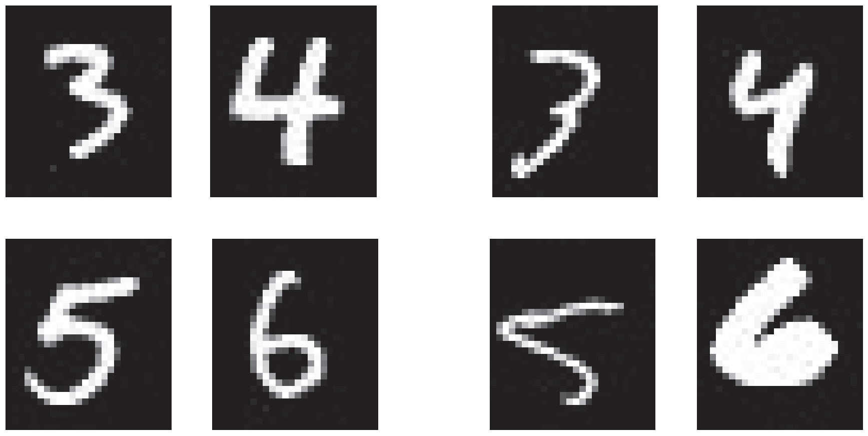


Figure 7.2: Uncertainty

model is hence improved. In information theory [104], the information gain by revealing a random variable is usually defined as the entropy of that random variable:

Definition 17 (Uncertainty)**.** *The uncertainty of a training instance* $\mathbf{x_i}$ *for a model* $f_{\mathbf{w}}$ *is defined as the entropy of the* $f_{\mathbf{w}}(\mathbf{x_i})$*, i.e.*

$$U(\mathbf{w}, \mathbf{x_i}) = -\sum_{y} Pr(y|f_{\mathbf{w}}(\mathbf{x_i})) \log Pr(y|f_{\mathbf{w}}(\mathbf{x_i})) \tag{7.3}$$

Second, not all the uncertain training instances contribute equally to the model performance. For example, though the model may always be uncertain about the label for noisy data, this does not mean that noisy data are more helpful to improve the model performance. This is because the uncertainty measure only evaluates the information inside the label, but not its contribution to the model. Therefore, we introduce another measure called *significance* to evaluate the effectiveness of information transfer from the data to the model (i.e. conversion rate). Intuitively, the output of a noisy instance is not sensitive to the change of the parameter. When the output of a data instance is sensitive to the change of parameter, its loss will be significantly reduced even with tiny changes of the parameter, which provides a clear instruction on how to reduce the loss by tuning the parameter. The right hand side of Figure 7.3 shows the images that are noisy and with little significance.

Definition 18 (Significance)**.** *The significance of a training instance* $\mathbf{x_i}$ *for a*

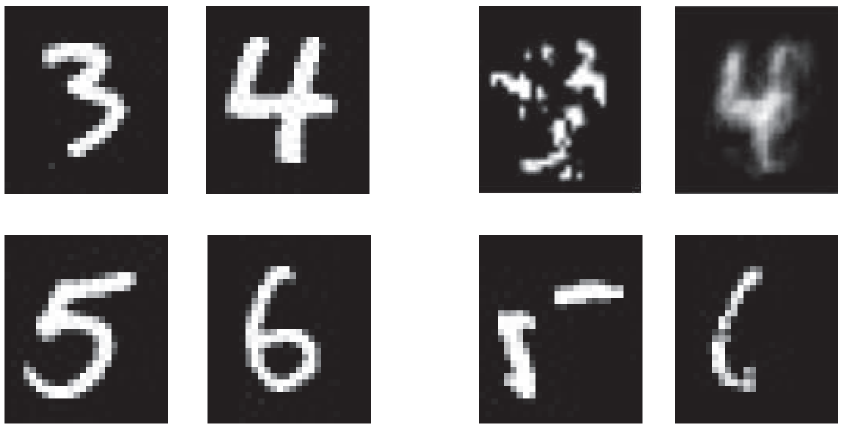


Figure 7.3: Significance

model $f_{\mathbf{w}}$ is defined as its sensitivity to parameter change:

$$S(\mathbf{w}, \mathbf{x_i}) = \|\nabla_{\mathbf{w}} f_{\mathbf{w}}(\mathbf{x_i})\|_2 \tag{7.4}$$

$\|\nabla_{\mathbf{w}} f_{\mathbf{w}}(\mathbf{x_i})\|_2$ is the maximal change of $f_{\mathbf{w}}(\mathbf{x_i})$ when the parameter $\mathbf{w}$ changes an unit distance.

Here, we give a comparison between uncertainty and significance: uncertainty measures the expectation of accuracy on the current model, while significance measures the potential improvement of accuracy by tuning the model. Therefore, instances that are easy to classify usually have low uncertainty but high significance; noisy instances usually have high uncertainty but low significance, while valuable border instances that have not been well learned usually have both high uncertainty and high significance.

Third, the information gain in one iteration may overlap with the information obtained in earlier training steps. For example, visiting the training instance that has just been trained usually does not provide extra information than what has been derived in the previous visit. Moreover, for a completely new instance that has not been trained, there is no overlap between the information gain and information in previous steps. Therefore, we use the visiting interval to measure the effect of information overlap:

Definition 19 (Interval)**.** *The visiting interval $I(\mathbf{w}, \mathbf{x_i})$ of a training instance $\mathbf{x_i}$ for a model $f_{\mathbf{w}}$ is defined as the number of iterations since the last time $\mathbf{x_i}$*

was used in training. A larger interval provides less information overlap and more pure information gain.

Combining all the three factors together, we define the information gain of model $f_{\mathbf{w}}$ from training instance $\mathbf{x_i}$ as $IG(\mathbf{w}, \mathbf{x_i})$:

$$IG(\mathbf{w}, \mathbf{x_i}) = U(\mathbf{w}, \mathbf{x_i}) * S(\mathbf{w}, \mathbf{x_i}) * I(\mathbf{w}, \mathbf{x_i}) \tag{7.5}$$

The objective of our ActiveSampler is to choose the training instance $\mathbf{x_i}$ with the largest $IG(\mathbf{w}, \mathbf{x_i})$.

7.2.3 Weighted SGD and Analysis

We now present how to adapt ActiveSampler to the SGD framework in detail.

Theorem 7.1 (Information Gain Maximization). *By choosing the largest $IG(\mathbf{w}, \mathbf{x_i})$ in each iteration, the sampling probability p_i of each training instance $\mathbf{x_i}$ should be proportional to its expectation of the gradient magnitude, i.e.*

$$p_i = \frac{E_y[\|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y)\|_2]}{\sum_i E_y[\|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y)\|_2]} \tag{7.6}$$

Proof Sketch: At each iteration, $IG(\mathbf{w}, \mathbf{x_i})$ for each instance will increase $U(\mathbf{w}, \mathbf{x_i})$ * $S(\mathbf{w}, \mathbf{x_i})$, and the instance with the largest $IG(\mathbf{w}, \mathbf{x_i})$ will be selected as sample. After an instance is sampled, its $IG(\mathbf{w}, \mathbf{x_i})$ will be set to zero as its $I(\mathbf{w}, \mathbf{x_i})$ becomes zero. Therefore, as the number of iterations grows, the sampling frequency for each instance should be proportional to its $U(\mathbf{w}, \mathbf{x_i})$ * $S(\mathbf{w}, \mathbf{x_i})$, considering that the largest $IG(\mathbf{w}, \mathbf{x_i})$ selected in each iteration should has a similar value. Meanwhile,

$$\begin{aligned}
& E_y[\|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y)\|_2] \\
= \quad & \textstyle\sum_y Pr(y|f_{\mathbf{w}}(\mathbf{x_i}))\|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y)\|_2 \\
= \quad & \textstyle\sum_y Pr(y|f_{\mathbf{w}}(\mathbf{x_i}))\frac{\partial}{\partial f_{\mathbf{w}}(\mathbf{x_i})} L(f_{\mathbf{w}}(\mathbf{x_i}), y)\|\nabla_{\mathbf{w}} f_{\mathbf{w}}(\mathbf{x_i})\|_2 \\
= \quad & \textstyle\sum_y (Pr(y|f_{\mathbf{w}}(\mathbf{x_i})) * \log Pr(y|f_{\mathbf{w}}(\mathbf{x_i})))\|\nabla_{\mathbf{w}} f_{\mathbf{w}}(\mathbf{x_i})\|_2 \\
= \quad & U(\mathbf{w}, \mathbf{x_i}) * S(\mathbf{w}, \mathbf{x_i})
\end{aligned} \tag{7.7}$$

