UNIVERSITY OF CALIFORNIA
RIVERSIDE

Divide and Conquer Algorithms for Machine Learning

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Michael John Izbicki

September 2017

Dissertation Committee:

Dr. Christian R. Shelton, Chairperson
Dr. Stefano Lonardi
Dr. Vagelis Papalexakis
Dr. James Flegal

The Dissertation of Michael John Izbicki is approved:

Committee Chairperson

University of California, Riverside

Acknowledgments

I would like to especially thank my advisor Christian Shelton for his mentorship and encouragement. I would also like to thank the rest of my dissertation committee (Stefano Lonardi, Vagelis Papalexakis, and James Flegal) and professors who served on my candidacy and proposal committees (Hamed Mohsenian-Rad, Ertem Tuncel, and Neal Young). Finally, I would like to thank the members of the RLAIR lab, including Juan Casse, Busra Celikkaya, Zhen Qin, Dave Gumboc, Matthew Zarachoff, Kazi Islam, Sepideh Azarnoosh, Chengkuan Hong, Sanjana Sandeep, Gaurav Jhaveri, Jacob Fauber, and Mehran Ghamaty.

To my wife Kristen.

ABSTRACT OF THE DISSERTATION

Divide and Conquer Algorithms for Machine Learning

by

Michael John Izbicki

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2017
Dr. Christian R. Shelton, Chairperson

This thesis improves the scalability of machine learning by studying **mergeable learning algorithms**. In a mergeable algorithm, many processors independently solve the learning problem on small subsets of the data. Then a master processor merges the solutions together with only a single round of communication. Mergeable algorithms are popular because they are fast, easy to implement, and have strong privacy guarantees.

Our first contribution is a novel fast cross validation procedure suitable for any mergeable algorithm. This fast cross validation procedure has a constant runtime independent of the number of folds and can be implemented on distributed systems. This procedure is also widely applicable. We show that 32 recently proposed learning algorithms are mergeable and therefore fit our cross validation framework. These learning algorithms come from many subfields of machine learning, including density estimation, regularized loss minimization, dimensionality reduction, submodular optimization, variational inference, and markov chain monte carlo.

We also provide two new mergeable learning algorithms. In the context of regularized loss minimization, existing merge procedures either have high bias or slow runtimes. We introduce the **optimal weighted average** (OWA) merge procedure, which achieves

both a low bias and fast runtime. We also improve the **cover tree** data structure for fast nearest neighbor queries by providing a merge procedure. In doing so, we improve both the theoretical guarantees of the cover tree and its practical runtime. For example, the original cover tree was able to find nearest neighbors in time $O(c_{\text{exp}}^{12} \log n)$, and we improve this bound to $O(c_{\text{hole}}^{4} \log n)$ for i.i.d. data. Here, c_{exp} and c_{hole} are measures of the "intrinsic dimensionality" of the data, and on typical datasets $c_{\text{exp}} > c_{\text{hole}}$. Experiments on large scale ad-click, genomics, and image classification tasks empirically validate these algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning has seen many recent successes due to the availability of large datasets. For example, Facebook collects more than 200 terabytes of user information per day (Vagata and Wilfong). This information includes text messages sent between users, photos uploaded by users, and the webpages that users visit. Machine learning algorithms use this data to predict who a user is friends with offline (Curtiss et al., 2013), identify users in uploaded images (Taigman et al., 2014), and determine which ads are the most profitable to display (He et al., 2014). These learning algorithms run on an estimated "hundreds of thousands" of servers located at six data centers around the world (Facebook Datacenter FAQ). Technology companies like Amazon, Apple, Google, Microsoft, Netflix, and Twitter all depend on machine learning algorithms operating on similarly large scale data. Developing algorithms that work at this scale is one of the central problems of modern machine learning research.

Parallel algorithms are needed to solve these large scale problems, but designing parallel algorithms is difficult. There are many architectures for parallel computing, and

most parallel algorithms work well only for certain architectures. The simplest architecture is the **shared memory model**, which corresponds to a typical desktop computer with multiple cores. Communication between cores has almost no overhead, so shared memory parallel algorithms may perform many rounds of communication without slowing down. The Hogwild learning algorithm (Recht et al., 2011) is a famous example. The **distributed model** of computing is an alternative where communication between processors is more expensive. In distributed algorithms, the dataset is too large to fit in the memory of a single machine. So the data is partitioned onto a cluster of machines connected by a network. Communication is relatively expensive and good distributed algorithms use clever tricks to minimize communication. Frameworks such as MPI (Message Passing Forum, 1994), MapReduce (Dean and Ghemawat, 2008) and Spark (Meng et al., 2016) have been developed to provide simple abstractions to manage this communication. In a typical distributed environment, all machines are located in the same building with high speed network connections; so communication takes on the order of milliseconds. But some examples are more extreme. Google recently proposed the **federated learning model** of computation where the processors are user cellphones (McMahan et al., 2017). Cellphones have only intermittent network connectivity, and so communication can take hours or days. The use of cellphones as processing nodes further raises privacy concerns.

This thesis studies a particularly simple class of parallel algorithms called **mergeable learning algorithms**. Mergeable learning algorithms work using the following divide and conquer procedure: first, each processor independently solves the learning problem on a small subset of data; then a master processor merges the solutions together with only

a single round of communication. Mergeable algorithms are growing in popularity due to three attractive properties:

1. *They have small communication complexity.* This makes them suitable for both the shared memory and distributed parallel environments.

2. *They are easy to implement.* Because each processor works independently on a small dataset, these processors can use standard libraries for solving small scale learning problems. For example, these processors could use Python's scikit-learn (Pedregosa et al., 2011), C++'s MLPack (Curtin et al., 2013a), or the libraries provided by the R (R Development Core Team, 2008) or Julia (Bezanson et al., 2017) languages. To implement a mergeable algorithm, the programmer need only implement the merge function. This is typically a simple task.

3. *They have strong privacy guarantees.* Differential privacy is the standard mathematical technique for privacy preserving data analysis (Dwork et al., 2014). Single processor differentially private algorithms are now well understood (Chaudhuri et al., 2011). Using these locally private methods ensures that confidential data doesn't leak between machines or leak to the outside world.

These three advantages are well known, and they are the primary motivation for a recent growth in popularity of mergeable algorithms. The first major contribution of this thesis is to show that mergeable algorithms have a fourth benefit:

4 *They have a fast cross validation algorithm.* Cross validation is a method for estimating the quality of a machine learning algorithm. Standard k-fold cross validation takes

time linear in k and cannot be used on large scale datasets. Chapter 2 shows that all mergeable algorithms have a fast cross validation method that takes constant time independent of k. This fast cross validation method is suitable even in the distributed environment where communication is expensive.

Chapter 2 emphasizes the wide applicability of this new fast cross validation method by describing 32 recently developed mergeable algorithms. They come from many subfields of machine learning, including density estimation, regularized loss minimization, dimensionality reduction, submodular optimization, variational inference, and Markov chain Monte Carlo. Much of this work appears to be unaware of the developments in mergeable estimators from other subfields of machine learning, and our survey points out how these fields can benefit from each other. For example, we show that mergeable algorithms for regularized loss minimization can be directly applied to solve variational inference problems, improving the known results for variational merge procedures. We emphasize that this fast cross validation algorithm is applicable to all 32 mergeable estimators that we present in Chapter 2. For many of these problems, no previous fast cross validation procedure was known, and so cross validation could not be performed on large scale datasets.

The remainder of this thesis develops two new mergeable estimators. Chapter 3 presents the **optimal weighted average** for regularized risk minimization (RLM). RLM is one of the most important learning paradigms. It includes important models such as logistic regression, kernel algorithms, and neural networks. As we show in the survey portion of Chapter 2, existing merge procedures for RLM either have poor statistical performance or poor computational performance. The OWA merge procedure is the first algorithm that has

both good statistical and computational performance. The key insight of OWA is that the merge procedure depends on the data, whereas all previous merge procedures for RLM do not depend on the data. We validate the OWA merge procedure empirically using logistic regression and convolutional neural network models.

Chapter 4 discusses the cover tree data structure for fast nearest neighbor queries in arbitrary metric spaces. Cover trees were first introduced by Beygelzimer et al. (2006), and they have seen widespread use in the machine learning community since then. This chapter shows that cover trees can be merged together, providing the first parallel construction and fast cross validation algorithms. We also improve the cover tree's theoretical and practical performance. As an example, the cover tree's original analysis showed that nearest neighbor queries could be answered in time $O(c_{\text{exp}}^{12} \log n)$, and we improve this bound to $O(c_{\text{hole}}^{4} \log n)$ for i.i.d. data. Here c_{exp} and c_{hole} are measures of the "intrinsic dimensionality" of the data. We will show that on typical datasets $c_{\text{exp}} > c_{\text{hole}}$, and we will argue that in pathological data sets c_{hole} more accurately represents our intuitive notion of dimensionality. We further provide practical improvements in the form of a new data structure called the **simplified cover tree**. These practical improvements include an easier to understand definition, reduced overhead in tree traversals, and improved cache performance. We use benchmark datasets with the Euclidean distance to show that our simplified cover tree implementation is faster than both existing cover tree implementations and other techniques specialized for Euclidean distance nearest neighbor queries such as kd-trees and locality sensitive hashing. We also compare the simplified and original cover trees on non-Euclidean protein interaction and computer vision problems.

Chapter 5 concludes the thesis with a summary of open problems. We argue that little is known about the best possible merge procedures, and that the existing merge procedures likely are suboptimal.

Chapter 2

Mergeable learning algorithms

Chapter 1 argued that mergeable learning algorithms are popular because they are fast, easy to implement, and provide good privacy guarantees. This chapter develops a novel fast cross validation method for mergeable algorithms and shows that 32 recently developed learning algorithms are mergeable. For most of these learning algorithms, no previous fast cross validation procedure was known. Our fast cross validation method can be used on all of them.

The chapter begins with a discussion of the classic ridge regression algorithm in Section 2.1. Ridge regression has well known distributed training and fast cross validation algorithms. Then Section 2.2 formally defines mergeable algorithms as algorithms that have nice computational properties similar to ridge regression. The main contribution of this chapter is Section 2.3, which introduces a novel fast cross validation procedure applicable to all mergeable algorithms. To motivate the need for fast cross validation, we first review existing methods for estimating the quality of an estimator. We show that these methods

Section	Problem	Number of merge methods
2.4.1	closed form estimators	3
2.4.2.2	averaging methods	9
2.4.2.3	principle component analysis	2
2.4.2.4	submodular optimization	6
2.4.3.1	parametric Bayesian inference	1
2.4.3.2	variational inference	3
2.4.3.3	Markov chain Monte Carlo	8

Table 2.1: Summary of the existing mergeable learning algorithms.

either do not scale to large datasets or have suboptimal statistical properties. Our fast cross validation procedure, however, is both highly scaleable and statistically efficient. Another advantage of our cross validation procedure is that it works in both the single machine and distributed environments. This improves previous methods for fast cross validation which worked only in the single machine environment and only on a much smaller class of models.

Section 2.4 argues that the framework of mergeable learning algorithms is widely applicably by describing 32 published examples. We divide these examples into three categories. Section 2.4.1 describes how to merge learning algorithms with closed form solutions such as linear exponential families and naive Bayes. Most estimators, however, do not have closed form solutions. Section 2.4.2 describes merge procedures for regularized loss minimization problems, which typically lack a closed form solution. The procedures in this section apply to a large class of learning models, one of the most important being logistic regression. The most general methods are based on parameter averaging, but we also present more methods designed specifically for principle component analysis and submodular optimization. Finally, Section 2.4.3 describes merge procedures for Bayesian methods. It includes subsections on parametric estimation, variational inference, and Markov chain

Monte Carlo. Table 2.1 summarizes these examples.

2.1 Ridge regression as a motivating example

Ridge regression is one of the earliest learning algorithms, but it is still widely studied and used. For example, Meng et al. (2014), Wang et al. (2016b), and Gascon et al. (2017) propose distributed learning algorithms in different settings. In this section, we describe some of the computationally attractive properties of the ridge regression algorithm. These properties enable easy distributed learning and fast cross validation. In subsequent sections we will see that mergeable algorithms are a generalization of ridge regression that share its nice computational properties.

In the **regression problem**, we are given a matrix of **covariates** $X : \mathbb{R}^{d \times mn}$ and a vector of **response variables** $Y : \mathbb{R}^{mn}$. Here we assume that the dimensionality of the problem is d and there are mn data points. The goal is to learn an unknown function $y^* : \mathbb{R}^d \to \mathbb{R}$. In a **linear regression problem**, the function y^* is defined to be

$$y^*(\mathbf{x}) = \mathbf{w}^{*\mathsf{T}}\mathbf{x} \tag{2.1}$$

where $\mathbf{w}^* : \mathbb{R}^d$ is an unknown vector of model parameters that we must learn. The standard **ridge regression algorithm** estimates $\mathbf{w}^*$ using the formula

$$\hat{\mathbf{w}}^{ridge} = \arg\min_{\mathbf{w} \in \mathbb{R}^d} \|Y - \mathbf{w}^{\mathsf{T}} X\|^2 + \lambda \|\mathbf{w}\|^2 \tag{2.2}$$

where λ is a **hyperparameter** that must be set by the user. We can solve this optimization

problem by taking the derivative inside the arg min and setting it to zero. The result has closed form solution

$$\hat{\mathbf{w}}^{ridge} = (X^\mathsf{T} X + \lambda I)^{-1} X^\mathsf{T} Y. \tag{2.3}$$

The matrix product $X^\mathsf{T} X$ take time $O(mnd^2)$, and the inverse takes time $O(d^3)$. Whenever $mn \gg d$, the product dominates the runtime and should be parallelized.

The closed form solution for $\hat{\mathbf{w}}^{ridge}$ lets us construct parallel algorithm using a divide and conquer procedure. First, we divide the data onto m separate machines as follows. Divide the covariate matrix X into m smaller matrices $X_1, ..., X_m$ each of dimension $d \times n$, and divide the response vector Y into m smaller vectors $Y_1, ..., Y_m$ each of dimension n. Then transfer X_i and Y_i to each machine i. Next, each machine i independently calculates the local statistics

$$A_i = X_i^\mathsf{T} X_i \qquad \text{and} \qquad B_i = X_i^\mathsf{T} \mathbf{y}_i. \tag{2.4}$$

The runtime of calculating these local statistics $O(nd^2)$, which is independent of the number of machines m. The local statistics are transmitted to a master server, which calculates

$$\hat{\mathbf{w}}^{ridge} = \left(\sum_{i=1}^{m} A_i + \lambda I \right)^{-1} \sum_{i=1}^{m} B_i. \tag{2.5}$$

By definition of A_i and B_i, Equations (2.3) and (2.5) are the same. Calculating the summations in (2.5) takes time $O(md^2)$ and the inverse takes time $O(d^3)$. The total runtime of the parallel procedure is $O((m+n)d^2 + d^3)$, which is faster than the runtime of the sequential procedure $O(mnd^2 + d^3)$.

This divide and conquer algorithm makes ridge regression easy to use on large scale

data. Mergeable learning algorithms are a generalization of ridge regression to all methods that support a similar divide and conquer approach to learning. In the next subsection, we formally define mergeable learning algorithms. Then, we describe a novel fast cross validation procedure that generalizes fast cross validation of ridge regression models (e.g. Golub et al., 1979) to all mergeable algorithms.

2.2 A formal definition of mergeable learning algorithms

We first provide a general definition of learning algorithms, then we define mergeable learning algorithms. Let $\mathcal{Z}$ be a set called the **data space**; let $P_{\mathcal{Z}}$ be an unknown distribution over $\mathcal{Z}$; let $\mathcal{W}$ be a set called the **parameter space**; and let $f : \mathcal{Z} \to \mathcal{W}$ be a function called the **model**. Our goal is to estimate $\mathbb{E}_{\mathbf{z}\sim P_{\mathcal{Z}}} f(\mathbf{z})$ based on a sample of points Z from $\mathcal{Z}$.[1] We call a function $\texttt{alg} : \{\mathcal{Z}\} \to \mathcal{W}$ a **machine learning algorithm** or equivalently an **estimator**[2] if

$$\texttt{alg}(Z) \approx \mathop{\mathbb{E}}_{\mathbf{z}\in Z} f(\mathbf{z}). \tag{2.6}$$

When the data set Z is understood from context, we will denote the parameter estimate by $\hat{\mathbf{w}}^{\texttt{alg}} = \texttt{alg}(Z)$. For example the formula (2.3) that defined $\hat{\mathbf{w}}^{ridge}$ in the previous section defined the ridge regression estimator. Notice that except in trivial models that lack randomness, the approximation in (2.6) can never be exact. The randomness ensures that for any finite data set Z, the left-hand side can be close to the right hand side with high probability, but they will be non-equal with probability 1.

[1] In most cases, the data set Z will be sampled i.i.d. from some $P_{\mathcal{Z}}$, but this is not a requirement.

[2] While the terms machine learning algorithm and estimator are interchangeable, we will typically refer to `alg` as a machine learning algorithm when analyzing its computational properties and as an estimator when analyzing its statistical properties.

We say a learning algorithm `alg` is **mergeable** if it can be decomposed into two functions $\texttt{map} : \{\mathcal{Z}\} \to \mathcal{W}^{local}$ and $\texttt{reduce} : \{\mathcal{W}^{local}\} \to \mathcal{W}$ such that for any collection of disjoint data sets $Z_1, ..., Z_m$ satisfying $\cup_{i=1}^m Z_i = Z$,

$$\texttt{reduce}\{\texttt{map}(Z_i)\}_{i=1}^m \approx \texttt{alg}(\cup_{i=1}^m Z_i). \tag{2.7}$$

In the ridge regression example, Equation (2.4) defined the `map` procedure and Equation (2.5) defined the `reduce` procedure. Unlike the approximation in (2.6), the approximation in (2.7) can be exact (as it is for ridge regression).

Algorithm 1 shows that all mergeable estimators can be distributed using the popular MapReduce framework (Dean and Ghemawat, 2008). However not all algorithms that can be implemented with MapReduce are mergeable. General MapReduce procedures can require many iterations of the `map` and `reduce` functions, whereas mergeable algorithms are allowed only a single iteration. Previous work has systematically studied machine learning algorithms that use an arbitrary number of MapReduce iterations (Chu et al., 2007) and algorithms that use a fixed constant number of iterations (Karloff et al., 2010). This thesis is the first work to study MapReduce learning algorithms that use exactly 1 iteration.[3] Despite this lack of systematic study, we shall see that many existing papers have developed mergeable algorithms for specific problems. The key contribution of this study is shown in Section 2.3, where we present a new fast cross validation procedure applicable to all mergeable algorithms.

[3] With the exception of Izbicki (2013) on which this thesis is based.

Algorithm 1 `dist_learn`(learning algorithm f, data sets Z_i)

prerequisite: each machine i has dataset Z_i stored locally

1: each machine i independently and in parallel:
2: compute $\hat{\mathbf{w}}_i^{\mathtt{alg}} = \mathtt{map}(Z_i)$
3: transmit $\hat{\mathbf{w}}_i^{\mathtt{alg}}$ to the master
4: master machine:
5: compute $\hat{\mathbf{w}}^{\mathtt{alg}} = \mathtt{reduce}\{\hat{\mathbf{w}}_i^{\mathtt{alg}}\}_{i=1}^m$

2.3 Model selection and fast/distributed cross validation

Model selection is the process of selecting a good machine learning algorithm for a particular task. Unfortunately, the most accurate model selection methods have high computational cost and so cannot be used in large scale learning systems. Most research on large scale learning has focused only on accelerating the learning algorithm, with the model selection process receiving comparatively little attention. In this section, we review popular model selection procedures with emphasis on their statistical and computational tradeoffs. A summary is shown in Table 2.2. We show in particular that cross validation has the best statistical properties, but that standard cross validation techniques are too slow for large scale data. We develop two new cross validation techniques that address this shortcoming. In particular, we show that for mergeable learning algorithms cross validation can be performed with only a constant computational overhead independent of the size of the problem, and that this cross validation can be distributed. Effectively, this results in "free" cross validation estimate whenever we train a distributed model.

In the **model selection** problem, we are given a set of models $\mathcal{F} : \{\mathcal{Z} \to \mathcal{W}\}$ and a dataset Z. Our goal is to select a good model $\hat{f} \in \mathcal{F}$. To formally measure the quality of

method	serial runtime	distributed runtime	bias	variance	prerequisites
empirical risk	$O(mn)$	$O(n)$	high	high	–
SRM	$O(mn)$	$O(n)$	low	high	excess risk bound
validation set	$O(mn)$	$O(n)$	low	high	–
bootstrap	$O(kmn)$	$O(kn)$	low	low	–
k-fold cross validation	$O(kmn)$	$O(kn)$	low	low	–
fast k-fold cross validation	$O(mn)$	$O(n)$	low	low	mergable

Table 2.2: Accuracy of different methods for estimating a model's true error. To simplify the notation, we assume that: the total number of data points is mn; in the distributed environment, there are m machines each with n data points; the runtime of `alg` is $O(mn)$, `map` is $O(n)$, and `reduce` is $O(1)$; and the communication cost of distributed learning is negligible. Notice that our fast k-fold cross validation method has both the best runtimes and statistcal properties.

a model, we require a **loss function** $\mathcal{L} : \{\mathcal{Z}\} \times \mathcal{W} \to \mathbb{R}$.[4] The best model f^* is given by

$$f^* = \arg\min_{f \in \mathcal{F}} \text{err}^*(f(Z)) \qquad \text{where} \qquad \text{err}^*(f(Z)) = \mathop{\mathbb{E}}_{\mathbf{z} \sim P_{\mathcal{Z}}} \mathcal{L}(\mathbf{z}, f(Z)). \tag{2.8}$$

Because the distribution over Z is unknown, the distribution over $f(Z)$ is also unknown; so $\text{err}^*(f(Z))$ cannot be directly computed and must be estimated. Given a loss estimator $\widehat{\text{err}}^g \approx \text{err}^*$, we can define an estimator of f^* as

$$\hat{f}^g = \arg\min_{f \in \mathcal{F}} \widehat{\text{err}}^g(f(Z)). \tag{2.9}$$

The minimization (2.9) is highly nonconvex, and a number of algorithms have been proposed to solve it. In the popular but naive **grid search** algorithm, the space of functions $\mathcal{F}$ is discretized into a finite grid $F \subset \mathcal{F}$, and the optimization is performed over F. Grid search tends to work well when $\mathcal{F}$ is one dimensional, but the performance degrades in

[4] Many specific models assume that $\mathcal{L}$ is convex or satisfies linearity properties, but in general this need not be the case.

higher dimensions. More advanced techniques involve randomly subsampling $\mathcal{F}$ (Bergstra and Bengio, 2012) and the so-called **Bayesian optimization** techniques (e.g. Snoek et al., 2012; Feurer et al., 2015). Each of these techniques requires as a subroutine a good estimate $\widehat{\text{err}}^g$ of the true error. We focus in the remainder of this section on these loss estimators.

The most obvious estimator of err^* is the **empirical risk**

$$\widehat{\text{err}}^{emp} = \mathcal{L}(Z, \texttt{alg}(Z)). \tag{2.10}$$

Unfortunately, the empirical risk $\widehat{\text{err}}^{emp}$ is heavily biased because it evaluates the model on the same data it was trained on. **Structural risk minimization** (SRM) is a technique that uses a debiased empirical risk $\widehat{\text{err}}^{emp}$ for model selection (Chapter 7 of Shalev-Shwartz and Ben-David, 2014). The idea is to create a function $b : \{\mathcal{Z}\} \to \mathbb{R}$ that upper bounds the **excess risk**. That is,

$$\text{err}^* - \widehat{\text{err}}^{emp} \leq b(Z). \tag{2.11}$$

These bounds are often given in terms of the learning algorithm's VC-dimension or Rademacher complexity. The SRM risk estimator is then given by

$$\widehat{\text{err}}^{srm} = \widehat{\text{err}}^{emp} + b(Z). \tag{2.12}$$

The main advantage of the SRM estimator is its computational simplicity. SRM is not often used in practice because deriving the bound function b can be difficult, and the known bounds are typically not tight enough to give good approximations.

The most commonly used risk estimator in large scale learning is the **validation**

set estimator (Chapter 11 of Shalev-Shwartz and Ben-David, 2014). This method divides the training data set into two disjoint sets Z^{train} and Z^{test}. The risk is then estimated by

$$\widehat{\text{err}}^{valid} = \mathcal{L}(Z^{test}, f(Z^{train})). \tag{2.13}$$

This estimator has low bias but high variance. In particular, the variance of $\widehat{\text{err}}^{valid}$ is determined by the number of samples in the validation set: as the number of samples increases, the variance decreases. Unfortunately, increasing the size of the validation set decreases the size of the test set, increasing the model's true error. There is no general procedure for finding the optimal balance between the size of the training and test sets, although in practice it is common to use about 10% of the data in the validation set. The validation set method shares the favorable computational properties of SRM, but is easier to use because there is no need to derive a bound on the excess risk. When the bound is loose, as is often the case in practice, the validation set method will also have improved statistical performance. Whenever the learning algorithm is parallelizable, then both SRM and the validation set method are parallelizable as well.

The **bootstrap** improves the statistical properties of the validation set estimator but is more expensive computationally (Efron et al., 1979). The full procedure is shown in Algorithm 2. The idea is to use repeated subsampling to create many overlapping train/validation test splits, then average the results. This averaging reduces the variance of the error estimate. The runtime is linear in the number of resampling iterations k, which is too expensive for large scale problems. Kleiner et al. (2012) and Kleiner et al. (2014) introduce the **bag of little bootstraps** (BLB) as an efficiently computatable alternative

Algorithm 2 bootstrap(learning algorithm f, data set Z, number of samples t)

for $i = 1$ **to** k **do**
    let $Z^{(i)}$ be a sample of mn data points from Z with replacement
    let $\widehat{\text{err}}^{(i)} = \mathcal{L}(Z - Z^{(i)}, f(Z^{(i)}))$
end for
return $\widehat{\text{err}}^{boot} = \frac{1}{k}\sum_{i=1}^{k} \widehat{\text{err}}^{(i)}$

Algorithm 3 `cv`(learning algorithm f, data set Z, number of folds k)

partition Z into k disjoint sets $Z_1, ..., Z_k$
for $i \in \{1, ..., k\}$ **do**
    let $Z_i^{train} = Z - Z_i$
    let $\hat{\mathbf{w}}_{-i} = f(Z_i^{train})$
    let $\widehat{\text{err}}_i^{emp} = \mathcal{L}(\hat{\text{model}}_i, Z_i)$
end for
return $\frac{1}{k}\sum_{i=1}^{k} \widehat{\text{err}}_i^{emp}$

to the bootstrap. The BLB subsamples some small fraction of the data set in each iteration before performing the full bootstrap test. This increases the bias of the estimator but makes it suitable for large scale problems. As there are no data dependencies between each of the iterations, both the standard bootstrap and BLB are easy to parallelize even when the learning algorithm is not parallelizable.

Cross validation is a family of techniques that also improves the validation set method (see Arlot et al., 2010, for a survey). A standard version called k-fold cross validation is shown in Algorithm 3. The data set Z is partitioned into k smaller data sets $Z_1, ..., Z_k$. For each of these data sets, the validation set estimator is used to approximate the true error. The results are then averaged and returned.

Naive k-fold cross validation is an expensive procedure taking time linear in the number of folds k. This makes it unsuitable for large scale learning problems. Algorithm

Algorithm 4 `fast_cv`(learning algorithm f, data set Z, number of folds k)

partition Z into k equally sized disjoint sets $Z_1, ..., Z_k$
// calculate local models
for $i = 1$ **to** k **do**
    $\hat{\mathbf{w}}_i^{\texttt{alg}} = \texttt{map}(Z_i)$
end for
// calculate prefixes
for $i = 1$ **to** k **do**
    $\hat{\mathbf{w}}_{prefix,i} = \texttt{reduce}(\hat{\mathbf{w}}_{prefix,i-1}, \hat{\mathbf{w}}_i^{\texttt{alg}})$
end for
// calculate suffixes
for $i = k$ **to** 1 **do**
    $\hat{\mathbf{w}}_{suffix,i} = \texttt{reduce}(\hat{\mathbf{w}}_{suffix,i+1}, \hat{\mathbf{w}}_i^{\texttt{alg}})$
end for
// merge models and calculate estimated risk
for $i = 1$ **to** k **do**
    $\hat{\mathbf{w}}_{-i}^{\texttt{alg}} = \texttt{reduce}(\hat{\mathbf{w}}_{prefix,i}, \hat{\mathbf{w}}_{suffix,i})$
    $\widehat{\text{err}}^{(i)} = \mathcal{L}(\hat{\mathbf{w}}_{-i}^{\texttt{alg}}, Z_i)$
end for
return $\widehat{\text{err}}^{cv} = \frac{1}{k}\sum_{i=1}^{k} \widehat{\text{err}}^{(i)}$

4 presents a faster method for cross validation that uses the $\texttt{reduce}_f$ function (Izbicki, 2013). The algorithm is divided into four loops. In the first loop, a "local model" $\hat{\mathbf{w}}_i^{\texttt{alg}}$ is trained on each partition Z_i. The second loop calculates "prefix models," which satisfy the property that $\hat{\mathbf{w}}_{prefix,i} \approx f(\cup_{j=1}^{i-1} Z_j)$. The third loop calculates "suffix models," which satisfy the property that $\hat{\mathbf{w}}_{prefix,i} \approx f(\cup_{j=i+1}^{k} Z_j)$. The final loop merges the prefix and suffix models so that $\hat{\mathbf{w}}_{-i}^{\texttt{alg}} \approx f(Z - Z_i)$. The computed error is then an unbiased estimate of the true error. When the $\texttt{reduce}_f$ function is exact, then `fast_cv` will return exactly the same answer as the standard `cv` procedure. When $\texttt{reduce}_f$ is not exact, then `fast_cv` will be an approximate version of cross validation, and the approximation depends on the accuracy of $\texttt{reduce}_f$.

Standard cross validation can be easily parallelized as there are no data dependen-

Algorithm 5 `dist_cv`(learning algorithm f, data set Z, number of folds k)

prerequisite: each machine i has dataset Z_i stored locally

each machine i:
    calculate $\hat{\mathbf{w}}_i = \texttt{map}(Z_i)$
    broadcast $\hat{\mathbf{w}}_i$ to each other machine
each machine i:
    compute $\mathbf{w}_{-i} = \texttt{reduce}\{\mathbf{w}_1, ..., \mathbf{w}_{i-1}, \mathbf{w}_{i+1}, ..., \mathbf{w}_k\}$
    compute $\widehat{\text{err}}_i^{emp} = \mathcal{L}(\hat{\mathbf{w}}_{-i}, Z_i)$
    transmit $\widehat{\text{err}}_i^{emp}$ to the master
the master machine:
    compute $\widehat{\text{err}}^{emp} = \frac{1}{k} \sum_{i=1}^{k} \widehat{\text{err}}_i^{emp}$
    return $\widehat{\text{err}}^{emp}$

cies between each iteration of the foor loop. The `fast_cv` method is slightly more difficult to parallelize due to the data dependencies in the first and second for loops.[5] Algorithm 5 presents a distributed version of fast cross validation. This algorithm is closely related to the distributed learning algorithm of Algorithm 1 and has essentially the same runtime. In other words, when learning an embarrassingly parallel model on a distributed architecture, we can also perform cross validation with essentially no computational overhead. The only difference between the distributed cross validation and distributed learning algorithms is that in the second round of communication, each machine (rather than only the master) computes the model $\mathbf{w}_{-i}$. As these machines were sitting idle in the distributed training procedure in Algorithm 1, having them perform work does not increase the runtime. The runtime of the cross validation procedure is increased slightly by the fact that an additional round of communication is needed where the master averages the validation errors calculated on each machine.

[5] The standard parallel prefix-sum algorithm can be used to compute the $\hat{\mathbf{w}}_{prefix,i}$ and $\hat{\mathbf{w}}_{prefix,j}$ vectors (Ladner and Fischer, 1980; Blelloch, 1990), which would parallelize the `fast_cv` method. These parallel algorithms are designed for shared memory systems, and it is unclear how they generalize to the distributed environment where communication is expensive. Thus the `dist_cv` algorithm uses a different approach.

Many other fast cross validation procedures have been developed for specific models. Arlot et al. (2010) provide a survey of results, describing fast cross validation methods for ridge regression, kernel density estimation, and nearest neighbor classification. Our fast cross validation framework includes each of these models as a special case, and many more besides. Joulani et al. (2015) develop a cross validation framework that is closely related to our own. Their framework is suitable for any incremental learning algorithm (whereas ours is suitable for mergable algorithms). All previous work on fast cross validation only considered the non-distributed case, but we have shown that our fast cross validation framework is suitable for the distributed case.

Both the bootstrap and cross validation are strictly better than the validation set method of error estimation, but standard no-free-lunch results (e.g. Chapter 8 of Shalev-Shwartz and Ben-David, 2014) ensure that there is no universally best method for estimating an estimator's true error for all distributions. On real world datasets, however, k-fold cross validation seems to work better and is the method of choice. Kohavi (1995) empirically compare the bootstrap to k-fold cross validation, and determine that on real world datasets k-fold cross validation has the best performance. When computational requirements are ignored, it is tempting to make k as large as possible. One of the limitations of the `dist_cv` method is that the number of folds must be the same as the number of processors. This will be relatively small compared to the number of data points. Conveniently, this turns out to be a good number of folds to use in practice. Rao et al. (2008) shows that for very small k values, increasing k improves the estimation of the model's accuracy; but as k increases beyond a certain threshold, the error of cross validation increases as well. Thus the `dist_cv`

method is both computationally cheap and statistically effective for real world data.

2.4 Example mergeable algorithms

Many proposed distributed algorithms fit our mergeable framework and therefore get a fast cross validation algorithm "for free." This section reviews this previous work. Section 2.4.1 discusses algorithms with closed form solution; Section 2.4.2 discusses approximate merge procedures for loss minimization problems; and Section 2.4.3 discusses approximate merge procedures for Bayesian learning problems. In total, we discuss 32 algorithms. None of the papers that originally presented these algorithms has fast cross validation procedures,[6] but our fast cross validation procedure is applicable to all these algorithms.

2.4.1 Estimators with closed form solutions

We now show that many popular machine learning algorithms are mergeable. We begin by looking at models that are exactly mergeable. That is, given data sets $Z_1, ..., Z_m$,

$$\texttt{reduce}(\{\texttt{alg}(Z_i)\}_{i=1}^m) = \texttt{alg}(\cup_{i=1}^m Z_i). \tag{2.14}$$

Notice that (2.14) is the same as (2.7) which defined the invariant that `reduce` should obey approximate equality, except the $\approx$ symbol has been replaced by $=$. In this section, we show that the RLM estimator for linear exponential families, Bayesian classifers, and ridge regression all have exact merge functions. A key feature of each of these models is that the RLM estimator has a closed form solution. This closed form solution is what allows

[6] For some of these algorithms, fast cross validation procedures do exist, but they were developed in an ad-hoc manner independent of the mergeable distributed algorithm.

the development of an exact merge function. We also show that many other estimators have closed form solutions not based on the RLM, and these estimators are also exactly mergeable.

