WASHINGTON UNIVERSITY IN ST. LOUIS

School of Engineering and Applied Science
Department of Computer Science and Engineering

Learning in the Real World: Constraints on Cost, Space, and Privacy

by
Matt J. Kusner

A dissertation presented to the
Graduate School of Arts and Sciences
of Washington University in
partial fulfillment of the
requirements for the degree
of Doctor of Philosophy

August 2016
Saint Louis, Missouri

Table of Contents

List of Tables

List of Figures

Acknowledgments

Throughout my graduate work there were four forces in my life to which I owe the most credit: Sonia, Kilian, Eddie, and my family. I had no publications before I met Sonia. She taught me how to be a dedicated realist, a crucial skill for readying papers for submission, for delving into new research topics, and for living a life outside of graduate study. She is that person who really fights to make me better than I am. She has brought me balance. To my best friend.

To my advisor Kilian, who really taught me about professionalism, a person who was genius at putting himself in another person's shoes, a perfectly strict editor, the guy always testing the limits of your abilities and wondering why he had hit their boundaries.

When I joined Kilian's lab I was not prepared to do research in machine learning. Eddie saw to it that I would be. Eddie would be my model for a successful graduate student. I have known no one to work harder on research than him. Eddie's generous offers to include me on projects early in my Ph.D. were truly the foundation of my research confidence.

There were frequently large decisions that I needed outside opinions on. My family: Dad, Mom, David, Jon, and Anna, were constant coaches, always trying hard to figure out the outcome that would benefit me the most. Thank you Dad for always being confident in me, Mom for your endless excitement about anything having to do with me, David for humorously pointing out the absurdity of things, and Jon for keeping me honest. I thank Anna for the inspiration to fight against odds, to never put boundaries on yourself, and for the perspective to realize there are problems larger than I have ever known; and that even these can be overcome.

Thank you to my lab-mates: Yu Sun, who has an insatiable curiosity and a uniquely critical mind. To Gao Huang, who is never defeated by anything. To Dor Kedem, for his attention to detail and his friendship. To Stephen Tyree, for being a counter-weight to both Jake and my excessive optimism and for his cooking advice. To Jake Gardner, for his humor and discussions on all things Bayesian. To Nick Kolkin, for keeping an eye on me and his ready assistance and clear mind. To Wenlin Chen, for being my submodular consultant, constant fellow-student inspiration, and strategic advisor. To Minmin Chen, who set the bar for research productivity. To Quan Zhou, with whom I had insightful discussions about Chinese politics, Eastern medicine, and the convenience of sitting in a sunlit room. To Shuang Liu, for his big heart. To Felix Wu, for his ferocious energy and enthusiasm. To Chuan Guo, for helpfully questioning anything I said that lacked justification. To Geoff Pleis, who is a superb listener. To Jack Hessel, for welcoming me to Ithaca and his ability to get to the point of something.

Thanks to my collaborators, Roman Garnett, who has astoundingly detailed research intuitions and his good-natured wit. Gustavo Malkomes, for his humility and pointed questions. Ben Moseley, who can draft a paper in a day and for his professional advice. Karthik Sridharan, for never being unfamiliar with any theory and his effortless humor. Jason Barnes, for his honesty and his appreciation of good food. Yixuan Li, who can explain anything. Paul Upchurch, for whom it is never too late to change or improve. Vlad Niculae, who codes beautifully and is a true friend. Kunal Agrawal, for her superhuman writing and editing skills. To John Cunningham, Kavita Bala, John Hopcroft, and Olivier Chapelle.

To the perfect and tireless staff: Kelli Eckman, Lauren Huffman, Jayme Moehle, Cheryl Sickinger, Sharon Matlock, Myrna Harbinson, and Madeline Hawkins.

Thank you all.

Matt J. Kusner

Washington University in Saint Louis

August 2016

ABSTRACT OF THE DISSERTATION

Learning in the Real World: Constraints on Cost, Space, and Privacy
by
Matt J. Kusner
Doctor of Philosophy in Computer Science
Washington University in St. Louis, 2016
Professor Kilian Q. Weinberger

The sheer demand for machine learning in fields as varied as: healthcare, web-search ranking, factory automation, collision prediction, spam filtering, and many others, frequently outpaces the intended use-case of machine learning models. In fact, a growing number of companies hire machine learning researchers to rectify this very problem: to tailor and/or design new state-of-the-art models to the setting at hand.

However, we can generalize a large set of the machine learning problems encountered in practical settings into three categories: *cost*, *space*, and *privacy*. The first category (cost) considers problems that need to balance the accuracy of a machine learning model with the cost required to evaluate it. These include problems in web-search, where results need to be delivered to a user in under a second and be as accurate as possible. The second category (space) collects problems that require running machine learning algorithms on low-memory computing devices. For instance, in ecological mapping or urban planning operations we may opt to use many small unmanned aerial vehicles (UAVs) equipped with machine learning algorithms to classify an area of land. These algorithms should be small to fit within the physical memory limits of the UAV (and be energy efficient) while reliably classifying the landscape. The third category (privacy) considers problems where one wishes to run machine learning algorithms on sensitive data. It has been shown that seemingly innocuous analyses on such data can be exploited to reveal data that individuals would prefer to keep private.

Thus, nearly any algorithm that runs on patient or economic data falls under this set of problems.

We devise solutions for each of these problem categories including (i) a fast tree-based model for explicitly trading off accuracy and model evaluation time, (ii) a compression method for the k-nearest neighbor classifier, and (iii) a private causal inference algorithm that protects sensitive data.

Chapter 1

Introduction

Over the years the power of machine learning methods to produce solutions to various applications has made it one of the biggest success stories of computer science and mathematics. On a wide range of datasets and problems, machine learning (ML) models have surpassed expectations, even outperforming humans [84]. Companies all over the world have begun hiring machine learning researchers, and ML-based startups are being created at break-neck pace. From applications as diverse as forecasting injury in car crashes [37] to automated recipe recommendation [167] machine learning is making a mark.

In the past 10 years, application areas for ML have exploded in number. Many of these applications test the ability of ML to learn *while being constrained by other resources.* In this thesis we tackle three such constraints: **Cost** - cases in which one would like to make a machine learning prediction within some cost budget (e.g., time, dollars, carbon emissions); **Space** - settings that constrain the size of the machine learning model; **Privacy** - when the data on which an ML algorithm is trained is desired to be kept private (even if the model or predictions are not).