□

While the result in Theorem 7.1 is similar to the result in importance sampling SGD [121], but there is one major difference between them. In important sampling, the sampling frequency is proportional to the exact gradient magnitude where the label is known. However, in ACTIVESAMPLER, the sampling frequency is proportional to the expectation of gradient magnitude where the label is unknown. This is because from the prospective of active learning, the information from one instance is independent of the actual value of label. Therefore, a wrongly classified instance with high confidence will have higher frequency in importance sampling than that in ACTIVESAMPLER. In such cases, ACTIVESAMPLER may be more robust to contaminated data or outliers. However, there is no direct way to evaluate which method is better from theoretical angle. We compare importance sampling with our basic approach experimentally, showing that we have comparable results in general and sometimes better results; we also show that the practical model we present in the next section which focuses more on the accuracy in practical approximation obtains considerably better results.

While the above intuition suggests that different training instances should be sampled at different frequencies, directly changing the sampling frequency will result in a bias in the target of an optimization.

$$\begin{aligned} E_i[g_i(\mathbf{w})] &= \sum_i p_i g_i(\mathbf{w}) \\ &= \nabla_{\mathbf{w}} \sum_i p_i L(f_{\mathbf{w}}(\mathbf{x_i}), y_i) \end{aligned} \tag{7.8}$$

The loss function to be minimized is $\sum_i p_i L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)$ instead of $\sum_i \frac{1}{n} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)$. The weight for each training instance is unequal, which may affect the accuracy of the model. For this reason, using $\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)$ directly as $g_i(\mathbf{w})$ is inappropriate. Instead, we should guarantee that $E_i[g_i(\mathbf{w})]$ is $\nabla_{\mathbf{w}} L(\mathbf{w})$ by doing the following adjustment.

Theorem 7.2 (Weighted SGD). *Given any sampling distribution* $\{p_1, ..., p_i, ..., p_n\}$*, to get a SGD algorithm that optimizes* $L(\mathbf{w})$*,* $g_i(\mathbf{w})$ *should be reweighted to* $\frac{\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{n * p_i}$.

Proof. To get a SGD algorithm that optimizes $L(\mathbf{w})$, we need $E_i[g_i(\mathbf{w})]$ to be

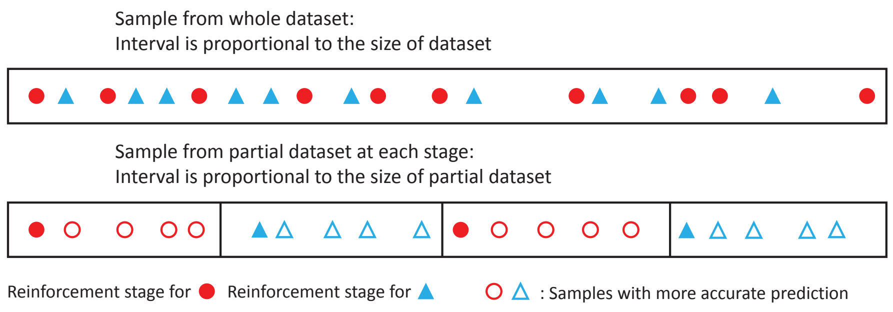

Sample from whole dataset:
Interval is proportional to the size of dataset
Sample from partial dataset at each stage:
Interval is proportional to the size of partial dataset
Reinforcement stage for ●
Reinforcement stage for ▲
○ △ : Samples with more accurate prediction

Figure 7.4: History Reinforcement

$\nabla_{\mathbf{w}} L(\mathbf{w})$. By scaling $g_i(\mathbf{w})$ w_i times and solve $E_i[g_i(\mathbf{w})] = \nabla_{\mathbf{w}} L(\mathbf{w})$, we have:

$$\begin{aligned} E_i[g_i(\mathbf{w})] &= \nabla_{\mathbf{w}} L(\mathbf{w}) \\ \Rightarrow \quad \sum_i p_i w_i \nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i) &= \sum_i \frac{1}{n} \nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i) \\ \Rightarrow \quad p_i w_i &= 1/n \end{aligned} \tag{7.9}$$

Therefore, $w_i = 1/(n * p_i)$. □

7.2.4 Practical Implementation Issues

As discussed in Chapter 2, the main goal of optimizing the SGD algorithm is to reduce the variance of $g_i(\mathbf{w})$, while keeping the computation cost per iteration light-weight. We have already shown how to minimize the variance of $g_i(\mathbf{w})$ by using ActiveSampler. In this subsection, we will discuss some practical issues in implementing ActiveSampler onto real systems, which may significantly affect the computation time in each iteration.

Sampling based on History

According to Theorem 7.1, the ideal goal of ActiveSampler is to sample each instance $\langle \mathbf{x_i}, y_i \rangle$ with $Pr(\mathbf{x_i})$ which is proportional to $E_y[\|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y)\|_2]$. For each instance $\langle \mathbf{x_i}, y_i \rangle$, $E_y[\|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y)\|_2]$ can be represented as:

$$\frac{\sum_y Pr(y|f_{\mathbf{w}}(\mathbf{x_i})) \log Pr(y|f_{\mathbf{w}}(\mathbf{x_i}))}{\log Pr(y_i|f_{\mathbf{w}}(\mathbf{x_i}))} \|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)\|_2 \tag{7.10}$$

However, use of the exact distribution, which requires all n gradients to be evaluated at each step, is obviously not practical. Instead, we can predict

Algorithm 11: ASSGD (ActiveSampler SGD)

```
Input: Initial $\mathbf{w_0}$, $T$, $\beta$, $EGrad[]$, $SumEGrad = \sum_i EGrad[i]$
Output: Final $\mathbf{w_T}$
1 for $t = 1, ..., T$ do
2     sample $i$ from $\{1, ..., n\}$ based on distribution $\{p_1, ..., p_n\}$ where
      $p_i = \beta/n + (1-\beta)EGrad[i]/SumEGrad$;
3     $g_i(\mathbf{w}) = \nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)/np_i$;
4     $\mathbf{w_t} = \mathbf{w_{t-1}} - \eta \nabla_{\mathbf{w}} \rho_f(\mathbf{w}) - \eta g_i(\mathbf{w})$;
5     $SumEGrad = SumEGrad - EGrad[i]$;
6     $EGrad[i] = \frac{\sum_y Pr(y|f_{\mathbf{w}}(\mathbf{x_i})) \log Pr(y|f_{\mathbf{w}}(\mathbf{x_i}))}{\log Pr(y_i|f_{\mathbf{w}}(\mathbf{x_i}))} \|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)\|_2$;
7     $SumEGrad = SumEGrad + EGrad[i]$;
```

the gradient magnitude for each training instance using historical data. A straightforward approach to the problem is to remember the latest expectation $E_y[\|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y)\|_2]$ of each instance and use it as an approximation. Considering that the actual expectation may change and the historical data is only an approximation, a smoothing term is required. For example, if one instance contributes a zero expectation of gradient at any iteration of the model training when it is sampled, that sample will never be visited afterward if there is no smoothing, notwithstanding this instance may become valuable for refinement of the model at later stages.