2.4.1.1 Exponential family distributions

Exponential family distributions have nice computational properties and are used frequently in practice. Many popular distributions (e.g. Gaussian, Dirichlet, Poisson, exponential, and categorical) are in the exponential family. In this section we will see that a particularly nice subclass of the exponential family called the linear exponential family is exactly mergeable, but that other subclasses called curved exponential families and stratified exponential families are not. Many of the merge procedures in later sections will depend on the mergeability of the linear exponential family.

The **exponential family** (EF) of distributions is defined to be the set $\mathcal{P}^{exp}$ of all distributions whose density can be written in the form

$$p\left(\mathbf{z}|\mathbf{w}\right) = h(\mathbf{z}) \exp\left(\mathbf{w}^{\mathsf{T}} T(\mathbf{z}) - \psi(\mathbf{w})\right) \tag{2.15}$$

where $\mathbf{z} \in \mathcal{Z}$ and $\mathbf{w} \in \mathcal{W}$. We make no assumption on the set $\mathcal{Z}$, but require $\mathcal{W}$ to be an open subset of a Hilbert space $\mathcal{H}$. We will ignore the technical details of infinite dimensional Hilbert spaces and denote the inner product using $^{\mathsf{T}}$ in analogy with finite dimensional vectors. The function $h : \mathcal{Z} \to \mathbb{R}$ is called the **base measure**, $T : Z \to \mathcal{W}$ the **sufficient statistic**, and $\psi : \mathcal{W} \to \mathbb{R}$ the **log partition function**. Many authors require ψ to be strongly convex, but we will require only the weaker condition that the derivative

be invertible.

The standard method to estimate parameters in the exponential family is with **maximum likelihood estimation** (MLE). MLE is equivalent to RLM with the regularization strength $\lambda = 0$ and loss equal to the negative log likelihood. That is,

$$\ell(\mathbf{z}; \mathbf{w}) = -\log p\left(\mathbf{z}|\mathbf{w}\right) = -\log h(\mathbf{z}) - \mathbf{w}^{\mathsf{T}} T(\mathbf{z}) + \psi(\mathbf{w}), \tag{2.16}$$

and the parameter estimate is given by

$$\hat{\mathbf{w}}^{exp} = \underset{\mathbf{w}\in\mathcal{W}}{\arg\min} \sum_{\mathbf{z}\in Z} \Big(\psi(\mathbf{w}) - \mathbf{w}^{\mathsf{T}} T(\mathbf{z})\Big). \tag{2.17}$$

The $-\log h(\mathbf{z})$ term in (2.16) does not depend on the parameter $\mathbf{w}$ and so does not appear in the optimization (2.17).

In the special case when $\mathcal{W} = \mathcal{H} = \mathbb{R}^d$, the distribution is said to be in the **linear exponential family** (LEF).[7] Let Z be a dataset with mn elements. Then the parameter estimate (2.17) has closed form solution

$$\hat{\mathbf{w}}^{exp} = \psi'^{-1}\left(\frac{1}{mn}\sum_{\mathbf{z}\in Z} T(\mathbf{z})\right), \tag{2.18}$$

which follows from setting the derivative of the objective in (2.17) to zero and solving for $\mathbf{w}$. Because the equation for $\hat{\mathbf{w}}^{exp}$ can be given in closed form, an exact merge procedure exists. Decompose the data set Z into m data sets $Z_1, ..., Z_m$ each of size n. On each local

[7] The LEF is sometimes called the **full exponential family** (FEF) when the parameters are identifiable.

data set, the parameter estimate is defined to be

$$\hat{\mathbf{w}}_i^{exp} = \psi'^{-1}\left(\frac{1}{n}\sum_{\mathbf{z}\in Z_i} T(\mathbf{z})\right) \tag{2.19}$$

and the merge procedure is then

$$\texttt{reduce}(\{\hat{\mathbf{w}}_i^{exp}\}_{i=1}^m) = \psi'^{-1}\left(\frac{1}{m}\sum_{i=1}^m \psi'\left(\hat{\mathbf{w}}_i^{rlm}\right)\right). \tag{2.20}$$

Substituting (2.19) into (2.20) gives the standard MLE (2.18). That is, the merge function is exact. For the linear exponential family, we can therefore parallelize training and perform fast cross validation.

Other important subfamilies of the exponential family do not have exact merge procedures. When the parameter space $\mathcal{W}$ is a manifold in the underlying Hilbert space $\mathcal{H}$, we say the distribution is in the **curved exponential family** (CEF). The CEF was introduced by Efron (1975) and Amari (1982). Amari (2016) provides a modern treatment of the CEF from the perspective of information geometry. The CEF is used to represent dependency relationships between random variables, and a particularly important subset of the CEF is the set of undirected graphical models with no hidden variables. Liu and Ihler (2012) prove that no exact merge procedure exists for CEF distributions.

When the parameter space $\mathcal{W}$ is an algebraic variety (i.e. the set of roots of low dimensional polynomials embedded in high dimensional space.), then the distribution is in the **stratified exponential family** (SEF). Geiger and Meek (1998) and Geiger et al. (2001) introduced SEFs and show that the SEF is equivalent to the set of directed or undirected

graphical models with hidden variables. They further show that

$$\text{LEF} \subset \text{CEF} \subset \text{SEF} \subset \text{EF}, \tag{2.21}$$

where the set inequalities in (2.21) are strict. An immediate consequence of (2.21) is that exact merge procedures do not exist for general SEF distributions since they do not exist for CEF distributions. The **deep exponential family** (DEF) of distributions was proposed by Ranganath et al. (2015) and is a subset of the SEF. A distribution is in the DEF if it is the distribution resulting from placing EF priors on the natural parameters of an EF distribution. Although exact merge procedures do not exist for the CEF, SEF, or DEF, approximate merge procedures can be used. See Section 2.4.2 for examples of approximate merge procedures.

2.4.1.2 Naive Bayes

Generative classifiers are some of the oldest and simplest machine learning algorithms. They model the data as a probability distribution, and parameter estimation of the classifier is just parameter estimation of the underlying distribution. Whenever the distribution has a mergeable learning algorithm, then the generative classifier does as well. Fast cross validation methods for generative classifiers seem to be "folk knowledge" in the statistical community, but we know of no publications earlier than Izbicki (2013) that describe a fast cross validation method.

Formally, in **classification** we assume that the data space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$, where $\mathcal{X}$ is the space of features and $\mathcal{Y}$ is a finite set of class labels. The goal of classification is to

learn an unknown function $\hat{y} : \mathcal{X} \to \mathcal{Y}$. In a **generative classifier**, the function $\hat{y}$ has the form

$$\hat{y}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} p(y|\mathbf{x}) \tag{2.22}$$

where $p(y|\mathbf{x})$ is a distribution from a known family with unknown parameters. By Bayes theorem, we have that

$$p(y|\mathbf{x}) = \frac{p(y)p(\mathbf{x}|y)}{p(\mathbf{x})}, \tag{2.23}$$

and so (2.22) can be rewritten as

$$\hat{y}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} p(y)p(\mathbf{x}|y). \tag{2.24}$$

The $1/p(\mathbf{x})$ factor is dropped as it does not affect the optimization. To learn a generative classifier, we need to learn the parameters of the distributions $p(y)$ and $p(\mathbf{x}|y)$. Whenever these distributions have mergeable learning algorithms, then the generative classifier does as well.

The **naive Bayes** model is an important special case of a generative classifier that makes the following simplifying assumptions: the feature space $\mathcal{X} = \mathbb{R}^d$, and each feature $\mathbf{x}^{(i)}$ is independent given y. In notation,

$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(\mathbf{x}^{(i)}|y). \tag{2.25}$$

Substituting into (2.24) gives the naive Bayes classification rule

$$\hat{y}(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} p(y) \prod_{i=1}^{d} p(\mathbf{x}^{(i)}|y). \tag{2.26}$$

The distribution $p(y)$ is typically assumed to be the categorical distribution, and the distributions $p(\mathbf{x}^{(i)}|y)$ are typically assumed to be univariate categorical distributions for discrete data or univariate normal distributions for continuous data. All of these distributions are in the linear exponential family, so naive Bayes has an exact mergeable learning algorithm.

2.4.2 Approximate regularized loss minimization

The **regularized loss minimizer** (RLM) is a large and popular family of learning algorithms. It is defined to be

$$\hat{\mathbf{w}}^{rlm} = \arg\min_{\mathbf{w} \in \mathcal{W}} \mathcal{L}(Z; \mathbf{w}) + \lambda r(\mathbf{w}) \tag{2.27}$$

where λ is a hyperparameter and $r : \mathcal{W} \to \mathbb{R}$ a regularization function. In the common case where the data Z is drawn i.i.d. from $\mathcal{Z}$, the loss function can be decomposed as

$$\mathcal{L}(Z; \mathbf{w}) = \sum_{\mathbf{z} \in Z} \ell(\mathbf{z}; \mathbf{w}) \tag{2.28}$$

where $\ell : \mathcal{Z} \to \mathcal{W}$ is also called the loss function. The RLM estimator is then equivalently written as

$$\hat{\mathbf{w}}^{rlm} = \arg\min_{\mathbf{w} \in \mathcal{W}} \sum_{\mathbf{z} \in \mathcal{Z}} \ell(\mathbf{z}; \mathbf{w}) + \lambda r(\mathbf{w}). \tag{2.29}$$

Ridge regression (Section 2.1) is an example RLM where

$$\ell(\mathbf{x}, y; \mathbf{w}) = \|y - \mathbf{w}^\mathsf{T}\mathbf{x}\|^2 \qquad \text{and} \qquad r(\mathbf{w}) = \|\mathbf{w}\|^2. \tag{2.30}$$

Unlike the ridge regression case, most RLM problems have no closed form solution. Therefore, no exact merging procedure exists, but a large body of work has formed that develops approximate merge procedures. Before presenting these merge procedures, we begin with a discussion about when approximate merge procedures are appropriate.

2.4.2.1 When are approximate merge procedures appropriate?

Approximate merge procedures are appropriate when the approximation has a small effect on an estimator's statistical error. When $\mathcal{W}$ is a normed space, we define the **statistical error** of an estimator f to be $\|\mathbf{w}^* - \hat{\mathbf{w}}^f\|$ where $\mathbf{w}^*$ is called the **true parameter** and defined to be

$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathcal{W}} \mathop{\mathbb{E}}_{\mathbf{z} \sim P_\mathcal{Z}} \mathcal{L}(\mathbf{z}; \mathbf{w}). \tag{2.31}$$

We assume for notational convenience that $\mathbf{w}^*$ is unique. A low statistical error ensures that the estimator has low loss for any reasonable loss function.

Standard results on the RLM estimator show that if the size of the training set Z is mn, then $\|\mathbf{w}^* - \hat{\mathbf{w}}^f\| \leq O(1/\sqrt{mn})$ under mild conditions on f (e.g. Lehmann, 1999). This rate of convergence is known to be optimal in the sense that no estimator can have faster convergence. But alternative estimators can also have this optimal convergence rate. Therefore, the goal when designing an approximate merge method is to design an algorithm that matches this convergence rate.

Exact merge procedures for RLM problems automatically inherit the $O(1/\sqrt{mn})$ convergence rate. If an approximate merge procedure also has an $O(1/\sqrt{mn})$ convergence rate, then it is just as good as the standard RLM procedure from a statistical perspective. We shall see that existing merge procedures are only able to achieve this rate under special conditions. In the Chapter 3, we present a novel algorithm called the optimal weighted average that achieves this optimal convergence rate in the most general setting. We divide existing approximate merge methods into three categories. Section 2.4.2.2 presents merge procedures based on averaging. Section 2.4.2.3 presents merge procedures specialized for principle component analysis (PCA), and Section 2.4.2.4 presents merge procedures for submodular losses. In Chapter 3, we present a more general merge procedure with better statistical and computational properties than existing methods.

2.4.2.2 Averaging based methods

The averaging based merge procedures are the simplest. Most of these procedures require only that the parameter space $\mathcal{W}$ have vector space structure. This is common in many problems, the standard example being logistic regression. This section presents 9 published merge methods (Merugu and Ghosh, 2003; McDonald et al., 2009; Zinkevich et al., 2010; Zhang et al., 2012, 2013b; Liu and Ihler, 2014; Battey et al., 2015; Han and Liu, 2016; Jordan et al., 2016; Lee et al., 2017). There is no previously published fast cross validation method for these learning algorithms, but the method presented in Section 2.3 can be applied to all of them. A common theme of these merge procedures is that they do not depend on the data. In the next chapter, we present a novel improvement to these estimators that does depend on the data and so has better approximation guarantees.

The simplest and most popular mergeable estimator is the **naive averaging estimator**. The `map` procedure is given by

$$\hat{\mathbf{w}}_i^{rlm} = \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{\mathbf{z}\in\mathcal{Z}_i} \ell(\mathbf{z};\mathbf{w}) + \lambda r(\mathbf{w}) \tag{2.32}$$

where $\hat{\mathbf{w}}_i^{rlm}$ is called the **local regularized risk minimizer** on machine i, and the `reduce` procedure is given by

$$\hat{\mathbf{w}}^{ave} = \frac{1}{m}\sum_{i=1}^{m} \hat{\mathbf{w}}_i^{rlm}. \tag{2.33}$$

Naive averaging was first studied by McDonald et al. (2009) for the case L_2 regularized maximum entropy models. They use the triangle inequality to decompose the statistical error as

$$\|\mathbf{w}^* - \hat{\mathbf{w}}^{ave}\| \leq \|\mathbf{w}^* - \mathbb{E}\,\hat{\mathbf{w}}^{ave}\| + \|\mathbb{E}\,\hat{\mathbf{w}}^{ave} - \hat{\mathbf{w}}^{ave}\|. \tag{2.34}$$

We call the $\|\mathbf{w}^* - \mathbb{E}\,\hat{\mathbf{w}}^{ave}\|$ the **bias** of $\hat{\mathbf{w}}^{ave}$ and the $\|\mathbb{E}\,\hat{\mathbf{w}}^{ave} - \hat{\mathbf{w}}^{ave}\|$ term the **variance**[8] of $\hat{\mathbf{w}}$. They provide concentration inequalities for the estimation error, showing that the variance reduces at the optimal rate of $O((mn)^{-1/2})$, but that the bias reduces only as $O(n^{-1/2})$. Therefore naive averaging works well on unbiased models, but poorly on models with large bias. Subsequent work has focused on merge methods that improve both the bias and the variance.

Under more stringent assumptions, it can be shown that the bias of $\hat{\mathbf{w}}^{ave}$ will reduce at a faster rate. In particular, when the bias of $\hat{\mathbf{w}}_i^{rlm}$ shrinks faster than $O(n^{1/2})$, then naive averaging can be an effective merge procedure. Zhang et al. (2012) show that the

[8] It is also common to call the term $\|\mathbb{E}\,\hat{\mathbf{w}} - \hat{\mathbf{w}}\|^2$ the **variance**, in which case $\|\mathbb{E}\,\hat{\mathbf{w}} - \hat{\mathbf{w}}\|$ could be called the **deviation**.

mean squared error (MSE) $\mathbb{E}\,\|\mathbf{w}^* - \hat{\mathbf{w}}^{ave}\|^2$ decays as $O((mn)^{-1}+n^{-2})$. This matches the optimal MSE of $\hat{\mathbf{w}}^{rlm}$ whenever $m < n$. Their analysis also requires limiting assumptions. For example, they assume the parameter space $\mathcal{W}$ is bounded. This assumption does not hold under the standard Bayesian interpretation of L2 regularization as a Gaussian prior of the parameter space. They further make strong convexity and $8th$ order smoothness assumptions which guarantee that $\hat{\mathbf{w}}_i^{rlm}$ is a "nearly unbiased estimator" of $\mathbf{w}^*$. Most recently, Rosenblatt and Nadler (2016) analyze $\hat{\mathbf{w}}^{ave}$ in the asymptotic regime as the number of data points $n \to \infty$. This analysis is more general than previous analyses, but it does not hold in the finite sample regime. Zinkevich et al. (2010) show that if the local training sets Z_i partially overlap each other (instead of being disjoint), then the resulting estimator will have lower bias.

Zhang et al. (2013b) show how to reduce bias in the special case of kernel ridge regression. By a careful choice of regularization parameter λ, they cause $\hat{\mathbf{w}}_i^{rlm}$ to have lower bias but higher variance, so that the final estimate of $\hat{\mathbf{w}}^{ave}$ has both reduced bias and variance. This suggests that a merging procedure that reduces bias is not crucial to good performance if we set the regularization parameter correctly. Typically there is a narrow range of good regularization parameters, and finding a λ in this range is expensive computationally. Our `dist_cv` method in Algorithm 5 can be used to more quickly tune this parameter.

Battey et al. (2015) and Lee et al. (2017) independently develop methods to reduce the bias for the special case of lasso regression.[9] In the lasso, the loss function is the squared

[9] Both Battey et al. (2015) and Lee et al. (2017) originally appeared on arXiv at the same time in 2015, but only Lee et al. (2017) has been officially published.

loss (as in ridge regression), but the regularization function is the L_1 norm (instead of the squared L_2 norm). The idea is to use an estimator with low bias and high variance to train a model on each local data set Z_i. Then, when the results are averaged, the bias will remain low, and the high variance will be reduced. They propose using the following **debiased lasso estimator** (DLE) on each dataset:

$$\hat{\mathbf{w}}_i^{dle} = \hat{\mathbf{w}}_i^{rlm} + \frac{1}{n}\hat{\Theta}X^\mathsf{T}(y - X\hat{\mathbf{w}}_i^{rlm}), \tag{2.35}$$

where $\hat{\Theta}$ is an approximate inverse of the empirical covariance $\hat{\Sigma} = X^\mathsf{T}X$. The resulting **debiased average** (DAVE) estimator is

$$\hat{\mathbf{w}}^{dave} = \frac{1}{m}\sum_{i=1}^{m}\hat{\mathbf{w}}^{dle}. \tag{2.36}$$

The bias and variance of this estimator both shrink at the optimal rate of $O((nm)^{-1/2}$.

Zhang et al. (2012) provide a debiasing technique that works for any estimator. It works as follows. Let $r \in (0, 1)$, and Z_i^r be a bootstrap sample of Z_i of size rn. Then the **bootstrap average estimator** is

$$\hat{\mathbf{w}}^{boot} = \frac{\hat{\mathbf{w}}^{ave} - r\hat{\mathbf{w}}^{ave,r}}{1-r}, \qquad \text{where} \qquad \hat{\mathbf{w}}^{ave,r} = \frac{1}{m}\sum_{i=1}^{m}\arg\max_{\mathbf{w}}\sum_{(\mathbf{x},y)\in Z_i^r}\ell(y, \mathbf{x}^\mathsf{T}\mathbf{w}) + \lambda r(\mathbf{w}).$$

The intuition behind this estimator is to use the bootstrap sample to directly estimate and correct for the bias. When the loss function is convex, $\hat{\mathbf{w}}^{boot}$ enjoys a mean squared error (MSE) that decays as $O((mn)^{-1} + n^{-3})$. There are two additional limitations to $\hat{\mathbf{w}}^{boot}$. First, the optimal value of r is not obvious and setting the parameter requires cross

validation on the entire data set. So properly tuning λ_2 is more efficient than r. Second, performing a bootstrap on an unbiased estimator increases the variance. This means that $\hat{\mathbf{w}}^{boot}$ could perform worse than $\hat{\mathbf{w}}^{ave}$ on unbiased estimators.

Jordan et al. (2016) propose to reduce bias by incoporating second order information into the average. In particular, they develop an approach that uses a single approximate Newton step in the merge procedure. As long as the initial starting point (they suggest using $\hat{\mathbf{w}}^{ave}$) is within $O(\sqrt{1/n})$ of the true parameter vector, then this approach converges at the optimal rate. When implementing Jordan et al.'s approach, we found it suffered from two practical difficulties. First, Newton steps can diverge if the starting point is not close enough. We found in our experiments that $\hat{\mathbf{w}}^{ave}$ was not always close enough. Second, Newton steps require inverting a Hessian matrix. In the experiments of Chapter 3, we consider a problem with dimension $d \approx 7 \times 10^5$; the corresponding Hessian is too large to practically invert.

Liu and Ihler (2014) propose a more Bayesian approach inspired by Merugu and Ghosh (2003). Instead of averaging the model's parameters, they directly "average the models" with the following KL-average estimator:

$$\hat{\mathbf{w}}^{kl} = \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{i=1}^{m} \mathrm{KL}\left(p(\cdot; \hat{\mathbf{w}}_i^{rlm}) \,\middle\|\, p(\cdot; \mathbf{w}) \right). \tag{2.37}$$

Liu and Ihler show theoretically that this is the best merge function in the class of functions that do not depend on the data. The main disadvantage of KL-averaging is computational. The minimization in (2.37) is performed via a bootstrap sample from the local models, which is computationally expensive. This method has three main advantages. First, it

is robust to reparameterizations of the model. Second, it is statistically optimal for the class of non-interactive algorithms. Third, this method is general enough to work for any model, even if the parameter space $\mathcal{W}$ is not a vector space. The main downside of the KL-average is that the minimization has a prohibitively high computational cost. Let n^{kl} be the size of the bootstrap sample. Then Liu and Ihler's method has MSE that shrinks as $O((mn)^{-1} + (nn^{kl})^{-1})$. This implies that the bootstrap procedure requires as many samples as the original problem for a single machine to get a MSE that shrinks at the same rate as the averaging estimator. Han and Liu (2016) provide a method to reduce the MSE to $O((mn)^{-1} + (n^2n^{kl})^{-1})$ using control variates, but the procedure remains prohibitively expensive. Their experiments show the procedure scaling only to datasets of size $mn \approx 10^4$, whereas our experiments involve a dataset of size $mn \approx 10^8$.

2.4.2.3 Principle component analysis

Principle component analysis (PCA) is a popular technique for dimensionality reduction. In this section, we describe how PCA works on a single machine, show that two existing techniques for distributed PCA are mergeable (Qu et al., 2002; Liang et al., 2013), and describe the importance of fast cross validation techniques for PCA.

Let Z denote the $mn \times d$ input data matrix. The goal of PCA is to find a $d \times k$ matrix with $k \ll d$ such that the distance between Z and $Z\mathbf{w}\mathbf{w}^\dagger$ is minimized. Formally,

$$\hat{\mathbf{w}}^{pca} = \arg\min_{\mathbf{w} \in \mathbb{R}^{d \times k}} \|Z - Z\mathbf{w}\mathbf{w}^\dagger\|^2. \tag{2.38}$$

The optimization (2.38) is non-convex, but the solution can be calculated efficiently via the

singular value decomposition (SVD). The SVD of Z is

$$Z = UDV^{\mathsf{T}}, \tag{2.39}$$

where U is an orthogonal matrix of dimension $mn \times mn$, D is a diagonal matrix of dimension $mn \times d$, and V is an orthogonal matrix dimension $d \times d$. The columns of U are called the left singular vectors, the columns of V the right singular vectors, and the entries in D the singular values. The solution to (2.38) is given by the first k columns of V. Halko et al. (2011) provides a method for efficiently calculating approximate SVDs on a single machine when only the first k singular values/vectors are needed.

Qu et al. (2002) and Liang et al. (2013) introduce essentially the same algorithm for distributed PCA. Each machine i locally calculates the SVD of its local dataset $Z_i = U_i D_i V_i^{\mathsf{T}}$. Let $D_i^{(k)}$ denote the $k \times k$ submatrix of D_i containing the first k singular values, and $V_i^{(k)}$ denote the $d \times k$ submatrix of V_i containing the first k columns. The local machines each transmit $D_i^{(k)}$ and $V_i^{(k)}$ to the master machine. The master calculates

$$S = \sum_{i=1}^{m} V_i^{(k)} D_i^{(k)} {V_i^{(k)}}^{\mathsf{T}} \tag{2.40}$$

Performing the SVD on S then gives the approximate principle components of the entire data set Z. Qu et al. (2002) further provide a modification to the merge procedure that approximately centers the data, but they do not provide any theoretical guarantees on the performance of their algorithm relative to the single machine oracle. Liang et al. (2013) do not consider the possibility of centering the data, but they do show that their algorithm is

a $1 + \varepsilon$ approximation of the single machine algorithm, where ε depends on properties of the data and the choice of k.

A major difficulty in PCA (distributed or not) is selecting a good value for k. There are two reasons to choose a small value of k. The most obvious is computational. When k is small, future stages in the data processing pipeline will be more efficient because they are working in a lower dimensional space. But there is a more subtle statistical reason. When there is a large noise component in the data, using fewer dimensions removes this noise and improves the statistical efficiency of later stages of the data pipeline. Perhaps the simplest method of determining k is the scree test (Cattell, 1966), where the data's singular values are plotted and the analyst makes a subjective judgement. More robust methods make distributional assumptions (Bartlett, 1950). Under these assumptions, the noise in the data can be estimated directly and k determined appropriately. When these distributional assumption do not hold, however, the resulting k value can be arbitrarily poor. The most robust solution uses cross validation and the PRESS statistic. Unfortunately, this is also the most expensive technique computationally. Considerable work has been done to improve both the theoretical guarantees of cross validation and improve its runtime via approximations (Wold, 1978; Eastment and Krzanowski, 1982; Krzanowski, 1987; Mertens et al., 1995; Diana and Tommasi, 2002; Engelen and Hubert, 2004; Josse and Husson, 2012; Camacho and Ferrer, 2012). Notably, none of these fast cross validation techniques work in the distributed setting. Thus the distributed fast cross validation technique due to the mergeability of Qu et al. (2002) and Liang et al. (2013) is both novel and useful.

2.4.2.4 Submodular optimization

Submodular functions are a class of set function that share many important properties with convex functions (Lovász, 1983). In particular, they are easy to optimize and have many applications. The maximization of submodular functions has become an important technique in the approximation of NP-hard problems (Krause and Golovin, 2014). Applications in machine learning include clustering, sparse nonparametric regression, image segmentation, document summarization, and social network modeling (see references within Mirzasoleiman et al., 2016). The last five years has seen work focused on scaling up submodular optimization via distributed algorithms. Six of these methods create mergeable estimators that fit our framework (Mirzasoleiman et al., 2013; Barbosa et al., 2015; Malkomes et al., 2015; Bhaskara et al., 2016; Barbosa et al., 2016; Mirzasoleiman et al., 2016) and so induce fast cross validation procedures. This is the first work addressing fast cross validation in submodular learning algorithms. In this section we first define submodularity, then introduce the distributed optimizers.

For a set function $f : \{\mathcal{Z}\} \to \mathbb{R}$, a set $Z \subset \mathcal{Z}$ and an element $e \in \mathcal{Z}$, we define the **discrete derivative** of f at Z with respect to e to be

$$f'(e; Z) = f(Z \cup \{e\}) - f(Z). \tag{2.41}$$

We call the function f **monotone** if for all e and Z, $f'(e; Z) \geq 0$. We further call f **submodular** if for all $A \subseteq B \subseteq Z$ and $e \in Z - B$,

$$f'(e; A) \geq f'(e; B). \tag{2.42}$$

Algorithm 6 `greedy`(data set Z, constraint size k)

1: $S \leftarrow \{\}$
2: **for** $i = 1$ **to** k **do**
3: $\quad \mathbf{z}_i \leftarrow \arg\max_{\mathbf{z} \in Z - S} f'(\mathbf{z}; S)$
4: $\quad S \leftarrow S \cup \{\mathbf{z}_i\}$
5: **end for**
6: **return** S

We first consider the problem of monotone submodular maximization subject to cardinality constraints. That is, we want to solve

$$\hat{S} = \arg\max_{S \subseteq Z} f(S) \qquad \text{s.t.} \qquad |S| \leq k. \tag{2.43}$$

Solving (2.43) is NP-hard in general, so it is standard to use the greedy approximation algorithm introduced by Nemhauser et al. (1978). The procedure (which we will refer to as `greedy`) is shown in Algorithm 6. The `greedy` algorithm is known to be a $1 - 1/e$ approximation algorithm[10], and no better approximation algorithm exists unless $P = NP$ (Krause and Golovin, 2014).

Mirzasoleiman et al. (2013) introduced the first method of merging independently computed solutions in their `GreeDi` (GREEdy DIstributed) algorithm. `GreeDi` works by first running the `greedy` algorithm locally on each node to compute local solutions $\hat{\mathbf{w}}_i^{greedy}$. These solutions are transmitted to the master machine. The master combines the local solutions into a set of size km and reruns `greedy` on the combined set. In notation, the

[10] The symbol e is Euler's constant and not an approximation variable.

merge procedure is given by

$$\texttt{GreeDi}(\hat{\mathbf{w}}_1^{greedy}, ..., \hat{\mathbf{w}}_m^{greedy}) = \texttt{greedy}(\cup_{i=1}^{m} \hat{\mathbf{w}}_i^{greedy}, k). \tag{2.44}$$

Mirzasoleiman et al. (2013) show that in the worst case, `GreeDi` achieves an approximation guarantee of

$$f(\hat{\mathbf{w}}^{GreeDi}) \geq \frac{(1-1/e)^2}{\min\{m,k\}} f(\mathbf{w}^*). \tag{2.45}$$

Barbosa et al. (2015) improve the analysis of `GreeDi` to show that in expectation

$$f(\hat{\mathbf{w}}^{GreeDi}) \geq \frac{1-1/e}{2} f(\mathbf{w}^*), \tag{2.46}$$

which matches the guarantee of the optimal `greedy` centralized algorithm up to the $1/2$ constant factor. Notably, the approximation is independent of the number of machines m or the size of the problem k.

Subsequent work has extended the `GreeDi` framework to apply to more general submodular optimization problems. Malkomes et al. (2015) solves the k-centers clustering problem; Bhaskara et al. (2016) solves the column subset selection problem; and both Barbosa et al. (2016) and Mirzasoleiman et al. (2016) solve submodular problems with matroid, p-system, and knapsack constraints. The algorithms presented in each of these papers follows the same basic pattern: the `greedy` algorithm used by the local machines and the merge procedure is replaced by an alternative algorithm that is more appropriate for the new problem setting.

As with continuous statistical optimization problems, the difficulty of submodular

optimization is known to depend on the curvature of the problem (Vondrák, 2010). The **submodular curvature** is defined to be

$$c = 1 - \min_{e \in Z} \frac{f'(e; Z - e)}{f(e)}. \tag{2.47}$$

When the curvature is small, the `greedy` algorithm will perform better. In particular, when the $c = 0$ the problem is said to be **modular** and the `greedy` algorithm returns an optimal solution. Vondrák (2010) shows that for all $c > 0$, `greedy` returns a $(1 - e^{-c})/c$ approximate solution, and that no better approximation is possible. There is as yet no work discussing the relationship of curvature to the difficulty of merging local solutions. It seems likely, however, that a bound analogous to the continuous bound provided by Liu and Ihler (2014) will hold.

2.4.3 Bayesian methods

In Bayesian inference, we treat the data set Z as observed variables and assume there is a hidden variable θ on which the data depends. For ease of notation, we will assume that $\theta \in \mathbb{R}^d$ and the data set Z contains mn i.i.d. data points. That is, the distribution of Z is $p(Z|\theta) = \prod_{i=1}^{mn} p(\mathbf{z}_i|\theta)$.[11] Our goal is to calculate the **posterior distribution** $p(\theta|Z)$. In this section we discuss three general techniques for learning the posterior. We warm-up with the Bernstein-von Mises (BvM) method. BvM uses a simple model based on a parametric approximation. The method of merging linear exponential families from Section 2.4.1.1 is then used to merge the posteriors. Next we discuss variational inference

[11] In general these conditions can be relaxed at the expense of more complicated notation.

(VI). VI also makes a parametric approximation, but uses a more complex optimization procedure to choose the parameters. We shall see that the merge procedures for VI are closely related to those for RLM, and in particular that all the merge procedures for RLM of Section 2.4.2 can be directly applied to the VI problem. Previous work has not noticed this connection. The final method is Markov chain Monte Carlo (MCMC). MCMC uses sampling to approximate the posterior. The merge procedures for MCMC share little in common with merge procedures for any other problem.

2.4.3.1 Bernstein-von Mises

When the posterior distribution $p(\theta|Z_i)$ has a parametric form in the linear exponential family, then the local posteriors can easily be combined (see Section 2.4.1.1). Under mild conditions, the **Bernstein-von Mises theorem**[12] states that as the number of samples $n \to \infty$, the distribution $p(\theta|Z_i)$ converges to a normal distribution. See for example Chapter 10.2 of van der Vaart (1998) for a formal statement of the theorem with conditions. Neiswanger et al. (2014) use this result to create a simple distributed learning procedure that can be thought of as the Bayesian version of naive parameter averaging (see Section 2.4.2.2).

The full procedure is as follows. By Bayes theorem, we have that the posterior can be written as

$$p(\theta|Z) = \frac{p(Z|\theta)p(\theta)}{p(Z)} \tag{2.48}$$

where $p(\theta)$ is the prior distribution over the parameters θ. Neiswanger et al. (2014) define

[12] The Bernstein-von Mises theorem is also often called the Bayesian central limit theorem.

the **subposterior** distribution for a dataset Z_i to be

$$p(\theta|Z_i) = \frac{p(Z|\theta)p(\theta)^{1/m}}{p(Z_i)}. \tag{2.49}$$

Notice the subposterior uses the underweighted prior $p(\theta)^{1/m}$. Some merge procedures use this underweighted prior on the subposterior, and some use the unmodified prior. Here, our choice of the underweighted prior means that we can rewrite the full posterior as

$$p(\theta|Z) \propto p(\theta)p(Z|\theta) = p(\theta)\prod_{i=1}^{m} p(Z_i|\theta) \propto p(\theta)\prod_{i=1}^{m} p(\theta|Z_i)p(\theta)^{-1/m} = \prod_{i=1}^{m} p(\theta|Z_i). \tag{2.50}$$

By the Berstein-von Mises theorem, we also have that the subposterior can be approximated as

$$p\left(\theta|Z_i\right) \approx \mathcal{N}(\theta; \hat{\mu}_i, \hat{\Sigma}_i) \tag{2.51}$$

where $\hat{\mu}_i$ and $\hat{\Sigma}_i$ are local mean and covariance parameter estimates computed by either variational inference (Section 2.4.3.2) or Markov chain Monte Carlo (Section 2.4.3.3), and $\mathcal{N}(\theta; \hat{\mu}_i, \hat{\Sigma}_i)$ is the density of the multivariate normal distribution with mean μ_i and covariance $\hat{\Sigma}_i$. Combining (2.50) and (2.51) gives our final parameter estimate

$$p\left(\theta|Z\right) \propto \prod_{i=1}^{m} p\left(\theta|Z_i\right) \approx \prod_{i=1}^{m} \mathcal{N}(\theta; \hat{\mu}_i, \hat{\Sigma}_i) = \mathcal{N}(\theta; \hat{\mu}, \hat{\Sigma}), \tag{2.52}$$

where

$$\hat{\Sigma} = \left(\sum_{i=1}^{m} \hat{\Sigma}_i^{-1}\right)^{-1} \qquad \text{and} \qquad \hat{\mu} = \hat{\Sigma}\left(\sum_{i=1}^{m} \hat{\Sigma}^{-1}\mu_i\right). \tag{2.53}$$

When the number of samples per machine n is much greater than the number of machines

m and the dimensionality of the problem d, then (2.53) provides a good estimate of the true posterior distribution. Like the naive averaging method of Section 2.4.2.2, however, (2.53) will be highly biased when n is small. Variational inference and Markov chain Monte Carlo typically provide much better results.