Our first algorithm, *Approximately Submodular Tree of Classifiers* (ASTC) directly learns an accurate ML classifier within a cost constraint by formulating the inherent optimization problem as a series of approximately submodular optimization problems. We nest these optimization problems (each of which produce an ML classifier) within a tree-structure so that different data points have 'specialty' classifiers designed for them. We show how ASTC matches and outperforms state-of-the-art methods while also being significantly faster to train.

Our second algorithm, *Stochastic Neighbor Compression* (SNC) learns a compressed dataset for the purpose of k-nearest neighbor classification (in which model size is the dataset size). We use a stochastic relaxation of the 1-nearest neighbor rule called the 'stochastic neighborhood' [86] and derive a continuous and smooth objective that can be easily optimized via conjugate gradient descent with simple matrix updates. We demonstrate that we can compress datasets to as low as 4% of their original size *without* sacrificing model accuracy.

Our third contribution is a technique to privatize causal inference, the first such method we are aware of. We consider a popular model for bivariate causal inference called the additive noise model (ANM) [89]. We show that we can efficiently and without noticeable losses in accuracy apply noise-addition techniques to the result of ANM causal inference to ensure that personal data remains private. We make use of the robust and widely-adopted framework of differential privacy [55] to do this. On causal inference tasks randomly chosen from a set of competition benchmarks [78] we show we can, with high probability, simultaneously release the same results as the ANM privately.

1.1 Motivation

In large part, there is often a wide disconnect between state-of-the-art machine learning done in a research context, and the machine learning models that are used in the real-world.

Whereas researchers in academia are interested primarily in optimizing some objective, the practitioner is often faced with additional *budget* constraints.

1.1.1 Real-World Examples

For instance, consider three likely real-world machine learning scenarios:

1. You are a machine learning specialist at a web-search company. You are tasked with designing a model that achieves Precision@5 (a ranking criterion) of at least α *and* that returns a result in under one millisecond. Even if there are multiple models in the machine learning literature that could reach this result, it is unlikely the strict time constraint also happens to be satisfied, especially because *it was not simultaneously optimized for.* How should you proceed?

2. You are an environmental researcher working to automate ecological identification. For instance, you may be interested in finding out what sorts of plants exist, what the climate is like, and what sorts of wildlife live in the area. For such identification purposes, you would like to deploy many small unmanned aerial vehicles (UAVs) and make sensor readings (e.g., temperature sensors, photographs, laser range-finding measurements). Each UAV can take readings of the region it is currently searching, and can communicate them back to a centralized base of operations. However, this communication requires a non-trivial amount of energy and limits search time and thus should be done sparingly. To determine if imagery is worth sending back you would like to automatically classify readings *on board* the UAV. To do so, you wish to deploy a nearest-neighbor classifier using a set of canonical readings that have been classified manually by experts. Unfortunately, the UAV has very restricted computation power

and memory. How should you limit the size of the dataset so that it runs quickly and fits within the UAV memory *while* maximizing classification accuracy?

3. You are a medical researcher who would like to leverage recent work in causal inference to determine if a new procedure is in fact causing a new-found deadly infection. You would like to publish your findings but you are worried that it will leak private information about the patients you collected data from (namely that they needed to have the procedure and/or they have the infection). How can you release the causal result *and* with high probability not reveal private patient data?

In each of the above settings the modeler is confronted with a trade-off between *maximizing an objective* (often requiring a higher cost, space, and/or publicity) and *minimizing a budget* (often forcing a lower objective). Instead of dealing with such problems as they arise, machine learning researchers need to think hard about how to explicitly design models that can be optimized *under real-world budget constraints*. The focus of this thesis is to do just that. Specifically, we will consider three specific types of budgets highlighted in the above examples: *cost*, *space*, and *privacy*. These already encompass a variety of machine learning scenarios from recommendation, face recognition, bankruptcy prediction, weather forecasting, stock market modeling, advertising, real-time machine translation, and many others. We address each budget in the following chapters and design models to directly address the trade-off at hand. To demonstrate the benefit from explicitly considering the objective/budget trade-off, we show how well our models perform on a number of the settings mentioned above.

1.2 Mathematical Background

In this section we describe the mathematical concepts that will be important for understanding the key contributions of the thesis.

1.2.1 Empirical Risk Minimization

Many results in machine learning used in practical settings can be grouped by the umbrella term *empirical risk minimization* (ERM). While ERM encompasses a broad range of machine learning models we will focus in this discussion on *linear models*. To describe this adequately we will first describe the subfield of *supervised learning* and introduce important notation that will be used in all chapters of the thesis.

Supervised Learning

In supervised learning we first assume we are given a set of data $\{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$, called a 'training set', and another $\{(\mathbf{x}^{(1)'}, y^{(1)'}), \ldots, (\mathbf{x}^{(n)'}, y^{(n)'})\}$, called a 'test set'. Each element $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$ (where usually $\mathcal{X} \subseteq \mathbb{R}^d$) is a vector of d real numbers usually referred to as a 'feature vector' or an 'input', and each of the d elements are themselves called 'features'. The other elements $y, y' \in \mathcal{Y}$ are referred to as 'labels'. They may be integer-valued, in which case this sort of learning is called 'classification' (either 'binary' if two-class or 'multi-class' otherwise), or real-valued, which is called 'regression'. This distinction is made because often the models that perform classification are quite different from those used for regression. The goal in supervised learning is to learn a function $g : \mathcal{X} \rightarrow \mathcal{Y}$, mapping from the features $\mathbf{x}, \mathbf{x}'$ to the labels y, y'. In linear ERM, we restrict g to be a function of linear parameters: $g(\mathbf{x}) := h(\mathbf{w}^\top \mathbf{x} + b)$ (where h may be a function that thresholds $\mathbf{w}^\top \mathbf{x} + b$ to the interval

$[0, 1]$, such as the sigmoidal function, or the identity function). Thus the problem of learning g reduces to learning a set of parameters $\mathbf{w}$ and a so-called bias b. Note that it is possible to incorporate the bias b into the parameter vector $\mathbf{w}$ by appending a 1 onto every $\mathbf{x}, \mathbf{x}'$ as an additional feature and setting $\mathbf{w} := [\mathbf{w}, b]^\top$. We therefore will assume b is always incorporated into $\mathbf{w}$ in this way throughout the thesis. Finally, we will assume there exists a true function $f : \mathcal{X} \to \mathcal{Y}$ that we are trying to match as closely as possible when learning g. We will learn g using the training set and evaluate how accurate it is (or how close it is to f) using the test set.