Definition 20 (History Approximation)**.** *Let t_i be the latest step where training instance $\langle \mathbf{x_i}, y_i \rangle$ is visited and let $\mathbf{w_{t_i}}$ denote the parameter value at step t_i. The sampling probability for each sample $\langle \mathbf{x_i}, y_i \rangle$ in a practical* ActiveSampler *with smoothing is defined as:*

$$p_i = (1-\beta)\frac{E_y[\|\nabla_{\mathbf{w_{t_i}}} L(f_{\mathbf{w_{t_i}}}(\mathbf{x_i}), y_i)\|_2]}{\sum_i E_y[\|\nabla_{\mathbf{w_{t_i}}} L(f_{\mathbf{w_{t_i}}}(\mathbf{x_i}), y_i)\|_2]} + \frac{\beta}{n} \tag{7.11}$$

The scheme ensures that every training instance has at least β times the average sampling probability (i.e. $1/n$) being sampled. Algorithm 11 describes the ActiveSampler using the history approximation. In each iteration, only its gradient magnitude needs to be updated.

We observe that by using the history length to denote the number of iterations since the last step an instance is sampled, its history approximation becomes less accurate with the increase of the history length. Meanwhile, the expectation of history length for one instance is $1/p_i$, and the average of p_i is $1/n$. Consequently, the history approximation will become less accurate when data size becomes larger. To address this issue, we propose **History Reinforcement**, whose key idea is illustrated in Figure 7.4. History Reinforcement algorithm trains the model using a set of stages, each of which contains a large amount of SGD iterations. Within a stage, it first samples a subset of training data which consists of m instances, and then uses them as the training set in its SGD iterations. During the training of each stage, the sampling probability for the instances is n/m times larger than that for training all the instances together. Therefore, the approximation will be much more accurate except the first time in a stage when one instance is sampled. The only drawback of History Reinforcement is that it may lead to a bias in the training of a stage, as only partial data are sampled for training. However, [61] presents an effective scheme in an analogous context to reduce this bias by adding a regularizer to limit the change of parameter in every stage. Below, we formally define the concept of History Reinforcement.

Definition 21 (History Reinforcement)**.** *History Reinforcement algorithm consists of multiple stages. In each stage s, it draws a subset I_s of training instances, which contains m random instances from the whole dataset, and trains the parameter $\mathbf{w_s}$ using g SGD iterations. The loss function used in each stage s is:*

$$L(\mathbf{w_t}) = \sum_{\langle \mathbf{x_i}, y_i \rangle \in I_s} \frac{L(f_{\mathbf{w_{t_i}}}(\mathbf{x_i}), y_i)}{m} + \frac{\gamma_s}{2} \|\mathbf{w_{s-1}} - \mathbf{w_s}\|_2^2 \tag{7.12}$$

where γ_s is a parameter in [61] calculated based on m, s and $Var(g_i(\mathbf{w}))$.

The correctness and effectiveness of this batch training is given in Theorem 1 of [61] (by considering each stage as a batch step). The expected number of visits for one instance in a stage is g/m. Therefore, $1 - m/g$ proportion of the iterations in a training stage will benefit from a more accurate approximation. In essence, there is a trade-off between the bias involved by using partial data and the accuracy gain in gradient approximation. For larger datasets, the bias becomes less significant and the accuracy gain by using History Reinforcement becomes more valuable. On the contrary, the approximation of gradient in small datasets is fairly accurate, and therefore, directly sampling from the whole

Algorithm 12: ASSGD with History Reinforcement

```
   Input: Initial $\mathbf{w_0}$, $S$, $m$, $g$
   Output: Final $\mathbf{w_S}$
 1 for $s = 1, ..., S$ do
 2     $I_s = \emptyset$;
 3     for $i = 1, ..., m$ do
 4         sample $s_i$ uniformly from $\{1, ..., n\} - I_s$;
 5         $I_s = I_s \cup \{s_i\}$;
 6     Compute $\gamma_s$ based on [61];
 7     Train $\mathbf{w_s}$ using Algorithm 11 for $g$ iterations, using $\mathbf{w_{s-1}}$ as initial
       $\mathbf{w_0}$, using $I_s$ as the training set and using $\rho_f(\mathbf{w}) + \frac{\gamma_s}{2}\|\mathbf{w_{s-1}} - \mathbf{w}\|_2^2$ as
       the regularization term;
```

dataset is advantageous.

Efficient Vectorized Computation

To reduce the variance of the stochastic gradient, a widely adopted solution for most large-scale optimization problems is to employ mini-batch training, which averages the stochastic gradients of multiple training samples. Thanks to the effect of vectorized computation and the constant communication cost when the computations are parallelized, the training time per iteration for mini-batch SGD is much smaller than b times the training time per iteration for the standard SGD. ActiveSampler is orthogonal to mini-batch SGD, so we could use both techniques simultaneously. However, to integrate them together, we need to compute the average of $g_i(\mathbf{w})$ for b training samples in an efficient vectorized way, as well as to obtain the gradient magnitude for each training instance.

Definition 22 (Mini-batch SGD / ActiveSampler).
At each iteration t, mini-batch SGD uniformly draws b samples $I_t = \{t_1, ..., t_b\}$ from n training instances, and uses the averaged gradient as the stochastic gradient.

$$g_t(\mathbf{w}) \quad = \quad \sum_{i \in I_t} \frac{\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{b} \tag{7.13}$$

At each iteration t, mini-batch ActiveSampler *repeats the sample selection in Theorem 7.1 for b times and get b samples $I_t = \{t_1, ..., t_b\}$, and uses the* *averaged gradient as the stochastic gradient.*

$$g_t(\mathbf{w}) \;=\; \sum_{i \in I_t} \frac{\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{bnp_i} \tag{7.14}$$

Similar to mini-batch SGD, the variance of $g_t(\mathbf{w})$ in mini-batch ActiveSampler *is reduced by b times.*

In mini-batch SGD, the main advantage stemmed from vectorized computation is that the actual gradients from all samples do not need to be calculated individually and then aggregated. Here we use a multi-layer perceptron [22] (MLP) model to illustrate how the stochastic gradient of mini-batch SGD is computed, and how mini-batch ActiveSampler can be computed in a similar light-weight manner. Note that general linear models ($f_{\mathbf{w}}(\mathbf{x}) = \mathbf{w^T x}$) are usually generalized as a multi-class classification problem, and their parameter $\mathbf{w}$ is also a matrix, which is similar to the hidden layer in MLP. Therefore, general linear models can be viewed as a single layer perceptron with a small difference in the loss function and hence all the optimization techniques discussed below can be applied to these models as well.

Definition 23 (Multi-Layer Perceptron (MLP))**.** *Multi-Layer Perceptron [22] is a feed forward neural network. It consists of one input layer $H^{(0)}$, h hidden layers ($H^{(k)}$, $k = 1..., h$) and a loss layer to compute the loss $L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)$ based on the prediction $H^{(h)}$ for $\mathbf{x_i}$. Each hidden layer k is a vector of units, and the calculation is formalized as follows:*

$$Z^{(k+1)} = W^{(k)} H^{(k)} + B^{(k)} \tag{7.15}$$

$$H^{(k+1)} = \sigma(Z^{(k+1)}) \tag{7.16}$$

where $\sigma(\cdot)$ is the activation function. The gradient is computed via back-propagation:

$$\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{\partial W^{(k)}} = \frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{\partial Z_i^{(k+1)}} \times H_i^{(k)T} \tag{7.17}$$

$$\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{\partial H^{(k)}} = \frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{\partial Z_i^{(k+1)}} \times W^{(k)} \tag{7.18}$$

$$\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{\partial Z_p^{(k)}} = \sigma'(Z_p^{(k)}) \frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{\partial H_p^{(k)}} \tag{7.19}$$