2.4.3.2 Variational inference

Variational inference (VI) is a popular method for approximating intractable posterior distributions (Jordan et al., 1999; Blei et al., 2017). There are three existing methods of distributed VI that fit our merging framework (Broderick et al., 2013; Campbell and How, 2014; Neiswanger et al., 2015). To describe these methods, we will present VI as a special case of regularized loss minimization (RLM). This is a non-standard presentation of VI, but it highlights the similarities between distributed VI methods and the distributed RLM methods already presented. In particular, the 9 mergeable RLM estimators of Section 2.4.2.2 can be applied directly to the VI problem. These estimators have several advantages over the VI-specific merging procedures, including explicit regret bounds. Each of these merge procedures induces a fast cross validation procedure (by Section 2.3), and we believe these are the first published fast cross validation procedures for VI.

Variational inference uses optimization to create a deterministic approximation to the posterior that is easy to compute. The first step is to select a surrogate family of densities $Q = \{q(\theta|\mathbf{w}) : \mathbf{w} \in \mathcal{W}\}$, then the optimal density $q(\theta|\hat{\mathbf{w}}^{vi})$ is given by solving the optimization

$$\hat{\mathbf{w}}^{vi} = \arg\min_{\mathbf{w}\in\mathcal{W}} \mathrm{KL}(q(\theta|\mathbf{w}) \parallel p(\theta|Z)). \tag{2.54}$$

Solving (2.54) is equivalent to the RLM problem

$$\hat{\mathbf{w}}^{vi} = \arg\min_{\mathbf{w}\in\mathcal{W}} \sum_{\mathbf{z}\in Z} \ell(\mathbf{z};\mathbf{w}) + r(\mathbf{w}) \tag{2.55}$$

where

$$\ell(\mathbf{z};\mathbf{w}) = -\int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log p(\mathbf{z}|\theta)\, \mathrm{d}\theta \qquad \text{and} \qquad r(\mathbf{w}) = \mathrm{KL}(q(\theta|\mathbf{w}) \parallel p(\theta)). \tag{2.56}$$

The loss ℓ above is commonly known as the negative cross entropy. Practitioners commonly choose the distributions p and q to be conjugate members of the exponential family, in which case ℓ and r will have closed form representations. The equivalence between (2.54) and (2.55) follows because

$$\mathrm{KL}(q(\theta|\mathbf{w}) \parallel p(\theta|Z)) \tag{2.57}$$

$$= \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log \frac{q(\theta|\mathbf{w})}{p(\theta|Z)}\, \mathrm{d}\theta \tag{2.58}$$

$$= \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log q(\theta|\mathbf{w})\, \mathrm{d}\theta - \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log p(\theta|Z)\, \mathrm{d}\theta \tag{2.59}$$

$$= \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log q(\theta|\mathbf{w})\, \mathrm{d}\theta - \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log \frac{p(Z|\theta)p(\theta)}{p(Z)}\, \mathrm{d}\theta \tag{2.60}$$

$$= \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log \frac{q(\theta|\mathbf{w})}{p(\theta)}\, \mathrm{d}\theta - \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log p(Z|\theta)\, \mathrm{d}\theta + \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log p(Z)\, \mathrm{d}\theta \tag{2.61}$$

$$= \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log \frac{q(\theta|\mathbf{w})}{p(\theta)}\, \mathrm{d}\theta - \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log p(Z|\theta)\, \mathrm{d}\theta + \log p(Z) \tag{2.62}$$

$$= \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log \frac{q(\theta|\mathbf{w})}{p(\theta)}\, \mathrm{d}\theta - \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log \prod_{\mathbf{z}\in Z} p(\mathbf{z}|\theta)\, \mathrm{d}\theta + \log p(Z) \tag{2.63}$$

$$= \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log \frac{q(\theta|\mathbf{w})}{p(\theta)}\, \mathrm{d}\theta - \sum_{\mathbf{z}\in Z} \int_{\theta\in\Theta} q(\theta|\mathbf{w}) \log p(\mathbf{z}|\theta)\, \mathrm{d}\theta + \log p(Z) \tag{2.64}$$

$$= r(\mathbf{w}) + \sum_{\mathbf{z} \in Z} \ell(\mathbf{z}; \mathbf{w}) + \log p(Z) \tag{2.65}$$

Finally, since the term $\log p(Z)$ does not depend on $\mathbf{w}$ it can be removed from the optimization in (2.55). Because of this reduction of VI to RLM, the 9 techniques for merging RLM estimators can be applied directly to the VI problem. No previous literature has noticed this connection.

We now present three methods for distributed VI that are not derived from the RLM problem. Broderick et al. (2013) proposed the first method, **streaming distributed asynchronous Bayes** (SDA-Bayes). SDA-Bayes is effectively a form of naive parameter averaging (see Section 2.4.2.2) that also updates the regularization. Each local machine calculates the variational approximation $q(\theta|\hat{\mathbf{w}}_i^{vi})$ locally, then the merged parameters are given by

$$\hat{\mathbf{w}}^{sda} = (1-m)\hat{\mathbf{w}}_0^{vi} + \sum_{i=1}^{m} \hat{\mathbf{w}}_i^{vi} \tag{2.66}$$

where $\hat{\mathbf{w}}_0^{vi}$ denotes the hyper parameters of the prior distribution $p(\theta)$, which is assumed to be in the same exponential family as the variational approximation $q(\theta|\hat{\mathbf{w}}_i^{vi})$.

The merge formula for SDA-Bayes has the following probabilistic justification. We can factor the global posterior into local posteriors and apply the exponential family assumption to get

$$p(\theta|Z) = p(\theta|Z_1, ..., Z_m) \tag{2.67}$$

$$\propto \left(\prod_{i=1}^{m} p(Z_i|\theta) \right) p(\theta) \tag{2.68}$$

$$\propto \left(\prod_{i=1}^{m} \frac{p(\theta|Z_i)}{p(\theta)}\right) p(\theta) \tag{2.69}$$

$$= p(\theta)^{1-m} \prod_{i=1}^{m} p(\theta|Z_i) \tag{2.70}$$

$$\approx p(\theta)^{1-m} \prod_{i=1}^{m} q(\theta|\hat{\mathbf{w}}_i^{vi}) \tag{2.71}$$

$$= h(\theta) \exp\left(\left((1-m)\hat{\mathbf{w}}_0^{vi} + \sum_{i=1}^{m} \hat{\mathbf{w}}_i^{vi}\right)^{\mathsf{T}} T(\theta) - (1-m)\psi(\hat{\mathbf{w}}_0^{vi}) - \sum_{i=1}^{m} \psi(\hat{\mathbf{w}}_i^{vi})\right) \tag{2.72}$$

$$\approx h(\theta) \exp\left(\left((1-m)\hat{\mathbf{w}}_0^{vi} + \sum_{i=1}^{m} \hat{\mathbf{w}}_i^{vi}\right)^{\mathsf{T}} T(\theta) - \psi\left((1-m)\hat{\mathbf{w}}_0^{vi} - \sum_{i=1}^{m} \hat{\mathbf{w}}_i^{vi}\right)\right) \tag{2.73}$$

$$= h(\theta) \exp\left(\hat{\mathbf{w}}^{sda\mathsf{T}} T(\theta) - \psi(\hat{\mathbf{w}}^{sda})\right) \tag{2.74}$$

$$= q(\theta|\hat{\mathbf{w}}^{sda}) \tag{2.75}$$

Notice that in (2.69), SDA-Bayes does not assume the underweighted prior in the subposterior $p(\theta|Z_i)$ as we assumed in the Bernstein-von Mises merge procedure. The approximation of line (2.71) is the variational approximation, and the approximation of line (2.73) assumes linearity of ψ. In general, ψ may be highly non-linear and so SDA-Bayes provides no guarantees on how well $\hat{\mathbf{w}}^{sda}$ approximates the single machine oracle VI parameters $\hat{\mathbf{w}}^{vi}$. A major advantage of using the RLM procedures instead is that they do come with these guarantees. For example, even the simple naive average guarantees optimal reduction in variance (Section 2.4.2.2).

Other work on mergeable VI has made the SDA algorithm more widely applicable. One limitation of SDA-Bayes is that it only works when the parameters of the variational family are identifiable. We say that the variational family is **identifiable** if for

all $\mathbf{w}_1 \neq \mathbf{w}_2 \in W$, $q(\theta|\mathbf{w}_1) \neq q(\theta|\mathbf{w}_2)$. One important class of nonidentifiability is parameter symmetry. Let S_d denote the group of permutation matrices of dimension d (called the symmetric group of order d), and recall that the variational parameter space $\mathcal{W} = \mathbb{R}^d$. We say that the parameter space exhibits **symmetry** if for any $P \in S_d$, $q(\theta|\mathbf{w}) = q(\theta|P\mathbf{w})$. Symmetry is commonly found in mixture models, and it complicates their learning. Campbell and How (2014) propose an extension to SDA-Bayes called **approximate merging of posteriors with symmetries** (AMPS). The AMPS estimator is given by

$$\hat{\mathbf{w}}^{amps} = (1-m)\hat{\mathbf{w}}_0^{vi} + \sum_{i=1}^{m} P_i \hat{\mathbf{w}}_i^{vi} \tag{2.76}$$

where the P_i are given by

$$\{P_i\} = \underset{P_i \in S_d}{\arg\max}\, \psi\left((1-m)\hat{\mathbf{w}}_0^{vi} + \sum_{i=1}^{m} P_i \hat{\mathbf{w}}_i^{vi}\right) \tag{2.77}$$

The intuition behind AMPS is that we should permute each machine's parameters in such a way that they have the highest probability of being aligned correctly. Maximizing the log partition function ψ does this. The idea of optimizing over permutation matrices to handle symmetry non-identifiabilities is useful in general RLM optimization problems, not only variational methods. This idea has not been applied to general RLM problems, but all the merge methods of Section 2.4.2.2 could be augmented with this capability relatively easily.

A second limitation of SDA-Bayes is that it requires the prior and variational families be conjugate. Gershman et al. (2012) introduce **nonparametric variational infer-**

ence (NVI), which is a single machine method that does not require conjugacy. Neiswanger et al. (2015) propose to combine SDA-Bayes with the NVI model. The central idea is that the variational family should be a mixture of normal distributions. That is,

$$Q^{nvi} = \left\{ \frac{1}{k} \prod_{i=1}^{k} \mathcal{N}(\theta; \mu, \Sigma) : \mu \in \mathbb{R}^d, \Sigma \text{ is a } d \text{ dimensional covariance matrix} \right\} \tag{2.78}$$

and k is a hyperparameter determining the number of mixture components. Since Q^{nvi} is a family of mixture models, it exhibits parameter symmetry. Rather than optimizing over rotation matrices like AMPS, Neiswanger et al. (2015) choose to use a merge procedure that uses sampling to align the components. As the details are rather complicated, we do not describe them here. We do note, however, that the sampling is done locally on the master machine with no communication. The method therefore fits our framework.

In summary, we argue that VI problems should be seen as a special type of RLM problem. This lets the merge procedures of Section 2.4.2.2 apply directly to the VI problem. No experiments have yet been performed with these merge procedures on VI problems; but we might hope they will perform well because they have strong theoretical guarantees whereas the existing VI merge procedures completely lack theoretical guarantees.

2.4.3.3 Markov chain Monte Carlo

Markov chain Monte Carlo (MCMC) is a stochastic method for approximating the posterior of a distribution (e.g. Andrieu et al., 2003). MCMC provides a way to generate samples from the posterior distribution, and these samples can be used to estimate statistics like the expectation or mode of the distribution. MCMC is typically slower than variational

inference but more accurate. As the number of MCMC samples approaches infinity, the approximation error of MCMC shrinks to zero; whereas for variational methods, the approximation error is always some nonzero constant determined by the variational distribution. Because MCMC is computationally expensive, many techniques have been developed to scale MCMC to large datasets. In this section, we will see that seven distributed MCMC algorithms use merge procedures and fit our framework (Wang and Dunson, 2013; Minsker et al., 2014; Neiswanger et al., 2014; Wang et al., 2015; White et al., 2015; Srivastava et al., 2015; Nemeth and Sherlock, 2016; Scott et al., 2016). Another popular approach to reducing the computational burden of MCMC samplers is via fast cross validation methods (Marshall and Spiegelhalter, 2003; Bhattacharya et al., 2007; Bornn et al., 2010; Held et al., 2010; Vehtari et al., 2012; Li et al., 2016). None of the existing fast cross validation algorithms is suitable for the distributed environment. Our framework of mergeable learning algorithms unifies these two approaches to faster MCMC estimators and provides the first distributed fast cross validation methods for MCMC estimators.

We now formally describe the problem of merging MCMC samples. As in Section 2.4.3.1, all the MCMC methods use the underweighted subposterior prior $p(\theta|Z_i) \propto p(Z_i|\theta)p(\theta)^{1/m}$. So by Equation (2.50) gives us

$$p(\theta|Z) \propto \prod_{i=1}^{m} p(\theta|Z_i), \tag{2.79}$$

We use an MCMC sampler to create a series of t samples $\theta_{i,1}, ..., \theta_{i,t}$ from each subposterior i. The number of samples generated by each machine (t) need not have any relationship to the number of data points stored on each machine (n). Generating these samples can

be done independently on m separate machines without communication. Furthermore, any sequential MCMC sampler can be used. The optimal choice of sampler will depend on the particular problem. The `reduce` procedure takes as inputs samples from the m subposteriors and generates samples from the full posterior.

There are three families of merge procedures for MCMC. The simplest family directly merges the samples from the subposterior distributions. The **consensus Monte Carlo** (CMC) algorithm introduced by Scott et al. (2016)[13] is the primary example. CMC generates a sample by a weighted average of the subposteriors' samples. Formally, the jth sample from the full posterior is given by

$$\theta_j^{cmc} = \left(\sum_{i=1}^{m} \Sigma_i \right)^{-1} \sum_{i=1}^{m} \Sigma_i \theta_{i,j}^{mcmc}. \tag{2.80}$$

Where Σ_i is the sample covariance matrix of Z_i. Like the naive averaging estimator for optimization problems, CMC reduces the variance but not the bias of the samples. If we construct a pseudo-parameter vector $\hat{\mathbf{w}}_i^{mcmc} = (\theta_{i,1}, ..., \theta_{i,r})$ by concatenating the samples from the subposteriors, then we can use all the techniques of merging RLM parameters to merge samples as well. None of the RLM merge procedures (other than averaging) have been applied to MCMC, so it is unclear if they would offer any benefit.

A second category of merge functions uses a nonparametric estimate of the subposterior density. This results in significantly more complicated merge functions. These merge functions trade better statistical performance for worse computational performance.

[13] The CMC method appears to be the earliest mergeable MCMC method despite the 2016 official publication date. An early version of the CMC paper was first released online in 2013 and is cited by many other mergeable MCMC papers. Since the earlier papers are not on arXiv or otherwise available online. We have chosen to cite the official journal publication.

Instead of merging the samples directly, an approximate posterior distribution is created by multiplying the nonparametric estimates of the subposteriors. MCMC samples are then generated by sampling from the approximate posterior. The first method in this catgory was given by Neiswanger et al. (2014) and called the **nonparametric density product estimator** (NDPE). NDPE uses the samples from subposterior i to approximate the subposterior density using the **kernel density estimator** (KDE) with a Gaussian kernel:

$$p^{kde}(\theta|Z_i) = \frac{1}{n}\sum_{j=1}^{t}\mathcal{N}(\theta;\theta_{i,j},h^2I_d), \tag{2.81}$$

where I_d is the d dimensional identity matrix and h is a global bandwidth parameter for the gaussian kernel. The full posterior is then approximated by substituting (2.81) into (2.79):

$$p^{ndpe}(\theta|Z) \propto \prod_{i=1}^{m} p^{kde}(\theta|Z_i) = \frac{1}{n^m}\prod_{i=1}^{m}\sum_{j=1}^{t}\mathcal{N}(\theta;\theta_{i,j},h^2I_d) \tag{2.82}$$

Naively sampling from (2.82) is computationally expensive as there are n^m mixture components. Neiswanger et al. (2014) show that many of these components are redundant, and the NDPE samples form a mixture of only mt components, which can be tractably sampled from. The advantage of nonparametric estimators is that they are unbiased when n is fixed and $m \to \infty$, but the disadvantage is that they take a long time to converge. Neiswanger et al. (2014) also propose the **semiparametric density product estimator** (SDPE) which has faster convergence. SDPE approximates the subposterior distributions as the product of a gaussian distribution and the KDE of (2.81). SDPE exhibits both faster convergence and asymptotic efficiency. Wang and Dunson (2013) propose an improvement

to the SDPE called the **Weierstrass sampler** (WS).[14] The WS method approximates the subposteriors using the weierstrass transform of the KDE, which is the convolution (instead of the product) of a gausian distribution and KDE. Nemeth and Sherlock (2016) report that using a gaussian process to approximate the subposteriors has improved empirical performance compared to both the SDPE and WS methods, although they provide little theoretical justification for this claim. Wang et al. (2015) introduce the **parallel aggregation random trees** (PART) merging method. PART uses a kd-tree to nonparametrically represent the subposterior distributions. This has the advantage are that there is no kernel hyperparameter that needs tuning. PART is the only method that provides finite sample guarantees on the quality of the emitted samples. Finally, White et al. (2015) study the NPDE method in the more general setting of **approximate baysian computation** (ABC) where the likelihood function $p(Z|\theta)$ is either unknown or too expensive to compute.

The final category of samplers is based on taking the geometric median or mean of the subposteriors. The earliest example in this category is due to Minsker et al. (2014) and called the **median of subposterior measures** (MSM). The MSM embeds the subposteriors into a reproducing kernel Hilbert space. The approximated posterior is the median of the subposteriors with respect to the distance function in the Hilbert space. Srivastava et al. (2015) propose the **waserstein posterior** (WASP) method. WASP returns the barycenter of the subposterior distributions with respect to the wasserstein distance (the continuous analog of the earth mover distance). WASP shows that this problem can be formulated as a sparse linear program and so solved efficiently in practice.

[14] Wang and Dunson (2013) has an earlier citation date than Neiswanger et al. (2014), but was actually created later. Neiswanger et al. (2014) first appeared on arXiv in 2013 and later was published in a journal in 2014. The improvements of Wang and Dunson (2013) were released on arXiv between these two dates.

2.5 Conclusion

We have seen many examples of mergeable learning algorithms. Researchers have developed these algorithms while working on distributed learning problems, but we have shown that these algorithms also come with a fast cross validation procedure. In the next two chapters, we design new mergeable learning algorithms. Chapter 3 presents the optimal weighted average (OWA) algorithm for merging regularized loss minimizers. Unlike the methods for RLM presented in Section 2.4.2.2, OWA has statistical guarantees and is fast to compute. Chapter 4 presents the cover tree data structure, which is used to speed up nonparametric learning algorithms like nearest neighbor queries and kernel density estimation. On of the improvements to the cover tree is a merge procedure. This merge procedure provides the first parallel construction algorithm and fast cross validation algorithm for cover tree based learning methods.

Chapter 3

The optimal weighted average

This chapter presents a new merge procedure called the **optimal weighted average** (OWA). OWA improves on the "averaging based" merge procedures of Section 2.4.2.2 from the previous chapter. In particular: OWA either has stronger statistical guarantees, is applicable to more models, or is more computationally efficient. To simplify the exposition, we will first present the OWA procedure for linear models. Then, we will show an example application of OWA to deep neural networks.

This chapter can be read independently of the previous chapters. Section 3.1 formally introduces the needed notation and describes the problem of merging linear models. Section 3.2 describes the OWA merge procedure. We take special care to show how OWA can be implemented with off-the-shelf optimizers. Section 3.3 provides a simple proof that OWA achieves the optimal $O(\sqrt{d/mn})$ error. Our main condition is that the single machine parameter vectors have a "sufficiently gaussian" distribution. This is a mild condition known to hold in many situations of interest. In order to clarify the constant factors involved in our

algorithm, we also introduce a novel measure of the smoothness of a loss function. Section 3.4.2 shows empirically that OWA performs well on synthetic and real world advertising data. We demonstrate that OWA is robust to the strength of regularization, which is one of the reasons it performs well in practice. Section 3.5 then generalizes the OWA algorithm to deep neural networks. Preliminary experiments on the MNIST data set show that this nonlinear version of OWA works well.

3.1 The problem of merging linear models

Let $\mathcal{Y} \subseteq \mathbb{R}$ be the space of response variables, $X \subseteq \mathbb{R}^d$ be the space of covariates, and $\mathcal{W} \subseteq \mathbb{R}^d$ be the parameter space. We assume a linear model where the loss of data point $(\mathbf{x}, y) \in \mathcal{X} \times \mathcal{Y}$ given the parameter $\mathbf{w} \in \mathcal{W}$ is denoted by $\ell(y, \mathbf{x}^\mathsf{T}\mathbf{w})$. We define the true loss of parameter $\mathbf{w}$ to be $\mathcal{L}^*(\mathbf{w}) = \mathbb{E}\,\ell(y; \mathbf{x}^\mathsf{T}\mathbf{w})$, and the optimal parameter vector $\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathcal{W}} \mathcal{L}^*(\mathbf{w})$. We do not require that the model be correctly specified, nor do we require that ℓ be convex with respect to $\mathbf{w}$. Let $Z \subset \mathcal{X} \times \mathcal{Y}$ be a dataset of mn i.i.d. observations. Finally, let $r : \mathcal{W} \to \mathbb{R}$ be a regularization function (typically the L1 or L2 norm) and $\lambda \in \mathbb{R}$ be the regularization strength. Then the **regularized loss minimizer** (RLM) is

$$\hat{\mathbf{w}}^{rlm} = \arg\max_{\mathbf{w}\in\mathcal{W}} \sum_{(\mathbf{x},y)\in Z} \ell(y, \mathbf{x}^\mathsf{T}\mathbf{w}) + \lambda r(\mathbf{w}). \tag{3.1}$$

Assume that the dataset Z has been partitioned onto m machines so that each machine i has dataset Z_i of size n, and all the Z_i are disjoint. Then each machine calculates the local RLM

$$\hat{\mathbf{w}}_i^{rlm} = \arg\max_{\mathbf{w}\in\mathcal{W}} \sum_{(\mathbf{x},y)\in Z_i} \ell(y, \mathbf{x}^\mathsf{T}\mathbf{w}) + \lambda r(\mathbf{w}). \tag{3.2}$$

Solving for $\hat{\mathbf{w}}_i^{rlm}$ requires no communication with other machines. Our goal is to merge the $\hat{\mathbf{w}}_i^{rlm}$s into a single improved estimate.

We now recall some facts from Chapter 2 about measuring the quality of merged estimators. Section 2.4.2.2 presented a number of existing baseline merge procedures. The simplest is the naive averaging estimator:

$$\hat{\mathbf{w}}^{ave} = \frac{1}{m}\sum_{i=1}^{m} \hat{\mathbf{w}}_i^{rlm}. \tag{3.3}$$

OWA will improve on naive averaging by taking a carefully selected weighted average. The quality of an estimator $\hat{\mathbf{w}}$ can be measured by the estimation error $\|\hat{\mathbf{w}} - \mathbf{w}^*\|$. We can use the triangle inequality to decompose this error as

$$\|\hat{\mathbf{w}} - \mathbf{w}^*\| \leq \|\hat{\mathbf{w}} - \mathbb{E}\,\hat{\mathbf{w}}\| + \|\mathbb{E}\,\hat{\mathbf{w}} - \mathbf{w}^*\|. \tag{3.4}$$

We refer to $\|\hat{\mathbf{w}} - \mathbb{E}\,\hat{\mathbf{w}}\|$ as the variance of the estimator and $\|\mathbb{E}\,\hat{\mathbf{w}} - \mathbf{w}^*\|$ as the bias. For the single machine oracle estimator $\hat{\mathbf{w}}^{rlm}$, both the variance and bias decrease at the rate $O(\sqrt{d/mn})$. For the naive averaging estimator, the variance decreases at the optimal $O(\sqrt{d/mn})$ rate, but the bias decreases only as $O(\sqrt{d/n})$. In other words, the bias of $\hat{\mathbf{w}}^{ave}$ is independent of the number of machines m. We shall see that for the OWA estimator, both the bias and variance decrease as $O(\sqrt{d/mn})$. Thus OWA has better guarantees than naive averaging and similar guarantees as the single machine oracle.

3.2 The algorithm

The **optimal weighted average** (OWA) merge procedure uses a second round of optimization to calculate the optimal linear combination of the $\hat{\mathbf{w}}_i^{rlm}$s. This second optimization occurs over a small fraction of the dataset, so its computational and communication cost is negligible.

3.2.1 Warmup: the full OWA estimator

To motivate the OWA estimator, we first present a less efficient estimator that uses the full dataset for the second round of optimization. Define the matrix $\hat{W} : \mathbb{R}^{d \times m}$ to have its ith column equal to $\hat{\mathbf{w}}_i^{rlm}$. Now consider the estimator

$$\hat{\mathbf{w}}^{owa,full} = \hat{W}\hat{\mathbf{v}}^{owa,full}, \qquad \text{where} \qquad \hat{\mathbf{v}}^{owa,full} = \arg\max_{\mathbf{v} \in \mathbb{R}^m} \sum_{(\mathbf{x},y) \in Z} \ell\left(y, \mathbf{x}^\mathsf{T}\hat{W}\mathbf{v}\right) + \lambda r(\hat{W}\mathbf{v}). \tag{3.5}$$

Notice that $\hat{\mathbf{w}}^{owa,full}$ is just the regularized loss minimizer when the parameter space $\mathcal{W}$ is restricted to the subspace $\hat{\mathcal{W}}^{owa} = \text{span}\{\hat{\mathbf{w}}_i^{rlm}\}_{i=1}^m$. In other words, the $\hat{\mathbf{v}}^{owa,full}$ vector contains the optimal weights to apply to each $\hat{\mathbf{w}}_i^{rlm}$ when averaging. Figure 3.1 shows graphically that no other estimator in $\hat{\mathcal{W}}^{owa}$ can have lower regularized empirical loss than $\hat{\mathbf{w}}^{owa,full}$.

3.2.2 The OWA estimator

Calculating the weights $\hat{\mathbf{v}}^{owa,full}$ directly is infeasible because it requires access to the full dataset. Fortunately, we do not need to consider all the data points for an accurate esti-

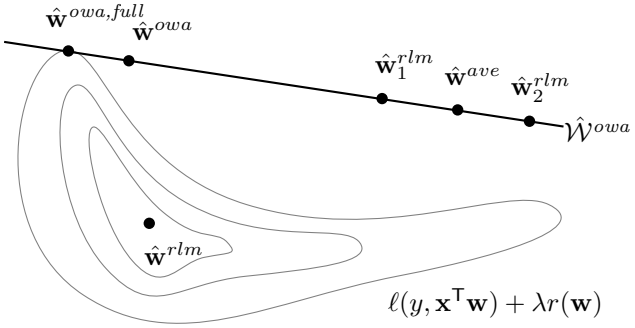

Figure 3.1: Graphical depiction of the OWA estimator's parameter space. $\hat{\mathbf{w}}^{owa,full}$ is the estimator with best loss in $\hat{\mathcal{W}}^{owa}$, and $\hat{\mathbf{w}}^{owa}$ is close with high probability.

mator. In a linear model, the number of data points needed is proportional to the dimensionality of the underlying space. The parameter space $\hat{\mathcal{W}}^{owa}$ is m-dimensional, so we need only $O(m)$ samples. In general, $m \ll n$, so this second optimization requires only a tiny fraction of the original data. This intuition motivates the OWA estimator. Let $Z_i^{owa} \subset Z_i$ be a set of n^{owa} data points uniformly sampled from Z_i without replacement, and let Z^{owa} be the union of the Z_i^{owa}s. Then the OWA estimator is defined to be

$$\hat{\mathbf{w}}^{owa} = \hat{W}\hat{\mathbf{v}}^{owa}, \qquad \text{where} \qquad \hat{\mathbf{v}}^{owa} = \operatorname*{arg\,max}_{\mathbf{v}\in\mathbb{R}^m} \sum_{(\mathbf{x},y)\in Z^{owa}} \ell\left(y, \mathbf{x}^\mathsf{T}\hat{W}\mathbf{v}\right) + \lambda r(\hat{W}\mathbf{v}). \tag{3.6}$$

We present two algorithms for calculating $\hat{\mathbf{w}}^{owa}$ in a distributed setting. These algorithms are minor specializations of the general distributed learning algorithm for mergeable estimators (Algorithm 1). Algorithm 7 uses only a single round of communication, but it requires that a predesignated master machine already have a copy of the Z^{owa} dataset. Each machine calculates $\hat{\mathbf{w}}_i^{rlm}$ independently, then transfers the result to the master. The master then has all the information needed to solve (3.6). A total of $O(dm)$ bits are transfered to the server. (The parameter vector has d dimensions and there are m machines.)

Algorithm 7 Calculating $\hat{\mathbf{w}}^{owa}$ in one round

Preconditions:
  each machine i already has dataset Z_i
  the master machine additionally has Z^{owa}
Round 1, each machine i independently:
  calculates $\hat{\mathbf{w}}_i^{rlm}$ using (3.2)
  transmits $\hat{\mathbf{w}}_i^{rlm}$ to the master
The master calculates $\hat{\mathbf{w}}^{owa}$ using (3.6)
  (optionally) master uses approximation (3.7)

Algorithm 8 Calculating $\hat{\mathbf{w}}^{owa}$ in two rounds

Preconditions:
  each machine i already has dataset Z_i
Round 1, each machine i independently:
  calculates $\hat{\mathbf{w}}_i^{rlm}$ using (3.2)
  broadcasts $\hat{\mathbf{w}}_i^{rlm}$ to all other machines
Round 2, each machine i independently:
  constructs $\hat{W} = (\hat{\mathbf{w}}_1^{rlm}, ..., \hat{\mathbf{w}}_m^{rlm})$
  samples a dataset $Z_i^{owa} \subset Z_i$ of size n^{owa}
  calcs $Z_i^{proj} = \{(\mathbf{x}^\mathsf{T}\hat{W}, y) : (\mathbf{x}, y) \in Z^{owa}\}$
  sends Z_i^{proj} to a master machine
The master calculates $\hat{\mathbf{w}}^{owa}$ using (3.6)
  (optionally) master uses approximation (3.7)

The averaging estimator transfers the same information, and so has the same $O(dm)$ communication complexity. The only difference between the two algorithms is the way the master machine merges the local estimates.

If the master machine does not already have a copy of Z^{owa}, then we must transfer a copy to the master. Algorithm 8 is a two round version of OWA that exploits the structure of the $\hat{\mathbf{w}}^{owa}$ estimator to transmit this data efficiently. In the first round, each machine calculates $\hat{\mathbf{w}}_i^{rlm}$ independently. The result is then broadcast to every other machine, instead of just the master. A total of $O(dm^2)$ bits are transmitted in this round. (The parameter vector has d dimensions, there are m machines, and each machine transmits to each other

machine.) In the second round, each machine projects its local dataset Z_i^{owa} onto the space $\hat{\mathcal{W}}^{owa}$. These projected data points are then transmitted to the master. A total of $O(m^2 n^{owa})$ bits are transmitted. (The projected data points each have dimension m, there are m machines, and there are n^{owa} data points per machine.) The analysis in Lemma 2 (see Section 3.3) suggests that n^{owa} should be set to $O(mn/d)$. So the total data transmitted in both rounds is $O(dm^2 + m^3 n/d)$.

3.2.3 Implementing OWA with existing optimizers

Equations 3.5 and 3.6 cannot be solved directly using off-the-shelf optimizers because existing optimizers do not support the non-standard regularization term $r(\hat{W}\mathbf{v})$. In practice, it is sufficient to approximate this term by L2 regularization directly on the $\mathbf{v}$ vector:

$$\lambda r(\hat{W}\mathbf{v}) \approx \lambda_2 \|\mathbf{v}\|^2, \tag{3.7}$$

where λ_2 is a new hyperparameter. We provide two justifications for this approximation:

1. When we want the parameter vector $\mathbf{w}$ to be sparse (and so the regularizer r is the L1 norm), we have no reason to believe that the $\mathbf{v}$ vector should be sparse. The desired sparsity is induced by the regularization when solving for $\hat{\mathbf{w}}_i^{rlm}$s and maintained in any linear combination of the $\hat{\mathbf{w}}_i^{rlm}$s.

2. As the size of the dataset increases, the strength of the regularizer decreases. In this second optimization, the dimensionality of the problem is small, so it is easy to add more data to make the influence of the regularization negligible.

The new λ_2 regularization parameter should be set by cross validation. This will be a fast procedure, however, because there are only $mn^{owa} \ll mn$ data points to optimize over, they have dimensionality $m \ll d$, and the L2 regularized problem is much easier to solve than the L1 problem. Furthermore, this cross validation can be computed locally on the master machine without any communication. The experiments in Section 3.4.2 provide an example where the first round of optimization to compute the $\hat{\mathbf{w}}_i^{rlm}$s takes about a day, but the second round of optimization (including cross validation over λ_2) takes only several minutes.

3.3 Analysis

In this section, we outline the argument that OWA's generalization error $\mathcal{L}^*(\hat{\mathbf{w}}^{owa}) - \mathcal{L}^*(\mathbf{w}^*)$ and estimation error $\|\hat{\mathbf{w}}^{owa} - \mathbf{w}^*\|$ both decay as $O(\sqrt{d/mn})$. Full proofs of all theorems can be found in the supplemental material. Our analysis depends on the single machine estimators obeying the following mild condition.

Definition 1. We say that an estimator $\hat{\mathbf{w}}$ that has been trained on n data points of dimension d satisifies the *sub-gaussian tail (SGT) condition* if, for all $t > 0$, with probability at least $1 - \exp(-t)$, $\|\hat{\mathbf{w}} - \mathbf{w}^*\| \leq O(\sqrt{dt/n})$.

The SGT condition is a high-level condition that is well established in the statistical literature. All generalized linear models (such as logistic regression and ordinary least squares regression) satisfy the SGT condition, and many less standard models (such as linear models with a sigmoid loss) also satisfy the condition. There are many ways to prove that an estimator satisfies the SGT condition. In the asymptotic regime when $n \to \infty$,

very strong results of this form have been known since the 1960s. Chapter 7 of Lehmann (1999) provides an elementary introduction to this work. Lehman requires only that ℓ be three times differentiable, that the data points be i.i.d., and that $\mathbf{w}^*$ be identifiable. More recent results establish the SGT condition in the non-asymptotic regime $n < \infty$. The strongest non-asymptotic results known to the authors are due to Spokoiny (2012). Spokoiny's only assumption is that the empirical loss admit a local approximation via the "bracketing device," which is a generalization of the Taylor expansion. The full explanation of the bracketing device is rather technical, so we do not present it here. Instead, we highlight that Spokoiny's results do not require a convex loss or even that the data be i.i.d.

The first lemma is an easy consequence of the SGT condition for the $\hat{\mathbf{w}}_i^{rlm}$s. It formalizes the key idea that $\hat{\mathcal{W}}^{owa}$ is a good subspace to optimize over because it contains a point close to the true parameter vector.