Overfitting. The reason we split the data into training and test sets is because, if we learn g on the training set, the training set will often give us an overly optimistic view of the accuracy of g: it will lead us to think that g is more accurate than it truly is. This is because g was trained specifically to get the training set correct. In doing so, we may accidentally learn a g that captures the noise of the training set, alongside the true mapping given by f. Thus, to test this, we evaluate our learned function g on the test set to see if we *overfit* to the training set. This problem of overfitting has long plagued machine learning algorithms, and was especially prevalent in the early use of multi-layer perceptron neural networks in the 1980s.

Loss and Regularization

In part, empirical risk minimization (ERM) techniques were derived in response to the problems of overfitting. There are two essential components to ERM: (1) a *loss function*; used to minimize the error of the mapping $g : \mathcal{X} \to \mathcal{Y}$ and (2) a *regularization term*; used to prevent overfitting to the training set. Below we describe each of these components in more detail and then form the generic optimization problem characteristic of ERM.

Table 1.1: Example loss functions.

loss	equation	type	popular models
0-1	$\mathbf{1}_{y \neq \mathbf{w}^\top \mathbf{x}}$	CLASSIFICATION	-
SQUARED	$(y - \mathbf{w}^\top \mathbf{x})^2$	REGRESSION	[113, 168]
HINGE	$\max\{0, 1 - y\mathbf{w}^\top \mathbf{x}\}$	CLASSIFICATION	[39]
LOGISTIC	$\log(1 + e^{-y\mathbf{w}^\top \mathbf{x}})$	CLASSIFICATION	[121]
EXPONENTIAL	$e^{-y\mathbf{w}^\top \mathbf{x}}$	CLASSIFICATION	[63]
HUBER	$\begin{cases} \frac{1}{2}(y - \mathbf{w}^\top \mathbf{x})^2 & \text{IF } \lvert y - \mathbf{w}^\top \mathbf{x} \rvert \leq \delta \\ \delta \lvert y - \mathbf{w}^\top \mathbf{x} \rvert - \frac{1}{2}\delta^2 & \text{OTHERWISE} \end{cases}$	REGRESSION	[90]

Loss. A loss function $\ell : \mathcal{X} \times \mathcal{Y} \times \mathcal{G} \rightarrow \mathbb{R}_{\geq 0}$ maps an input $\mathbf{x}$, a label y and a function g to a non-negative real number that describes how 'accurately' $g(\mathbf{x})$ 'predicts' y (as $g(\mathbf{x})$ approaches y the loss $\ell(\mathbf{x}, y, g)$ should decrease, and vice-versa). Table 1.1 gives examples of popular loss functions for linear ERM (i.e., $g(\mathbf{x}) := h(\mathbf{w}^\top \mathbf{x})$ for a possibly non-linear function h). The 0-1 loss is rarely used because of the fact that it is not differentiable at 0 and everwhere else has a derivative of 0. In regression, the squared loss is commonly used but has the draw-back that it is very sensitive to outlier preditions. This means that gradient-based optimization methods are heavily encouraged (by the size of the gradient) to push these outlier predictions smaller, rather than to push somewhat incorrect predictions to be (near) perfect. The Huber loss aims to remedy this by smoothly interpolating between the squared loss when predictions $\mathbf{w}^\top \mathbf{x}$ are close to the class label y, and the absolute loss $|y - \mathbf{w}^\top \mathbf{x}|$ when predictions are very different from the class label.

In classification, the hinge loss has been hugely influential in support vector machines (SVMs) [39]. It naturally enforces a 'large margin' between the decision boundary and the data, which tends to reduce overfitting. Given a binary classification problem between classes -1 and 1, the logistic loss learns a classifier that reports the probability of a point being in the positive class via $1/(1 + \exp(-\mathbf{w}^\top \mathbf{x}))$. Finally, the exponential loss, used in boosting [63], is designed to heavily penalize outlier predictions.

Table 1.2: Example regularization functions.

regularizer	equation	popular models
ℓ_0	$\sum_j \mathbf{1}_{w_j \neq 0}$	-
ℓ_1	$\sum_j \lvert w_j \rvert$	[168, 111]
ℓ_2^2	$\sum_j w_j^2$	[113, 39]
ℓ_1/ℓ_2 MIXED NORM	$\sum_{G \in \mathcal{G}} \sqrt{\sum_{j \in G} w_j^2}$	[13, 188, 106]
LAPLACIAN	$\sum_{k \mid k \in \mathrm{NN}(i)} (\mathbf{w}^\top \mathbf{x}^{(i)} - \mathbf{w}^\top \mathbf{x}^{(k)})^2$	[15, 179]

Regularization. A regularizer $r : \mathcal{G} \rightarrow \mathbb{R}_{\geq 0}$ maps a function g to a non-negative real number and describes how complex the function g is (or how much g is able to vary). The larger r is the more complex g is. Table 1.2 describes popular regularization functions and what they are used for. Probably the most popular regularizer is the squared ℓ_2 norm (denoted ℓ_2^2 in the table). It tends to favor weight vectors that have all small weights, heavily penalizing large weights just like the squared loss penalizing outlier predictions. If instead we wanted to force some weights to be exactly 0 we could use the ℓ_0 regularizer. It is however discontinous and thus rarely used. Instead, its convex envelope, the ℓ_1 norm [10], is used instead. It tends to drive down weights that are unimportant for prediction, making it popularly used for feature selection applications. The ℓ_1/ℓ_2 mixed norm enforces group-sparsity. Specifically, imagine our weight vector is partitioned into groups where we denote the set of all groups as $\mathcal{G} = \{G_1, G_2, \ldots, G_t\}$, and each G_i is a set of indices of the weight vector (e.g., $G_i = [35, 36, \ldots, 45]$). Then the ℓ_1/ℓ_2 first sums over each possible group in $\mathcal{G}$, and then, inside the square-root, sums over the squared elements of the weight vector for a particular group. This will force the weights of certain groups (e.g., G_3, G_8, G_9) to be practically 0 (i.e., numerically equivalent to 0), creating group-based sparsity patterns. Thus this is the group-level extension of the ℓ_1 norm. Finally the Laplacian norm enforces that similar inputs have similar predictions. Specifically, let $\mathrm{NN}(i)$ return the set of indices for inputs that are closest in Euclidean distance to $\mathbf{x}^{(i)}$. This means that for every $k \in \mathrm{NN}(i)$, the distance $\|\mathbf{x}^{(i)} - \mathbf{x}^{(k)}\|_2$ is smaller than for any of the remaining inputs in the dataset.