Algorithm 13: Batch Computation for ActiveSampler

```
    Input: $H^{(k)}_{b\times l}$, $Z^{(k+1)}_{b\times m}$, $W^{(k)}_{m\times l}$, $\nabla H^{(k+1)}_{b\times m}$
    Output: $\nabla H^{(k)}_{b\times l}$, $\nabla W^{(k)}_{m\times l}$, $\|\nabla_{W^{(k)}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)\|_2$
 1  foreach $i \in \{0, ..., b-1\}$ do
 2      foreach $p \in \{0, ..., m-1\}$ do
 3          $\nabla Z^{(k+1)}[i][p] = \sigma'(Z^{(k+1)}[i][p]) \nabla H^{(k+1)}[i][p]$;
 4  $\nabla H^{(k)}_{b\times l} = \nabla Z^{(k+1)}_{b\times m} \times W^{(k)}_{m\times l}$;
 5  $\nabla W^{(k)}_{m\times l} = \frac{1}{b} (\nabla Z^{(k+1)})^{T}_{m\times b} \times H^{(k)}_{b\times l}$;
 6  // line 1-5 : compute stochastic gradient $O(bml)$
 7  foreach $i \in \{0, ..., b-1\}$ do
 8      $SumGZ = 0$, $SumH = 0$;
 9      foreach $p \in \{0, ..., m-1\}$ do
10          $SumGZ = SumGZ + (\nabla Z^{(k+1)}[i][p])^2$;
11      foreach $q \in \{0, ..., l-1\}$ do
12          $SumH = SumH + (H^{(k)}[i][q])^2$;
13      $\|\nabla_{W^{(k)}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)\|_2 = \sqrt{SumGZ * SumH}$
14  // line 7-13 : compute gradient magnitude $O(b(m+l))$
```

We now analyze the computation of gradient for one layer k in mini-batch SGD. Using m to denote the number of units in $H^{(k+1)}$, and l to denote the number of units in $H^{(k)}$, the parameter $W^{(k)}$ is an $m \times l$ matrix, and $g_t(W^{(k)})$ is also an $m \times l$ matrix.

$$\begin{aligned} g_t(W^{(k)}) &= \sum_{i\in I_t} \frac{\nabla_{W^{(k)}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{b} \\ &= \frac{1}{b} \sum_{i\in I_t} \frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{\partial Z_i^{(k+1)}} \times {H_i^{(k)}}^T \end{aligned} \tag{7.20}$$

However, directly computing the b gradients $\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}),y_i)}{\partial Z_i^{(k+1)}} \times {H_i^{(k)}}^T$ one by one is not efficient, as each gradient is an $m \times l$ matrix. Instead, using $H^{(k)}$ to denote the $b \times l$ lower layer feature matrix $[H_1^{(k)}, ..., H_b^{(k)}]^T$, and using $\frac{\partial L(f_{\mathbf{w}}(\mathbf{x}),y)}{\partial Z^{(k+1)}}$ to denote the $b \times m$ higher layer gradient matrix $[\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}),y_i)}{\partial Z_1^{(k+1)}}, ..., \frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}),y_i)}{\partial Z_b^{(k+1)}}]^T$, we have:

$$\begin{aligned} g_t(W^{(k)}_{pq}) &= \frac{1}{b}\sum_{i\in I_t}\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)}{\partial Z^{(k+1)}_{ip}} \times H^{(k)}_{iq} \\ \Rightarrow \qquad g_t(W^{(k)}) &= \frac{1}{b}(\frac{\partial L(f_{\mathbf{w}}(\mathbf{x}), y)}{\partial Z^{(k+1)}})^T \times H^{(k)} \end{aligned} \tag{7.21}$$

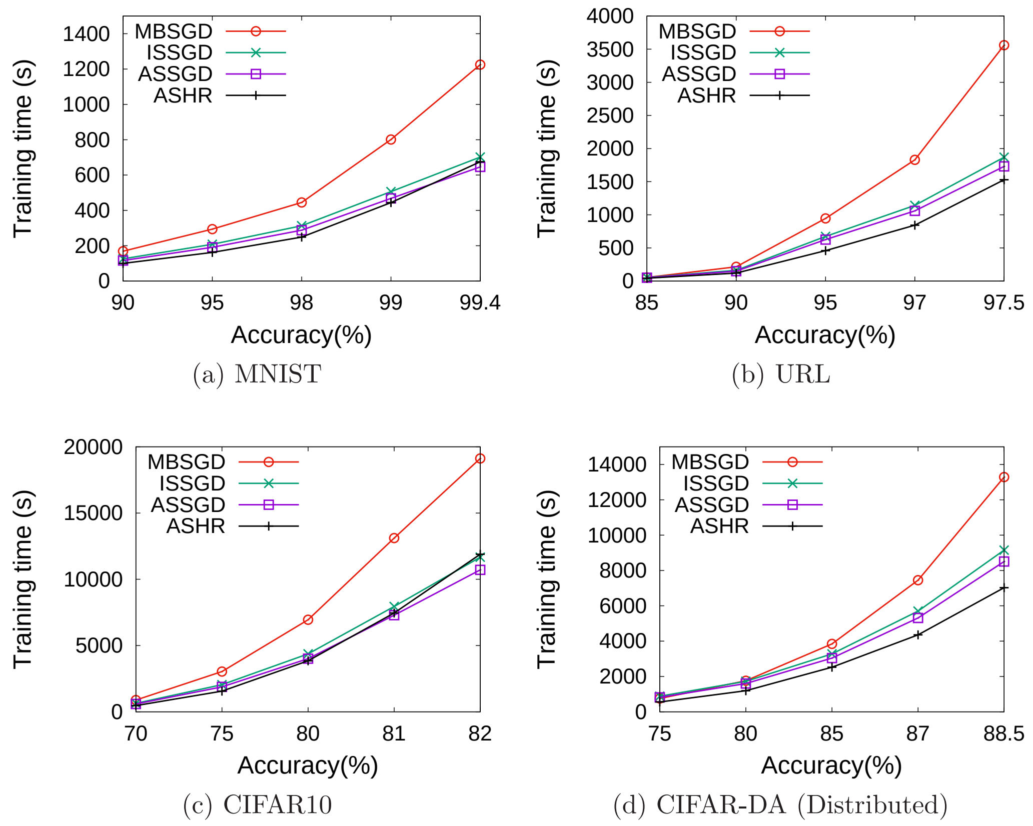
Therefore, it is computed by performing matrix multiplication for an $m \times b$ matrix and a $b \times l$ matrix. Obviously, this is more efficient than the previous method, which computes multiple vector-vector multiplications. It also reduces the memory cost from $b \times m \times l$ to $m \times l$. The computation for $\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}),y_i)}{\partial H^{(k)}}$ is analogous.

In mini-batch ActiveSampler, there are two differences compared to mini-batch SGD. First, ActiveSampler needs to provide each instance a weight based on $1/np_i$. Second, ActiveSampler needs to compute the gradient magnitude for every training instance. For the first problem, the solution is quite straightforward – putting the weight in the loss function by scaling the value of loss by $1/np_i$ times before calculating its gradient for the parameters.

For the calculation of the gradient magnitude, we exploit the following equation to avoid explicitly calculating the gradient of each training instance:

$$\begin{aligned} & \|\nabla_{W^{(k)}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)\|_2^2 \\ = \quad & \textstyle\sum_{p\in m}\sum_{q\in l}(\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}),y_i)}{\partial W^{(k)}_{pq}})^2 \\ = \quad & \textstyle\sum_{p\in m}\sum_{q\in l}(\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}),y_i)}{\partial Z^{(k+1)}_{ip}} H^{(k)}_{iq})^2 \\ = \quad & (\textstyle\sum_{p\in m}\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}),y_i)}{\partial Z^{(k+1)}_{ip}}^2)(\sum_{q\in l} H^{(k)}_{iq}{}^2) \end{aligned} \tag{7.22}$$

Therefore, we just need to compute the product of the square sum of $\frac{\partial L(f_{\mathbf{w}}(\mathbf{x_i}),y_i)}{\partial Z^{(k+1)}_{i}}$ and $H^{(k)}_i$, which are all from intermediate results during the computation of mini-batch SGD. Its computation complexity is just $O(b(m+l))$, which is extremely light-weight considering that the cost for calculating the gradient is $O(bml)$. Algorithm 13 shows the vectorized computation of ActiveSampler in each layer of MLP. For deep models which contain multiple layers, the square of the gradient magnitude with respect to parameters from whole layers can be computed by summing the square of gradient magnitude with respect to pa-



(a) MNIST (b) URL (c) CIFAR10 (d) CIFAR-DA (Distributed)

Figure 7.5: Overall Training Time

rameters from each layer, i.e.

$$\|\nabla_{\mathbf{w}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)\|_2^2 \;=\; \sum_k \|\nabla_{W^{(k)}} L(f_{\mathbf{w}}(\mathbf{x_i}), y_i)\|_2^2 \tag{7.23}$$

7.3 Experimental Study

7.3.1 Experiment Setup

We evaluate the speedup of ActiveSampler using a set of popular benchmark tasks, namely the hand-written digit classification on MNIST dataset using SVM [96], malicious URL detection on URL dataset using feature selection [63], and image classification on CIFAR10 dataset using CNN [54]. Furthermore, we also test the scalability of ActiveSampler using the CIFAR10 dataset with data augmentation, where the size of the training data is increased by 128x. Table 7.1 summarizes the datasets and models used in our experimental study.