Lemma 2. Assume the $\hat{\mathbf{w}}_i^{rlm}$s satisfy the SGT condition. Let $\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^*$ denote the vector in $\hat{\mathcal{W}}^{owa}$ with minimum distance to $\mathbf{w}^*$. Let $t > 0$. Then with probability at least $1 - \exp(-t)$,

$$\|\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^* - \mathbf{w}^*\| \leq O(\sqrt{dt/mn}). \tag{3.8}$$

Proof. Using independence of the $\hat{\mathbf{w}}_i^{rlm}$s and the SGT condition, we have that

$$\Pr\left[\|\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^* - \mathbf{w}^*\| \leq O(\sqrt{dt/mn})\right] \geq \Pr\left[\min_{i=1\ldots m} \|\hat{\mathbf{w}}_i^{rlm} - \mathbf{w}^*\| \leq O(\sqrt{dt/mn})\right] \tag{3.9}$$

$$= 1 - \Pr\left[\min_{i=1\ldots m} \|\hat{\mathbf{w}}_i^{rlm} - \mathbf{w}^*\| > O(\sqrt{dt/mn})\right] \tag{3.10}$$

$$= 1 - \left(\Pr\left[\|\hat{\mathbf{w}}_1^{rlm} - \mathbf{w}^*\| > O(\sqrt{dt/mn})\right]\right)^m \tag{3.11}$$

$$= 1 - \left(1 - \Pr\left[\|\hat{\mathbf{w}}_1^{rlm} - \mathbf{w}^*\| \leq O(\sqrt{dt/mn})\right]\right)^m \tag{3.12}$$

$$\geq 1 - \big(1 - (1 - \exp(-t/m))\big)^m \tag{3.13}$$

$$= 1 - \exp(-t). \tag{3.14}$$

□

Our next lemma shows that $\hat{\mathbf{w}}^{owa}$ is a good approximation of $\hat{\mathbf{w}}^{owa,full}$ as long as $n^{owa} = mn/d$. In linear models, the number of dimensions used typically grows proportionally to the number of data points, so this lemma formalizes the intuition that relatively little data is needed in the second optimization. In this lemma, we require that both $\hat{\mathbf{v}}^{owa,full}$ and $\hat{\mathbf{v}}^{owa}$ satisfy the SGT condition. Recall from Equations 3.5 and 3.6 that both estimators are trained on a projected data set with dimension $\min(m, d)$. So the SGT conditions for $\hat{\mathbf{v}}^{owa,full}$ and $\hat{\mathbf{v}}^{owa}$ state that with probability at least $1 - \exp(-t)$,

$$\|\hat{\mathbf{v}}^{owa,full} - \mathbf{v}^*\| \leq O(\min(m, d)t/mn), \tag{3.15}$$

and

$$\|\hat{\mathbf{v}}^{owa} - \mathbf{v}^*\| \leq O(\min(m,d)t/mn^{owa}), \tag{3.16}$$

where $\mathbf{v}^*$ is the parameter vector in $\hat{\mathcal{W}}^{owa}$ that minimizes $\mathcal{L}^*$. (Note that in general, $\mathbf{v}^* \neq \pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^*$). The results of Spokoiny (2012) can be used to show that $\hat{\mathbf{v}}^{owa,full}$ and $\hat{\mathbf{v}}^{owa}$ satisfy the SGT condition.

Lemma 3. Assume that $\hat{\mathbf{v}}^{owa,full}$ and $\hat{\mathbf{v}}^{owa}$ satisfy the SGT condition. Let $n^{owa} = mn/d$. Let $t > 0$. Then with probability at least $1 - \exp(-t)$, $\|\hat{\mathbf{w}}^{owa,full} - \hat{\mathbf{w}}^{owa}\| \leq O(dt/mn)$.

Proof. By the SGT condition for $\hat{\mathbf{v}}^{owa,full}$, we have that

$$\|\hat{\mathbf{v}}^{owa,full} - \mathbf{v}^*\| \leq O(\sqrt{\min(m,d)t/mn}) = O(\sqrt{dt/mn}). \tag{3.17}$$

Similarly, by the SGT condition for $\hat{\mathbf{v}}^{owa}$, we have that So,

$$\|\hat{\mathbf{v}}^{owa} - \mathbf{v}^*\| \leq O(\sqrt{\min(m,d)t/mn^{owa}}) = O(\sqrt{mt/mn^{owa}}) = O(\sqrt{dt/mn}). \tag{3.18}$$

The result follows by combining Equations 3.17 and 3.18 using the triangle inequality:

$$\|\hat{\mathbf{w}}^{owa,full} - \hat{\mathbf{w}}^{owa,*}\| = \|\hat{\mathbf{v}}^{owa,full} - \hat{\mathbf{v}}^{owa}\| \leq \|\hat{\mathbf{v}}^{owa} - \mathbf{v}^*\| + \|\mathbf{v}^* - \hat{\mathbf{v}}^{owa}\| \leq O(\sqrt{dt/mn}). \tag{3.19}$$

□

In order to connect the results of Lemmas 1 and 2 to the generalization error of our estimator, we need to introduce a smoothness condition on the true loss function $\mathcal{L}^*$.

Definition 4. We say that $\mathcal{L}^*$ is β-*Lipschitz* continuous if for all $\mathbf{w}_1$ and $\mathbf{w}_2$,

$$|\mathcal{L}^*(\mathbf{w}_1) - \mathcal{L}^*(\mathbf{w}_2)| \leq \beta \|\mathbf{w}_1 - \mathbf{w}_2\|. \tag{3.20}$$

We can now present our first main result.

Theorem 5. *Assume the* $\hat{\mathbf{w}}_i^{rlm}$*s,* $\hat{\mathbf{v}}^{owa,full}$*, and* $\hat{\mathbf{v}}^{owa}$ *all satisfy the SGT condition. Further assume that* $\mathcal{L}^*$ *is* β*-Lipschitz. Let* $n^{owa} = mn/d$*. Let* $t > 0$*. Then with probability at least* $1 - \exp(-t)$*,*

$$\mathcal{L}^*(\hat{\mathbf{w}}^{owa}) - \mathcal{L}^*(\mathbf{w}^*) \leq O(\beta\sqrt{dt/mn}). \tag{3.21}$$

Proof. We have that

$$\begin{aligned}
\mathcal{L}^*(\hat{\mathbf{w}}^{owa,full}) - \mathcal{L}^*(\mathbf{w}^*) &\leq \mathcal{L}^*(\hat{\mathbf{w}}^{owa,full}) - \mathcal{L}^*(\hat{W}\mathbf{v}^*) + \mathcal{L}^*(\hat{W}\mathbf{v}^*) - \mathcal{L}^*(\mathbf{w}^*) && (3.22)\\
&\leq \mathcal{L}^*(\hat{W}\hat{\mathbf{v}}^{owa,full}) - \mathcal{L}^*(\hat{W}\mathbf{v}^*) + \mathcal{L}^*(\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^*) - \mathcal{L}^*(\mathbf{w}^*) && (3.23)\\
&\leq \beta\|\hat{W}\hat{\mathbf{v}}^{owa,full} - \hat{W}\mathbf{v}^*\| + \beta\|\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^* - \mathbf{w}^*\| && (3.24)\\
&\leq \beta\|\hat{\mathbf{v}}^{owa,full} - \mathbf{v}^*\| + \beta\|\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^* - \mathbf{w}^*\| && (3.25)\\
&\leq O(\beta\sqrt{dt/mn}). && (3.26)
\end{aligned}$$

The last line follows by applying the SGT condition with respect to $\hat{\mathbf{w}}^{owa,full}$ on the left

term and Lemma 2 on the right term. We then have that

$$\mathcal{L}^*(\hat{\mathbf{w}}^{owa}) - \mathcal{L}^*(\mathbf{w}^*) \leq \mathcal{L}^*(\hat{\mathbf{w}}^{owa}) - \mathcal{L}^*(\hat{\mathbf{w}}^{owa,full}) + \mathcal{L}^*(\hat{\mathbf{w}}^{owa,full}) - \mathcal{L}^*(\mathbf{w}^*) \tag{3.27}$$

$$\leq \beta \|\hat{\mathbf{w}}^{owa} - \hat{\mathbf{w}}^{owa,full}\| + \mathcal{L}^*(\hat{\mathbf{w}}^{owa,full}) - \mathcal{L}^*(\mathbf{w}^*) \tag{3.28}$$

$$\leq O(\sqrt{dt/mn}). \tag{3.29}$$

The last line follows from Lemma 3 on the left term and Eq. (3.26) on the right term. □

The use of Lipschitz continuity in Theorem 1 sacrifices generality for interpretability. While many losses are β-Lipschitz (e.g. log loss, hinge loss, and sigmoid loss), many other losses are not (e.g. squared loss and exp loss). Therefore Theorem 1 is not as general as it could be. To improve generality of our next result, we introduce the following novel smoothness condition.

Definition 6. Let $R \subseteq \mathcal{W}$. We call the true loss function $\mathcal{L}^*$ *locally quadratically bracketed in* R (LQB-R) if for all points $\mathbf{w} \in R$,

$$\alpha_{\text{lo}} \|\mathbf{w} - \mathbf{w}^*\|^2 \leq \mathcal{L}(\mathbf{w}) - \mathcal{L}(\mathbf{w}^*) \leq \alpha_{\text{hi}} \|\mathbf{w} - \mathbf{w}^*\|^2. \tag{3.30}$$

The LQB-R condition is quite general. The leftmost inequality in (3.30) is related to (but more general than) the more familiar notion of α_{lo}-strong convexity. (We say $\mathcal{L}^*$ is α_{lo}-strongly convex if $\nabla^2 \mathcal{L}^*(\mathbf{w}) \geq \alpha_{\text{lo}} I$ for all $\mathbf{w}$.) Inequality (3.30), however, requires only that $\nabla^2 \mathcal{L}^*(\mathbf{w}) \geq \alpha_{\text{lo}} I$ at the minimum point $\mathbf{w} = \mathbf{w}^*$. The function may (for example) have local maxima elsewhere. When R has finite radius (i.e. $\|\mathbf{w}_1 - \mathbf{w}_2\|$ is finite for all $\mathbf{w}_1, \mathbf{w}_2 \in R$), then all continuous functions with a unique global minimum will satisfy the

leftmost inequality for some value of α_{lo}. The rightmost inequality of (3.30) states that $\mathcal{L}^*$ must not grow too quickly within R. If R has finite radius, and there is an open ball around $\mathbf{w}^*$ that does not intersect R, then any continuous function satisfies the rightmost inequality for some α_{hi}. In particular, models using the log loss, hinge loss, sigmoid loss, squared loss, and exp loss all satisfy the LQB-R condition for any finite set R.

Theorem 7. *Assume the $\hat{\mathbf{w}}_i^{rlm}$s, $\hat{\mathbf{v}}^{owa,full}$, and $\hat{\mathbf{v}}^{owa}$ all satisfy the SGT condition. Further assume that $\mathcal{L}$ is locally quadratically bracketed in $R = \{\hat{W}\mathbf{v}^*, \pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^*\}$. Let $t > 0$. Then with probability at least $1 - \exp(-t)$,*

$$\|\hat{\mathbf{w}}^{owa} - \mathbf{w}^*\| \leq O\left(\sqrt{(\alpha_{hi}/\alpha_{lo})(dt/mn)}\right). \tag{3.31}$$

Proof. By local quadratic bracketing, we have that

$$\alpha_{\text{lo}}\|\hat{W}\mathbf{v}^* - \mathbf{w}^*\|^2 \leq \mathcal{L}(\hat{W}\mathbf{v}^*) - \mathcal{L}(\mathbf{w}^*) \leq \mathcal{L}(\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^*) - \mathcal{L}(\mathbf{w}^*) \leq \alpha_{\text{hi}}\|\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^* - \mathbf{w}^*\|^2.$$

Simplifying and applying Lemma 2 gives

$$\|\hat{W}\mathbf{v}^* - \mathbf{w}^*\| \leq \sqrt{\alpha_{\text{hi}}/\alpha_{\text{lo}}}\|\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^* - \mathbf{w}^*\| \leq O(\sqrt{(\alpha_{\text{hi}}/\alpha_{\text{lo}})(dt/mn)}). \tag{3.32}$$

Applying the triangle inequality, Equation (3.32), and Lemma 3 gives

$$\|\hat{\mathbf{w}}^{owa} - \mathbf{w}^*\| \leq \|\hat{\mathbf{w}}^{owa} - \hat{\mathbf{w}}^{owa,full}\| + \|\hat{\mathbf{w}}^{owa,full} - \hat{W}\mathbf{v}^*\| + \|\hat{W}\mathbf{v}^* - \mathbf{w}^*\| \tag{3.33}$$

$$\leq O(\sqrt{(\alpha_{\text{hi}}/\alpha_{\text{lo}})(dt/mn)}). \tag{3.34}$$

The last line follows from Lemma 2 on the left, the SGT conditionin the middle, and (3.32) on the right. □

Our choice of R in Theorem 2 gives us a reasonable interpretation of the ratio $\alpha_{\text{hi}}/\alpha_{\text{lo}}$ as a generalized condition number. We know from the SGT condition and Lemma 1 that as the number of data points per machine n goes to infinity, then the distances $\|\hat{W}\mathbf{v}^* - \mathbf{w}^*\|$ and $\|\pi_{\hat{\mathcal{W}}^{owa}}\mathbf{w}^*\|$ go to zero. The lower and upper bounding quadratic functions then converge to the 2nd order Taylor approximation of $\mathcal{L}^*$ at $\mathbf{w}^*$. The ratio $\alpha_{\text{hi}}/\alpha_{\text{lo}}$ is then the condition number of the Hessian of $\mathcal{L}^*$ at $\mathbf{w}^*$.

We claim that the convergence rate in (3.31) is optimal because up to the constant factor $\alpha_{\text{hi}}/\alpha_{\text{lo}}$, it matches the convergence rate of the single machine oracle $\hat{\mathbf{w}}^{rlm}$.

3.4 Experiments

We evaluate OWA on synthetic and real-world logistic regression tasks. In each experiment, we compare $\hat{\mathbf{w}}^{owa}$ with four baseline estimators: the naive estimator using the data from only a single machine $\hat{\mathbf{w}}_i^{rlm}$; the averaging estimator $\hat{\mathbf{w}}^{ave}$; the bootstrap estimator $\hat{\mathbf{w}}^{boot}$; and the oracle estimator of all data trained on a single machine $\hat{\mathbf{w}}^{rlm}$. The $\hat{\mathbf{w}}^{boot}$ estimator has a parameter r that needs to be tuned. In all experiments we evaluate $\hat{\mathbf{w}}^{boot}$ with $r \in \{0.005, 0.01, 0.02, 0.04, 0.1, 0.2\}$, which is a set recommended in the original paper (Zhang et al., 2012), and then report only the value of r with highest true likelihood. Thus we are reporting an overly optimistic estimate of the performance of $\hat{\mathbf{w}}^{boot}$, and as we shall see $\hat{\mathbf{w}}^{owa}$ still tends to perform better.

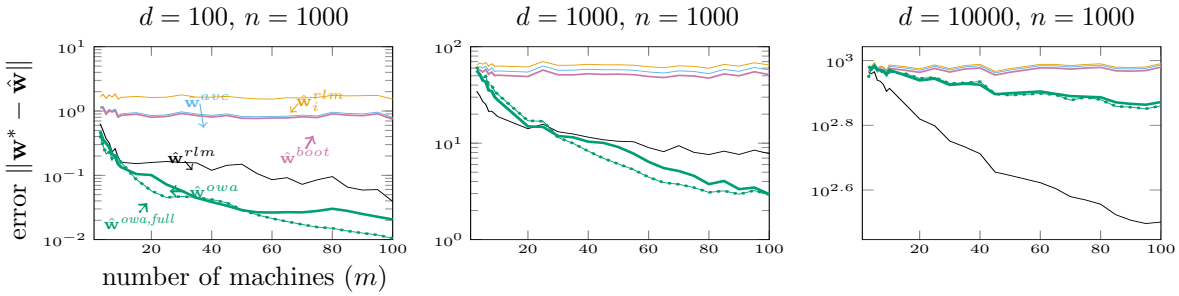

Figure 3.2: OWA's performance on synthetic logistic regresion data. The left figure shows scalability in the low dimension regime, the middle figure in a medium dimension regime, and the right figure in a high dimension regime. $\hat{\mathbf{w}}^{owa}$ scales well with the number of machines in all cases. Surprisingly, $\hat{\mathbf{w}}^{owa}$ outperforms the oracle estimator trained on all of the data $\hat{\mathbf{w}}^{rlm}$ in some situations.

3.4.1 Synthetic data

We generate the data according to a sparse logistic regression model. Each component of $\mathbf{w}^*$ is sampled i.i.d. from a spike and slab distribution. With probability 0.9, it is 0; with probability 0.1, it is sampled from a standard normal distribution. The data points are then sampled as

$$\mathbf{x}_i \sim \mathcal{N}(0, I) \qquad \text{and} \qquad y_i \sim \text{Bernoulli}\left(1/\left(1 + \exp(-\mathbf{x}_i^\mathsf{T}\mathbf{w}^*)\right)\right). \tag{3.35}$$

The primary advantage of synthetic data is that we know the model's true parameter vector. So for each estimator $\hat{\mathbf{w}}$ that we evaluate, we can directly calculate the error $\|\hat{\mathbf{w}} - \mathbf{w}^*\|$. We run two experiments on the synthetic data. In both experiments, we use the L1 regularizer to induce sparsity in our estimates of $\mathbf{w}^*$. Results are qualitatively similar when using a Laplace, gaussian, or uniform prior on $\mathbf{w}^*$, and with L2 regularization.

Our first experiment shows how the estimators scale as the number of machines m increases. We fix $n = 1000$ data points per machine, so the size of the dataset mn

grows as we add more machines. This simulates the typical "big data" regime where data is abundant, but processing resources are scarce. For each value of m, we generate 50 datasets and report the average of the results. Our $\hat{\mathbf{w}}^{owa}$ estimator was trained with $n^{owa} = 128$. The results are shown in Figure 3.2. As the analysis predicted, the performance of $\hat{\mathbf{w}}^{owa}$ scales much better than $\hat{\mathbf{w}}^{ave}$ and $\hat{\mathbf{w}}^{boot}$. Surprisingly, in the low dimensional regimes, $\hat{\mathbf{w}}^{owa}$ outperforms the single machine oracle $\hat{\mathbf{w}}^{rlm}$.

One issue that has been overlooked in the literature on non-interactive distributed estimation is how to best set λ. There are two natural choices. The first is: for each λ in the grid, perform the full training procedure including all communications and merges. Then select the λ with lowest cross validation error. Unfortunately, this requires many rounds of communication (one for each λ we are testing). This extra communication largely negates the main advantage of non-interactive learners. The second is: each machine independently uses cross validation to select the λ that best fits the data locally when calculating $\hat{\mathbf{w}}_i^{rlm}$. In our experiments we use this second method due to three advantages. First, there is no additional communication because model selection is a completely local task. Second, existing optimizers have built-in model selection routines which make the process easy to implement. We used the default model selection procedure from Python's SciKit-Learn (Pedregosa et al., 2011). Third, the data may be best fit using different regularization strengths for each machine. It is unclear which method previous literature used to select λ. As mentioned in Section 3.2.3, OWA requires an additional round of cross validation during the master's merge procedure to set λ_2. This cross validation is particularly fast and requires no communication with other machines.

Our second experiment shows the importance of proper λ selection. We evaluate the performance of the estimators with λ varying from 10^{-4} to 10^4 on a grid of 80 points. Figure 3.3 shows the results. The $\hat{\mathbf{w}}^{owa}$ estimator is more robust to the choice of λ than the other distributed estimators.

3.4.2 Real world advertising data

We evaluate the estimators on real world data from the KDD 2012 Cup (Niu et al., 2012). The goal is to predict whether a user will click on an ad from the Tencent internet search engine. This dataset was previously used to evaluate the performance of $\hat{\mathbf{w}}^{boot}$ (Zhang et al., 2012). This dataset is too large to fit on a single machine, so we must use distributed estimators, and we do not provide results of the oracle estimator $\hat{\mathbf{w}}^{rlm}$ in our figures. There are 235,582,879 distinct data points, each of dimension 741,725. The data points are sparse, so we use the L1 norm to encourage sparsity in our final solution. The regularization strength was set using cross validation in the same manner as for the synthetic data. For each test, we split the data into 80 percent training data and 20 percent test data. The training data is further subdivided into 128 partitions, one for each of the machines used. It took about 1 day to train the local model on each machine in our cluster.

Our first experiment tests the sensitivity of the n^{owa} parameter on large datasets. We fix $m = 128$, and allow n^{owa} to vary from 2^0 to 2^{20}. Recall that the number of data points used in the second optimization is mn^{owa}, so when $n^{owa} = 2^{20}$ nearly the entire data set is used. We repeated the experiment 50 times, each time using a different randomly selected set Z^{owa} for the second optimization. Figure 3.4 shows the results. Our $\hat{\mathbf{w}}^{owa}$ estimator has lower loss than $\hat{\mathbf{w}}^{ave}$ using only 16 data points per machine (approximately

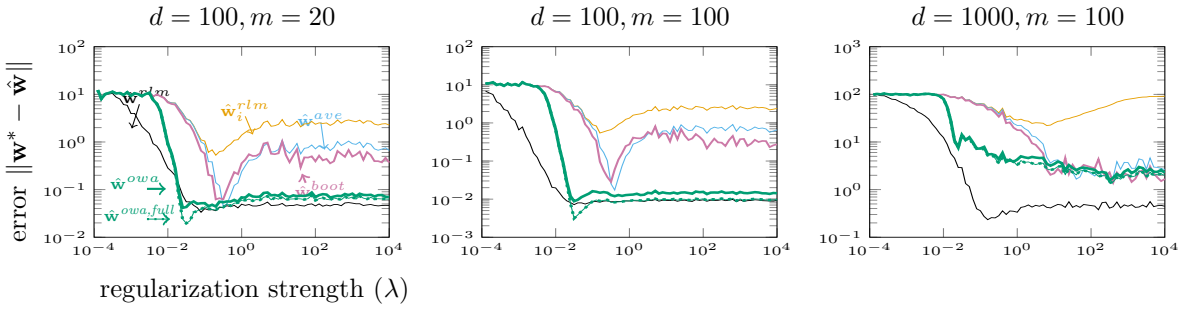

Figure 3.3: OWA is robust to the regularization strength used to solve $\hat{\mathbf{w}}_i^{rlm}$. Our theory states that as $m \to d$, we have that $\hat{\mathcal{W}}^{owa} \to \mathcal{W}$, and so $\hat{\mathbf{w}}^{owa} \to \hat{\mathbf{w}}^{rlm}$. This is confirmed in the middle experiment. In the left experiment, $m < d$, but $\hat{\mathbf{w}}^{owa}$ still behaves similarly to $\hat{\mathbf{w}}^{rlm}$. In the right experiment, $\hat{\mathbf{w}}^{owa}$ has similar performance as $\hat{\mathbf{w}}^{ave}$ and $\hat{\mathbf{w}}^{boot}$ but over a wider range of λ values.

4×10^{-8} percent of the full training set) and $\hat{\mathbf{w}}^{owa}$ has converged to its final loss value with only 1024 data points per machine (approximately 2.7×10^{-6} percent of the full training set). This justifies our claim that only a small number of data points are needed for the second round of optimization, and so the communication complexity of $\hat{\mathbf{w}}^{owa}$ is essentially the same as $\hat{\mathbf{w}}^{ave}$. The computation is also very fast due to the lower dimensionality and L2 regularization in the second round of optimization. When $n^{owa} = 2^{10}$, computing the merged model took only minutes (including the cross validation time to select λ_2). This time is negligible compared to the approximately 1 day it took to train the models on the individual machines.

Our last experiment shows the performance as we scale the number of machines m. The results are shown in Figure 3.4. Here, $\hat{\mathbf{w}}^{owa}$ performs especially well with low m. For large m, $\hat{\mathbf{w}}^{owa}$ continues to slightly outperform $\hat{\mathbf{w}}^{boot}$ without the need for an expensive model selection procedure to determine the r parameter.

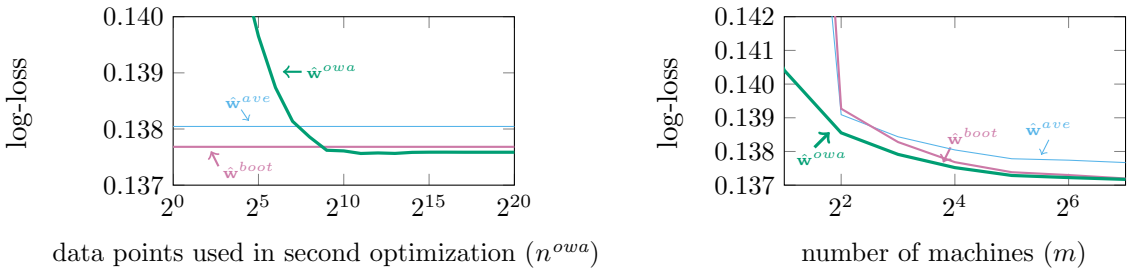

Figure 3.4: OWA's performance on real world ad-click data. (*left*) Relatively few data points are needed in the second round of optimization for $\hat{\mathbf{w}}^{owa}$ to converge. (*right*) Performance of the parallel estimators on advertising data as the number of machines m increases.

3.5 Nonlinear OWA

We now generalize the linear OWA procedures from the previous sections to a nonlinear deep neural network problem. Neural networks are among the most complicated models used in machine learning, and large neural networks require distributed training with many GPUs for days or weeks. Cross validation is rarely used even on the smallest networks due to the computational overhead. A merge method would simplify the distributed training procedure and let practitioners use cross validation to evaluate their models. Due to the complexity of neural networks, very little prior work attempts to merge them together. The only example we know of is McMahan et al. (2017), which uses the naive averaging merge procedure. This section will improve these results using the OWA merge procedure. We first introduce new notation needed for neural network problems, then extend OWA to this setting, and conclude with preliminary experiments on the MNIST data set.

3.5.1 Notation

We assume our network architecture has p layers. For each layer $i \in \{1, ..., p\}$, there is an associated dimension $d_i \in \mathbb{N}$, activation function $\sigma_i : \mathbb{R}^{d_i} \to \mathbb{R}^{d_i}$, and weight matrix

$W_i : \mathbb{R}^{d_i \times d_{i-1}}$. The input to the network is a vector $\mathbf{x} \in \mathbb{R}^{d_0}$. The output of layer i is then recursively given by

$$f_i(\mathbf{x}) : \mathbb{R}^{d_i} = \begin{cases} \mathbf{x} & i = 0 \\ \sigma_i(W_i f_{i-1}(\mathbf{x})) & i > 0 \end{cases} \tag{3.36}$$

and $f_p(\mathbf{x})$ is the final output of the network. In supervised learning problems, we are given a dataset $Z \subset \mathbb{R}^{d_0} \times \mathbb{R}^p$ with mn data points, and our goal is to approximately solve

$$W = \arg\min_W \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{Z}} \ell(\mathbf{y}, f_p(\mathbf{x})) + r(W), \tag{3.37}$$

where ℓ is the loss function and r is the regularization function. We make no assumptions on the training algorithm used to minimize (3.37). For example, stochastic gradient descent may be used with any learning rate schedule, with or without drop out (Srivastava et al., 2014), and with or without batch normalization (Ioffe and Szegedy, 2015). We divide Z into m disjoint smaller datasets $\{Z^{(1)}, ..., Z^{(m)}\}$ each with n points. Each dataset $Z^{(a)}$ is transfered to processor a, which solves the local learning problem

$$W^{(a)} = \arg\min_W \sum_{(\mathbf{x},\mathbf{y}) \in Z^{(a)}} \ell(\mathbf{y}, f_p(\mathbf{x})) + r(W). \tag{3.38}$$

Each machine solves (3.38) without communicating with other machines using any optimizer appropriate for the network architecture and data. Our goal is to develop a merge procedure that combines the $W^{(a)}$ local parameter estimates into a single global parameter estimate with small loss.

3.5.2 Merging neural networks

A simple baseline merge procedure is the naive averaging estimator

$$W_i^{ave} = \frac{1}{m} \sum_{a=1}^{m} W_i^{(a)}. \tag{3.39}$$

Google's recent federated learning architecture uses naive averaging to merge models together that have been independently trained on users' cellphones (McMahan et al., 2017). Because neural networks have more complicated geometry than linear models, the averaging estimator has not been formally analyzed in this setting.

We will define an improved merge procedure based on a weighted average of the local parameter estimates. This requires some tensor notation. We define for each layer i in the network the 3rd-order tensor $W_i^{stacked} : \mathbb{R}^m \times \mathbb{R}^{d_i} \times \mathbb{R}^{d_{i-1}}$, where the (a, b, c)th component of $W_i^{stacked}$ is defined to be the (b, c)th component of $W_i^{(a)}$. In words, $W_i^{stacked}$ is the 3rd-order tensor constructed by stacking the local parameter estimates $W_i^{(a)}$ along a new dimension. We also define the function $\texttt{contract} : (\mathbb{R}^m, \mathbb{R}^m \times \mathbb{R}^{d_i} \times \mathbb{R}^{d_{i-1}}) \to \mathbb{R}^{d_i} \times \mathbb{R}^{d_{i-1}}$ to be the tensor contraction along the first dimension. That is, if $V : \mathbb{R}^m$, then the (b, c)th component of $\texttt{contract}(V, W_i^{stacked})$ is equal to $\sum_{a=1}^m V(a) W_i^{(a)}(b, c)$. In particular, if each component of V equals $1/m$, then $\texttt{contract}(V, W_i^{stacked}) = \frac{1}{m} \sum_{a=1}^m W_i^{(a)} = W_i^{ave}$.

The nonlinear OWA procedure then works as follows. First define the modified

neural network

$$f_i^{mod}(\mathbf{x}) : \mathbb{R}^{d_i} = \begin{cases} \mathbf{x} & i = 0 \\ \sigma_i(W_i^{mod} f_{i-1}^{mod}(\mathbf{x})) & i > 0 \end{cases} \quad \text{where} \quad W_i^{mod} = \texttt{contract}(V_i, W_i^{stacked}). \tag{3.40}$$

Then select a small subset of the data Z^{owa} (i.e. $|Z^{owa}| \ll |Z|$) and train the network f^{mod} over only the parameters V_i. That is, we solve the optimization problem

$$V^{owa} = \arg\min_{V} \sum_{(\mathbf{x},\mathbf{y}) \in Z^{owa}} \ell(\mathbf{y}, f_p^{mod}(\mathbf{x})) + r(W). \tag{3.41}$$

The parameter matrices $W_i^{owa} = \texttt{contract}(V_i^{owa}, W_i^{stacked})$ can then be used in the original neural network. Intuitively, we need only a small number of data points in the optimization of (3.41) because the number of parameters is significantly smaller than in the original optimization (3.37). That is, the dimension of V^{owa} is much less than the dimension of W. When the network contains no hidden layers,

To solve (3.41), we can use the same optimization procedure that we used to solve the local optimization problem (3.38). Our implementation in TensorFlow (Abadi et al., 2016). Given any neural network model as input, it can define an appropriate merge procedure automatically.

3.5.3 Experiments

McMahan et al. (2017) evaluated the naive averaging merge procedure on MNIST, and we perform a similar experiment here. For our neural network, we use a default convolutional

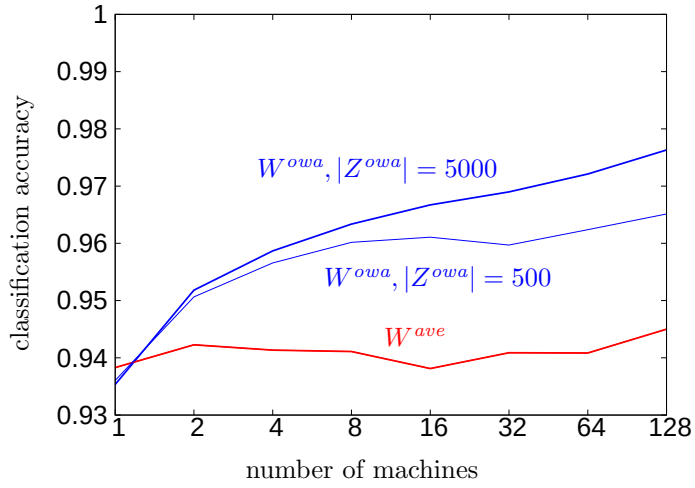

Figure 3.5: Comparison of naive averaging an OWA when merging convolutional neural networks on MNIST data. As expected, naive averaging has the lowest performance, and OWA's accuracy improves when we use more data in the second round of optimization.

neural network (CNN) provided by TensorFlow (Abadi et al., 2016). The network is specifically designed for MNIST and closely related to AlexNet (Krizhevsky et al., 2012). It has 8 layers and is trained using the Adam optimizer (Kingma and Ba, 2014) and dropout (Srivastava et al., 2014). We performed no hyperparameter tuning and simply used the default hyperparameters provided by TensorFlow. MNIST has a standard training set of 55000 data points, and we further divide the training set into 128 subsets containing either 429 or 430 data points. The 10 class labels are evenly distributed throughout the original training set, but we made no effort to ensure they were evenly distributed throughout the subsets. That means on average, each machine has access to only 43 examples from each class, but most machines will have significantly fewer examples for some classes. Under such an extreme paucity of data, it is unlikely for a single machine to be achive high classification accuracy.

Figure 3.5 shows the classification accuracy as the number of machines used varies from 2 to 128. Each experiment is repeated 5 times, and the average is shown. Since the number of data points per machine is fixed, adding more machines adds more data, so we should expect the classification accuracy to increase for a good merge procedure. We see that the OWA algorithm outperforms naive averaging. The OWA algorithm does not perform as well as the oracle network trained on all the data (which has > 0.99 accuracy). This is because of the difficulty of the local learning problems when MNIST is split into such small data sets. McMahan et al. (2017) performs several rounds of naive averaging to improve the accuracy of their federated learners. A similar iterated procedure can be done with OWA, but fewer iterations should be needed due to OWA's improved accuracy.

3.6 Conclusion

OWA is a new mergeable learning algorithm. In the linear case, OWA has strong theoretical guarantees and good practical performance. Preliminary experiments suggest that this practical performance may carry over to complex neural network models, but more extensive large scale experiments need to be done to validate this claim.

Chapter 4

The cover tree

The cover tree is a data structure for fast nearest neighbor queries in arbitrary metric spaces. Cover trees were first introduced by Beygelzimer et al. (2006), and they have seen widespread use in the machine learning community since then (Section 4.3). This chapter improves the cover tree's runtime both in theory and practice. Our first contribution is to improve the cover tree's runtime bounds. As an example, the original cover tree was able to find nearest neighbors in time $O(c_{\text{exp}}^{12} \log n)$, and we improve this bound to $O(c_{\text{hole}}^{4} \log n)$ for i.i.d. data. Here c_{exp} and c_{hole} are measures of the "intrinsic dimensionality" of the data. We will show that on typical datasets $c_{\text{exp}} \approx c_{\text{hole}}$, and we will argue that in pathological data sets c_{hole} more accurately represents our intuitive notion of dimensionality. We further provide practical improvements to the cover tree's implementation. Experiments show that with these improvements, the cover tree is the fastest technique available for nearest neighbor queries on many tasks.