Thus the Laplacian norm is small so long as the prediction $\mathbf{w}^\top \mathbf{x}^{(i)}$ is close to the prediction $\mathbf{w}^\top \mathbf{x}^{(k)}$, for all $k \in \mathrm{NN}(i)$.

Optimization problem

Given a loss function ℓ and a regularizer r, linear empirical risk minimization consists of solving an optimization problem over a data sample which balances error reduction (through the loss) and model complexity (through the regularizer) as follows:

$$\min_{g} \sum_{i=1}^{n} \ell(y^{(i)}, \mathbf{w}^\top \mathbf{x}^{(i)}) + \lambda r(\mathbf{w}).$$

Solving the optimization problem. There has been a wealth of research devoted to solving the above optimization problem, recent work includes [142, 38], an excellent reference is [160]. In large part, gradient-based methods such as conjugate gradient [122], the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [8], and stochastic gradient descent [25], are used to solve the above optimization.

Selecting λ. In fact, this optimization problem is the Lagrangian of an equivalent constrained optimization problem and λ is referred to as a Lagrange multiplier (see [27] for more details on the Lagrangian). Intuitively, λ controls the trade-off between model error and complexity and is usually set by finding the value of λ that produces a model g with minimum error on a validation set (this validation set is usually split off from the original training set prior to training). This λ value can be found via grid search, random search [19], or often more quickly via Bayesian optimization [158, 66].

1.2.2 Submodularity

Submodularity is a property of discrete functions over sets that has seen broad application in machine learning [118, 12, 91, 155]. Consider a function f that maps from a discrete set of items V to the reals $\mathbb{R}$. The set V is often referred to as the *ground set*. For example, f could be the price of a set of merchandise items, or the utility of a set of users for initiating a viral marketing campaign.

To give a better intuition of the definition of submodularity, we begin by defining modularity.

Definition 1. *Let V be the ground set of all items. A function h is called **modular** if it satisfies the following property: $h(A) = \sum_{a \in A} h(a)$ for any set $A \subseteq V$.*

Effectively, a modular function is the equivalent of a linear function in discrete space, as each item contributes linearly to the total value of the set. In fact, any modular function assigns a fixed value to each item in the set (similar to a set of grocery items). Given this definition, a submodular function f is, in a sense, less than a modular function in that the value of a set A assigned by f is less than the sum of the values of the individual items:

Definition 2. *Given ground set V, a function f is called **submodular** if it satisfies one of the following equivalent definitions [131]:*

$$
\begin{aligned}
f(A) &\leq \sum_{a \in A} f(a) && \forall A \subseteq V \\
f(A) + f(B) &\geq f(A \cup B) + f(A \cap B) && \forall A, B \subseteq V \\
f(A \cup k) - f(A) &\geq f(B \cup k) - f(B) && \forall A \subseteq B \subseteq V,\ \ \forall k \in V \setminus B.
\end{aligned}
$$

The last definition describes a 'diminishing returns' property of submodular functions: the marginal change incurred by k is larger for a set A than for B, a superset of A. Intuitively,

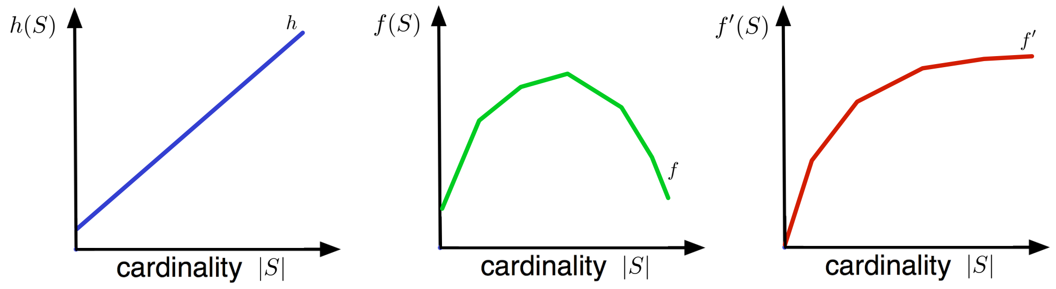


Figure 1.1: Types of set functions. (*Left.*) A modular function, in which each element contributes a certain value, independent of other elements. (*Center.*) A submodular set function, notice that the diminishing marginal returns characteristic of submodular functions becomes negative in this case. (*Right.*) A non-decreasing submodular set function.

a submodular function describes the cost of purchasing in bulk, where often the price of a product decreases per unit weight as more is purchased at once.

Figure 1.1 shows examples of modular and submodular functions. A classic result given by Nemhauser says that non-negative submodular functions that are also non-decreasing (Figure 1, *Right.*) are computationally efficient to approximately maximize [131]. Specifically, imagine we wish to solve the following optimization problem

$$\max_{S \subseteq V} f'(S) \;\; s.t., \;\; |S| \leq k, \tag{1.1}$$

where f' is a non-negative, non-decreasing submodular set function. Then, we can simply select k elements using the following greedy algorithm: At any point, given we've already selected a set A, the next element we select is j such that,

$$\operatorname*{argmax}_{j \in V \setminus A} f'(A \cup j) - f'(A).$$

Then we have the following guarantee,

Theorem 1 ([131]). *Given a non-negative, non-decreasing submodular set function f' and let G be the set selected by the above greedy algorithm. Then we have that,*

$$f'(G) \geq (1 - e^{-1}) \max_{S \subseteq V, |S| \leq k} f'(S)$$

This is a surprising result given that the optimization in eq. (1.1) is NP-hard. The intuition here is that, for submodular functions, essentially it is crucial that the set items with the largest marginal return are selected (the items selected first by the greedy algorithm). After these are selected we are given a guarantee (via submodularity) that there isn't some secret element that if only we selected it we would have received a huge gain over what we already selected. As long as we select the initial elements that provide us with the most gain, the worse we can do, Nemhauser details, is select inputs that perform $(1 - e^{-1})$ worse than optimal.