Dataset	# Examples	Size	Model	Test Error
MNIST	60K	57MB	kernel SVM	0.6%
URL	2.4M	950MB	Lasso	2.5%
CIFAR10	60K	161MB	DCNN	18%
CIFAR-DA	7.6M	14.8GB	DCNN	11.5%

Table 7.1: Datasets and Models

- **MNIST:** MNIST is a benchmark dataset of handwritten digits classification, consisting of a training set of 60000 images and a test set of 10000 images. Each image contains 28*28 gray pixels. Pegasos [96] is a mini-batch SGD solver for kernel SVM model. The test error of kernel SVM in MNIST dataset is 0.60% [96].
- **URL:** URL [67] is a dataset for malicious URL detection. It consists of 2.4 million URLs and 3.2 million features. Each URL contains around 100 non-zero features and hence its features are quite sparse. Lasso regression [63] is a popular feature selection model as described in Table 2.1. Its test error in the URL dataset is around 2.5%.
- **CIFAR10:** CIFAR10 is a dataset for image classification, consisting of a training set of 60000 images and a test set of 10000 images. It is the benchmark dataset commonly used for the evaluation of deep convolutional neural network (DCNN) [54] models. Each image contains 32*32 colored pixels. Its test error without data augmentation is 18%.
- **CIFAR-DA:** Data augmentation is a standard technique to increase the size of training data. It generates additional images by slightly translating the original images. We use the data augmentation version of the CIFAR10 dataset (CIFAR-DA) to study the scalability of ACTIVESAMPLER. It contains 128x images compared with CIFAR10. Its test error for DCNN model is 11.5%. However, as the number of training images increases, DCNN model takes significantly longer time to achieve its best performance.

All the models are trained under the SGD framework. The standard mini-batch SGD (**MBSGD**) algorithm is used as the baseline. We also implement the mini-batch ACTIVESAMPLER for comparison, with the same size of mini-batch. Unless otherwise specified, the size of mini-batch is set to 128. The validation accuracy is tested per 100 mini-batch iterations. To study the effect of History Reinforcement strategy described in Section 7.2.4, we have implemented two versions of ActiveSampler, with and without History Reinforcement. **ASSGD**, the version without History Reinforcement, is expected to to perform well for moderate size of training examples, while **ASHR**, the version with History Reinforcement, is expected to yield better performance for large-scale training sets. In ASHR, the whole dataset is randomly split into 16 large batches, and examples are trained 16 times on average at each stage. We also compare our algorithm with a recently proposed state-of-the-art weighted SGD algorithm called SGD with importance sampling (**ISSGD**) [121]. For the fair of comparison, we implement ISSGD as a mini-batch version using the technique proposed in Section 7.

All of the algorithms are implemented in C++, compiled using GCC O2, and OpenBlas is adopted to accelerate linear algebra operations. Experiments for MNIST, URL and CIFAR10 datasets are carried on an Intel Xeon 24-core server with 500GB memory.

Distributed Environment and Scalability Test:
SGD training algorithms can be easily distributed to clusters via the parameter server [60] architecture. We conduct our scalability study using CIFAR-DA dataset on the Apache SINGA system [109], which is a general distributed deep learning platform. We follow all the default distributed settings of CIFAR on SINGA, where the mini-batch size is set to be 512. The distributed environment is a 32-node cluster, where each machine is equipped with an Intel Xeon 4-core CPU and 8GB memory.

7.3.2 Overall Performance

Figure 7.5 shows the training time to reach a certain accuracy for MBSGD, ISSGD, ASSGD and ASHR. 99.4%, 82%, 97.5% and 88.5% are the best accuracies achieved in these four tasks respectively. Generally, the convergence rates of ASSGD and ASHR are significantly faster than MBSGD, and are 15%-25% faster than ISSGD. To reach the optimal test error, ASSGD and ASHR save about 40% to 60% of the training time. The speedup is especially great in the latter stages of training, as evidenced by the bigger difference in the slope between the algorithms in the right hand portion of each graph. A possible explanation to this phenomenon is that the models typically have smaller changes in the end stage of training. As a result, the larger variance in MBSGD would have more serious negative effect, while ASSGD and ASHR algorithm would get even better approximation of the scale of gradient. There are also some notable differences between the performance of ASSGD and ASHR. First, ASHR converges much faster than ASSGD in the two large datasets (URL and CIFAR-DA), demonstrating that ASHR provides more accurate approximation of the scale of gradient in large-scale datasets. Meanwhile, the speedup of ASHR is smaller than that of ASSGD in the two small datasets (MNIST and CIFAR10), probably due to the small training dataset used by ASHR which only contains about 3000 training examples. Second, ASHR converges slightly faster at the beginning, and a bit slower towards the end. This is because ASSGD needs to visit the whole dataset at least once before enjoying the benefit of smaller variance, while ASHR only needs to visit a subset of the dataset.

For the scalability test on CIFAR-DA, the training time of all the approaches are significantly smaller than that on CIFAR10 due to distributed training. ActiveSampler still works as expected: ASHR speeds up the training process by 1.9x, and ASSGD speeds up the process by 1.6x. This is because the benefit of ActiveSampler is derived from the total number of iterations used to achieve a certain accuracy, instead of the improvement of training time per iteration. Since the total number of iterations is not affected by the training architecture, the speedup of ActiveSampler is applicable to all kinds of SGD frameworks as long as its overhead in the training time per iteration is small. Conversely, the number of training examples does affect the performance of ASSGD, since its approximation becomes less accurate when the number of training examples increases. However, ASHR scales well in all cases.

7.3.3 Variance of Stochastic Gradients

From the stochastic optimization view point, the benefit of using ActiveSampler is derived mainly from the reduction of the variance of stochastic gradient. We therefore evaluate the average variance of MBSGD, ISSGD, ASSGD and ASHR and summarize the results in Figure 7.6. Since the absolute value of variance may change dramatically during the training process, we use the variance of MBSGD as the baseline and report the relative ratio of the variance of ISSGD, ASSGD and ASHR compared to the baseline. The results show that ASHR has the smallest variance, less than 40% of the variance of MBSGD on