Most work on nearest neighbor queries focuses on Euclidean space, but a major

strength of the cover tree is that it works in the more general setting of metric spaces. Section 4.1 formally introduces metric spaces and their importance in machine learning. Section 4.2 then provides a mathematically detailed discussion of four measures of the "size" of a metric space. These quantities are important to bound the runtime of operations on the cover tree. We review standard results that demonstrate that the expansion constant c_{exp} is not robust to small changes in the dataset. We then review basic properties of the more robust doubling constant. Unfortunately, we show that the doubling constant is not suitable for measuring the difficulty of exact nearest neighbor queries because there exists a data set of size n with doubling dimension 2 that requires $O(n)$ distance computations to find a nearest neighbor. We introduce the hole constant c_{hole} as a dimension that partially captures the robustness of the doubling dimension but is suitable for bounding the runtime of exact nearest neighbor queries. We also provide new results on the aspect ratio of a data set (ratio of largest to smallest distance) that will be needed to bound the runtime of the cover tree. Section 4.3 then reviews the history of techniques for faster nearest neighbor queries. The main theme we emphasize is that the expansion constant is widely used to bound the runtime of exact nearest neighbor queries, and the doubling constant is widely used to bound the runtime of approximate nearest neighbor queries.

Section 4.4 formally describes the original cover tree data structure. We use the mathematical techniques developed in previous sections to provide novel runtime bounds for querying and inserting data points. We also provide a novel algorithm for approximate nearest neighbor queries that improves on the algorithm of Beygelzimer et al. (2006). Section 4.5 then presents the simplified cover tree. The simplified cover tree is more awkward to

analyze theoretically but has a number of practical advantages. In particular: (i) the invariants are simpler; (ii) there are fewer nodes in the tree resulting in smaller constant factors; (iii) we introduce a new invariant that helps maintain tree balance; (iv) we show how to make the tree cache efficient; and (v) we show how to merge two trees together, resulting in a parallel tree construction algorithm and fast cross validation algorithm (based on the results in Chapter 2). Finally, Section 4.6 concludes with a series of five experiments. We use benchmark datasets with the Euclidean distance to show that our simplified cover tree implementation is faster than both existing cover tree implementations and other techniques specialized for Euclidean distance nearest neighbor queries. We also compare the simplified and original cover trees on non-Euclidean protein interaction and computer vision problems.

4.1 Definitions

This section first defines metric spaces and the nearest neighbor problem. A set $\mathcal{X}$ equipped with a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a **metric space** if it obeys the following three properties:

1. **Indiscernability**. For all $x_1, x_2 \in \mathcal{X}$, $d(x_1, x_2) = 0$ if and only if $x_1 = x_2$.

2. **Symmetry**. For all $x_1, x_2 \in \mathcal{X}$, $d(x_1, x_2) = d(x_2, x_2)$.

3. **Triangle inequality**. For all $x_1, x_2, x_3 \in \mathcal{X}$, $d(x_1, x_2) + d(x_2, x_3) \geq d(x_1, x_3)$.

We will use the script $\mathcal{X}$ notation for metric spaces that may be either infinite or finite, and the plain X for metric spaces that must be finite. In particular, X will typically denote a data set sampled from some larger space $\mathcal{X}$, as in the following definition of the nearest

neighbor problem.

Let $\mathcal{X}$ be a metric space, $X \subset \mathcal{X}$, and x be in $\mathcal{X}$ but not X. We call a point $y^* \in X$ a **nearest neighbor** of x if it is contained in $\arg\min_{y \in X} d(x, y)$. In most cases y^* will be unique (so we will often refer to *the* nearest neighbor of x). For all $\varepsilon \geq 0$, we call a point $\hat{y}$ a $(1+\varepsilon)$**-approximate nearest neighbor** (**ann**) of x if $\hat{y}$ satisfies $d(x, \hat{y}) \leq (1+\varepsilon) \cdot d(x, y^*)$. In particular, if $\hat{y}$ is a 1-ann, then $\hat{y}$ is a nearest neighbor.

4.2 Measuring the size of a metric space

The runtime of nearest neighbor queries depends on the dimension of the data, and there are many ways to define the dimension of arbitrary metric spaces. The expansion and doubling dimensions are the two most popular. The expansion dimension can be used to bound the runtime of either exact or approximate nearest neighbor queries, but it is not robust to small changes in the data. This lack of robustness can make runtime bounds based on the expansion dimension trivial (i.e. linear). On the other hand, the doubling dimension can only be used to bound the runtime of approximate nearest neighbor queries, but it is more robust than the expansion dimension. Therefore bounds using the doubling dimension are stronger than those using the expansion dimension. The original runtime bounds for the cover tree were in terms of the expansion dimension, but Section 4.4 will improve these bounds to use the doubling dimension in the case of approximate nearest neighbor searches. This section begins with a detailed discussion of these two dimensions, including important lemmas used in the proofs in Section 4.4. Then we introduce a novel third dimension called the hole dimension. We show in this section that the hole dimension is more robust than the

expansion dimension, and in Section 4.4 the hole dimension is used to bound the runtime of exact nearest neighbor queries. This section concludes with a discussion of the aspect ratio of a metric space. The log of the aspect ratio will bound the height of the cover tree (Lemma 26 in Section 4.4). In this section, we provide a novel bound showing that under mild regularity conditions the aspect ratio of i.i.d. data is $O(n^3)$, and thus the height of the cover tree is $O(\log n)$. No existing results show that the cover tree is well-balanced in this way.

4.2.1 Expansion dimension

The expansion dimension is the only notion of dimensionality that has been used for bounding the runtime of exact nearest neighbor queries. It was introduced by Karger and Ruhl (2002), and was subsequently used in Krauthgamer and Lee (2004), Beygelzimer et al. (2006), Ram et al. (2009), and Curtin et al. (2015). In this section, we define the expansion constant and show that it is not a robust measure of a metric's dimension.

Throughout this section we will work with the metric space $(\mathcal{X}, d)$ where the set $\mathcal{X}$ may be either finite or infinite. We let $B(x, \delta)$ denote the **ball** centered around x of radius δ. That is,

$$B(x, \delta) = \{x' : x' \in \mathcal{X}, d(x, x') \leq \delta\}. \tag{4.1}$$

The expansion dimension of a metric is only valid for metrics that also have a notion of volume, which is given by the function $\mu : \{\mathcal{X}\} \to \mathbb{R}^+$.[1] In finite metric spaces, μX is defined to be the number of points in X. In the Euclidean space $\mathcal{X} = \mathbb{R}^d$, μ is defined to

[1] Formally, μ is a measure and $\{\mathcal{X}\}$ is a σ-algebra on $\mathcal{X}$. Since we are primarily interested in the properties of finite sets, we will not need to formally use measure theory.

be the standard Euclidean volume, and in particular $\mu B(x, \delta) = \Theta(\delta^d)$.

The **expansion dimension** of a metric space $\mathcal{X}$ is defined as

$$\dim_{\text{exp}} = \log_2 c_{\text{exp}}, \tag{4.2}$$

where c_{exp} is called the **expansion constant** and defined as

$$c_{\text{exp}} = \max_{x \in \mathcal{X}, \delta \in \mathbb{R}^+} \frac{\mu B(x, 2\delta)}{\mu B(x, \delta)}. \tag{4.3}$$

In words, the expansion dimension measures how the volume of a ball changes as you increase the radius. The intuition behind why this is a good notion of a metric's dimension is that it agrees with the standard dimension in Euclidean space. This is shown formally in the following standard lemma.

Lemma 8. Let $\mathcal{X} = \mathbb{R}^d$ with the standard L_2 distance function. Then $\dim_{\text{exp}} \mathcal{X} = \Theta(d)$.

Proof (sketch). The volume of a Euclidean ball with radius δ is $\Theta(\delta^d)$. The expansion constant is the maximum of the ratio $\mu B(x, 2\delta)/\mu B(x, \delta)$. In Euclidean space this ratio is independent of x and δ. Letting $\delta = 1$ gives the ratio is $\mu B(x, 2)/\mu B(x, 1) = \Theta(2^d)$. Taking the log gives that $\dim_{\text{exp}} \mathcal{X} = \Theta(d)$. □

Unfortunately, the expansion dimension is known to have many defects which make runtime bounds less useful. The algorithmic consequence of these defects is that runtime bounds based on the expansion dimension are often trivial.

Example 9. A subset of a metric space may have arbitrarily large expansion dimension

compared to the host space. Let $x_i = 2^{-i}$ for all $i \in \mathbb{N}$, and $X_n = \{x_0, x_1, ..., x_n\}$. X_n is a subset of $\mathbb{R}$, which has expansion number $O(1)$, but X_n has expansion number $O(n)$.

Example 10. Adding a single point to a space may change the expansion dimension by an arbitrary amount. Let X be a finite metric space with n elements and expansion constant $c_{\text{exp}} \ll n$, and let $\text{diam}(X)$ be the maximum distance between any two points in X. Construct a new space $Y = X \cup \{y\}$ where y is a point not in X. Define the distance between y and any point in X to be $\text{diam}(X)$. This distance satisfies the triangle inequality, and so Y is a metric space. The expansion dimension of Y is n.

The doubling dimension discussed next does not suffer from either of the defects in Examples 9 or 10. Unfortunately, it cannot be used to measure the difficulty of exact nearest neighbor queries. The hole dimension (introduced in two sections) sits "in between" the doubling dimension and the expansion dimension. It does not suffer from the defect of Example 9, but it does suffer from the defect of Example 10. Unlike the doubling dimension, however, the hole dimension is suitable for measuring the difficulty of exact nearest neighbor queries.

4.2.2 Doubling dimension

The doubling dimension is more robust than the expansion dimension, but cannot be used to provide non-trivial runtime bounds for exact nearest neighbor queries (see Theorem 15 below). It has, however, seen widespread application in bounding the runtime of approximate nearest neighbor queries. Krauthgamer and Lee (2005) showed that the doubling constant is the correct way to measure the difficulty of $(1 + \varepsilon)$-ann queries in the following

sense: a $(1+\varepsilon)$-ann query can be answered in polylogarithmic time using polynomial space if and only if $c_{\text{doub}} = O(\log n)$. No similar result pairing exact nearest neighbor queries to a metric dimension is known.

The doubling dimension is widely used in computer science and mathematics. Before formally defining it, we briefly review these many uses. The term doubling dimension was first used by Gupta et al. (2003) in their study of low dimensional embeddings for nearest neighbor queries, but the main idea was introduced by Assouad (1979) to study fractals.[2] In the context of machine learning, the doubling dimension has found use both statistically and algorithmically. Statistically, the doubling dimension is a standard tool for proving regret bounds. For example, Luxburg and Bousquet (2004) uses the doubling dimension to provide the first large margin bounds for classifiers in metric spaces, and Chapter 27 of Shalev-Shwartz and Ben-David (2014) shows how to bound the Rademacher complexity of a classifier using the doubling dimension. A particularly apropos work in this vein is that of Kontorovich and Weiss (2015), which uses the doubling dimension to derive the first Bayes-consistent 1-nearest neighbor algorithm. Algorithmically, the doubling dimension is used to develop approximation algorithms. Chapter 32 of Goodman et al. (2017) provides a survey of how the doubling dimension is used to approximate a variety of NP-hard problems. In the context of machine learning, Gottlieb et al. (2014b) shows the first approximation algorithm for sample compression, and Gottlieb et al. (2014a) and Gottlieb et al. (2017) both create efficiently realizable approximation algorithms for classification in metric spaces.

The doubling dimension is closely related to the ideas of covering and packing

[2] There are many terms in the literature with essentially the same meaning as the doubling dimension. For example, the terms Assoud dimension and Minkowski dimensions are both used in fractal geometry; and the terms covering number, packing number, ε-nets, and metric entropy are all used in computer science.

numbers, so we introduce these ideas first. A **δ-covering** of a metric space $\mathcal{X}$ is a set $\{x_1, x_2, ..., x_n\} \subseteq \mathcal{X}$ such that for all $x \in \mathcal{X}$, there exists an x_i such that $d(x, x_i) < \delta$. The **δ-covering number** $N_\delta(\mathcal{X})$ is the cardinality of the smallest δ-covering. A **δ-packing** of a metric space $\mathcal{X}$ is a set $\{x_1, x_2, ..., x_M\} \subseteq \mathcal{X}$ such that $d(x_i, x_j) > \delta$ for all $i \neq j$. The **δ-packing number** $M_\delta(\mathcal{X})$ is the cardinality of the largest δ-packing. The covering number and packing number of a set are closely related. We will make heavy use of the following standard lemma in the analysis of the cover tree.

Lemma 11. For any metric space $\mathcal{X}$ and any $\delta > 0$,

$$M_{2\delta}(\mathcal{X}) \leq N_\delta(\mathcal{X}) \leq M_\delta(\mathcal{X}). \tag{4.4}$$

Proof. To prove the first inequality, let P be a 2δ-packing and C be a δ-cover of $\mathcal{X}$. For every point $p \in P$, there must exist a $c \in C$ such that $d(p, c) \leq \delta$. No other $p' \in P$ can also satisfy $d(p', c) \leq \delta$, because then by the triangle inequality

$$d(p', p) \leq d(p', c) + d(p, c) \leq 2\delta, \tag{4.5}$$

which would contradict that P is a 2δ-packing. In other words, for each $c \in C$, there is at most one $p \in P$. So $N_\delta \geq |C| \geq |P| \geq M_{2\delta}$.

To prove the second inequality, let $\mathcal{X}' \subseteq \mathcal{X}$ be a maximal δ-packing. Then there does not exist an $x \in \mathcal{X}$ such that for all $x' \in \mathcal{X}'$, $d(x, x') > \delta$. (Otherwise, $\mathcal{X}' \cup \{x\}$ would be a packing larger than $\mathcal{X}'$.) Hence, $\mathcal{X}'$ is also a δ-cover, and the smallest δ-cover can be no larger. □

The **doubling dimension** of a metric space $\mathcal{X}$ is defined to be

$$\dim_{\text{doub}} = \log_2 c_{\text{doub}}, \tag{4.6}$$

where c_{doub} is called the **doubling constant** and is defined to be

$$c_{\text{doub}} = \max_{x \in \mathcal{X}, r \in \mathbb{R}^+} N_r(B_{\mathcal{X}}(x, 2r)). \tag{4.7}$$

Whereas the expansion dimension measures how the volume of a ball changes as the radius changes, the doubling dimension measures how the packing number of a ball changes. The following lemma shows that the doubling dimension (like the expansion dimension) agrees with the dimension in Euclidean space.

Lemma 12. Let $\mathcal{X} = \mathbb{R}^d$ with the standard L_2 distance function. Then $\dim_{\text{doub}} \mathcal{X} = \Theta(d)$.

Proof (sketch). Follows by direct calculation as in Lemma 8. □

The major advantage of the doubling dimension over the expansion dimension is its robustness. The following standard lemma shows that when a data set changes by a small amount, the doubling dimension also changes by a small amount.

Lemma 13. Let $\mathcal{X}$ be a metric space, $X \subset \mathcal{X}$, and $x \in \mathcal{X}$. Then,

$$c_{\text{doub}}(X \cup \{x\}) \leq c_{\text{doub}} X + 1. \tag{4.8}$$

Proof. Let $y \in X$ and $\delta > 0$. Let C be a minimal δ-covering of $B_X(y, \delta)$. Then the set

$C \cup \{x\}$ must be a δ-covering for $B_{X\cup\{x\}}(y, \delta)$. So we have that

$$N_\delta(B_X(y, \delta)) \leq N_\delta(B_{X\cup\{x\}}(y, \delta)) + 1. \tag{4.9}$$

Taking the maximum of both sides with respect to y and δ gives (4.8). □

Gupta et al. (2003) and Krauthgamer and Lee (2004) provide the following lemma relating the size of the expansion and doubling dimensions.

Lemma 14. Every finite metric $(\mathcal{X}, d)$ satisfies $c_{\text{doub}} \leq c_{\text{exp}}^4$.

Proof. Fix some ball $B(x, \delta)$. We will show that $B(x, \delta)$ can be covered by c_{exp}^4 balls of radius δ. Let Y be a δ-cover of $B(x, 2\delta)$. Then,

$$B(x, 2\delta) \subseteq \bigcup_{y\in Y} B(y, \delta) \subseteq B(x, 4\delta) \tag{4.10}$$

Also, for every $y \in Y$,

$$|B(x, 4\delta)| \leq |B(y, 8\delta)| \leq c_{\text{exp}}^4 |B(y, \delta/2)|. \tag{4.11}$$

We also have that $|B(y, \delta/2)| = 1$. □

This lemma can be used to convert runtime bounds using the doubling dimension into bounds using the expansion dimension. We will not need to directly appeal to this lemma in our proofs. The following theorem shows that the doubling dimension is not suitable for bounding the runtime of exact nearest neighbor queries.

Theorem 15. *There exists an n-point metric with doubling constant* $c_{\text{doub}} = O(1)$ *such that answering exact nearest neighbor queries takes time* $O(n)$*.*

Proof (sketch). Take n points arranged in a circle in $\mathbb{R}^2$, and let the query point x be the center of the circle. Select one point y and reduce the distance from x to y by some small value ϵ, leaving all other values constant. The resulting distance is non-Euclidean, but still satisfies the requirements of a metric space. There is no way to tell which point y was modified in this way without checking all the points, thus the worst case nearest neighbor runtime is $O(n)$. □

4.2.3 Hole dimension

The hole dimension is a novel measure of a set's dimensionality. The idea is to create a dimension that is suitable for measuring the difficulty of exact nearest neighbor queries while being more robust than the expansion dimension. To do this, we modify the definition of the doubling dimension so that we ignore "holes" around each point in the space. In particular, the **hole number** is defined as

$$c_{\text{hole}} = \max_{x \in \mathcal{X}, \epsilon \geq 0, \delta > 0} N_\delta\big(B(x, \epsilon + 2\delta) \setminus B(x, \epsilon + \delta)\big) \tag{4.12}$$

$$\text{s.t.} \quad B(x, \epsilon + \delta) \setminus B(x, \delta) = \{\}. \tag{4.13}$$

The **hole dimension** is then

$$\dim_{\text{hole}} = \log_2 c_{\text{hole}}. \tag{4.14}$$

The value of the hole dimension is related to the doubling dimension by the following lemma.

Lemma 16. For any finite metric space, $c_{\text{doub}} \leq c_{\text{hole}} + 1$.

Proof. The inequality is tight when $\epsilon = 0$ in (4.12). More generally, when the value of ϵ in (4.12) is small, then the hole dimension is close to the doubling dimension. When the value of ϵ is large, then the hole dimension is larger than the doubling dimension. □

We justify calling the hole dimension a "dimension" by the following lemma.

Lemma 17. Let $\mathcal{X} = \mathbb{R}^d$ with the standard L_2 distance function. Then $\dim_{\text{hole}} \mathcal{X} = \Theta(d)$.

Proof. For all points $x \in \mathcal{X}$, the value of ϵ in (4.12) is 0. Therefore the hole dimension is equivalent to the doubling dimension. The result follows by Lemma 12, which bounds the doubling dimension of Euclidean space. □

The following example shows that the hole dimension is more robust than the expansion dimension.

Example 18. The hole dimension of a metric space may be arbitrarily smaller than the expansion dimension of the space. We show this using the same metric space as from Example 9. Let $x_i = 2^{-i}$ for all $i \in \mathbb{N}$, and $X_n = \{x_0, x_1, ..., x_n\}$. Then X_n has expansion number $O(n)$ and hole dimension $O(1)$.

In Section 4.4, we provide novel runtime bounds for the cover tree based on the hole dimension. We show that the runtime in terms of the hole dimension is significantly smaller than the runtime in terms of the expansion dimension ($O(c_{\text{hole}}^4)$ vs $O(c_{\text{exp}}^{12})$. Combined with the fact that c_{hole} can be much smaller than c_{exp}, this implies a bound that is potentially much stronger for certain datasets.

4.2.4 Aspect ratio

Unlike the expansion, doubling, and hole constants, the aspect ratio should not be thought of as a measure of dimension. This is because the aspect ratio is only applicable to finite sets. Instead, it should be thought of as a measure of how evenly spread the points in a metric space are. We will use the aspect ratio to bound the depth of the cover tree.

The **diameter** of $\mathcal{X}$ is the maximum distance between any two points. In notation,

$$\text{diam}(\mathcal{X}) = \max_{x_1, x_2 \in \mathcal{X}} d(x_1, x_2). \tag{4.15}$$

The **codiameter** of $\mathcal{X}$ is the minimum distance between any two points. In notation,

$$\text{codiam}(\mathcal{X}) = \min_{x_1 \neq x_2 \in \mathcal{X}} d(x_1, x_2). \tag{4.16}$$

The **aspect ratio** of $\mathcal{X}$, denoted by $\Delta_{\mathcal{X}}$, is the ratio of the diameter to the codiameter. Sets with small aspect ratio are called **fat**, and sets with large aspect ratio are called **skinny**.

There is little necessary relationship between the aspect ratio and the inherent dimension of a space. We emphasize this point with three examples.

Example 19. Let $\mathcal{Y} = \{y_1, ..., y_n\}$ be the discrete metric space of size n; that is,

$$d(y_i, y_j) = \begin{cases} 0 & i = j \\ 1 & \text{otherwise} \end{cases}. \tag{4.17}$$

Then the aspect ratio of $\mathcal{Y}$ is 1 (i.e. as small as possible), and both the expansion and

doubling constants of $\mathcal{Y}$ are $n-1$ (i.e. arbitrarily large).

Example 20. Now construct the set $\mathcal{Y}' = \{y'_1, y'_2, y'_3\}$. Let $r > 2$, and define the distance function to be

$$d(y'_i, y'_j) = \begin{cases} 0 & i = j \\ 1 & i = 1, j = 2 \\ r & i = 1, j = 3 \\ r & i = 2, j = 3 \end{cases}. \tag{4.18}$$

Then the aspect ratio is r (i.e. arbitrarily large), but the expansion constant is always 2 and the doubling constant always 1.

Example 21. Let $Y = \{\frac{1}{2}, \frac{1}{4}, ..., \frac{1}{2^n}\}$, and $d(y_1, y_2) = |y_1 - y_2|$. Then the aspect ratio is 2^{n-1} (i.e. arbitrarily large), the expansion constant is $n-1$ (i.e. arbitrarily large), and the doubling constant is 1.

The following lemma provides the only known relationship.

Lemma 22 (Krauthgamer and Lee (2004))**.** For any metric space $\mathcal{X}$, we have that $|\mathcal{X}| \leq \Delta_{\mathcal{X}}^{O(\dim_{\text{doub}}\mathcal{X})}$.

The aspect ratio and the expansion number share the unattractive property that adding a single point to a dataset can increase these quantities arbitrarily. Under mild assumptions, however, we can show that this is unlikely to happen. Specifically, let $\mathcal{X}$ be a metric space, and let $X = \{x_1, ..., x_n\} \subset \mathcal{X}$ be a sample of n i.i.d. points from a distribution $\mathcal{D}$ over $\mathcal{X}$. Our goal is to show that the aspect ratio grows polynomially in n. We will later

show that the log of the aspect ratio bounds the depth of the cover tree (see Lemma 26), and so the depth of the cover tree will be logarithmic in n.

We begin by bounding the diameter of X. We say the distribution $\mathcal{D}$ has **finite expected distance** if there exists an $\bar{x} \in \mathcal{X}$ such that $\mu = \mathbb{E}\, d(\bar{x}, x_i)$ is finite. Note that this is a mild condition satisfied by most standard distributions on Euclidean space. For example, the uniform, Gaussian, exponential, Weibull, and Pareto distributions all have finite expected distance. Notice that the Weibull and Pareto distributions have heavy tails but still satisfy the condition. One standard distribution which does not satisfy this property is the Cauchy distribution (the distribution of the reciprocal of a standard Gaussian random variable). The following lemma shows that finite expected distance is a sufficient condition for the diameter to grow polynomial in n.

Lemma 23. Let $\mathcal{X}$ and X be defined as above, and assume that $\mathcal{D}$ has finite expected distance. Then, $\mathbb{E}\,\mathrm{diam}(X) \leq 2n\mu$.

Proof. By the triangle inequality, we have that

$$\mathrm{diam}(X) = \max_{i,j} d(x_i, x_j) \leq \max_{i,j}(d(\bar{x}, x_i) + d(\bar{x}, x_j)) = 2 \max_i d(\bar{x}, x_i).$$

We now remove the max using the union bound. This gives

$$\Pr\left[\mathrm{diam}(X) > t\right] \leq \Pr\left[\max_i 2d(\bar{x}, x_i) > t\right] \leq \sum_{i=1}^{n} \Pr\left[2d(\bar{x}, x_i) > t\right] = n \Pr\left[2d(\bar{x}, x_1) > t\right].$$

The rightmost equality follows because the x_is are i.i.d. Finally, since the distances are

always nonnegative, we have that

$$\mathbb{E}\,\mathrm{diam}(X) = \int_0^\infty \Pr\left[\mathrm{diam}(X) > t\right]\,\mathrm{d}t \leq \int_0^\infty n\Pr\left[2d(\bar{x}, x_1) > t\right]\,\mathrm{d}t = 2n\,\mathbb{E}\,d(\bar{x}, x_1).$$

□

Next we show that the codiameter cannot shrink too fast. We say that the distribution $\mathcal{D}$ has B**-bounded density** if for all $x \in \mathcal{X}$, the density of $d(x, x_i)$ is bounded by B. An immediate consequence is that

$$\max_{x\in\mathcal{X}} \Pr\left[d(x, x_i) \leq t\right] \leq Bt. \tag{4.19}$$

All the standard distributions in Euclidean space satisfy this condition. The following lemma shows that having a B-bounded density is sufficient to lower bound the codiameter.

Lemma 24. Let $\mathcal{X}$ and X be defined as above, and assume that $\mathcal{D}$ has B-bounded density. Then, $\mathbb{E}\,\mathrm{codiam}(X) \geq (2n^2B)^{-1}$.

Proof. We have that

$$\Pr\left[\mathrm{codiam}(X) \leq t\right] = \Pr\left[\min\{d(x_i, x_j) : i \in \{1, ..., n\}, j \in \{i+1, ..., n\}\} \leq t\right] \tag{4.20}$$

$$\leq \sum_{i=1}^{n}\sum_{j=i+1}^{n} \Pr\left[d(x_i, x_j) \leq t\right] \tag{4.21}$$

$$\leq n^2 \max_{x\in X} \Pr\left[d(x_1, x) \leq t\right] \tag{4.22}$$

$$\leq n^2 Bt \tag{4.23}$$

Equation (4.21) follows from the union bound, (4.22) from the fact that the x_is are i.i.d., and (4.23) from the definition of B-bounded. We further have that since probabilities are always no greater than 1,

$$\Pr\left[\operatorname{codiam}(X) \leq t\right] \leq \min\{1, n^2 B t\}. \tag{4.24}$$

Finally, since $\operatorname{codiam}(X)$ is nonnegative, we have that

$$\begin{aligned}
\mathbb{E}\operatorname{codiam}(X) &= \int_0^\infty \left(1 - \Pr\left[\operatorname{codiam}(X) \leq t\right]\right) \mathrm{d}t && (4.25)\\
&\geq \int_0^\infty \left(1 - \min\{1, n^2 B t\}\right) \mathrm{d}t && (4.26)\\
&= \int_0^{(n^2 B)^{-1}} \left(1 - n^2 B t\right) \mathrm{d}t && (4.27)\\
&= \frac{1}{2n^2 B}. && (4.28)
\end{aligned}$$

□

An immediate consequence of Lemmas 23 and 24 is the following bound on the aspect ratio.

Lemma 25. Let $\mathcal{X}$ and X be defined as above. Assume that $\mathcal{D}$ has finite expected distance and B-bounded density. Then, $\mathbb{E}\,\Delta \leq 4B\mu n^3$.

4.3 Review of methods for faster nearest neighbors

This section explores three strategies for faster nearest neighbor queries: sample compression, embedding, and spatial¿ data structures. Sample compression works well when there are many data points (called **tall** data sets). Embedding works well when the data is high

dimensional (called **wide** data sets). Both techniques are simple and easy to implement, but are only suitable for approximate nearest neighbor queries. Spacial data structures work well for both tall and wide data, and they support both exact and approximate queries. Their downside is that they are typically more complicated to implement.

A recurring theme throughout this section is that the doubling dimension is widely used in approximation bounds and the expansion dimension widely used in exact bounds. Krauthgamer and Lee (2005) showed that the doubling constant exactly characterizes the difficulty of approximate nearest neighbor queries. In particular, they show that a $(1+\varepsilon)$-ann can be found in polylogarithmic time using polynomial space if and only if $c_{\text{doub}} = O(\log n)$. Despite the expansion constant's use in exact nearest neighbor queries, no similar result is known. This lack motivates our introduction of the hole constant to characterize the difficulty of nearest neighbor queries.

4.3.1 Sample compression

The method of compression was introduced by Hart (1968), and is arguably the simplest method for speeding up nearest neighbor classification. Given a data set X, compression selects a subset X' of X satisfying two properties: First, $|X'|$ should be much smaller than $|X|$. This ensures that nearest neighbor queries in X' take much less time than queries in X. Second, given any query point y, the nearest neighbor of y in X and X' should have the same class label. Finding an optimal subset X' for exact queries is known to be NP-hard (Zukhba, 2010), so a number of heuristics have been developed for approximate queries. Hart (1968) gave a heuristic that runs in time $O(n^3)$, and Angiulli (2005) provided an improved heuristic that runs in time $O(n^2)$. Unfortunately, neither of these heuristics provides any guarantee

that the nearest neighbors in X' are likely to have the same class label as the nearest neighbor in X. Gottlieb et al. (2014b) introduced the first sample compression scheme providing an approximation guarantee. They also showed that no compression scheme can perform significantly better than theirs unless P=NP. Their algorithm works in arbitrary metric spaces and has an impressive $O(\text{poly}(c_{\text{doub}})n)$ runtime. Sample compression can be easily combined with other techniques to further speed up nearest neighbor queries on the set X'.

4.3.2 Embeddings and locality sensitive hashing

High dimensional data can be slow in two ways. First, calculating distances in high dimensions takes longer than calculating distances in low dimensions. But more importantly, the **curse of dimensionality** causes high dimensional data to be "less structured" than low dimensional data. In particular, the variation in the distance between data points decreases as the dimensionality increases. This causes spatial¿ data structures to become less efficient. Embeddings provide a technique to reduce these two effects of high dimensional data. Specifically, given a data set X, an embedding is a function $f : X \to X'$ where the target X' is a lower dimensional space. If the embedding f has low distortion (i.e., distances in the low dimensional space are close to distances in the high dimensional space), then exact nearest neighbor queries in the low dimensional space serve as approximate queries in the high dimensional space.

The target space is typically a low dimensional L_p space. After applying the embedding, standard techniques for low dimensional L_p spaces like kd-trees can be used to quickly find the nearest neighbor. The most celebrated embedding is given by the Johnson-

Lindenstrauss (JL) lemma in L_2 space. The JL lemma states that if f is a random projection from $\mathbb{R}^d \to \mathbb{R}^{O(1)}$, then f embeds any set of n data points with $O(\log n)$ distortion with high probability (Johnson and Lindenstrauss, 1984). Further research has either simplified the proof of the JL lemma (Dasgupta and Gupta, 2003; Baraniuk et al., 2008) or reduced the computation needed to generate and apply the embedding (Achlioptas, 2001). Indyk and Motwani (1998) and Gionis et al. (1999) introduced the term locality sensitive hashing (LSH) for random projections where the target space is a low dimensional L_1 space (sometimes called the Hamming cube). Wang et al. (2014) and Wang et al. (2016a) provide good surveys on LSH techniques with particular emphasis on embeddings that depend on the data (in contrast to the the JL embedding which does not).

Low dimensional L_p spaces are also popular targets when the host space is a generic metric space. Bourgain (1985) showed that any n-point doubling metric can be embedded into Euclidean space with $O(\text{poly}(c_{\text{doub}}) \log n)$ distortion. Gupta et al. (2003) improve on Bourgain's result by showing an embedding with $O(\log c_{\text{doub}} \log n)$ distortion, and Krauthgamer et al. (2004) further improve this result by showing an embedding with $O(\sqrt{\log c_{\text{doub}} \log n})$ distortion. The best known embeddings into Euclidean space are due to Chan et al. (2010). They show that every n point metric can be embedded into $O(\log c_{\text{doub}} \log \log n)$ dimensional Euclidean space with $O(\log n / \sqrt{\log \log n})$ distortion. It is unfortunately not possible to embed metric spaces into L_1 or L_2 spaces with arbitrarily small distortion; however, Neiman (2016) shows how to embed finite metrics into L_∞ with arbitrary small distortion.

Embedding techniques are widely used in practice because they are easy to imple-

ment and work well in real world data. Somewhat surprisingly, however, they are provably non-optimal. ODonnell et al. (2014) provide lower bounds on the quality of LSH embeddings of Euclidean data, and Andoni et al. (2014) proposed a data structure with performance better than these lower bounds.

4.3.3 Spacial data structures

The simplest and oldest of spatial¿ data structure is the **ball tree** (Omohundro, 1989). The ball tree is a type of binary search tree that divides data points into subtrees using a number of possible heuristic rules. Although attractive for its simplicity, ball trees provide only the trivial runtime guarantee that queries will take time $O(n)$. In practice, a good choice of heuristic can greatly speed up some queries, and a number of heuristics have been developed. See Zezula et al. (2006) and Mao et al. (2016) for surveys. Despite the lack of guarantees, ball trees continue to be used in practice. For example, they are the only method for fast nearest neighbor queries in metric spaces provided in Python's popular `scikit-learn` library (Pedregosa et al., 2011). Unfortunately there are no detailed experimental results justifying the use of these heuristic data structures over methods with more theoretical justification.

Clarkson (1997) introduced the first data structure to provide theoretical guarantees. Clarkson's data structure, called the **M(S,Q)** solved the all-nearest neighbor problem in expected time $O(\text{poly}(c_{\text{doub}})n(\log n)^2(\log \Delta)^2)$ using space $O(n \log \Delta)$. The M(S,Q) structure works only in probability and requires that query data points have a similar distribution to the data used in construction. Clarkson also introduced a novel notion of dimensionality called the γ-dominator bound. This bound is closely related to the dou-

bling constant, but somewhat harder to use in practice. No subsequent work uses this bound on dimension. The next advance came when Karger and Ruhl (2002) introduced the **metric skip list** and the expansion dimension. This data structure requires $O(n \log n)$ space, finds nearest neighbors in time $O(\text{poly}(c_{\text{exp}}) \log n)$, and inserts new data in time $O(\text{poly}(c_{\text{exp}}) \log n \log \log n)$. The expansion dimension is not as good a notion of dimensionality as the doubling dimension.

The **navigating net** was the first data structure to use the doubling dimension. It requires quadratic space, polynomial dependency on the doubling constant, and logarithmic dependence on the aspect ratio (Krauthgamer and Lee, 2004). The navigating net inspired a flurry of subsequent results. Hildrum et al. (2004) modified the navigating net into a randomized data structure requiring only linear space. Krauthgamer and Lee (2005) further improve on the navigating net by replacing the logarithmic dependence on the aspect ratio with a logarithmic dependence on the number of data points. Har-Peled and Mendel (2006) provides a data structure for approximate nearest neighbors. Cole and Gottlieb (2006) provides a data structure for approximate nearest neighbor search with dynamic point sets that does not depend on the aspect ratio.