Chapter 2

Cost: Explicitly optimizing the accuracy/time-cost trade-off

Here is text of a conversation that no one in the field of machine learning (ML) has had with a practitioner (P) at in an industrial setting:

- ML: "I have a state-of-the-art model for your problem. In fact, I know everything about your problem."
- P: "Well that's great! So when can I see the results!"
- ML: "So, I should say that it takes roughly, give or take, 1 year to run."
- P: "That's no problem at all! We don't expect things to change much in a year so we can wait! Let's start right away! Not that it matters!"

Of course this is extreme. But consider that companies often rely on methods that are both *accurate* and *fast*. If a new web-search ranking algorithm is slightly better at ordering the webpages that appear in the 100-200 ranks in a list of thousands of results, and requires 30 seconds per search query, that algorithm is decidedly worse than the original algorithm. This is because web-search companies must return accurate results in under a few seconds.

In general, improvements in accuracy come at the cost of a slower algorithm. This may be due to (a) improvements in the algorithm itself that require a lengthier evaluation procedure, this sort of cost we call *evaluation cost*. Or (b) the cost of extracting specialized features, which we call *feature extraction cost*. In web-search ranking, features may describe how useful a webpage is, such as the click-through rate of the page (a relatively cheap feature) or a classification about the overall quality of the webpage (a possibly very expensive feature). Usually, the feature extraction cost dominates the runtime of a web-search ranking classifier.

As maximizing accuracy (or objective) and minimizing test-time cost are often diametrically opposed there exists a trade-off between these two quantities. In this work we aim to design models that best optimize this trade-off. We will refer to this general learning problem as **resource-efficient learning** or **budgeted learning**. We will begin by framing the accuracy/cost trade-off as an optimization problem, and formulate an exact expression for the feature extraction cost. We will then briefly describe a prior model that replaces the cost with a continuous relaxation and uses gradient-based optimization to solve for the best parameters of the model. In contrast to this tricky procedure, we reformulate the optimization as an approximately submodular set function optimization, which allows us to speed-up the optimization while matching (and sometimes outperforming) the performance of the original model.

2.1 Approximately Submodular Tree of Classifiers

In this chapter we introduce *Approximately Submodular Tree of Classifiers* (ASTC) a method to carefully extract features only if they are useful towards correct classification *and* fall within a pre-defined cost budget. We build a tree of classifiers; the internal nodes of the tree are trained to send different types of inputs to different parts of the tree. This way, features are specialized towards different subsets of inputs that they are good at predicting cheaply.

We build off the prior work of [184] who are the first to consider training a tree of cost-sensitive classifiers. In it they form a joint optimization problem over all classifiers in the entire tree and use block coordinate gradient descent to solve one node at a time. This means that when we compute the gradient with respect to one classifier we must take into account how changing it affects how data points are sent to all of its child nodes. This results in an optimization procedure that is difficult to implement, and sensitive to convergence thresholds. Their method achieves state-of-the-art trade-offs in accuracy-per-computational cost. However, it still poses notable difficulties for practitioners trying to implement and debug an algorithm in real-world settings.

We show that this complex optimization procedure can be entirely avoided. In fact, a cost-sensitive tree can be learned entirely greedily, without loss in accuracy. The key to why the greedy solution works is due to approximate submodularity [43, 77]. This is equivalent to submodularity that is off by a (small) multiplicative factor. In our case, this multiplicative factor comes in when two features do not independently contribute to a prediction, but have small synergistic effects. Specifically, if the two features together improve prediction by more than each feature individually, this is a non-submodular (or supermodular) effect. For the most part, however, feature information is redundant and the greedy algorithm achieves an approximation that is also near-optimal (similar to the submodular guarantee, see Section 1.2.2).

We begin by introducing resource-efficient or budgeted learning, where we must take into account costs incurred by the algorithm during classification. We will formulate the problem as a general constrained optimization problem, and introduce our notation.

2.1.1 Resource-Efficient Learning

In this section we formalize the general setting of resource-efficient learning and introduce our notation. The goal of resource-efficient learning is to design an optimization problem that balances two goals that are usually opposing: (1) minimizing error and (2) minimizing cost. Imagine we have functions that give the error and cost for a given model $\mathcal{M}$ and data $\mathcal{D}$: $e(\mathcal{M}, \mathcal{D})$ (error of model $\mathcal{M}$ on data $\mathcal{D}$) and $c(\mathcal{M}, \mathcal{D})$ (cost). We can then consider solving the following optimization problem,

$$\min_{\mathcal{M}} \mathbb{E}_{\mathcal{D}}[e(\mathcal{M}, \mathcal{D})] \ \text{ subject to } \mathbb{E}_{\mathcal{D}}[c(\mathcal{M}, \mathcal{D})] \leq B \tag{2.1}$$

where $\mathbb{E}_{\mathcal{D}}$ is the expectation over dataset $\mathcal{D}$ and B is a predefined cost budget that the model cannot exceed (in expectation). In practice we will use the sample average (over a fixed dataset) to approximate these expectations[*].

While there are many examples of resource-efficient algorithms that trade-off time-cost and accuracy, there is nothing that prevents algorithmic cost from being something unrelated to time, such as (a) money necessary to perform medical procedures required for the model, or (b) the amount of carbon required to generate the electricity to run the model, among many other costs.

Notation

We are given an independent and identically distributed (i.i.d.) training dataset with inputs and class labels $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{n} = (\mathbf{X}, \mathbf{y}) \in \mathcal{R}^{(n,d)} \times \mathcal{Y}^n$ (for instance $\times\mathcal{Y}=\{-1, 1\}$ for binary classification or for regression: $\mathcal{Y}=\mathcal{R}$). For every input $\mathbf{x}^{(i)}$, to obtain the value of the

[*]As an aside we note that we could also consider a model in which the *worst-case* cost is minimized, as opposed to the expected cost. We leave this direction for future work.

jth feature: $x_j^{(i)}$, we incur a cost of $c(j)$, which is the *feature extraction cost*. Importantly, once we extract this feature, we needn't pay for it again as we may cache its value for future use. Our goal is to learn a model $\mathcal{M}$ that accurately predicts the class label y of an input $\mathbf{x}$ *and* in doing so, $\mathcal{M}$ does not exceed budget B.