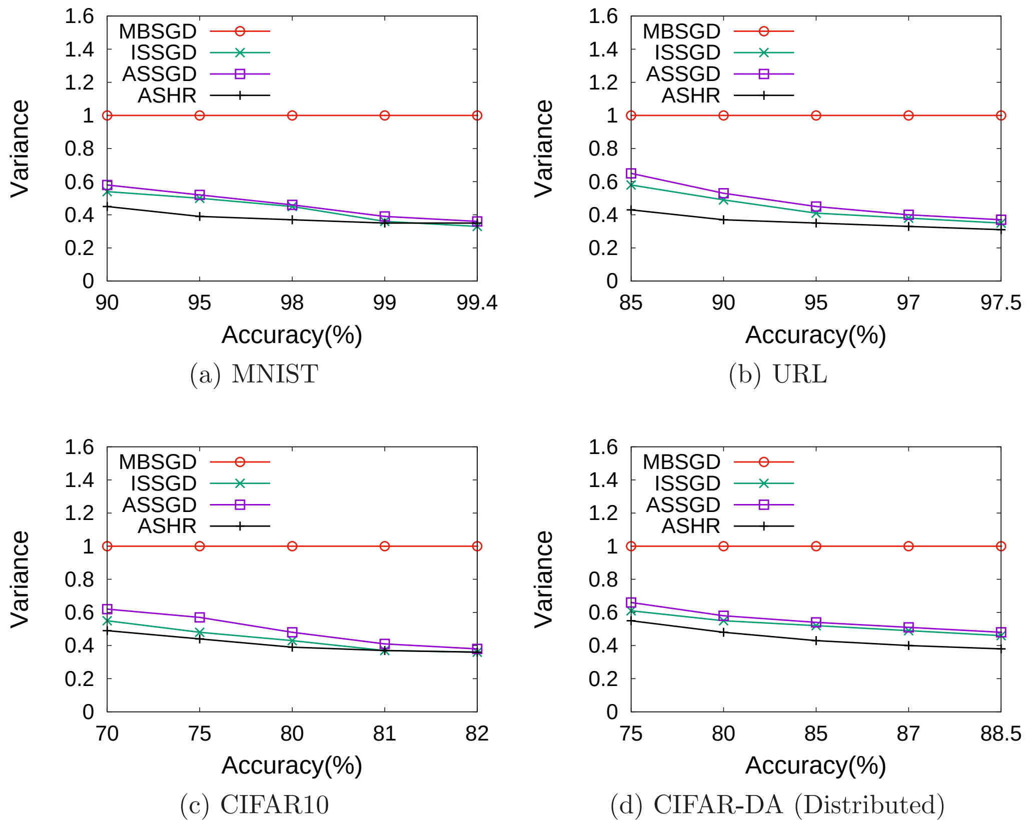


Figure 7.6: Variance at Different Training Stages

average. ISSGD, which supposes to have the smallest variance, has no significant advantage, probably because its approximation on gradient magnitude is less accurate than ASHR. The variance of ASSGD is slightly larger than that of ASHR, especially in the two large datasets, URL and CIFAR10, mainly because its gradient approximation is less accurate in larger datasets. However, the variance of ASSGD is still smaller than half of the variance of MBSGD with the same min-batch size. Another observable trend is that the variance ratio of ISSGD, ASSGD and ASHR are getting smaller with the increase of training time, suggesting that the history gradient approximation is getting more and more accurate when the training proceeds.

We note that in MBSGD, the variance is proportional to the reverse of the size of mini-batch. Therefore, to get the same level of variance in the stochastic gradient as ActiveSampler, MBSGD needs to increase its min-batch size by 2-3x. From this angle, ActiveSampler is a much more efficient method to reduce the variance of stochastic gradient, instead of relying on the use of a

Dataset	MBSGD	ASSGD	ASHR	ISSGD
MNIST	0.179s	0.208s	0.205s	0.205s
URL	0.080s	0.092s	0.092s	0.091s
CIFAR10	0.245s	0.308s	0.311s	0.309s
CIFAR-DA	0.110s	0.129s	0.129s	0.128s

Table 7.2: Training Time per Iteration

larger mini-batch.

7.3.4 Training Time Analysis

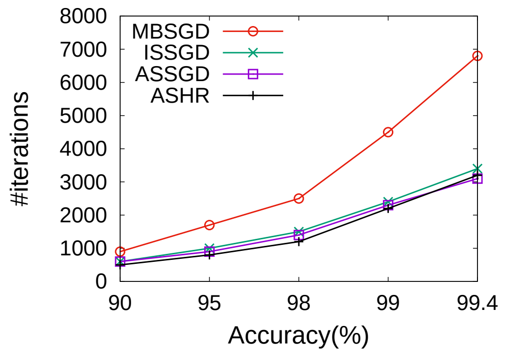
The overall training time is determined by the product of the training time per iteration and the number of iterations to reach a certain accuracy. Here we present a detailed study of how ActiveSampler performs from these two aspects.

Table 7.2 shows the training time per iteration of MBSGD, ISSGD, ASSGD and ASHR. Obviously, MBSGD is the fastest one since ISSGD, ASSGD and ASHR entail additional computations. However, the difference is not significant. ISSGD, ASSGD and ASHR only require 10%-25% more time than MBSGD, while providing the stochastic gradient with much smaller variance. There are no major differences among ISSGD, ASSGD and ASHR, since they have almost the same vectorized computation logics inside each iteration.

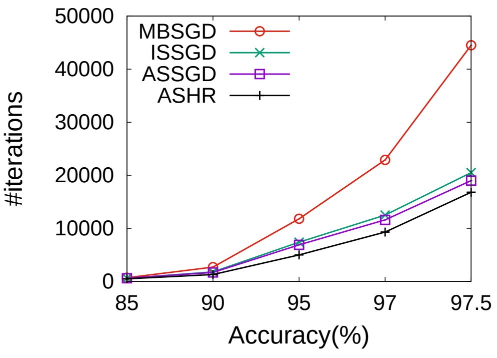
Figure 7.7 shows the number of iterations to reach a certain accuracy. The number of iterations required by ISSGD, ASSGD and ASHR is around 40% to 60% of the number of iterations required by MBSGD. ASSGD and ASHR outperforms ISSGD by reducing 10%-25% of the number of iterations. The reduced proportion of iterations varies with different datasets. More iterations can be saved when the contribution from training examples are highly biased.

7.4 Summary

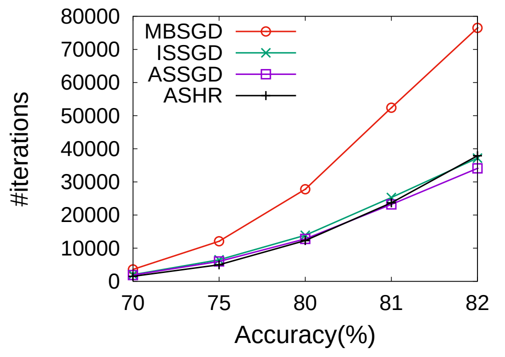
SGD algorithms are playing a central role in the model training of complex data analytics, where sampled training data are used at each training iteration. Uniform sampling and sequential access have been commonly used due to their simplicity. In this chapter, we study how the sampling method can affect the training speed as a means to facilitate analytics at scale. Based on



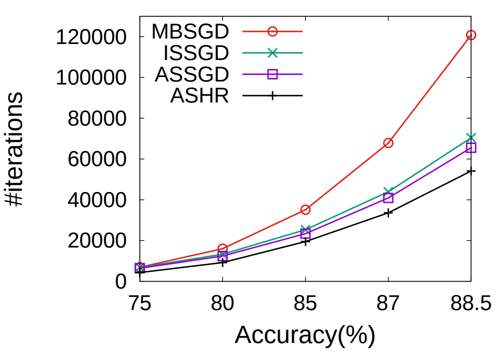
(a) MNIST



(b) URL



(c) CIFAR10



(d) CIFAR-DA (Distributed)

Figure 7.7: Number of Iterations with respect to Accuracy

the inspiration from active learning, we propose ActiveSampler which for each training point the sampling frequency is proportional to its expectation of gradient magnitude. We also develop a set of schemes to make the algorithm light-weight and fully vectorized. Experiments show that ActiveSampler can accelerate the training procedure of SVM, feature selection and deep learning by 1.8-2.3x, compared with the uniform sampling. Results also demonstrate that ActiveSampler has a significant effect on reducing the variance of the stochastic gradient, making the training process more stable.

CHAPTER 8

CONCLUSION

The aim of this thesis is to conduct a systematic study of the training data management infrastructure for data transformation, from structured but still raw data to processable informative data, by leveraging affordable human effort and computational resources. Two challenging problems are discussed and addressed. The first challenge is to increase the information value in Big Data, mainly by acquiring extra supervised information from data annotation. The second challenge is to squeeze and reorganize the data to a processable form without losing much information inside the original data, which typically includes representing, compressing, indexing and sampling the data to increase the computational efficiency.