Beygelzimer et al. (2006) introduce the **cover tree**, which also uses only linear space but is fully deterministic. The downside of the cover tree is that the analysis is in terms of the less robust expansion constants. Ram et al. (2009) and Curtin et al. (2015) improve the runtime bounds of cover trees and extend the use of cover trees to non-nearest neighbor problems. Curtin et al. (2013c) provide a generic framework for solving distance problems. Of all the spatial¿ data structures, the cover tree has seen the most widespread adoption.

For example, cover trees have been used to speed up support vector machines (Segata and Blanzieri, 2010), dimensionality reduction (Lisitsyn et al., 2013), gaussian processes (Moore and Russell, 2014), and reinforcement learning (Tziortziotis et al., 2014). This widespread adoption is likely due to the fast, easy to use reference implementation provided by Beygelzimer et al. (2006).

4.4 The original cover tree

The cover tree data structure was first described by Beygelzimer et al. (2006). The original presentation was in terms of an infinitely large tree, only a finite subset of which actually gets implemented. In this section we provide an alternative (but equivalent) definition that is more concrete. This presentation more closely matches the implementation in code and serves to highlight the differences between the original cover tree and the simplified cover tree presented in the next section. In addition to presenting the original cover tree, we present a novel algorithm for $(1 + \varepsilon)$-ann queries on the original cover tree that improves the runtime of the algorithm developed by Beygelzimer et al. (2006).

A **cover tree** is a tree data structure where each node in the tree contains a single point in the underlying metric space. When needed for disambiguation, $\text{dp}(p)$ will refer to the data point stored at node p; however, when clear from context we will omit the dp function. Thus the distance between the two points stored in nodes p_1 and p_2 in a cover tree can be specified by either $d(\text{dp}(p_1), \text{dp}(p_2))$ or more succinctly as $d(p_1, p_2)$. A valid cover tree must satisfy the following three invariants.

1. **Nesting invariant**. Every node p has an associated integer $\text{level}(p)$. For all nodes

$q \in \text{children}(p)$, $\text{level}(q) = \text{level}(p) - 1$. If $\text{children}(p)$ is non-empty, then $\text{children}(p)$ contains a node with the same data point as p.

2. **Covering invariant**. Every node p has an associated real number $\text{covdist}(p) = 2^{\text{level}(p)}$. For all nodes $q \in \text{children}(p)$, $d(p, q) \leq \text{covdist}(p)$.[3]

3. **Separating invariant**. For all nodes $q_1, q_2 \in \text{descendants}(p)$ satisfying $\text{level}(q_1) = \text{level}(q_2) = i$, $d(q_1, q_2) \geq 2^i$.

It will be useful to define the function

$$\text{maxdist}(p) = \underset{q \in \text{descendants}(p)}{\arg\max} \; d(p, q). \tag{4.29}$$

In words, $\text{maxdist}(p)$ is the greatest distance from p to any of its descendants. This value is upper bounded by $2^{\text{level}(p)+1}$, and its exact value can be cached within the data structure.

4.4.1 Properties of the cover tree

Before we present algorithms for manipulating the cover tree, we present two lemmas that bound the shape of the tree. These lemmas are a direct consequence of the cover tree's invariants and motivate the invariants' selection. Beygelzimer et al. (2006) present similar lemmas bounding the shape of the cover tree in terms of the expansion number. The bounds in this section, however, are in terms of the aspect ratio and doubling number.

Lemma 26. Let p be any non-leaf node in a cover tree. Denote by $\text{height}(p)$ the maximum number of edges between p and any of its descendants. We have that $\text{height}(p) \leq \log_2 \Delta_X$.

[3] Beygelzimer et al. (2006) observe that practical performance is improved on most datasets by redefining $\text{covdist}(p) = 1.3^{\text{level}(p)}$. All of our experiments use this modified definition.

Proof. We will show that the following chain of inequalities holds:

$$\Delta_X = \frac{\text{diam}(X)}{\text{codiam}(X)} \overset{(1)}{\geq} \frac{\text{diam}(X)}{\text{covdist}(p)/2^{\text{height}(p)-1}} \overset{(2)}{\geq} \frac{\text{diam}(X)}{\text{diam}(X)/2^{\text{height}(p)}} = 2^{\text{height}(p)}. \quad (4.30)$$

Solving for height(p) then gives the desired result. For inequality (1), consider a point q that is exactly i edges away from p. By the covering invariant,

$$d(q, \text{parent}(q)) \leq \text{covdist}(\text{parent}(q)) \leq \text{covdist}(\text{parent}(\text{parent}(q)))/2 \leq \text{covdist}(p)/2^{i-1}. \quad (4.31)$$

In particular, if ℓ is a leaf node height(p) edges away from p, then $\text{codiam}(X) \leq d(p, \ell) \leq 2^{\text{height}(p)-1}$. For inequality (2), observe that there must exist a child q of p such $d(p, q) \geq \text{covdist}(p)/2$. Otherwise, level$(p)$ could be reduced by one, which would violate the leveling invariant. Therefore, $\text{diam}(X) \geq d(p, q) \geq \text{covdist}(p)/2$. □

Lemma 27. For every node p in a cover tree, we have that $|\text{children}(p)| \leq c_{\text{doub}}^2$.

Proof. To simplify notation, we let $\delta = \text{covdist}(p)$. The separating invariant ensures that the children of p form a $\delta/2$-packing of $B(p, \delta)$. So by the definition of $M_{\delta/2}$ and Lemma 11, we have

$$|\text{children}(p)| \leq M_{\delta/2}(B(p, \delta)) \leq N_{\delta/4}(B(p, \delta)). \quad (4.32)$$

We now show that $N_{\delta/4}(B(p, \delta)) \leq c_{\text{doub}}$. Let Y be a $\delta/2$-covering of $B(p, \delta)$. For each $y_i \in Y$, let Y_i be a minimum $\delta/4$-covering of $B(y_i, \delta/2)$. The union of the Y_is is a $\delta/4$-covering of $B(p, \delta)$. There are at most c_{doub} Y_is, and each Y_i contains at most c_{doub} elements. So their union contains at most c_{doub}^2 elements. □

4.4.2 Approximate nearest neighbor query for a single point

We present two algorithms for answering $(1+\varepsilon)$-ann queries. The first is shown in Algorithm 9 as the `findnn_orig` function. This algorithm was first presented by Beygelzimer et al. (2006) in the original cover tree paper. Here, we provide an improved runtime analysis. Whereas Beygelzimer et al. (2006) show that exact nearest neighbor queries can be found in time $O(c_{\text{exp}}^{12} \log n)$, we show that exact nearest neighbor queries can be found in time $O(c_{\text{hole}}^{4} \log n)$. Lemma 16 showed that $c_{\text{hole}} \leq c_{\text{exp}}$, so this is a major improvement. The second algorithm is a novel modification to `findnn_orig` that improves the runtime and is described in the next subsection.

The `findnn_orig` function finds a $(1+\varepsilon)$-ann of a data point x in a set of cover trees P. The function would typically be called with only a single tree in the input set P; but in subsequent recursive calls, P may contain many subtrees. On each call, `findnn_orig` iterates over Q (the children of nodes in P), and constructs a set $Q_x \subseteq Q$ of subtrees that will be recursively searched. We typically have that $|Q_x| \ll |Q|$, so only a small portion of the tree is searched. We now prove the algorithm's correctness and runtime. The correctness proof is essentially the same as that of Beygelzimer et al. (2006), but the runtime proof is novel.

Theorem 28. `findnn_orig`$(\{p\}, x, \varepsilon)$ *returns a* $(1+\varepsilon)$*-ann of* x *in* data(p)*. In particular,* `findnn_orig`$(\{p\}, x, 0)$ *returns the exact nearest neighbor of* x*.*

Proof. We show that every $\hat{q}$ satisfying the if statement in line 3 is a $(1+\varepsilon)$-ann. For the first case where $Q = \{\hat{q}\}$, we show that $\hat{q}$ must be the true nearest neighbor. Let y^* denote

Algorithm 9 `findnn_orig`(set of cover trees P, query point x, tolerance ε)

returns a $(1+\varepsilon)$-ann of x in data(P)

1: let $Q = \{q : q \in \text{children}(p), p \in P\}$
2: let $\hat{q} = \arg\min_{q \in Q} d(q, x)$
3: **if** $Q = \{\hat{q}\}$ **or** $d(x, \hat{q}) \geq 2 \cdot \text{covdist}(\hat{q})(1 + 1/\varepsilon)$ **then**
4: **return** $\hat{q}$
5: **else**
6: let $Q_x = \{q : q \in Q, d(x, q) \leq d(x, \hat{q}) + \text{maxdist}(q)\}$
7: **return** `findnn_orig`(Q_x, x, ε)
8: **end if**

the true nearest neighbor of x in data(p). Assume for induction that y^* is in data(P) and hence in data(Q). Let q^* denote the node in Q that contains y^*. Then we have that

$$d(x, q^*) \leq d(x, y^*) + d(y^*, q^*) \leq d(x, \hat{q}) + \text{maxdist}(q^*). \tag{4.33}$$

The set Q_x is defined to be all elements of q satisfying (4.33). So q^* is always included in Q_x, which becomes P in subsequent recursive calls.

Now consider the case where $d(x, \hat{q}) \geq 2 \cdot \text{covdist}(\hat{q})(1 + 1/\varepsilon)$. We have,

$$\begin{aligned} d(x, \hat{q}) &\leq d(x, q^*) && (4.34)\\ &\leq d(x, y^*) + d(y^*, q^*) && (4.35)\\ &\leq d(x, y^*) + 2 \cdot \text{covdist}(q^*) && (4.36)\\ &\leq d(x, y^*) + \frac{d(x, \hat{q})}{1 + 1/\varepsilon} && (4.37)\\ &= (1 + \varepsilon) d(x, y^*) && (4.38) \end{aligned}$$

Line (4.35) is the triangle inequality, line (4.36) uses the maximum distance between any

node and a descendent, line (4.37) uses the condition in the if statement, and line (4.38) follows from algebraic manipulations. □

Theorem 29. *For any $\varepsilon \geq 0$,* `findnn_orig`$(\{p\}, x, \varepsilon)$ *finds a $(1+\varepsilon)$-ann in time $O(c_{\text{hole}}^4 \log \Delta)$. In particular,* `findnn_orig` *finds exact nearest neighbors in this time.*

Proof. The total runtime is bounded by the product of the number of recursive calls and the time spent in each call. The number of recursions is bounded by the height of the tree because `findnn_orig` descends one level of the tree with each recursion. This height is bounded by $O(\log \Delta)$ in Lemma 26. The cost of each call is $O(|Q|)$, since we perform one distance computation for each node in Q. The size of Q is the size of P times the number of children per node. The maximum number of children per node is $O(c_{\text{doub}}^2) = O(c_{\text{hole}}^2)$ by Lemma 27. The size of P is the same as the size of Q_x in the previous recursion. So we have that the overall runtime is $O(|Q_x| c_{\text{doub}}^2 \log \Delta)$, and all that remains is to bound $|Q_x|$.

Let $\gamma = d(x, \hat{q})$ and $\delta = \text{covdist}(\hat{q})$. By definition of Q_x on line 6, all $q \in Q_x$ satisfy

$$d(x, q) \leq d(x, \hat{q}) + \text{maxdist}(q) \leq \gamma + 2\delta. \tag{4.39}$$

So Q_x is a subset of $A(x, \gamma, 2\delta)$, and by the global separating invariant Q_x is a δ-packing of this annulus. By Lemma 11,

$$|Q_x| \leq M_\delta A(x, \gamma, 2\delta) \leq N_{\delta/2} A(x, \gamma, 2\delta) \leq c_{\text{hole}}^2. \tag{4.40}$$

The overall runtime is then bounded by $O(c_{\text{hole}}^4 \log \Delta)$. □

Algorithm 10 `findnn`(set of cover trees P, query point x, tolerance ε)

returns a $(1+\varepsilon)$-ann of x in data(P) faster than `findnn_orig`

let $Q = \{q : q \in \text{children}(p), p \in P\}$
let $\hat{q} = \arg\min_{q \in Q} d(q, x)$
if $Q = \{\hat{q}\}$ **or** $d(x, \hat{q}) \geq 2 \cdot \text{covdist}(\hat{q})(1 + 1/\varepsilon)$ **then**
    return $\hat{q}$
else
    let $Q_x = \{q : q \in Q, d(x, q) \leq d(x, \hat{q})\ /(1+\varepsilon)\ + \text{maxdist}(q)\} \cup \{\hat{q}\}$
    return `findnn`(Q_x, x, ε)
end if

4.4.3 Improved approximate nearest neighbor query for a single point

Algorithm 10 presents the `findnn` function, which is a novel improvement to the `findnn_orig` algorithm. The changes are minor and highlighted in yellow. Recall that the set Q_x contains all the points that will be descended on each recursive call. The improved function `findnn` uses a smaller set Q_x, reducing the total runtime of the algorithm. Proving the correctness of `findnn` requires an entirely different strategy.

Theorem 30. `findnn`$(\{p\}, x, \varepsilon)$ *returns an* $(1+\varepsilon)$*-ann of* x *in* p.

Proof. Let y^* denote the true nearest neighbor of x in data(p). We will show that on each recursion, either $y^* \in Q_x$ or $\hat{q}$ is an $(1+\varepsilon)$-ann. Since the algorithm only terminates when Q_x is empty, the returned $\hat{q}$ must be an $(1+\varepsilon)$-ann.

In the first recursive call where $y^* \notin \text{data}(Q_x)$, we must have that $y^* \in \text{data}(Q)$. Let q^* be the node in Q such that $y^* \in \text{data}(q^*)$. Then by the definition of Q_x, we have that

$$d(x, q^*) > d(x, \hat{q})/\varepsilon + \text{maxdist}(q^*). \tag{4.41}$$

Using the triangle inequality on the left and the definition of maxdist on the right gives

$$d(x, y^*) + d(y^*, q^*) \geq \overset{(4.41)}{\cdots} \geq d(x, \hat{q})/\varepsilon + d(y^*, q^*). \tag{4.42}$$

Subtracting $d(y^*, q^*)$ from each side gives

$$d(x, y^*) \geq d(x, \hat{q})/\varepsilon, \tag{4.43}$$

and so $\hat{q}$ is an $(1+\varepsilon)$-ann. In the recursive call on line 7, we call `findnn` with $P = Q_x \cup \{\hat{q}\}$ to ensure that on all future iterations, the distance between $\hat{q}$ and x is non-increasing. □

4.4.4 Inserting a single point

Our insertion algorithm is essentially the same as that of Beygelzimer et al. (2006), but we improve the runtime bounds. Whereas they show runtime bounds of $O(c_{\text{exp}}^6 \log n)$, we show a bound of $O(c_{\text{doub}}^3 \log n)$. Notice that our bound for insertion is in terms of the doubling dimension, which is the strongest notion of intrinsic dimensionality available for metric spaces. In particular, Lemma 16 shows that $c_{\text{doub}} \leq c_{\text{exp}}$, and Example 21 shows that c_{doub} can be arbitrarily smaller than c_{exp}.

The algorithm for inserting points is split into two functions. The most important is `insert_loop` (Algorithm 12). Like `findnn`, `insert_loop` recursively traverses a set of input trees to find a suitable location for insertion. On each iteration, a set Q_x is created that contains all the subtrees that will be descended. The global separating invariant requires that we traverse several subtrees simultaneously to ensure that the distance between x is

Algorithm 11 `insert`(cover tree p, data point x)

1: $\hat{q} = \arg\min_{q \in Q} d(q, x)$
2: **if** $d(p, x) > \text{covdist}(p)$ **then**
3: create a cover tree rooted at x with level $= \lceil \log_2 d(p, x) \rceil$ and children $= \{p\}$
4: **else if** $d(\hat{q}, x) \geq \text{covdist}(p)/2$ **then**
5: insert x as a child into p
6: **else**
7: `insert_loop`(children$(p), x$)
8: **end if**

Algorithm 12 `insert_loop`(set of cover trees P, data point x)

Precondition 1: $d(x, \hat{p}) \leq \text{covdist}(\hat{p})$
Precondition 2: P contains all nodes at level i with distance to x less than 2^i

1: let $Q = \{q : q \in \text{children}(p), p \in P\}$
2: let $\hat{q} = \arg\min_{q \in Q} d(q, x)$
3: **if** $d(\hat{q}, x) \geq \text{covdist}(\hat{q})$ **then**
4: let $\hat{p} = \arg\min_{p \in P} d(p, x)$
5: insert x as a child into $\hat{p}$
6: **else**
7: let $Q_x = \{q : q \in Q, d(q, x) \leq \text{covdist}(q)\}$
8: `insert_loop`(Q_x, x)
9: **end if**

not too close to other nodes on the same level in a different subtree. (A simple depth first search as done in Algorithm 13 for the simplified cover tree would not ensure this property.) `insert_loop` requires that the inserted data point be "close enough" to one of the sub trees to be inserted as a child. This prerequisite will not be satisfied by all data, so `insert` (Algorithm 11) is the main interface for insertion. It handles the case when the inserted data point is relatively far away, then calls `insert_loop` if the data point is close and a recursive descent is required. We now prove the correctness and runtime of insertion.

Theorem 31. *If p is a valid cover tree, then* `insert`(p, x) *inserts x into p.*

Proof. If x is too far from p to be a child, then line 3 creates a new node at x with p as its

only child. The level of this new node is calculated to exactly satisfy the covering invariant. If x can be inserted as a child of p without breaking the separating invariant, line 5 does so. Otherwise, x must be inserted as a descendent of some $p \in P$. The fact that x did not satisfy the condition on line 4 means that children(p) and x must satisfy the preconditions of `insert_loop`. So `insert` calls this helper function.

`insert_loop` takes as input a set of cover trees P and inserts a data point x into one of them. Let $i+1$ be the level of the nodes in Q. The if statement on line 3 checks whether adding x to level $i+1$ would violate the global separating invariant. Precondition 2 ensures that all such nodes in level $i+1$ are children of some node in p. If the separating invariant cannot be satisfied, then x must be inserted at a lower level. In the else clause on line 6, we form a set Q_x of nodes that x may be inserted under, and recursively call `insert_loop`. The definition of Q_x ensures that `insert_loop`'s preconditions will be met. □

Theorem 32. *The runtime of* `insert`(p, x) *is* $O(c_{\text{doub}}^3 \log \Delta)$.

Proof. The runtime of `insert` excluding the call to `insert_loop` takes only constant time. So the overall runtime is the runtime of `insert_loop`. The bound on the runtime of `insert_loop` follows the same pattern as the bound on `findnn`.

The total runtime of `insert_loop` is bounded by the product of the number of recursive calls and the time spent in each call. The number of recursions is bounded by the height of the tree because `insert_loop` descends one level of the tree with each recursive call. This height is bounded by $O(\log \Delta)$ in Lemma 26.

The cost of each recursive call is $O(|Q|)$, since we perform one distance computation for each node in Q. The size of Q is the size of P times the of children per node. To bound

the size of P, we will bound the size of Q_x (since P is always Q_x of the previous iteration). Let $\delta = 2^{i-1}$. Then Q_x is a δ-packing of $B(x, \delta)$ by construction. So,

$$|Q_x| \leq M_\delta(B(x, \delta)) \leq N_{\delta/2}(B(x, \delta)) \leq c_{\text{doub}}. \tag{4.44}$$

Each node in P has at most c_{doub}^2 children by Lemma 27, so $|Q| \leq c_{\text{doub}}^3$. □

4.5 The simplified cover tree

The **simplified cover tree** was introduced by Izbicki and Shelton (2015). It maintains three invariants that are simpler than the invariants of the original cover tree.

1. **Leveling invariant**. Every node p has an associated integer $\text{level}(p)$. For all nodes $q \in \text{children}(p)$, $\text{level}(q) = \text{level}(p) - 1$.

2. **Covering invariant**. Every node p has an associated real number $\text{covdist}(p) = 2^{\text{level}(p)}$. For all nodes $q \in \text{children}(p)$, $d(p, q) \leq \text{covdist}(p)$.

3. **Local separating invariant**. For all nodes $q_1, q_2 \in \text{children}(p)$, $d(q_1, q_2) \geq \text{covdist}(p)/2$.

The covering invariant is the same as in the original cover tree, but the leveling and local separating invariants are modified.

The leveling invariant does not require that data points exist at multiple levels of the tree. This is in contrast to the nesting invariant that requires all non-leaf nodes to have a child containing the same data point. Therefore, the simplified cover tree has fewer nodes, which reduces both the overhead from traversing the data structure and the number of required distance comparisons. Figure 4.1 shows the reduction in nodes by using

the simplified cover tree on benchmark datasets taken from the MLPack test suite (Curtin et al., 2013a). Section 4.6 contains more details on these datasets, and Figure 4.3 in the same section shows how this reduced node count translates into improved query performance.

The local separating invariant only requires that children of the same node be separated from each other. This is in contrast to the separating invariant of the original cover tree that requires all nodes at a particular level to be separated even if they do not share a parent. As the following subsections show, the local separating invariant decreases the complexity of cover tree construction, but increases the complexity of nearest neighbor queries. Our experiments in Section 4.6 demonstrate that this is a useful tradeoff.

4.5.1 Properties of the simplified cover tree

The simplified cover tree obeys the following two structural lemmas with the same proofs as the original cover tree.

Lemma 33. Let p be any non-leaf node in a simplified cover tree. Denote by height(p) the number of edges between p and its most distant leaf. We have that height(p) $\leq \log_2 \Delta_X$.

Lemma 34. For every node p in a simplified cover tree, we have that $|\text{children}(p)| \leq c_{\text{doub}}^2$.

4.5.2 Approximate nearest neighbor queries

Algorithms 9 and 10 from Section 4.4 correctly answer $(1 + \varepsilon)$-ann queries in both the simplified and original cover trees. Unfortunately, Izbicki and Shelton (2015) incorrectly stated that the simplified cover tree obeys the same runtime bounds as the original cover tree for nearest neighbor queries. This is not true in the worst case. The proofs of Theorems

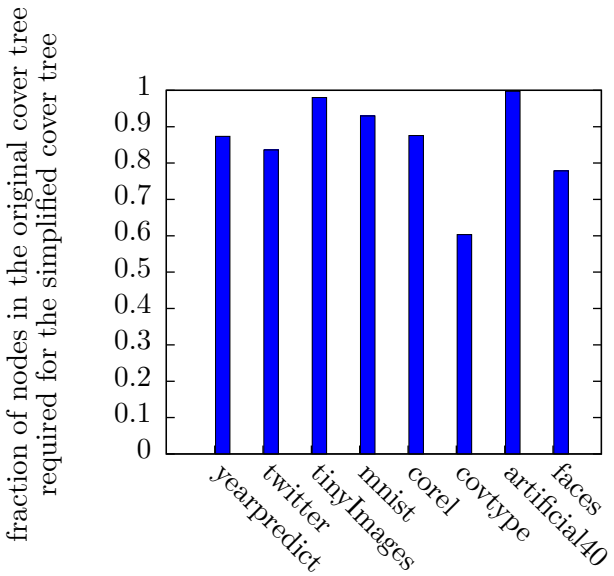

Figure 4.1: Fraction of nodes required for the simplified cover tree. Fewer nodes means less overhead from traversing the tree and fewer distance comparisons. See Table 4.1 for information on the datasets.

29 critically depend on the global nature of the separating invariant to bound the size of Q_x in each recursive call of `findnn_orig` and `findnn`. It is possible that an amortized or average case analysis could achieve similar runtimes, however the difficulty of such an analysis does not seem worth the effort. The experiments in Section 4.6 demonstrate that this theoretical difficulty is not an issue in practice as the simplified cover tree performs nearest neighbor queries faster than the original on real world problems.

4.5.3 Inserting a single point

Insertion into a simplified cover tree is faster and simpler than the original cover tree. Algorithms 13 and 14 show the procedure. It is divided into two cases. In the first case, we cannot insert our data point x into the tree without violating the covering invariant. So we raise the level of the tree p by taking any leaf node and using that as the new root. Because

Algorithm 13 `insert`(cover tree p, data point x)

```
1: if d(p, x) > covdist(p) then
2:     while d(p, x) > 2covdist(p) do
3:         Remove any leaf q from p
4:         p' ← tree with root q and p as only child
5:         p ← p'
6:     end while
7:     return tree with x as root and p as only child
8: end if
9: return insert_loop(p,x)
```

Algorithm 14 `insert_loop`(cover tree p, data point x)

prerequisite: $d(p, x) \leq \text{covdist}(p)$

```
1: for q ∈ children(p) do
2:     if d(q, x) ≤ covdist(q) then
3:         q' ← insert_loop(q, x)
4:         p' ← p with child q replaced with q'
5:         return p'
6:     end if
7: end for
8: return p with x added as a child
```

$\text{maxdist}(p) \leq 2\text{covdist}(p)$, we are guaranteed that $d(p, x) \leq \text{covdist}(x)$, and so we do not violate the covering constraint. In the second case, the `insert_loop` function recursively descends the tree. On each function call, we search through $\text{children}(p)$ to find a node we can insert into without violating the covering invariant. If we find such a node, we recurse; otherwise, we know we can add x to $\text{children}(p)$ without violating the separating invariant. In all cases, exactly one node is added per data point, so the resulting tree will have exactly n nodes. The following theorem shows that inserting into a simplified cover tree has a better dependence on c_{doub} than inserting into the original cover tree (c_{doub}^2 vs c_{doub}^3).

Theorem 35. *The runtime of* `insert_loop`(p, x) *is* $O(c_{\text{doub}}^2 \log \Delta)$. *For i.i.d. data satisfying the regularity conditions of Lemma 25, the expected runtime is* $O(c_{\text{doub}}^2 \log n)$.

Proof. On each recursive call, the algorithm visits each child node at most once, then either descends a level or terminates. The number of children is bounded by $O(c_{\text{doub}}^2)$ by Lemma 27, and the height of the tree is bounded by $O(\log \Delta)$ by Lemma 26. In the i.i.d. case, Lemma 25 states that $\Delta = O(n^3)$. □

4.5.4 The nearest ancestor invariant

A further advantage of the simplified cover tree is that it is easier to add new invariants that improve performance. In this section, we exploit a similarity between simplified cover trees and binary search trees (BSTs). Insertion into both trees follows the same procedure: Perform a depth first search to find the right location to insert the point. In particular, there is no rebalancing after the insertion. Many alternatives to plain BSTs produce better query times by introducing new invariants. These invariants force the insertion algorithm to rebalance the tree during the insertion step. Our definition of the simplified cover tree makes adding similar invariants easy. We now introduce one possible invariant.

A *nearest ancestor cover tree* is a simplified cover tree where every point p has the nearest ancestor invariant: If q_1 is an ancestor of p and q_2 is a sibling of q_1, then

$$d(p, q_1) \leq d(p, q_2)$$

In other words, the nearest ancestor cover tree ensures that for every data point, each of its ancestors is the "best possible" ancestor for it at that level.[4] Figure 4.2 shows a motivating one dimensional example.

[4] The nearest ancestor invariant is incompatible with the global separating invariant of the original cover

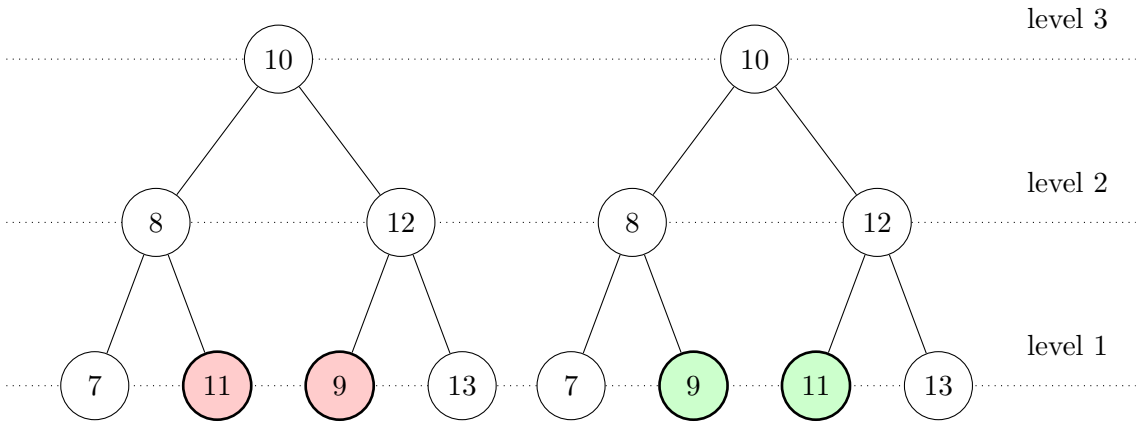

Figure 4.2: Example of a simplified cover tree. Using the metric $d(a, b) = |a - b|$, both trees are valid simplified cover trees; but only the right tree is a valid nearest ancestor cover tree. Moving the 9 and 11 nodes reduces the value of maxdist for their ancestor nodes. This causes pruning to happen more often during nearest neighbor queries, resulting in fewer distance comparisons.

Algorithm 15 shows how to insert a point into a nearest ancestor cover tree. It uses the same `insert` function as 13, but the helper function `insert_loop` is slightly modified in two ways (shown with an underline). First, we sort children(p) according to their distance from the data point x. This sorting ensures that our newly inserted point will satisfy the nearest ancestor invariant. But this new point x may cause other points to violate the nearest ancestor invariant. In particular, if x has a sibling q; q has a descendent r; and $d(r, x) < d(r, q)$; then r now violates the nearest ancestor invariant. Our second step is to call rebalance, which finds all these violating data points and moves them underneath x.

Most of the work of the `rebalance` function happens in the helper `rebalance_loop`. `rebalance_loop` returns a valid nearest ancestor cover tree and a set of points that still need to be inserted; `rebalance` just inserts those extra points. `rebalance_loop` takes two

tree. There are data sets that cannot simultaneously satisfy both.

nearest ancestor cover trees p and q (where p is an ancestor of q) and a point x. Its goal is to "extract" all points from q that would violate the nearest ancestor invariant if x became a sibling of p. It returns three values: a modified version of q, a set of points that cannot remain in any point along the path from p to q called the *moveset*, and a set of points that need to be reinserted somewhere along the path from p to q called the *stayset*. There are two cases. In the first case, the data point at node q must move. We then filter the descendants of q into the *moveset* or *stayset* as appropriate and return `null` for our modified q. In the second case, the data point at node q must stay. We recursively apply `rebalance_loop` to each of q's children; we use the results to update the corresponding child, the *moveset* and the *stayset* variables. Finally, we try to reinsert any nodes in *stayset*. If `rebalance_loop` was called with q a child of p, then the return value of *stayset* will be empty; any children that could not be directly reinserted must be in the *moveset*.

The `rebalance_loop` function loops over all $O(c_{\text{doub}}^2)$ children of a node, and the maximum depth of the recursion is $O(\log \Delta)$. Therefore the overall runtime is $O(c_{\text{doub}}^2 \log \Delta)$. It may be called up to $O(c_{\text{doub}}^2)$ times within `rebalance`, so the body of the for loop on line 14 executes $O(c_{\text{doub}}^4 \log \Delta)$ times. Unfortunately, we have no bound on the size of *moveset*, except to note that it is usually small in practice. On the datasets in our experiments (see Table 4.1), the value is usually zero or at worst in the single digits. Figure 4.3(a) shows that nearest ancestor cover tree construction is not that much slower in practice, and Figure 4.3(b) shows that this slowdown is overshadowed by the resulting speedup in nearest neighbor queries.

Algorithm 15 `insert_loop`(cover tree p, data point x)

```
1: for q ∈ children(p) sorted by distance to x do
2:     if d(q, x) ≤ covdist(q) then
3:         q' ← insert_loop(q, x)
4:         p' ← p with child q replaced with q'
5:         return p'
6:     end if
7: end for
8: return rebalance(p, x)
```

function `rebalance`(cover trees p, data point x)

prerequisites: x can be added as a child of p without violating the covering or separating invariants

```
1: create tree x' with root node x at level level(p) − 1 x' contains no other points
2: p' ← p
3: for q ∈ children(p) do
4:     (q', moveset, stayset) ← rebalance_loop(p, q, x)
5:     p' ← p' with child q replaced with q'
6:     for r ∈ moveset do
7:         x' ← insert(x', r)
8:     end for
9: end for
10: return p' with x' added as a child
```

Algorithm 16 `rebalance_loop`(cover trees p and q, point x)

prerequisites: p is an ancestor of q

```
1: if d(p, q) > d(q, x) then
2:     moveset, stayset ← ∅
3:     for r ∈ descendants(q) do
4:         if d(r, p) > d(r, x) then
5:             moveset ← moveset ∪ {r}
6:         else
7:             stayset ← stayset ∪ {r}
8:         end if
9:     end for
10:    return (null, moveset, stayset)
11: else
12:    moveset′, stayset′ ← ∅
13:    q′ ← q
14:    for r ∈ children(q) do
15:        (r′, moveset, stayset) ← rebalance_loop(p, r, x)
16:        moveset′ ← moveset ∪ moveset′
17:        stayset′ ← stayset ∪ stayset′
18:        if r′ = null then
19:            q′ ← q with the subtree r removed
20:        else
21:            q′ ← q with the subtree r replaced by r′
22:        end if
23:    end for
24:    for r ∈ stayset′ do
25:        if d(r, q)′ ≤ covdist(q)′ then
26:            q′ ← insert(q′, r)
27:            stayset′ ← stayset′ − {r}
28:        end if
29:    end for
30:    return (q′, moveset′, stayset′)
31: end if
```

4.5.5 Merging simplified cover trees

In this section we discuss parallelism on shared-memory, multiprocessor machines. Previous chapters discussed parallelism in the more restrictive model of distributed architectures. In distributed architectures, communication between processors is expensive and so must be minimized. In the shared memory model, communication is cheap and so we make no effort to minimize it.

Querying in parallel is easy. Since the results of neighbor queries for a data point do not depend on other data points, we can divide the points among the processors and have each processor traverse the tree independently. More difficult is constructing the tree in parallel. Our strategy is to split the data, create one cover tree on each processor, then merge these trees together. Previous work on parallelizing cover trees applied only to the GPU (Kumar and Ramareddy, 2010). Our approach is suitable for any shared-memory multiprocessor machine. We give a detailed description for merging simplified cover trees and discuss at a high level how to extend this procedure to nearest ancestor cover trees.

Algorithm 17 shows the merging procedure. The `merge` function's main purpose is to satisfy the prerequisites for `mergeHelper`, which has two phases. First, we find all the subtrees of q that can be inserted directly into p without violating any invariants, and we insert them. Second, we insert the remaining nodes from q into p directly via the `insert` function.

The `mergeHelper` function returns a partially merged tree and a set of nodes called the *leftovers* that still need to be inserted into the tree. The first phase uses the for loop

Algorithm 17 `merge`(cover tree p, cover tree q)

```
1: if level(q) > level(p) then
2:     swap p and q
3: end if
4: while level(q) < level(p) do
5:     move a node from the leaf of q to the root;
6:     this raises the level of q by 1
7: end while
8: (p, leftovers) ← mergeHelper(p, q)
9: for r ∈ leftovers do
10:    p ← insert(p, r)
11: end for
12: return p
```

starting on line 3 to categorize the children of q into three disjoint sets. The *uncovered* set contains all of q's children that would violate the covering invariant if inserted into p. The *sepcov* set contains all of q's children that would not violate the separating or covering invariants when inserted into p. Both of these sets are unused in the second phase of `mergeHelper`. Every child of q that is not inserted into the *uncovered* or *sepcov* sets gets merged with a suitable node in children(p). This is done by recursively calling the `mergeHelper` function. Any points that could not be inserted into the results of `mergeHelper` get added to the *leftovers* set.