Let us make the optimization problem in eq. (2.1) more concrete. Consider learning a linear model, i.e. $\mathcal{M} = \boldsymbol{\beta} \in \mathcal{R}^d$, that classifies an input $\mathbf{x}$ using the rule: $\boldsymbol{\beta}^\top \mathbf{x}$. As is common practice, to learn $\boldsymbol{\beta}$, instead of minimizing the expected error $\mathbb{E}_{\mathcal{D}}[e(\boldsymbol{\beta}, \mathcal{D})] = \mathbb{E}_{\mathbf{x},y}[\mathbf{1}_{y=\boldsymbol{\beta}^\top \mathbf{x}}]$ directly, consider minimizing a surrogate 'loss' $\ell(\cdot)$, which is an upper bound on the error: $\mathbb{E}_{\mathbf{x},y}[\ell(y, \boldsymbol{\beta}^\top \mathbf{x})]$. Any convex loss function can be used within our framework (see Section 1.2.1 for details on loss functions). Additionally, as we do not have access to the data distribution directly, we estimate the expectation using our i.i.d. samples as such,

$$\mathbb{E}_{\mathbf{x},y}[\ell(y, \boldsymbol{\beta}^\top \mathbf{x})] \approx \frac{1}{n} \sum_{i=1}^{n} \ell(y^{(i)}, \boldsymbol{\beta}^\top \mathbf{x}^{(i)})$$

As for the cost in eq. (2.1) note that, if at an index j our model has zero weight; $\beta_j = 0$, then we needn't extract feature j when applying the model at test time. Thus the cost of our model is just the cost of every feature for which $\boldsymbol{\beta}$ has a non-zero weight. Formally then, our goal is to solve the following optimization problem,

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \ell(y^{(i)}, \boldsymbol{\beta}^\top \mathbf{x}^{(i)}) \;\; \text{subject to} \sum_{j:|\boldsymbol{\beta}|>0} c(j) \leq B, \tag{2.2}$$

where $\sum_{j:|\boldsymbol{\beta}|>0} c(j)$ sums over the (used) features with non-zero weight in $\boldsymbol{\beta}$. This is the simplest model for budgeted learning and one downside of it is that regardless of the input $\mathbf{x}$, it always selects the exact same set of features once it has been trained (thus the expectation for the cost term can be removed). In the following we consider a model which instead uses a tree of classifiers to select different features for different sets of inputs.

2.1.2 A Simple Example

Consider that we want to predict whether an email is a spam email or not[†][‡]. For this dataset each label $y \in \{-1, 1\}$ (i.e., this is a classification task). Another way to write the optimization problem (2.2) is as an empirical risk minimization (ERM) problem with cost-weighted regularization (see Section 1.2.1 for more details on ERM):

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \ell(y^{(i)}, \boldsymbol{\beta}^\top \mathbf{x}^{(i)}) + \lambda \sum_{j=1}^{d} c(j) \|\beta_j\|_0 \tag{2.3}$$

where the l_0-norm $\|\cdot\|_0$ in the above equation is defined as follows,

$$\|a\|_0 = \begin{cases} 0 & \text{if } a = 0 \\ 1 & \text{otherwise.} \end{cases} \tag{2.4}$$

Equation 2.3 can be shown to be the Lagrangian of the constrained optimization problem in eq. 2.2, and thus have equivalent solutions (up to scaling of the Lagrange multiplier λ). A popular loss function for classification that is fully-differentiable is the logistic loss: $\ell(y, \boldsymbol{\beta}^\top \mathbf{x}) := \log(1 + e^{-y\boldsymbol{\beta}^\top \mathbf{x}})$, which we will use here.

This unconstrained optimization problem is still difficult to solve because of the discontinuous function $\|\cdot\|_0$. Thus, we will replace it with its convex envelope [10], the ℓ_1-norm: $\|\beta_j\|_1 = |\beta_j|$ to yield our final optimization problem,

$$\min_{\boldsymbol{\beta}} \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-y\boldsymbol{\beta}^\top \mathbf{x}}) + \lambda \sum_{j=1}^{d} c(j) |\beta_j| \tag{2.5}$$

[†]For this task we will use the UCI Spambase dataset: https://archive.ics.uci.edu/ml/datasets/Spambase
[‡]As our spam dataset does not come with feature costs we will set the cost of each feature equal to 1.

Solving this optimization problem yields the classifier $\boldsymbol{\beta}^*$, which can be used to classify inputs $\mathbf{x}$ via the following rule: If $(\boldsymbol{\beta}^*)^\top \mathbf{x} \geq 0$ we classify the point in class 1, otherwise we classify the point in class -1.

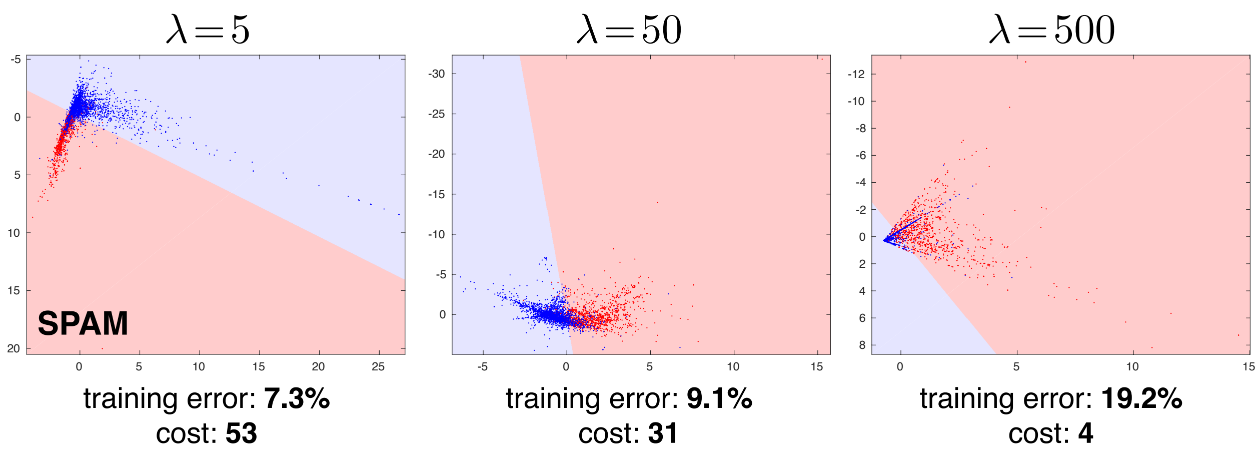


Figure 2.1: A visualization of different classifiers learned from the resource-efficient learning objective in equation (2.5).