To achieve this goal, we develop effective and efficient solutions to transform the Big Data into a processable and informative form. We propose a preference quantified model to annotate complex tasks where the supervised information is difficult to be represent by simple labels, and adapt an active learning approach to reduce the cost of human efforts. To further reduce the cost of data annotation by using crowdsourcing, we develop a cost-sensitive method for crowdsourced data quality management. We propose a hashing method to transform the training data into better compact representation, while preserving both internal information in each instances and external relations among those instances. Moreover, we index the data which are usually high-dimensional to support similarity queries based on the distance independent k-nearest neighbor measure. Finally, we study the effect of data sampling pattern on the efficiency of analytics model training, aiming to provide the most informative data in a processable size to the analytics model to speed up the model training procedure. All the methods are effective experimentally. The future works in this area may include establishing a automatic cost-sensitive infrastructure for training data management to address these challenges in an integrated manner, supporting multiple training tasks and acquiring suitable data annotations to benefit most tasks economically, discovering the most applicable similarity measure based on analytics tasks and adapting the hashing codes and index, providing diversified data in a processable volume to offer better coverage.

BIBLIOGRAPHY

[1] `http://www.mturk.com`.

[2] O. Alonso, D.E. Rose, and B. Stewart. Crowdsourcing for relevance evaluation. In *SIGIR Forum*, volume 42, pages 9–15. ACM, 2008.

[3] Alexandr Andoni, Piotr Indyk, Huy L Nguyen, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *SODA*, pages 1018–1028, 2014.

[4] Konstantin Andreev and Harald Racke. Balanced graph partitioning. *TOCS*, pages 929–939, 2006.

[5] Vassilis Athitsos, Jonathan Alon, Stan Sclaroff, and George Kollios. Boostmap: A method for efficient approximate similarity rankings. In *CVPR*, pages II–268, 2004.

[6] Vassilis Athitsos, Marios Hadjieleftheriou, George Kollios, and Stan Sclaroff. Query-sensitive embeddings. In *SIGMOD*, pages 706–717, 2005.

[7] Vassilis Athitsos, Michalis Potamias, Panagiotis Papapetrou, and George Kollios. Nearest neighbor retrieval using distance-based hashing. In *ICDE*, pages 327–336, 2008.

[8] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural computation*, pages 1373–1396, 2003.

[9] Jon Louis Bentley. K-d trees for semidynamic point sets. In *Symposium on Computational Geometry*, pages 187–197, 1990.

[10] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *ISMIR*, 2011.

[11] Dimitri P Bertsekas. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. *Optimization for Machine Learning*, 2011.

[12] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *COMPSTAT*. 2010.

[13] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[14] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. ICML, 2005.

[15] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 2010.

[16] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. Adapting ranking svm to document retrieval. In *SIGIR*, 2006.

[17] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM TIST*, 2011.

[18] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys*, pages 273–321, 2001.

[19] Xi Chen, Paul N Bennett, Kevyn Collins-Thompson, and Eric Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *WSDM*, 2013.

[20] David A Cohn. Neural network exploration using optimal experiment design. In *NIPS*, 1994.

[21] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[22] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. Large scale distributed deep networks. In *NIPS*, 2012.

[23] John E Dennis, Jr and Jorge J Moré. Quasi-newton methods, motivation and theory. *SIAM Review*.

[24] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *JMLR*, 2011.

[25] Tarek Elgamal, Maysam Yabandeh, Ashraf Aboulnaga, Waleed Mustafa, and Mohamed Hefeeda. spca: Scalable principal component analysis for big data on distributed platforms. In *SIGMOD*, 2015.

[26] A. Feng, M. Franklin, D. Kossmann, T. Kraska, S. Madden, S. Ramesh, A. Wang, and R. Xin. Crowddb: Query processing with the vldb crowd. *VLDB*, 4(12), 2011.

[27] M.J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. Crowddb: answering queries with crowdsourcing. In *SIGMOD*, pages 61–72, 2011.

[28] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *COLT*, pages 23–37, 1995.

[29] Atsushi Fujii, Takenobu Tokunaga, Kentaro Inui, and Hozumi Tanaka. Selective sampling for example-based word sense disambiguation. *Computational Linguistics*, 1998.

[30] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. Locality-sensitive hashing scheme based on dynamic collision counting. In *SIGMOD*, pages 541–552, 2012.

[31] Jinyang Gao, H V Jagadish, Wei Lu, and Beng Chin Ooi. Dsh: data sensitive hashing for high-dimensional k-nnsearch. In *SIGMOD*, pages 1127–1138, 2014.

[32] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[33] S. Guo, A. Parameswaran, and H. Garcia-Molina. So who won?: dynamic max discovery with the crowd. In *SIGMOD*, pages 385–396. ACM, 2012.

[34] Antonin Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[35] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *JMLR*, 2003.

[36] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. Spherical hashing. In *CVPR*, pages 2957–2964, 2012.

[37] Gisli R Hjaltason and Hanan Samet. Index-driven similarity search in metric spaces (survey article). *TODS*, pages 517–580, 2003.

[38] Chungwu Ho and Seth Zimmerman. On the number of regions in an m-dimensional space cut by n hyperplanes. *Gazette of Australian Math Society*, 2006.

[39] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *JMLR*, 2013.

[40] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.

[41] Yoonho Hwang, Bohyung Han, and Hee-Kap Ahn. A fast nearest neighbor search algorithm by nonlinear embedding. In *CVPR*, pages 3053–3060, 2012.

[42] Alexander Ilin and Tapani Raiko. Practical approaches to principal component analysis in the presence of missing values. *JMLR*, 2010.

[43] P.G. Ipeirotis, F. Provost, and J. Wang. Quality management on amazon mechanical turk. In *SIGKDD workshop*, pages 64–67. ACM, 2010.

[44] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. idistance: An adaptive B$^+$-tree based indexing method for nearest neighbor search. *TODS*, pages 364–397, 2005.

[45] Kevin G Jamieson and Robert Nowak. Active ranking using pairwise comparisons. In *NIPS*, 2011.

[46] Hervé Jégou, Laurent Amsaleg, Cordelia Schmid, and Patrick Gros. Query adaptative locality sensitive hashing. In *ICASSP*, pages 825–828, 2008.

[47] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*, 2013.

[48] Alexis Joly and Olivier Buisson. A posteriori multi-probe locality sensitive hashing. In *MM*, pages 209–218, 2008.

[49] Norio Katayama and Shin'ichi Satoh. The sr-tree: An index structure for high-dimensional nearest neighbor queries. In *SIGMOD*, pages 369–380, 1997.

[50] G. Kazai, J. Kamps, M. Koolen, and N. Milic-Frayling. Crowdsourcing for book search evaluation: impact of hit design on comparative system ranking. *SIGIR*, 2011.

[51] John Frank Charles Kingman. *Poisson processes*, volume 3. Oxford university press, 1992.

[52] A. Kittur, E.H. Chi, and B. Suh. Crowdsourcing user studies with mechanical turk. In *SIGCHI*, pages 453–456. ACM, 2008.

[53] Weihao Kong and Wu-Jun Li. Isotropic hashing. In *NIPS*, 2012.

[54] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[55] Brian Kulis and Kristen Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137, 2009.

[56] Arun Kumar, Jeffrey Naughton, and Jignesh M Patel. Learning generalized linear models over normalized data. In *SIGMOD*, 2015.

[57] David D Lewis and William A Gale. A sequential algorithm for training text classifiers. In *SIGIR*, 1994.

[58] Hang Li. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 2011.

[59] Hang Li. A short introduction to learning to rank. *IEICE Transactions*, 2011.

[60] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

[61] Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *SIGKDD*, 2014.

[62] Yue Lin, Rong Jin, Deng Cai, Shuicheng Yan, and Xuelong Li. Compressed hashing. In *CVPR*, pages 446–451, 2013.

[63] Jun Liu, Jianhui Chen, and Jieping Ye. Large-scale sparse logistic regression. In *SIGKDD*, 2009.