In the second phase of `mergeHelper`, we insert as many nodes as possible into our merged tree p'. First update the children with the subtrees in *sepcov*. Then insert the root of q. We know that $d(p, q) \leq \text{covdist}(p)$, so this insertion is guaranteed not to change the level of p'. Finally, we loop over the elements in *leftovers* and insert them into p' only if it would not change the level of p'. Any elements of *leftovers* that cannot be inserted into p' get inserted into *leftovers*$'$ and returned. It is important to do this insertion of *leftovers* at the lowest level possible (rather than wait until the recursion ends and have the insertion

Algorithm 18 `mergeHelper`(cover tree p, cover tree q)

prerequisite: level(p) = level(q), $d(p,q) \leq$ covdist(p)

```
 1: children' ← children(p)                                          ▷ Phase 1
 2: uncovered, sepcov, leftovers ← ∅
 3: for r ∈ children(q) do
 4:     if d(p,r) < covdist(p) then
 5:         foundmatch ←false
 6:         for s ∈ children' do
 7:             if d(s,r) ≤ sepdist(p) then
 8:                 (s', leftovers_s) ← mergeHelper(s,r)
 9:                 children' ← children' ∪ {s'} − {s}
10:                 leftovers ← leftovers ∪ leftovers_s
11:                 foundmatch ← true
12:                 break from inner loop
13:             end if
14:         end for
15:         if not foundmatch then
16:             sepcov ← sepcov ∪ {r}
17:         end if
18:     else
19:         uncovered ← uncovered ∪ {r}
20:     end if
21: end for
22: children' ← children' ∪ sepcov                                   ▷ Phase 2
23: p' ← tree rooted at p with children(p') = children'
24: p' ← insert(p', q)
25: leftovers' ← ∅
26: for r ∈ leftovers do
27:     if d(r,p)' ≤ covdist(p)' then
28:         p' ← insert(p', r)
29:     else
30:         leftovers' ← leftovers' ∪ {r}
31:     end if
32: end for
33: return (p', leftovers' ∪ uncovered)
```

performed in `merge`) to avoid unnecessary distance computations.

The `merge` function does not maintain the nearest ancestor invariant. A modified version of `merge` that calls the `rebalance` function appropriately could, however this would significantly complicate the procedure. Experimental evidence shows that this complication is not necessary. In our experiments below, we use the provided `merge` function in Algorithm 17 to parallelize both simplified and nearest ancestor tree construction. In practice, this retains the benefits of the nearest ancestor cover tree because the nearest ancestor invariant is violated in only a few places.

Providing explicit bounds on the runtime of `merge` is difficult. But in practice it is fast. When parallelizing on two processors, approximately 1% of the distance calculations occur within `merge`. So this is not our bottleneck. Instead, the main bottleneck of parallelism is cache performance. On modern desktop computers, last level cache is shared between all cores on a CPU. Cover tree construction results in many cache misses, and this effect is exaggerated when the tree is constructed in parallel.

4.5.6 Cache efficiency

One of the biggest sources of overhead in modern data structures is cache misses. Our last improvement is to make cover trees more cache efficient for queries. A simple way to reduce cache misses for tree data structures is to use the *van Emde Boas* tree layout (Frigo et al., 1999). This layout arranges nodes in memory according to a depth first traversal of the tree. This arrangement creates a *cache oblivious* data structure. That is, the programmer does not need any special knowledge about the cache sizes to obtain optimal speedup—the van Embde Boas tree layout works efficiently on any cache architecture. This layout has been

known for a long time in the data structures community, but it seems unused in machine learning libraries. Frigo et al. (1999) provides a detailed tutorial.

Our implementation of the cache oblivious cover tree is *static*. That is, we first construct the cover tree, then we call a function `pack` that rearranges the tree in memory. This means we do not get the reduced cache misses while constructing the tree, but only while querying the tree. The `pack` function is essentially free to run because it requires only a single traversal through the dataset and no distance computations. Figure 4.4 shows that the van Emde Boas tree layout reduces cache misses by 5 to 20 percent. This results in a reduction of stalled CPU cycles by 2 to 15 percent.

4.6 Experiments

We now validate our improvements to the cover tree empirically. Our first experiments use the Euclidean distance on a standard benchmark suite (described in Table 4.1). Our last experiments use non-Euclidean metrics on data from bioinformatics and computer vision. In each experiment, we use the **all nearest neighbors** experimental setup. That is, we first construct the cover trees on the dataset. Then, for each point in the dataset, we find its nearest neighbor. This is a standard technique for measuring the efficiency of nearest neighbor algorithms.

4.6.1 Cover tree type comparison

Our first experiment compares the performance of the three types of cover trees: original, simplified, and nearest ancestor. We measure the number of distance comparisons required

dataset	num data points	num dimensions
yearpredict	515345	90
twitter	583250	78
tinyImages	100000	384
mnist	70000	784
corel	68040	32
covtype	581012	55
artificial40	10000	40
faces	10304	20

Table 4.1: All MLPack benchmark datasets with at least 20 dimensions and 10000 points, arranged in descending order by runtime of all nearest neighbor search.

to build the tree on a dataset in Figure 4.3(a) and the number of distance comparisons required to find each data point's nearest neighbor in Figure 4.3(b). Distance comparisons are a good proxy measure of runtime performance because the majority of the algorithm's runtime is spent computing distances, and it ignores the possible unwanted confounding variable of varying optimization efforts. As expected, the simplified tree typically outperforms the original tree, and the nearest ancestor tree typically outperforms the simplified tree. We reiterate that this reduced need for distance comparisons translates over to all other queries provided by the space tree framework Curtin et al. (2013b).

4.6.2 Cover tree implementation comparison

We next compare our implementation against two good cover tree implementations currently in widespread use: the reference implementation used in the original paper Beygelzimer et al. (2006) and MLPack's implementation Curtin et al. (2013a). Both of these programs were written in C++ and compiled using `g++ 4.4.7` with full optimizations. Our implementation was written in Haskell and compiled with `ghc 7.8.4` also with full optimizations.[5] All tests

[5]Our code can be downloaded at `http://github.com/mikeizbicki/hlearn#covertree`.

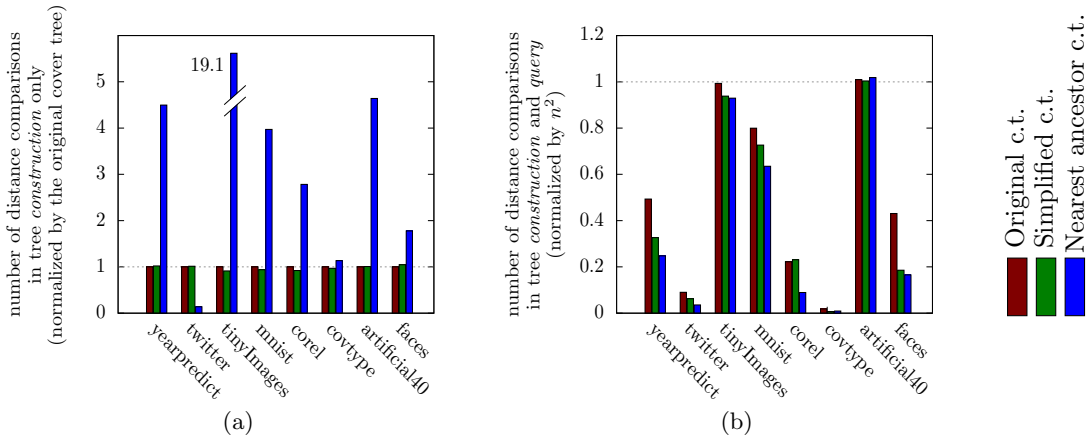

Figure 4.3:]
(a) Constructing a nearest ancestor query tree usually takes longer than the original cover tree and the simplified cover tree. (b) Construction *plus* querying is faster in the nearest ancestor cover tree. On most datasets, this faster query time more than offsets the increased construction cost, giving an overall speedup.

were run on an Amazon Web Services `c3.8x-large` instance with 60 GB of RAM and 32 Intel Xeon E5-2680 CPU cores clocked at 2.80GHz. Half of those cores are hyperthreads, so for simplicity we only parallelize out to 16 cores.

Since the reference implementation and MLPack only come with the Euclidean distance built-in, we only use that metric when comparing the three implementations. Figure 4.4 shows the cache performance of all three libraries. Figure 4.5 shows the runtime of all three libraries. Our implementation's cache performance and parallelization speedup is shown on the nearest ancestor cover tree. Neither the original implementation nor MLPack support parallelization.

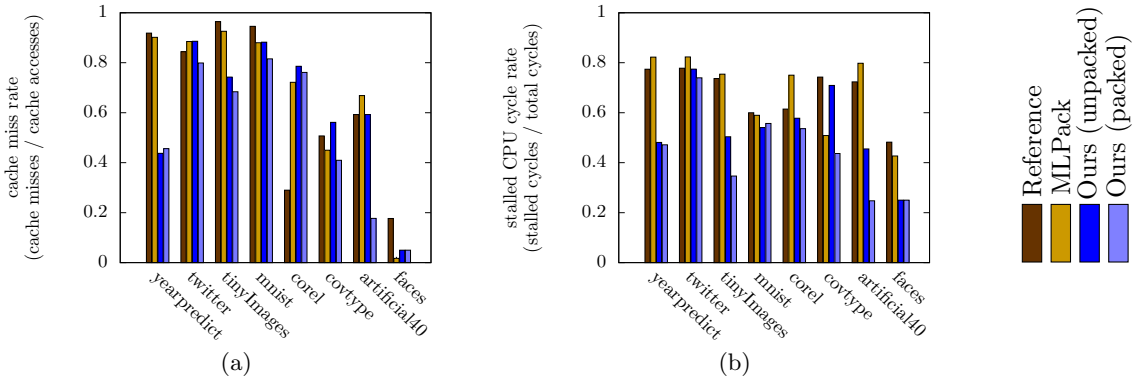

Figure 4.4: (a) Comparison of our packed nearest ancestor cover tree to our unpacked tree and other implementations, demonstrating better cache performance. (b) A stalled CPU cycle is when the CPU does no work because it must wait for a memory access. Reducing the number of cache misses results in fewer stalled cycles, and so faster runtimes. We used the Linux `perf stat` utility to measure the `cache-references`, `cache-misses`, `cycles`, and `stalled-cycles-frontend` hardware counters. `perf stat` uses a sampling strategy with negligible affect on program performance.

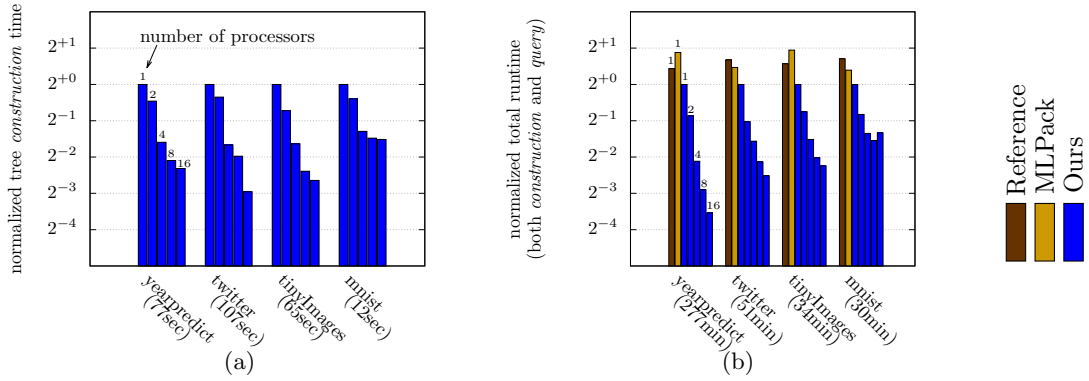

Figure 4.5: Run times on the "all nearest neighbor" procedure for only those datasets that take more than 5 minutes. (a) Tree construction. A single cover tree merge takes about 1% of the computation time; the main reason for the lack of perfect parallel speedup is the increased number of cache misses caused by inserting into multiple trees simultaneously. (b) Comparison on total performance to reference and MLPack implementations. Runtimes in both figures are divided by that of our single processor implementation (shown in parenthesis).

4.6.3 Alternative methods for Euclidean nearest neighbors

Next we compare our cover tree implementation to non-cover tree based nearest neighbor libraries. Specifically, we compare to a kd-tree implemented in the Fast Library for Approximate Nearest Neighbor (FLANN; Muja and Lowe, 2014), a kd-tree implemented in Julia (Bezanson et al., 2017), the original cover tree implementation (Beygelzimer et al., 2006), a kd-tree and cover tree implemented in MLPack (Curtin et al., 2013a), a kd-tree and ball tree implemented in Python's scikit learn (Pedregosa et al., 2011), and a kd-tree implemented in R (R Development Core Team, 2008). The results are shown in Figures 4.6 and 4.7. Without parallelization, our implementation is faster than most of the alternatives on most of the datasets. With parallelization it is faster than all of the alternatives on all of the datasets.

4.6.4 Graph kernels and protein function

An important problem in bioinformatics is to predict the function of a protein based on its 3d structure. State of the art solutions model the protein's 3d structure as a graph and use support vector machines (with a graph kernel) for prediction. Computing graph kernels is relatively expensive, however, so research has focused on making the graph kernel computation faster (Vishwanathan et al., 2010; Shervashidze et al., 2011). Such research makes graph kernels scale to *larger graphs*, but does not help in the case where there are *more graphs*. Our contribution is to use cover trees to reduce the number of required kernel computations, letting us scale to more graphs. The largest dataset in previous research contained about 1200 proteins. With our cover tree, we perform nearest neighbor queries

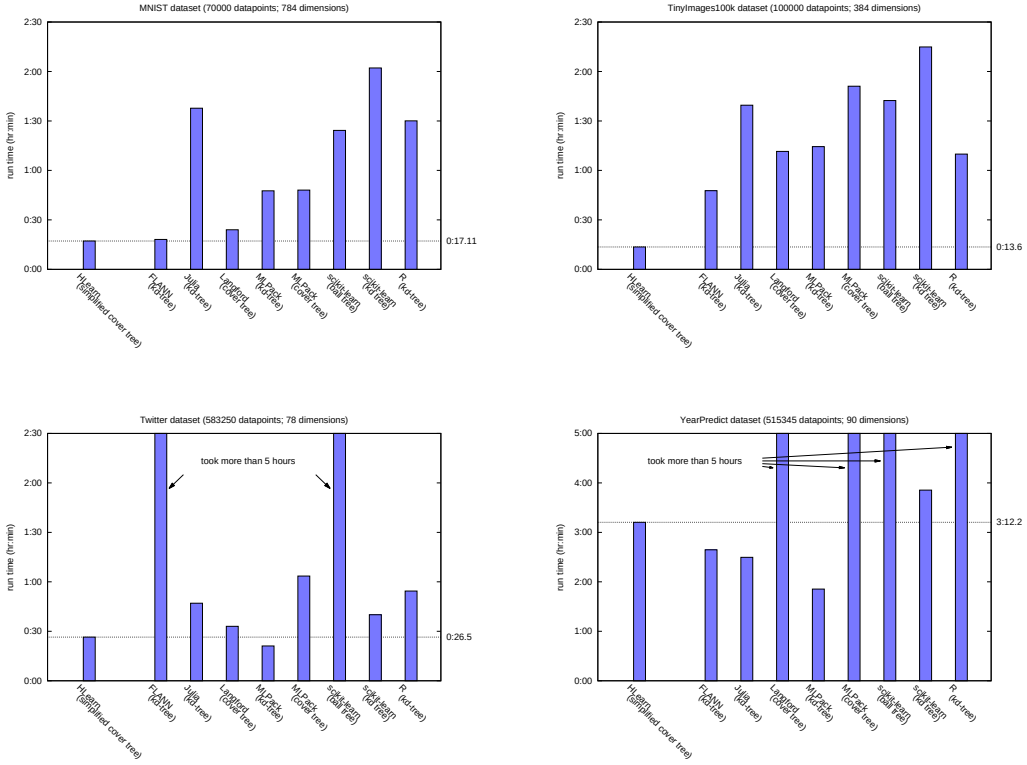

Figure 4.6: Using only a single processor, our nearest ancestor cover tree performs the fastest on each dataset except for YearPredict.

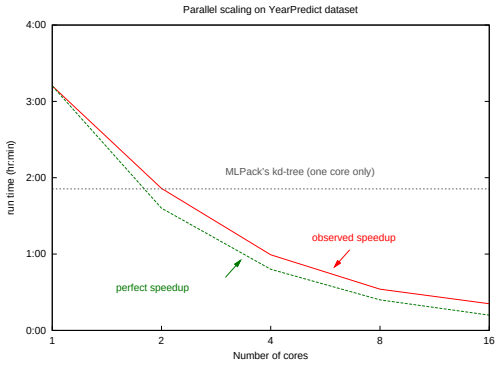

Figure 4.7: Using multiple processors, our implementation is now also the fastest on the YearPredict dataset by a wide margin. (FLANN is the only other library supporting parallelism, but in my tests with this dataset parallelism actually slowed it down for some reason.)

on all 100,000 proteins currently registered in the Protein Data Bank (Berman et al., 2000).

We use the random walk graph kernel in our experiment. It performs well on protein classification and is conceptually simple. See Vishwanathan et al. (2010) for more details. A naive computation of this kernel takes time $O(v^6)$, where v is the number of vertices in the graph. Vishwanathan et al. (2010) present faster methods that take time only $O(v^3)$. While considerably faster, it is still a relatively expensive distance computation.

The Protein Data Bank (Berman et al., 2000) contains information on the 3d primary structure of approximately one hundred thousand proteins. To perform our experiment, we follow a procedure similar to that used by the PROTEIN dataset used in the experiments in Vishwanathan et al. (2010). This procedure constructs secondary structure graphs from the primary structures in the Protein Data Bank using the tool VLPG (Schäfer et al., 2012). The Protein Data Bank stores the 3d structure of the atoms in the protein in a PDB file. From this PDB file, we calculate the protein's secondary structure using the DSSP tool (Joosten et al., 2011). Then, the tool VLPG (Schäfer et al., 2012) generates graphs from the resulting secondary structure. Some PDB files contain information for multiple graphs, and some do not contain enough information to construct a graph. In total, VLPG generates 250,000 graphs, and a typical graph has between 5-120 nodes and 0.1-3 edges per node. Figure 4.8 shows the scaling behavior of all three cover trees on this dataset. On all of the data, the total construction and query cost are 29% that of the original cover tree.

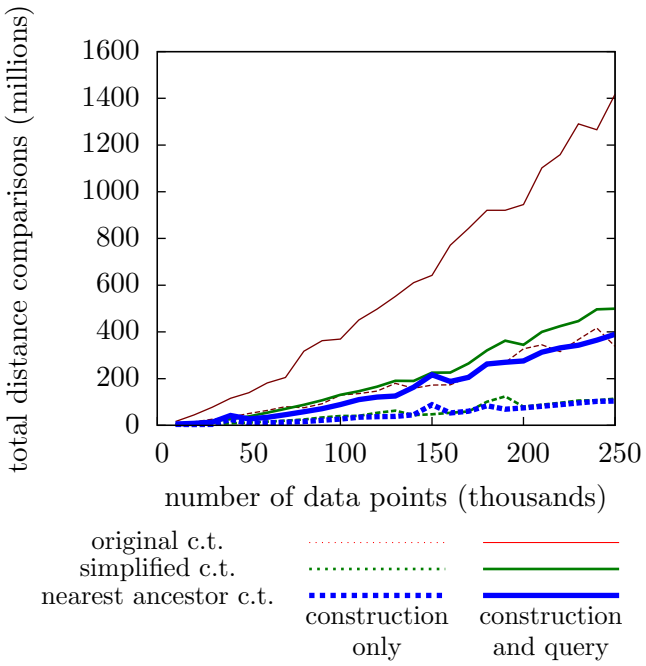

Figure 4.8: The effect on runtime as we increase the number of data points on the bionformatics data. The relationship is roughly linear, indicating protein graphs have a relatively low intrinsic dimensionality. As expected, the nearest ancestor cover tree performs the best.

4.6.5 Earth mover's distance

The Earth Mover's Distance (EMD) is a distance metric between histograms designed for image classification (Rubner et al., 1998). In our tests, we convert images into 3d histograms of the pixel values in LabCIE color space. LabCIE is a color space that represents colors in three dimensions. It is similar to the more familiar RGB and CMYK color spaces, but the distances between colors more accurately match human perception. We construct the histogram such that each dimension has 8 equally spaced intervals, for a total of 512 bins. We then create a "signature" of the histogram by recording only the 20 largest of the 512 bins.

Previous research on speeding up EMD focused on computing EMD distances

number of cores	simplified tree construction time	simplified tree construction speedup	nearest ancestor tree construction time	nearest ancestor tree construction speedup
1	70.7 min	1.0	210.9 min	1.0
2	36.6 min	1.9	94.2 min	2.2
4	18.5 min	3.8	48.5 min	4.3
8	10.2 min	6.9	25.3 min	8.3
16	6.7 min	10.5	12.0 min	17.6

Table 4.2: Parallel cover tree construction using the earth movers distance. On this large dataset with an expensive metric, we see better parallel speedup than on the datasets with the cheaper L2 metric. The nearest ancestor cover tree gets super-linear parallel speedup because we are merging with Algorithm 17, which does not attempt to rebalance.

faster. The EMD takes a base distance as a parameter. For an arbitrary base distance, EMD requires $O(b^3 \log b)$ time where b is the size of the histogram signature. Faster algorithms exist for specific base metrics. For example, with an L_1 base metric the EMD can be computed in time $O(b^2)$ (Ling and Okada, 2007); and if the base metric is a so-called "thresholded metric," we can get an order of magnitude constant factor speed up (Pele and Werman, 2009). We specifically chose the LabCIE color space because there is no known faster EMD algorithm, so it stress-tests our cover tree implementation.

In this experiment, we use the Yahoo! Flickr Creative Commons dataset. The dataset contains 1.5 million images in its training set, and we construct simplified and nearest ancestor cover trees in parallel on this data. Construction times are shown in Table 4.2. Using the cheap L2 distance with smaller datasets, tree construction happens quickly and so parallelization is less important. But with an expensive distance on this larger dataset, parallel construction makes a big difference.

4.7 Conclusion

This chapter presented a number of improvements to the cover tree data structure. We tightened the runtime bounds for the cover tree and introduced the simplified cover tree with improved practical performance. In particular, we introduced a merge procedure, so the simplified cover tree fits into the mergeable learning algorithm framework of Chapter 2. This merge procedure provides the first parallel construction algorithm and fast cross validation algorithms for the cover tree.

Chapter 5

Conclusion

This thesis has argued that mergeable learning algorithms are an important tool in designing large scale learning systems. Chapter 2 reviewed 32 published examples of mergeable algorithms and derived a novel fast cross validation procedure suitable for each of these algorithms. Then Chapters 3 and 4 introduced new mergeable algorithms that have better statistical guarantees than previously existing algorithms.

An important open question remains: What is the best possible merge procedure for models? Liu and Ihler (2014) provide a partial answer. As discussed in Section 2.4.2.2, they present a theorem showing that in the important special case of regularized loss minimization (RLM), merge procedures that do not depend on the data must incur a loss proportional to the statistical curvature of a problem. This is only a partial answer for two reasons. First, many learning problems do not fit the RLM framework. For example, there are no known bounds on the performance of merging MCMC estimators. Second, the OWA algorithm of Chapter 3 shows that good merge algorithms can be developed that depend on

the data. OWA's theoretical guarantees only apply in the linear RLM case. It is possible that similar data dependent merge procedures could work well in more general settings. We believe that the next few years is likely to see the development of new and better mergeable algorithms.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

Ittai Abraham, Yair Bartal, and Ofer Neimany. Advances in metric embedding theory. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 271–286. ACM, 2006.

Ittai Abraham, Yair Bartal, and Ofer Neiman. Embedding metrics into ultrametrics and graphs into spanning trees with constant average distortion. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 502–511. Society for Industrial and Applied Mathematics, 2007.

Ittai Abraham, Yair Bartal, Ofer Neiman, and Leonard J Schulman. Volume in general metric spaces. *Discrete & Computational Geometry*, 52(2):366–389, 2014.

Ittai Abraham, Shiri Chechik, Robert Krauthgamer, and Udi Wieder. Approximate Nearest Neighbor Search in Metrics of Planar Graphs. In *Approximation, Randomization, and Combinatorial Optimization.*, pages 20–42, 2015.

Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 274–281. ACM, 2001.

Radosław Adamczak and Witold Bednorz. Exponential concentration inequalities for additive functionals of Markov chains. *ESAIM: Probability and Statistics*, 19:440–481, 2015.

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

Sungjin Ahn and Max Welling. Bayesian posterior sampling via stochastic gradient Fisher scoring. In *International Conference of Machine Learning*, 2012.

Yacine Aït-Sahalia. Maximum likelihood estimation of discretely sampled diffusions: A closed-form approximation approach. *Econometrica*, 70(1):223–262, 2002.

S. Amari. *Information Geometry and Its Applications*. Springer Japan, 2016.

Shun-Ichi Amari. Differential geometry of curved exponential families-curvatures and information loss. *The Annals of Statistics*, pages 357–385, 1982.

Senjian An, Wanquan Liu, and Svetha Venkatesh. Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression. *Pattern Recognition*, 40(8): 2154–2162, 2007.

Alexandr Andoni, Dorian Croitoru, and Mihai Patrascu. Hardness of nearest neighbor under l-infinity. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 424–433. IEEE, 2008.

Alexandr Andoni, Piotr Indyk, Huy L Nguyn, and Ilya Razenshteyn. Beyond locality-sensitive hashing. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1018–1028. SIAM, 2014.

Alexandr Andoni, Thijs Laarhoven, Ilya Razenshteyn, and Erik Waingarten. Lower bounds on time-space trade-offs for approximate near neighbors. *arXiv preprint arXiv:1605.02701*, 2016.

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to MCMC for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

Christophe Andrieu, Matti Vihola, et al. Convergence properties of pseudo-marginal Markov chain Monte Carlo algorithms. *The Annals of Applied Probability*, 25(2):1030–1077, 2015.

Christophe Andrieu, Matti Vihola, et al. Establishing some order amongst exact approximations of MCMCs. *The Annals of Applied Probability*, 26(5):2661–2696, 2016.

Fabrizio Angiulli. Fast condensed nearest neighbor rule. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 25–32. ACM, 2005.

Sylvain Arlot, Alain Celisse, et al. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010.

Sunil Arya, Theocharis Malamatos, and David M Mount. Space-time tradeoffs for approximate nearest neighbor searching. *Journal of the ACM (JACM)*, 57(1):1, 2009.

Patrice Assouad. Étude d'une dimension métrique liée à la possibilité de plongements dans $\mathbb{R}^n$. In *C. R. Acad. Sci. Paris Sr*, 1979.

Laleh Badriasl, Thuraiappah Sathyan, and Sanjeev Arulampalam. A novel closed-form estimator for 3d tma using heterogeneous sensors. *IEEE Transactions on Signal Processing*, 2015.

John Baez and Mike Stay. Physics, topology, logic and computation: a rosetta stone. In *New Structures for Physics*, pages 95–172. Springer, 2010.

Natalia Bahamonde and Helena Veiga. A robust closed-form estimator for the garch (1, 1) model. *Journal of Statistical Computation and Simulation*, 86(8):1605–1619, 2016.

Zheng-Jian Bai, Raymond H Chan, and Franklin T Luk. Principal component analysis for distributed data sets with updating. In *International Workshop on Advanced Parallel Processing Technologies*, 2005.

C Bailer-Jones and K Smith. Combining probabilities. *Data Processing and Analysis Consortium (DPAS), GAIA-C8-TN-MPIA-CBJ-053*, 2011.

Richard Baraniuk, Mark Davenport, Ronald DeVore, and Michael Wakin. A simple proof of the restricted isometry property for random matrices. *Constructive Approximation*, 28 (3):253–263, 2008.

Rafael Barbosa, Alina Ene, Huy Nguyen, and Justin Ward. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning*, pages 1236–1244, 2015.

Rafael da Ponte Barbosa, Alina Ene, Huy L Nguyen, and Justin Ward. A new framework for distributed submodular maximization. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 645–654. Ieee, 2016.

Omer Barkol and Yuval Rabani. Tighter bounds for nearest neighbor search and related problems in the cell probe model. In *Proceedings of the Thirty-Second Annual ACM Symposium on Theory of Computing*, pages 388–396. ACM, 2000.

Yair Bartal, Nathan Linial, Manor Mendel, and Assaf Naor. On metric ramsey-type phenomena. In *Proceedings of the Thirty-Fifth Annual ACM Symposium on Theory of Computing*, pages 463–472. ACM, 2003.

Yair Bartal, Lee-Ad Gottlieb, and Ofer Neiman. On the impossibility of dimension reduction for doubling subsets of p. *SIAM Journal on Discrete Mathematics*, 29(3):1207–1222, 2015.

Maurice S Bartlett. Tests of significance in factor analysis. *British Journal of Mathematical and Statistical Psychology*, 3(2):77–85, 1950.

Heather Battey, Jianqing Fan, Han Liu, Junwei Lu, and Ziwei Zhu. Distributed estimation and inference with statistical guarantees. *arXiv preprint arXiv:1509.05457*, 2015.

Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The Protein Data Bank. *Nucleic Acids Research*, 28(1):235–242, 2000.

Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 97–104. ACM, 2006.

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017.

Aditya Bhaskara, Afshin Rostamizadeh, Jason Altschuler, Morteza Zadimoghaddam, Thomas Fu, and Vahab Mirrokni. Greedy column subset selection: New bounds and distributed algorithms. In *ICML*, 2016.

S Bhattacharya, J Haslett, et al. Importance re-sampling MCMC for cross-validation in inverse problems. *Bayesian Analysis*, 2(2):385–407, 2007.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 2017.

Guy E Blelloch. Prefix sums and their applications. *Technical Report*, 1990.

Luke Bornn, Arnaud Doucet, and Raphael Gottardo. An efficient computational approach for prior sensitivity analysis and cross-validation. *Canadian Journal of Statistics*, 38(1): 47–64, 2010.

Allan Borodin, Rafail Ostrovsky, and Yuval Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, pages 312–321. ACM, 1999.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Léon Bottou and Yann LeCun. Large scale online learning. In *Advances in Neural Information Processing Systems*, pages 217–224, 2004.

Jean Bourgain. On lipschitz embedding of finite metric spaces in hilbert space. *Israel Journal of Mathematics*, 52(1):46–52, 1985.

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

Oscar Boykin, Sam Ritchie, Ian O'Connell, and Jimmy Lin. Summingbird: A framework for integrating batch and online mapreduce computations. *Proceedings of the VLDB Endowment*, 7(13):1441–1451, 2014.

Mark Braverman, Ankit Garg, Tengyu Ma, Huy L Nguyen, and David P Woodruff. Communication lower bounds for statistical estimation problems via a distributed data processing inequality. *Symposium on the Theory of Computing*, 2016.

Tamara Broderick, Nicholas Boyd, Andre Wibisono, Ashia C Wilson, and Michael I Jordan. Streaming variational Bayes. In *Advances in Neural Information Processing Systems*, pages 1727–1735, 2013.

Kenneth S Brown. Semigroup and ring theoretical methods in probability. *Representations of Finite Dimensional Algebras and Related Topics in Lie Theory and Geometry*, 40:3–26, 2004.

José Camacho and Alberto Ferrer. Cross-validation in pca models with the element-wise k-fold (ekf) algorithm: theoretical aspects. *Journal of Chemometrics*, 26(7):361–373, 2012.

Trevor Campbell and Jonathan P How. Approximate decentralized bayesian inference. *arXiv preprint arXiv:1403.7471*, 2014.

Kevin Canini, Lei Shi, and Thomas Griffiths. Online inference of topics with latent dirichlet allocation. In *Artificial Intelligence and Statistics*, pages 65–72, 2009.

Raymond B Cattell. The scree test for the number of factors. *Multivariate Behavioral Research*, 1(2):245–276, 1966.

Gavin C Cawley and Nicola LC Talbot. Efficient approximate leave-one-out cross-validation for kernel logistic regression. *Machine Learning*, 71(2):243–264, 2008.

Hubert TH Chan, Anupam Gupta, Bruce M Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 762–771. Society for Industrial and Applied Mathematics, 2005.

T-H Hubert Chan, Anupam Gupta, and Kunal Talwar. Ultra-low-dimensional embeddings for doubling metrics. *Journal of the ACM (JACM)*, 57(4):21, 2010.

Yiu-Tong Chan and KC Ho. A simple and efficient estimator for hyperbolic location. *IEEE Transactions on Signal Processing*, 42(8):1905–1915, 1994.

Yiu-Tong Chan and Samuel M Thomas. An approximate maximum likelihood linear estimator of circle parameters. *Graphical Models and Image Processing*, 59(3):173–178, 1997.

Moses Charikar, Chandra Chekuri, Ashish Goel, Sudipto Guha, and Serge Plotkin. Approximating a finite metric by a small number of tree metrics. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 379–388. IEEE, 1998.

Kamalika Chaudhuri and Claire Monteleoni. Privacy-preserving logistic regression. In *Advances in Neural Information Processing Systems*, pages 289–296, 2009.

Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(Mar):1069–1109, 2011.

Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. Searching in metric spaces. *ACM Computing Surveys (CSUR)*, 33(3):273–321, 2001.

Peter Cheeseman, Bob Kanefsky, and William M Taylor. Where the really hard problems are. *IJCAI*, 1991.

Yilun Chen, Ami Wiesel, and Alfred O Hero. Shrinkage estimation of high dimensional covariance matrices. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 2937–2940. IEEE, 2009.

Cheng-Tao Chu, Sang K Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Kunle Olukotun, and Andrew Y Ng. Map-reduce for machine learning on multicore. In *Advances in neural information processing systems*, pages 281–288, 2007.

Kenneth L Clarkson. Nearest neighbor queries in metric spaces. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 609–617. ACM, 1997.

Kenneth L Clarkson. Nearest-neighbor searching and metric space dimensions. *Technical Report*, 2006.

Richard Cole and Lee-Ad Gottlieb. Searching dynamic point sets in spaces with bounded doubling dimension. In *Proceedings of the Thirty-Eighth Annual ACM Symposium on Theory of Computing*, pages 574–583. ACM, 2006.

Michael Collins, Sanjoy Dasgupta, and Robert E Schapire. A generalization of principal components analysis to the exponential family. In *Advances in Neural Information Processing Systems*, pages 617–624, 2002.

Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE Transactions on Visualization and Computer Graphics*, 16(4):599–608, 2010.

Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

Ryan R. Curtin, James R. Cline, Neil P. Slagle, William B. March, P. Ram, Nishant A. Mehta, and Alexander G. Gray. MLPACK: A scalable C++ machine learning library. *Journal of Machine Learning Research*, 14:801–805, 2013a.

Ryan R Curtin, William B March, Parikshit Ram, David V Anderson, Alexander G Gray, and Charles L Isbell Jr. Tree-independent dual-tree algorithms. In *Proceedings of The 30th International Conference on Machine Learning*, pages 1435–1443, 2013b.