We solve the above optimization problem for different values of $\lambda \in \{5, 50, 500\}$ and show the results in Figure 2.1. To visualize the results, we project the training set using principal components analysis (PCA) [53] down to two dimensions (these are the points in Figure 2.1, colored by their true classes). To visualize the decision boundary (the colored regions) we sample points in a grid around the training set and project them back to the original space (a 57-dimensional space, as this is the original dimensionality of spam). Once done, we classify these points into classes $1, -1$ (red, blue) via the above rule. Figure 2.1 shows how the decision boundary and the projected training set changes as λ changes. Initially, for $\lambda = 5$, we classify the training set quite accurately (7.3% error), but do so with a large cost (53 or 57 features are active). As λ is increased to 500 learned classifier becomes less accurate (19.2% error), but also more parsimonious, only using 4 of 57 features.

Table 2.1 shows the full results for many different λ. Depending on the application at hand, the practitioner can tune λ to trade-off model accuracy for cost.

Table 2.1: Training error and cost results for solving equation (2.5) on the spam dataset, under different regularization trade-offs λ.

λ	2.5	5	7.5	10	25	50	75	100	250	500
TRAIN ERR.	0.071	0.073	0.073	0.074	0.082	0.091	0.097	0.103	0.130	0.192
COST	54	53	51	48	39	31	27	25	13	4

2.1.3 Cost-Sensitive Tree of Classifiers (CSTC)

Recent work in resource-efficient learning [185] (CSTC) shows impressive results by learning multiple classifiers from eq. (2.2) which are arranged in a tree (depth D) $\boldsymbol{\beta}^1, \ldots, \boldsymbol{\beta}^{2^D-1}$. The CSTC model is shown in figure 2.2 (throughout the paper we consider the linear classifier version of CSTC). Each node v^k is a classifier whose predictions $\mathbf{x}^\top\boldsymbol{\beta}^k$ for an input $\mathbf{x}$ are thresholded by θ^k. The threshold decides whether to send $\mathbf{x}$ to the upper or lower child of v^k. An input continues through the tree in this way until arriving at a leaf node, which predicts its label.

Combinatorial Optimization. There are two road-blocks to learning the CSTC classifiers $\boldsymbol{\beta}^k$. First, because instances traverse different paths through the tree, the optimization is a complex combinatorial problem. In [185], they fix this by probabilistic tree traversal. Specifically, each classifier $\mathbf{x}^\top\boldsymbol{\beta}^k$ is trained using *all* instances, weighted by the probability that instances reach v^k. This probability is derived by squashing node predictions to the range $[0, 1]$ using the sigmoid function: $\sigma(\mathbf{x}^\top\boldsymbol{\beta}) = 1/(1 + \exp(-\mathbf{x}^\top\boldsymbol{\beta}))$.

To make the optimization more amenable to gradient-based methods, [185] convert the constrained optimization problem in eq. (2.2) to the Lagrange equivalent and minimize the *expected classifier loss* plus the *expected feature cost*, where the expectation is taken over the probability of an input reaching node v^k,

$$\min_{\boldsymbol{\beta}^k} \underbrace{\frac{1}{n}\sum_{i=1}^{n} p_i^k (y^{(i)} - \mathbf{x}^{(i)\top}\boldsymbol{\beta}^k)^2}_{\text{exp. squared loss}} + \rho\|\boldsymbol{\beta}^k\|_1 + \lambda \underbrace{\mathbb{E}[\mathcal{C}(\boldsymbol{\beta}^k)]}_{\text{exp. feature cost}}. \tag{2.6}$$

Here $p_i^k = \sigma(\mathbf{x}^{(i)\top}\boldsymbol{\beta}^k)$ is the probability that instance $\mathbf{x}^{(i)}$ traverses to v^k and ρ is the regularization constant to control overfitting. The last term is the expected feature cost of $\boldsymbol{\beta}^k$,

$$\mathbb{E}[\mathcal{C}(\boldsymbol{\beta}^k)] = \sum_{v^l \in \mathcal{P}^k} p^l \left[\sum_j c(j) \left\| \sum_{v^{k'} \in \pi^l} |\beta_j^{k'}| \right\|_0 \right], \tag{2.7}$$

The outer-most sum in the expectation (2.7) is over all leaf nodes $\mathcal{P}^k$ on paths that pass through $\boldsymbol{\beta}^k$ (see the white leaf nodes in Figure 2.3) and p^l is the probability of any input reaching such a leaf node $v^l \in \mathcal{P}^k$. The remaining terms describe the cost incurred for an input traversing to that leaf node. CSTC makes the assumption that, once a feature is extracted for an instance *it is free for future requests* for that instance. Therefore, CSTC sums over all features j and if any classifier along the path to leaf node v^l uses feature j, it is paid for exactly once (π^l is the set of nodes on the path to v^l).

ℓ_0 **norm and Differentiability.** The second optimization road-block to CSTC is that this feature cost term in eq. (2.7) is non-continuous, and is thus hard to optimize. Their solution is to derive a continuous relaxation of the ℓ_0 norm using the mixed-norm [106]. The final optimization is non-covex and not differentiable and the authors present a variational approach, introducing auxiliary variables for both ℓ_0 and ℓ_1 norms so that the optimization can be solved with cyclic block-coordinate descent.