[64] X. Liu, M. Lu, B.C. Ooi, Y. Shen, S. Wu, and M. Zhang. Cdas: a crowdsourcing data analytics system. *VLDB*, 5(10):1040–1051, 2012.

[65] Bo Long, Olivier Chapelle, Ya Zhang, Yi Chang, Zhaohui Zheng, and Belle Tseng. Active learning for ranking through expected loss optimization. In *SIGIR*, 2010.

[66] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*, pages 950–961, 2007.

[67] Justin Ma, Lawrence K Saul, Stefan Savage, and Geoffrey M Voelker. Identifying suspicious urls: an application of large-scale online learning. In *ICML*, 2009.

[68] A. Marcus, E. Wu, D.R. Karger, S. Madden, and R.C. Miller. Crowdsourced databases: Query processing with people. CIDR, 2011.

[69] Adam Marcus, Eugene Wu, David R. Karger, Samuel Madden, and Robert C. Miller. Demonstration of qurk: a query processor for humanoperators. In *SIGMOD Conference*, pages 1315–1318, 2011.

[70] Rajeev Motwani, Assaf Naor, and Rina Panigrahy. Lower bounds on locality sensitive hashing. *Discrete Mathematics*, pages 930–935, 2007.

[71] Yadong Mu, Wei Liu, and Wei Fan. Stochastic gradient made stable: A manifold propagation approach for large-scale optimization. *arXiv:1506.08350*, 2015.

[72] Yadong Mu, Jialie Shen, and Shuicheng Yan. Weakly-supervised hashing in kernel space. In *CVPR*, pages 3344–3351, 2010.

[73] Ion Muslea, Steven Minton, and Craig A Knoblock. Selective sampling with redundant views. In *AAAI*, 2000.

[74] Sahand Negahban, Sewoong Oh, and Devavrat Shah. Iterative ranking from pair-wise comparisons. In *NIPS*, 2012.

[75] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 2009.

[76] X Niyogi. Locality preserving projections. In *NIPS*, pages 153–160, 2004.

[77] Rina Panigrahy. Entropy based nearest neighbor search in high dimensions. In *SODA*, pages 1186–1195, 2006.

[78] A. Parameswaran and N. Polyzotis. Answering queries using humans, algorithms and databases. 2011.

[79] A. Parameswaran, A.D. Sarma, H. Garcia-Molina, N. Polyzotis, and J. Widom. Human-assisted graph search: it's okay to ask questions. *VLDB*, 4(5):267–278, 2011.

[80] A.G. Parameswaran, H. Garcia-Molina, H. Park, N. Polyzotis, A. Ramesh, and J. Widom. Crowdscreen: Algorithms for filtering data with humans. In *SIGMOD*, pages 361–372. ACM, 2012.

[81] Michael Prince. Does active learning work. a review of the research. *Journal of Engineering Education-Washington*, 2004.

[82] V.C. Raykar, S. Yu, L.H. Zhao, G.H. Valadez, C. Florin, L. Bogoni, and L. Moy. Learning from crowds. *JMLR*, 11:1297–1322, 2010.

[83] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 1951.

[84] Nicholas Roy and Andrew McCallum. Toward optimal active learning through monte carlo estimation of error reduction. *ICML*, 2001.

[85] Ruslan Salakhutdinov and Geoffrey E Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *AI and Statistics*, pages 412–419, 2007.

[86] Venu Satuluri and Srinivasan Parthasarathy. Bayesian locality sensitive hashing for fast similarity search. *VLDB*, pages 430–441, 2012.

[87] Robert E Schapire. The boosting approach to machine learning: An overview. *LECTURE NOTES IN STATISTICS*, pages 149–172, 2003.

[88] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. *arXiv:1309.2388*, 2013.

[89] David W Scott. *Multivariate density estimation: theory, practice, and visualization*, volume 383. John Wiley & Sons, 2009.

[90] J. Selke, C. Lofi, and W.T. Balke. Pushing the boundaries of crowd-enabled databases with query-driven schema expansion. *VLDB*, 5(6):538–549, 2012.

[91] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[92] Burr Settles. Active learning literature survey. *University of Wisconsin, Madison*, 2010.

[93] Burr Settles and Mark Craven. An analysis of active learning strategies for sequence labeling tasks. In *EMNLP*, 2008.

[94] H Sebastian Seung, Manfred Opper, and Haim Sompolinsky. Query by committee. In *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992.

[95] Gregory Shakhnarovich, Paul Viola, and Trevor Darrell. Fast pose estimation with parameter-sensitive hashing. In *ICCV*, pages 750–757, 2003.

[96] Shai Shalev-Shwartz, Yoram Singer, Nathan Srebro, and Andrew Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical Programming*, 2011.

[97] Nitish Srivastava and Ruslan Salakhutdinov. Multimodal learning with deep boltzmann machines. In *NIPS*, pages 2222–2230, 2012.

[98] Michael Stonebraker. The case for partial indexes. *SIGMOD Record*, pages 4–11, 1989.

[99] Michael Stonebraker and Lawrence A Rowe. *The design of Postgres*, volume 15. ACM, 1986.

[100] Xiaoyuan Su and Taghi M Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in artificial intelligence*, 2009.

[101] Omer Tamuz, Ce Liu, Serge Belongie, Ohad Shamir, and Adam Tauman Kalai. Adaptively learning the crowd kernel. *arXiv:1105.1033*, 2011.

[102] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. Quality and efficiency in high dimensional nearest neighbor search. In *SIGMOD*, pages 563–576, 2009.

[103] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *JMLR*, 2001.

[104] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.

[105] Chong Wang, Xi Chen, Alex J Smola, and Eric P Xing. Variance reduction for stochastic gradient optimization. In *NIPS*, 2013.

[106] J. Wang, T. Kraska, M.J. Franklin, and J. Feng. Crowder: crowdsourcing entity resolution. *VLDB*, 5(11):1483–1494, 2012.

[107] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. *TPAMI*, pages 2393–2406, 2012.

[108] Qing Wang, Sanjeev R Kulkarni, and Sergio Verdú. Divergence estimation for multidimensional densities via-nearest-neighbor distances. *Information Theory*, pages 2392–2405, 2009.

[109] Wei Wang, Gang Chen, Anh Tien Tuan Dinh, Jinyang Gao, Beng Chin Ooi, Kian-Lee Tan, and Sheng Wang. Singa: Putting deep learning in the hands of multimedia users. In *ACM MM*, 2015.

[110] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *VLDB*, pages 194–205, 1998.

[111] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[112] P. Welinder and P. Perona. Online crowdsourcing: rating annotators and obtaining cost-effective labels. In *CVPR workshop*, pages 25–32. IEEE, 2010.

[113] D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 2003.

[114] Feng Yan, Olatunji Ruwase, Yuxiong He, and Trishul Chilimbi. Performance modeling and scalability optimization of distributed deep learning systems. In *SIGKDD*, 2015.

[115] Tingxin Yan, Vikas Kumar, and Deepak Ganesan. Crowdsearch: exploiting crowds for accurate real-time image search on mobile phones. In *MobiSys*, pages 77–90, 2010.

[116] Jinfeng Yi, Rong Jin, Shaili Jain, and Anil Jain. Inferring users preferences from crowdsourced pairwise comparisons: A matrix completion approach. In *HCOMP*, 2013.

[117] Cui Yu, Beng Chin Ooi, Kian-Lee Tan, and H. V. Jagadish. Indexing the distance: An efficient method to knn processing. In *VLDB*, pages 421–430, 2001.

[118] Hwanjo Yu. Selective sampling techniques for feedback-based data retrieval. *DMKD*, 2011.

[119] Ce Zhang, Arun Kumar, and Christopher Ré. Materialization optimizations for feature selection workloads. In *SIGMOD*, 2014.

[120] Ce Zhang and Christopher Ré. Towards high-throughput gibbs sampling at scale: A study across storage managers. In *SIGMOD*, 2013.

[121] Peilin Zhao and Tong Zhang. Stochastic optimization with importance sampling for regularized loss minimization. In *ICML*, 2015.