Ryan R Curtin, William B March, Parikshit Ram, David V Anderson, Alexander G Gray, and Charles L Isbell Jr. Tree-independent dual-tree algorithms. *arXiv preprint arXiv:1304.4327*, 2013c.

Ryan R Curtin, Dongryeol Lee, William B March, and Parikshit Ram. Plug-and-play dual-tree algorithm runtime analysis. *Journal of Machine Learning Research*, 16:3269–3297, 2015.

Michael Curtiss, Iain Becker, Tudor Bosman, Sergey Doroshenko, Lucian Grijincu, Tom Jackson, Sandhya Kunnatur, Soren Lassen, Philip Pronin, Sriram Sankar, et al. Unicorn: A system for searching the social graph. *Proceedings of the VLDB Endowment*, 6(11): 1150–1161, 2013.

Artur Czumaj and Christian Sohler. Sublinear-time algorithms. *Property Testing*, 6390: 41–64, 2010.

Eric Dal Moro and Yuriy Krvavych. Probability of sufficiency of solvency ii reserve risk margins: Practical approximations. *ASTIN Bulletin: The Journal of the IAA*, pages 1–49, 2017.

Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

Trip Denton, M Fatih Demirci, Jeff Abrahamson, Ali Shokoufandeh, and Sven Dickinson. Selecting canonical views for view-based 3-d object recognition. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 273–276. IEEE, 2004.

Giancarlo Diana and Chiara Tommasi. Cross-validation methods in principal component analysis: a comparison. *Statistical Methods & Applications*, 11(1):71–82, 2002.

Chris Ding and Xiaofeng He. K-means clustering via principal component analysis. In *Proceedings of the Twenty-First International Conference on Machine learning*, page 29. ACM, 2004.

Mathias Drton, Bernd Sturmfels, and Seth Sullivant. *Lectures on algebraic statistics*, volume 39. Springer Science & Business Media, 2008.

John C Duchi, Michael I Jordan, Martin J Wainwright, and Yuchen Zhang. Optimality guarantees for distributed statistical estimation. *arXiv preprint arXiv:1405.0782*, 2014.

Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

HT Eastment and WJ Krzanowski. Cross-validatory choice of the number of components from a principal component analysis. *Technometrics*, 24(1):73–77, 1982.

B Efron et al. Bootstrap methods: Another look at the jackknife. *The Annals of Statistics*, 7(1):1–26, 1979.

Bradley Efron. Defining the curvature of a statistical problem (with applications to second order efficiency). *The Annals of Statistics*, pages 1189–1242, 1975.

Sanne Engelen and Mia Hubert. Fast cross-validation in robust pca. *Proceedings of Computational Statistics*, 2004.

Martin Erwig and Steve Kollmansberger. Functional pearls: Probabilistic functional programming in haskell. *Journal of Functional Programming*, 16(1):21–34, 2006.

Facebook Datacenter FAQ. URL http://www.datacenterknowledge.com/the-facebook-data-center-faq. Accessed: 05 September 2017.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015.

Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel. An algorithm for finding best matches in logarithmic expected time. *Transactions on Mathematical Software*, 3 (3):209–226, 1977.

Matteo Frigo, Charles E Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Foundations of Computer Science, 1999. 40th Annual Symposium on*, pages 285–297, 1999.

Dan Garber and Elad Hazan. Fast and simple pca via convex optimization. *arXiv preprint arXiv:1509.05647*, 2015.

Ankit Garg, Tengyu Ma, and Huy Nguyen. On communication cost of distributed statistical estimation and dimensionality. In *Advances in Neural Information Processing Systems*, pages 2726–2734, 2014.

Adria Gascon, Phillipp Schoppmann, Borja Balle, Mariana Raykova, Jack Doerner, Samee Zahur, and David Evans. Privacy-preserving distributed linear regression on high-dimensional data. *Proceedings on Privacy Enhancing Technologies*, 4:248–267, 2017.

Dan Geiger and Christopher Meek. Graphical models and exponential families. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 156–165. Morgan Kaufmann Publishers Inc., 1998.

Dan Geiger, David Heckerman, Henry King, and Christopher Meek. Stratified exponential families: graphical models and model selection. *Annals of Statistics*, pages 505–529, 2001.

Samuel Gershman, Matt Hoffman, and David Blei. Nonparametric variational inference. *ICML*, 2012.

Fabian Gieseke, Justin Heinermann, Cosmin Oancea, and Christian Igel. Buffer k-d trees: Processing massive nearest neighbor queries on gpus. In Tony Jebara and Eric P. Xing, editors, *Proceedings of the 31st International Conference on Machine Learning*, pages 172–180, 2014.

Jennifer A Gillenwater, Rishabh K Iyer, Bethany Lusch, Rahul Kidambi, and Jeff A Bilmes. Submodular hamming metrics. In *Advances in Neural Information Processing Systems*, pages 3141–3149, 2015.

Aristides Gionis, Piotr Indyky, and Rajeev Motwaniz. Similarity search in high dimensions via hashing. In *VLDB*, 1999.

Gene H Golub, Michael Heath, and Grace Wahba. Generalized cross-validation as a method for choosing a good ridge parameter. *Technometrics*, 21(2):215–223, 1979.

Jacob E Goodman, Joseph O'Rourke, and Csaba D Toth. *Handbook of discrete and computational geometry*. CRC press, 2017.

Alkis Gotovos, Hamed Hassani, and Andreas Krause. Sampling from probabilistic submodular models. In *Advances in Neural Information Processing Systems*, pages 1945–1953, 2015.

Lee-Ad Gottlieb. A light metric spanner. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 759–772. IEEE, 2015.

Lee-Ad Gottlieb and Liam Roditty. An optimal dynamic spanner for doubling metric spaces. In *European Symposium on Algorithms*, pages 478–489. Springer, 2008.

Lee-Ad Gottlieb and Shay Solomon. Light spanners for snowflake metrics. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, page 387. ACM, 2014.

Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. Efficient classification for metric data. *IEEE Transactions on Information Theory*, 60(9):5750–5759, 2014a.

Lee-Ad Gottlieb, Aryeh Kontorovich, and Pinhas Nisnevitch. Near-optimal sample compression for nearest neighbors. In *Advances in Neural Information Processing Systems*, pages 370–378, 2014b.

Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. Efficient regression in metric spaces via approximate lipschitz extension. *IEEE Transactions on Information Theory*, 2017.

Ben Green. Approximate algebraic structure. *International Congress of Mathematicians*, 2014.

Qintian Guo and Norman C Beaulieu. An approximate ml estimator for the location parameter of the generalized gaussian distribution with $p = 5$. *IEEE Signal Processing Letters*, 20(7):677–680, 2013.

Anupam Gupta, Robert Krauthgamer, and James R Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 534–543. IEEE, 2003.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

Jun Han and Qiang Liu. Bootstrap model aggregation for distributed statistical learning. *Advances in Neural Information Processing Systems*, 2016.

Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

Peter Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 1968.

Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, pages 1–9. ACM, 2014.

Michael J Healy. Colimits in memory: category theory and neural systems. In *Neural Networks, 1999. IJCNN'99. International Joint Conference on*, volume 1, pages 492–496. IEEE, 1999.

Michael J Healy. Category theory applied to neural modeling and graphical representations. In *Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, volume 3, pages 35–40. IEEE, 2000.

Michael J Healy and Thomas P Caudell. A categorical semantic analysis of art architectures. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 1, pages 38–43. IEEE, 2001.

Michael J Healy, Thomas P Caudell, and Yunhai Xiao. From categorical semantics to neural network design. In *Neural Networks, 2003. Proceedings of the International Joint Conference on*, volume 3, pages 1981–1986. IEEE, 2003.

Leonhard Held, Birgit Schrödle, and Håvard Rue. Posterior and cross-validatory predictive checks: a comparison of MCMC and inla. *Statistical Modelling and Regression Structures*, pages 91–110, 2010.

Martin Hildebrand et al. A survey of results on random random walks on finite groups. *Probability Surveys*, 2:33–63, 2005.

Kirsten Hildrum, John Kubiatowicz, Sean Ma, and Satish Rao. A note on the nearest neighbor in growth-restricted metrics. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 560–561. Society for Industrial and Applied Mathematics, 2004.

Matthew Hoffman, Francis R Bach, and David M Blei. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, pages 856–864, 2010.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

H Hotelling. Analysis of a complex of statistical variables into principle components. *Journal of Educational Psychology*, 1933.

Daniel Hsu, Sham M Kakade, Tong Zhang, et al. A tail inequality for quadratic forms of subgaussian random vectors. *Electron. Commun. Probab*, 17(52):1–6, 2012.

Yingyao Hu and Yuya Sasaki. Closed-form estimation of nonparametric models with non-classical measurement errors. *Journal of Econometrics*, 185(2):392–408, 2015.

Chung-Jung Huang, Chia-Wei Dai, Tsung-Yu Tsai, Wei-Ho Chung, and Ta-Sung Lee. A closed-form phase-comparison ml doa estimator for automotive radar with one single snapshot. *IEICE Electronics Express*, 10(7):20130086–20130086, 2013.

Zaijing Huang and Andrew Gelman. Sampling for bayesian computation with large datasets. *Available at SSRN: https://ssrn.com/abstract=1010107*, 2005.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 604–613. ACM, 1998.

Piotr Indyk and Assaf Naor. Nearest-neighbor-preserving embeddings. *ACM Transactions on Algorithms (TALG)*, 3(3):31, 2007.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

Rishabh Iyer and Jeffrey Bilmes. Submodular point processes with applications to machine learning. In *Artificial Intelligence and Statistics*, pages 388–397, 2015.

Mike Izbicki. Algebraic classifiers: a generic approach to fast cross validation, online training, and parallel training. In *ICML*, pages 1162–1170, 2013.

Mike Izbicki and Christian Shelton. Faster cover trees. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 1162–1170, 2015.

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

Joonas Jälkö, Onur Dikmen, and Antti Honkela. Differentially private variational inference for non-conjugate models. *arXiv preprint arXiv:1610.08749*, 2016.

William B Johnson and Joram Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary Mathematics*, 26(189-206):1, 1984.

William B Johnson and Assaf Naor. The johnson-lindenstrauss lemma almost characterizes hilbert space, but not quite. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 885–891. Society for Industrial and Applied Mathematics, 2009.

Robbie P. Joosten, Tim A. H. te Beek, Elmar Krieger, Maarten L. Hekkelman, Rob W. W. Hooft, Reinhard Schneider, Chris Sander, and Gert Vriend. A series of PDB related databases for everyday needs., January 2011.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

Michael I. Jordan, Jason D. Lee, and Yun Yang. Communication-efficient distributed statistical inference. *arXiv preprint arXiv:1605.07689*, 2016.

Julie Josse and François Husson. Selecting the number of components in principal component analysis using cross-validation approximations. *Computational Statistics & Data Analysis*, 56(6):1869–1879, 2012.

Pooria Joulani, András György, and Csaba Szepesvári. Fast cross-validation for incremental learning. In *IJCAI*, 2015.

Ravi Kannan, Santosh Vempala, and David Woodruff. Principal component analysis and higher correlations for distributed data. In Maria Florina Balcan, Vitaly Feldman, and Csaba Szepesvri, editors, *Proceedings of The 27th Conference on Learning Theory*, volume 35 of *Proceedings of Machine Learning Research*, pages 1040–1057, Barcelona, Spain, 13–15 Jun 2014. PMLR.

David R Karger and Matthias Ruhl. Finding nearest neighbors in growth-restricted metrics. In *Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing*, pages 741–750. ACM, 2002.

Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 938–948. Society for Industrial and Applied Mathematics, 2010.

Vishesh Karwa, Dan Kifer, and Aleksandra B Slavković. Private posterior distributions from variational approximations. *arXiv preprint arXiv:1511.07896*, 2015.

Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael Jordan. The big data bootstrap. *ICML*, 2012.

Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I Jordan. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 76(4):795–816, 2014.

Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *IJCAI*, 1995.

Imre Risi Kondor. *Group theoretical methods in machine learning*. Columbia University, 2008.

Aryeh Kontorovich and Roi Weiss. A Bayes consistent 1-nn classifier. In *Artificial Intelligence and Statistics*, pages 480–488, 2015.

D Kotschick. What is... a quasi-morphism. *Notices AMS*, 51(2):208–209, 2004.

Andreas Krause and Daniel Golovin. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, February 2014.

Robert Krauthgamer and James R Lee. Navigating nets: simple algorithms for proximity search. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 798–807. Society for Industrial and Applied Mathematics, 2004.

Robert Krauthgamer and James R Lee. The black-box complexity of nearest-neighbor search. *Theoretical Computer Science*, 348(2-3):262–276, 2005.

Robert Krauthgamer, James R Lee, Manor Mendel, and Assaf Naor. Measured descent: A new embedding method for finite metrics. In *Foundations of Computer Science, 2004. Proceedings. 45th Annual IEEE Symposium on*, pages 434–443. IEEE, 2004.

Dennis Kristensen and Oliver Linton. A closed-form estimator for the garch (1, 1) model. *Econometric Theory*, 22(2):323–337, 2006.

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

WJ Krzanowski. Cross-validation in principal component analysis. *Biometrics*, pages 575–584, 1987.

Alex Kulesza and Ben Taskar. k-dpps: Fixed-size determinantal point processes. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1193–1200, 2011.

Brian Kulis. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5 (4):287–364, 2013.

R. Deepak Kumar and K. Ramareddy. Design and implementation of cover tree algorithm on cuda-compatible gpu. *International Journal of Computer Applications*, 3(7):24–27, June 2010. Published By Foundation of Computer Science.

Richard E Ladner and Michael J Fischer. Parallel prefix computation. *Journal of the ACM (JACM)*, 27(4):831–838, 1980.

John Langford. Vowpal wabbit open source project. URL https://github.com/JohnLangford/vowpal_wabbit. Accessed: 05 September 2017.

Krzysztof Łatuszyński, Błażej Miasojedow, Wojciech Niemiro, et al. Nonasymptotic bounds on the estimation error of MCMC algorithms. *Bernoulli*, 19(5A):2033–2066, 2013.

Jason D Lee, Yuekai Sun, Qiang Liu, and Jonathan E Taylor. Communication-efficient sparse regression: a one-shot approach. *Journal of Machine Learning Research*, 2017.

Erich Leo Lehmann. *Elements of large-sample theory*. Springer Science & Business Media, 1999.

Longhai Li, Shi Qiu, Bei Zhang, and Cindy X Feng. Approximating cross-validatory predictive evaluation in bayesian latent variable models with integrated is and waic. *Statistics and Computing*, 26(4):881–897, 2016.

Mu Li, David G Andersen, and Jun Woo Park. Scaling distributed machine learning with the parameter server. In *OSDI*, 2014.

Yingyu Liang, Maria-Florina Balcan, and Vandana Kanchanapally. Distributed pca and k-means clustering. In *The Big Learning Workshop at NIPS*, 2013.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop (Vol. 8)*, 2004.

Hui Lin and Jeff Bilmes. Multi-document summarization via budgeted maximization of submodular functions. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 912–920. Association for Computational Linguistics, 2010.

Hui Lin and Jeff Bilmes. A class of submodular functions for document summarization. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 510–520. Association for Computational Linguistics, 2011.

Hui Lin and Jeff A Bilmes. Learning mixtures of submodular shells with application to document summarization. *arXiv preprint arXiv:1210.4871*, 2012.

Erik Lindström. Estimating parameters in diffusion processes using an approximate maximum likelihood approach. *Annals of Operations Research*, 151(1):269–288, 2007.

Haibin Ling and Kazunori Okada. An efficient earth mover's distance algorithm for robust histogram comparison. *PAMI*, 29:853, 2007.

Sergey Lisitsyn, Christian Widmer, and Fernando J. Iglesias Garcia. Tapkee: An efficient dimension reduction library. *Journal of Machine Learning Research*, 14:2355–2359, 2013.

Bo Liu, Xiao-Tong Yuan, Yang Yu, Qingshan Liu, and Dimitris N Metaxas. Decentralized robust subspace clustering. In *AAAI*, 2016.

Qiang Liu and Alexander Ihler. Distributed parameter estimation via pseudo-likelihood. *arXiv preprint arXiv:1206.6420*, 2012.

Qiang Liu and Alexander T Ihler. Distributed estimation, information loss and exponential families. In *Advances in Neural Information Processing Systems*, pages 1098–1106, 2014.

László Lovász. Submodular functions and convexity. In *Mathematical Programming The State of the Art*, pages 235–257. Springer, 1983.

Mario Lucic, Olivier Bachem, Morteza Zadimoghaddam, and Andreas Krause. Horizontally scalable submodular maximization. In *ICML*, pages 2981–2989, 2016.

Ulrike von Luxburg and Olivier Bousquet. Distance-based classification with lipschitz functions. *Journal of Machine Learning Research*, 5(Jun):669–695, 2004.

Chenxin Ma, Virginia Smith, Martin Jaggi, Michael I Jordan, Peter Richtárik, and Martin Takáč. Adding vs. averaging in distributed primal-dual optimization. *International Conference of Machine Learning*, 2015.

Yanyuan Ma and Anastasios A Tsiatis. On closed form semiparametric estimators for measurement error models. *Statistica Sinica*, pages 183–193, 2006.

Gustavo Malkomes, Matt J Kusner, Wenlin Chen, Kilian Q Weinberger, and Benjamin Moseley. Fast distributed k-center clustering with outliers on massive data. In *Advances in Neural Information Processing Systems*, pages 1063–1071, 2015.

Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61–68, 2014.

Rui Mao, Peihan Zhang, Xingliang Li, Xi Liu, and Minhua Lu. Pivot selection for metric-space indexing. *International Journal of Machine Learning and Cybernetics*, 2(7):311–323, 2016.

EC Marshall and DJ Spiegelhalter. Approximate cross-validatory predictive checks in disease mapping models. *Statistics in Medicine*, 22(10):1649–1660, 2003.

Jiří Matoušek. On variants of the Johnson–Lindenstrauss lemma. *Random Structures & Algorithms*, 33(2):142–156, 2008.

Peter McCullagh. What is a statistical model? *Annals of Statistics*, pages 1225–1267, 2002.

Ryan McDonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon S Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*, pages 1231–1239, 2009.

H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. Communication-efficient learning of deep networks from decentralized data. *AISTATS*, 2017.

Ted Meeds and Max Welling. Optimization Monte Carlo: Efficient and embarrassingly parallel likelihood-free inference. In *Advances in Neural Information Processing Systems*, pages 2080–2088, 2015.

Xiangrui Meng, Michael A Saunders, and Michael W Mahoney. Lsrn: A parallel iterative solver for strongly over-or underdetermined systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, 2014.

Xiangrui Meng, Joseph Bradley, Burak Yavuz, Evan Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, DB Tsai, Manish Amde, Sean Owen, et al. Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241, 2016.

Bart Mertens, Tom Fearn, and Michael Thompson. The efficient cross-validation of principal components applied to principal component regression. *Statistics and Computing*, 5(3): 227–235, 1995.

Srujana Merugu and Joydeep Ghosh. Privacy-preserving distributed clustering using generative models. In *Data Mining, 2003. ICDM 2003. Third IEEE International Conference on*, pages 211–218. IEEE, 2003.

Message Passing Forum. MPI: A message-passing interface standard. 1994.

Stanislav Minsker, Sanvesh Srivastava, Lizhen Lin, and David Dunson. Scalable and robust bayesian inference via the median posterior. In *International Conference on Machine Learning*, pages 1656–1664, 2014.

Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization: Identifying representative elements in massive data. In *Advances in Neural Information Processing Systems*, pages 2049–2057, 2013.

Baharan Mirzasoleiman, Amin Karbasi, Ashwinkumar Badanidiyuru, and Andreas Krause. Distributed submodular cover: Succinctly summarizing massive data. In *Advances in Neural Information Processing Systems*, pages 2881–2889, 2015.

Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. Distributed submodular maximization. *Journal of Machine Learning Research*, 17(238):1–44, 2016.

David A Moore and Stuart J Russell. Fast gaussian process posteriors with product trees. In *UAI*, pages 613–622, 2014.

Yadong Mu and Shuicheng Yan. Non-metric locality-sensitive hashing. In *AAAI*, 2010.

Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

Sahand Negahban, Bin Yu, Martin J Wainwright, and Pradeep K Ravikumar. A unified framework for high-dimensional analysis of m-estimators with decomposable regularizers. In *Advances in Neural Information Processing Systems*, pages 1348–1356, 2009.

Ofer Neiman. Low dimensional embeddings of doubling metrics. *Theory of Computing Systems*, 58(1):133–152, 2016.

Willie Neiswanger, Chong Wang, and Eric Xing. Asymptotically exact, embarrassingly parallel MCMC. *Uncertainty in Artificial Intelligence*, 2014.

Willie Neiswanger, Chong Wang, and Eric Xing. Embarrassingly parallel variational inference in nonconjugate models. *arXiv preprint arXiv:1510.04163*, 2015.

Christopher Nemeth and Chris Sherlock. Merging MCMC subposteriors through gaussian-process approximations. *arXiv preprint arXiv:1605.08576*, 2016.

George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14(1): 265–294, 1978.

Andrew Y Ng and Michael I Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems*, pages 841–848, 2002.

Richard Nickl and Benedikt M Pötscher. Bracketing metric entropy rates and empirical central limit theorems for function classes of besov-and sobolev-type. *Journal of Theoretical Probability*, 20(2):177–199, 2007.

Yanzhi Niu, Yi Wang, Gordon Sun, Aden Yue, Brian Dalessandro, Claudia Perlich, and Ben Hamner. The tencent dataset and KDD-cup'12. 2012.

David Novak, Martin Kyselak, and Pavel Zezula. On locality-sensitive indexing in generic metric spaces. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, pages 59–66. ACM, 2010.

Stephen Malvern Omohundro. Five balltree construction algorithms. Technical Report 89-063, International Computer Science Institute, December 1989.

Ryan ODonnell, Yi Wu, and Yuan Zhou. Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory (TOCT)*, 6(1):5, 2014.

R Kelley Pace and Dongya Zou. Closed-form maximum likelihood estimates of nearest neighbor spatial dependence. *Geographical Analysis*, 32(2):154–172, 2000.

Lior Pachter and Bernd Sturmfels. Tropical geometry of statistical models. *Proceedings of the National Academy of Sciences of the United States of America*, 101(46):16132–16137, 2004.

Lior Pachter and Bernd Sturmfels. *Algebraic statistics for computational biology*, volume 13. Cambridge University Press, 2005.

Tapio Pahikkala, Jorma Boberg, and Tapio Salakoski. Fast n-fold cross-validation for regularized least-squares. In *Scandenavian Conference on Artificial Intelligence (SCAI)*, 2006.

Rina Panigrahy, Kunal Talwar, and Udi Wieder. A geometric approach to lower bounds for approximate near-neighbor search and partial match. In *Foundations of Computer Science, 2008. FOCS'08. IEEE 49th Annual IEEE Symposium on*, pages 414–423. IEEE, 2008.

Mijung Park, James Foulds, Kamalika Chaudhuri, and Max Welling. Variational Bayes in private settings (vips). *arXiv preprint arXiv:1611.00340*, 2016.

Mihai Patrascu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011.

Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572, 1901.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Ofir Pele and Michael Werman. Fast and robust earth mover's distances. In *ICCV*, 2009.

Avi Pfeffer. Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 137:96, 2009.

Jun Qi and Javier Tejedor. Robust submodular data partitioning for distributed speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 2254–2258. IEEE, 2016.

Yongming Qu, George Ostrouchov, Nagiza Samatova, and Al Geist. Principal component analysis for dimension reduction in massive distributed data sets. In *ICDM*, 2002.

Matias Quiroz, Mattias Villani, and Robert Kohn. Speeding up MCMC by efficient data subsampling. *arXiv:1404.4178*, 2015.

R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

Parikshit Ram, Dongryeol Lee, William March, and Alexander G Gray. Linear-time algorithms for pairwise statistical problems. In *Advances in Neural Information Processing Systems*, pages 1527–1535, 2009.

Parikshit Ram, Dongryeol Lee, William March, and Alexander G. Gray. Linear-time Algorithms for Pairwise Statistical Problems. In *Advances in Neural Information Processing Systems*, 2010.

Norman Ramsey and Avi Pfeffer. Stochastic lambda calculus and monads of probability distributions. In *ACM SIGPLAN Notices*, volume 37, pages 154–165. ACM, 2002.

Rajesh Ranganath, Linpeng Tang, Laurent Charlin, and David Blei. Deep exponential families. In *Artificial Intelligence and Statistics*, pages 762–771, 2015.

R Bharat Rao, Glenn Fung, and Romer Rosales. On the dangers of cross-validation. an experimental evaluation. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 588–596. SIAM, 2008.

Benjamin Recht, Christopher Re, Stephen Wright, and Feng Niu. Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In *Advances in neural information processing systems*, pages 693–701, 2011.

Exequiel Rivas and Mauro Jaskelioff. Notions of computation as monoids. *arXiv preprint arXiv:1406.4823*, 2014.

Jonathan D Rosenblatt and Boaz Nadler. On the optimality of averaging in distributed statistical learning. *Information and Inference*, 5(4):379–404, 2016.

Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision*, pages 59–66, 1998.

Liyang Rui and KC Ho. Efficient closed-form estimators for multistatic sonar localization. *IEEE Transactions on Aerospace and Electronic Systems*, 51(1):600–614, 2015.

Tim Schäfer, Patrick May, and Ina Koch. Computation and Visualization of Protein Topology Graphs Including Ligand Information. In *German Conference on Bioinformatics 2012*, volume 26, pages 108–118, Dagstuhl, Germany, 2012.

Ioannis D Schizas and Abiodun Aduroja. A distributed framework for dimensionality reduction and denoising. *IEEE Transactions on Signal Processing*, 63(23):6379–6394, 2015.

Adam Ścibior, Zoubin Ghahramani, and Andrew D Gordon. Practical probabilistic programming with monads. In *ACM SIGPLAN Notices*, volume 50, pages 165–176. ACM, 2015.

Steven L Scott, Alexander W Blocker, Fernando V Bonassi, Hugh A Chipman, Edward I George, and Robert E McCulloch. Bayes and big data: The consensus Monte Carlo algorithm. *International Journal of Management Science and Engineering Management*, 11(2):78–88, 2016.

Nicola Segata and Enrico Blanzieri. Fast and scalable local kernel machines. *Journal of Machine Learning Research*, 11:1883–1926, August 2010.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

Ohad Shamir. Fundamental limits of online and distributed algorithms for statistical learning and estimation. In *Advances in Neural Information Processing Systems 27*, pages 163–171, 2014.

Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12:2539–2561, November 2011.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Ian Simon, Noah Snavely, and Steven M Seitz. Scene summarization for online image collections. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

Pinaki Sinha and Ramesh Jain. Extractive summarization of personal photos from life events. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.

Pinaki Sinha, Sharad Mehrotra, and Ramesh Jain. Summarization of personal photologs using multidimensional content and context. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, page 4. ACM, 2011.

Scott A Sisson and Yanan Fan. *Likelihood-free MCMC*. Chapman & Hall/CRC, New York.[839], 2011.

Vidyashankar Sivakumar, Arindam Banerjee, and Pradeep K Ravikumar. Beyond sub-gaussian measurements: High-dimensional structured estimation with sub-exponential designs. In *Advances in Neural Information Processing Systems*, pages 2206–2214, 2015.

Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

Vladimir Spokoiny. Parametric estimation. finite sample theory. *The Annals of Statistics*, 40(6):2877–2909, 2012.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of machine learning research*, 15(1):1929–1958, 2014.

Sanvesh Srivastava, Volkan Cevher, Quoc Dinh, and David Dunson. Wasp: Scalable Bayes via barycenters of subset posteriors. In *Artificial Intelligence and Statistics*, pages 912–920, 2015.

Seth Sullivant. *Algebraic statistics*. 2017.

Zoltan Szabo, Arthur Gretton, Barnabas Poczos, and Bharath Sriperumbudur. Two-stage sampled learning theory on distributions. In Guy Lebanon and S. V. N. Vishwanathan, editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, volume 38 of *Proceedings of Machine Learning Research*, pages 948–957, San Diego, California, USA, 09–12 May 2015. PMLR.

Zoltán Szabó, Bharath Sriperumbudur, Barnabás Póczos, and Arthur Gretton. Learning theory for distribution regression. *Journal of Machine Learning Research*, 17(152):1–40, 2016.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

Kunal Talwar. Bypassing the embedding: algorithms for low dimensional metrics. In *Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 281–290. ACM, 2004.

Yongqiang Tang. On the multiple imputation variance estimator for control-based and delta-adjusted pattern mixture models. *Biometrics*, 2017.

Eric J Tchetgen Tchetgen. On a closed-form doubly robust estimator of the adjusted odds ratio for a binary exposure. *American Journal of Epidemiology*, 177(11):1314–1316, 2013.

Eric Sadit Tellez and Edgar Chavez. On locality sensitive hashing in metric spaces. In *Proceedings of the Third International Conference on SImilarity Search and APplications*, pages 67–74. ACM, 2010.

Pascal Tesson and Denis Thérien. Monoids and computations. *International Journal of Algebra and Computation*, 14(05n06):801–816, 2004.

Pascal Tesson and Denis Thérien. Bridges between algebraic automata theory and complexity theory. *Bulletin of the EATCS*, 2006.

Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. Learning mixtures of submodular functions for image collection summarization. In *Advances in Neural Information Processing Systems*, pages 1413–1421, 2014.

Nikolaos Tziortziotis, Christos Dimitrakakis, and Konstantinos Blekas. Cover Tree Bayesian Reinforcement Learning. *Journal of Machine Learning Research*, 15, 2014.

Pamela Vagata and Kevin Wilfong. Scaling the facebook data warehouse to 300 pb. URL https://code.facebook.com/posts/229861827208629/scaling-the-facebook-data-warehouse-to-300-pb. Accessed: 05 September 2017.

A. W. van der Vaart. *Asymptotic statistics*. Cambridge Series in Statistical and Probabilistic Mathematics. 1998. ISBN 0-521-49603-9.

Aki Vehtari, Janne Ojanen, et al. A survey of bayesian predictive methods for model assessment, selection and comparison. *Statistics Surveys*, 6:142–228, 2012.

Roman Vershynin. Introduction to the non-asymptotic analysis of random matrices. In Y. Eldar and G. Kutyniok, editors, *Compressed Sensing, Theory and Applications*, chapter 5. 2012.

S. V. N. Vishwanathan, Nicol N. Schraudolph, Risi Kondor, and Karsten M. Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11:1201–1242, August 2010.

Jan Vondrák. Submodularity and curvature: The optimal algorithm (combinatorial optimization and discrete algorithms). *RIMS Kokyuroku Bessatsu*, 2010.

Chong Wang, John Paisley, and David Blei. Online variational inference for the hierarchical dirichlet process. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 752–760, 2011.

Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*, 2014.

Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big dataa survey. *Proceedings of the IEEE*, 104(1):34–57, 2016a.

Xiangyu Wang and David B Dunson. Parallelizing MCMC via weierstrass sampler. *arXiv preprint arXiv:1312.4605*, 2013.

Xiangyu Wang, Fangjian Guo, Katherine A Heller, and David B Dunson. Parallelizing MCMC with random partition trees. In *Advances in Neural Information Processing Systems*, pages 451–459, 2015.

Xiangyu Wang, David B Dunson, and Chenlei Leng. Decorrelated feature space partitioning for distributed sparse regression. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 802–810. Curran Associates, Inc., 2016b.

Max Welling. Herding dynamical weights to learn. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1121–1128. ACM, 2009.

SR White, Theodore Kypraios, and SP Preston. Piecewise approximate bayesian computation: fast inference for discretely observed Markov models using a factorised posterior distribution. *Statistics and Computing*, 25(2):289–301, 2015.

Svante Wold. Cross-validatory estimation of the number of components in factor and principal components models. *Technometrics*, 20(4):397–405, 1978.

David H Wolpert and David R Wolf. Estimating functions of probability distributions from a finite set of samples. *Physical Review E*, 52(6):6841, 1995.

Minjie Xu, Balaji Lakshminarayanan, Yee Whye Teh, Jun Zhu, and Bo Zhang. Distributed bayesian posterior sampling via moment sharing. In *Advances in Neural Information Processing Systems*, pages 3356–3364, 2014.

Eunho Yang, Aurelie Lozano, and Pradeep Ravikumar. Elementary estimators for high-dimensional linear regression. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 388–396, 2014.

Eunho Yang, Aurélie C Lozano, and Pradeep K Ravikumar. Closed-form estimators for high-dimensional generalized linear models. In *Advances in Neural Information Processing Systems*, pages 586–594, 2015.

Zhi-Sheng Ye and Nan Chen. Closed-form estimators for the gamma distribution derived from likelihood equations. *The American Statistician*, 2016.

Brent A Yorgey. Monoids: theme and variations (functional pearl). In *ACM SIGPLAN Notices*, volume 47, pages 105–116. ACM, 2012.

Jialin Yu. Closed-form likelihood approximation and estimation of jump-diffusions with an application to the realignment risk of the chinese yuan. *Journal of Econometrics*, 141(2): 1245–1280, 2007.

Jun Yu. Empirical characteristic function estimation and its applications. *Econometric Reviews*, 23(2):93–123, 2004.

Vicente Zarzoso, Frank Herrmann, and Asoke K Nandi. Weighted closed-form estimators for blind source separation. In *Statistical Signal Processing, 2001. Proceedings of the 11th IEEE Signal Processing Workshop on*, pages 456–459. IEEE, 2001.

Bohdan Zelinka. Tolerance in algebraic structures. *Czechoslovak Mathematical Journal*, 20 (2):179–183, 1970.

Bohdan Zelinka. Tolerance in algebraic structures. ii. *Czechoslovak Mathematical Journal*, 25(2):175–178, 1975.

Pavel Zezula, Giuseppe Amato, Vlastislav Dohnal, and Michal Batko. *Similarity search: the metric space approach*, volume 32. Springer Science & Business Media, 2006.

Yuchen Zhang, Martin J Wainwright, and John C Duchi. Communication-efficient algorithms for statistical optimization. In *Advances in Neural Information Processing Systems*, pages 1502–1510, 2012.

Yuchen Zhang, John Duchi, Michael I Jordan, and Martin J Wainwright. Information-theoretic lower bounds for distributed statistical estimation with communication constraints. In *Advances in Neural Information Processing Systems*, pages 2328–2336, 2013a.

Yuchen Zhang, John Duchi, and Martin Wainwright. Divide and conquer kernel ridge regression. In *Conference on Learning Theory*, pages 592–617, 2013b.

Yuchen Zhang, John C Duchi, and Martin J Wainwright. Divide and conquer kernel ridge regression: a distributed algorithm with minimax optimal rates. *Journal of Machine Learning Research*, 16:3299–3340, 2015.

Shen-Yi Zhao, Ru Xiang, Ying-Hao Shi, Peng Gao, and Wu-Jun Li. Scope: Scalable composite optimization for learning on spark. *AAAI*, 2017.

Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola. Parallelized stochastic gradient descent. In *Advances in Neural Information Processing Systems*, pages 2595–2603, 2010.

AV Zukhba. Np-completeness of the problem of prototype selection in the nearest neighbor method. *Pattern Recognition and Image Analysis*, 20(4):484–494, 2010.