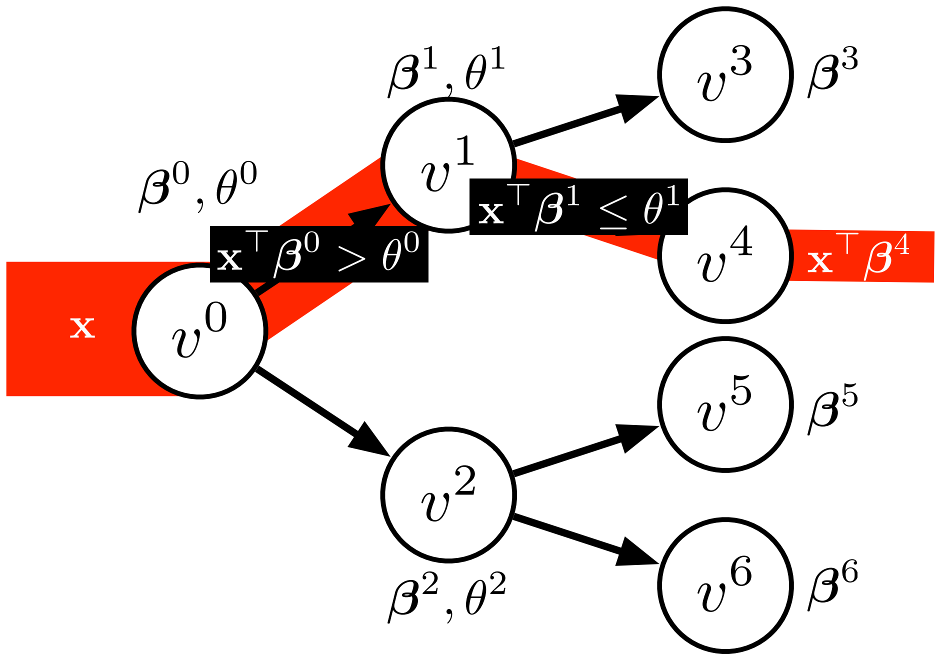


Figure 2.2: The CSTC tree (depth 3). Instances $\mathbf{x}$ are sent along a path through the tree (*e.g.*, in red) based on the predictions of node classifiers β^k. If predictions are above a threshold θ^k, $\mathbf{x}$ is sent to an upper child node, otherwise it is sent to a lower child. The leaf nodes predict the class of $\mathbf{x}$.

There are a number of practical difficulties that arise when using CSTC for a given dataset. First, optimizing a non-leaf node in the CSTC tree affects all descendant nodes via the instance probabilities p_i^k. This slows the optimization and is difficult to implement. Second, the optimization is sensitive to gradient learning rates and convergence thresholds, which require careful tuning. In the same vein, selecting appropriate ranges for hyperparameters λ and ρ may take repeated trial runs. Third, because CSTC needs to reoptimize all classifier nodes the training time is non-trivial for large datasets, making hyperparameter tuning on a validation set time-consuming. Additionally, the stationary point reached by block coordinate descent is initialization dependent. These difficulties may serve as significant barriers to entry, potentially preventing practitioners from using CSTC.

2.1.4 A Simplier Tree-Based Model

We propose a vastly simplified variant of the CSTC classifier, called Approximately Submodular Tree of Classifiers (*ASTC*). Instead of relaxing the expected cost term into a continuous function, we reformulate the entire optimization as an *approximately submodular set function optimization problem*.

ASTC nodes. We begin by considering an individual classifier $\boldsymbol{\beta}^k$ in the CSTC tree, optimized using eq. (2.6). If we ignore the effect of $\boldsymbol{\beta}^k$ on descendant leaf nodes $\mathcal{P}^k$ and previous nodes on its path π^k, the feature cost changes:

$$\mathbb{E}[\mathcal{C}(\boldsymbol{\beta}^k)] = \sum_j c(j)\|\beta_j^k\|_0. \tag{2.8}$$

This combined with the loss term is simply a weighted classifier with cost-weighted ℓ_0-regularization. We propose to *greedily* select features based on their performance/cost trade-off and to build the tree of classifiers top-down, starting from the root. We will solve one node at a time and set features 'free' that are used by parent nodes (as they need not be extracted twice). Figure 2.3 shows a schematic of the difference between the optimization of ASTC and the reoptimization of CSTC.

Resource-Constrained Submodular Optimization. An alternative way to look at the optimization of a single CSTC node is as an optimization over sets of features.[§] Let $[d] = \{1, \ldots, d\}$ be the set of all features. Define the loss function for node v^k, $\ell_k(A)$, over a set of

[§]Without loss of generality we assume from now on that for each feature vector $\mathbf{x}_j \in \mathcal{R}^n$ that $\|\mathbf{x}_j\|_2 = 1$, for all $j = 1, \ldots, d$, and that $\|\mathbf{y}\|_2 = 1$. Additionally $\mathbf{x}_j, \mathbf{y}$ also have zero mean.

features $A \subseteq [d]$ as such,

$$\ell_k(A) = \min_{\boldsymbol{\beta}^k} \frac{1}{n} \sum_{i=1}^{n} p_i^k (y_i - \delta_A(\mathbf{x}^{(i)})^\top \boldsymbol{\beta}^k)^2 \tag{2.9}$$

where we treat probabilities p_i^k as indicator weights: $p_i^k = 1$ if input $\mathbf{x}^{(i)}$ is sent to v^k, and to 0 otherwise. Define $\delta_A(\mathbf{x})$ as an element-wise feature indicator function that returns feature x_a if $a \in A$ and 0 otherwise. Thus, ℓ_k is the squared loss of the optimal model using only (a) inputs that reach v^k and (b) the features in set A. Our goal is to select a set of features A that have low cost, and simultaneously have a low optimal loss $\ell_k(A)$.

Certain problems in constrained set function optimization have very nice properties. Particularly, a class of set functions, called *submodular* set functions, have been shown to admit simple near-optimal greedy algorithms [131]. For the resource-constrained case, each feature (set item) j has a certain resource cost $c(j)$, and we would like to ensure that the cost of selected features fall under some resource budget B. For a submodular function s that is non-decreasing and non-negative the resource-constrained set function optimization,

$$\max_{A \subseteq [d]} s(A) \;\; \text{subject to} \;\; \sum_{j \in A} c(j) \leq B \tag{2.10}$$

can be solved near-optimally by greedily selecting set elements $j \in [d]$ that maximize s as such,

$$g_t = \operatorname*{argmax}_{j \in [d]} \left[\frac{s(G_{t-1} \cup j) - s(G_{t-1})}{c(j)} \right]. \tag{2.11}$$

Where we define the greedy ordering $G_{t-1} = (g_1, g_2, \ldots, g_{t-1})$. To find g_t we evaluate all remaining set elements $a \in [d] \setminus G_{t-1}$ and pick the element $g_t = \hat{j}$ for which $s(G_{t-1} \cup \hat{j})$ increases the most over $s(G_{t-1})$ *per cost*. Let $G_{\langle L \rangle} = (g_1, \ldots, g_T)$ be the largest feasible greedy set, having total cost L (*i.e.*, $\sum_{t=1}^{T} c(g_t) = L \leq B$ and $L + c(g_{T+1}) > B$). It has been proved [161] that for any non-decreasing and non-negative submodular function s and some budget B, eq. (2.11) gives an approximation ratio of $(1 - e^{-1}) \approx 0.63$ with respect to the optimal set with cost $L \leq B$. Call this set $\mathcal{C}^*_{\langle L \rangle}$. Then, $s(G_{\langle L \rangle}) \geq (1 - e^{-1}) s(\mathcal{C}^*_{\langle L \rangle})$ for the resource-constrained optimization (2.10).

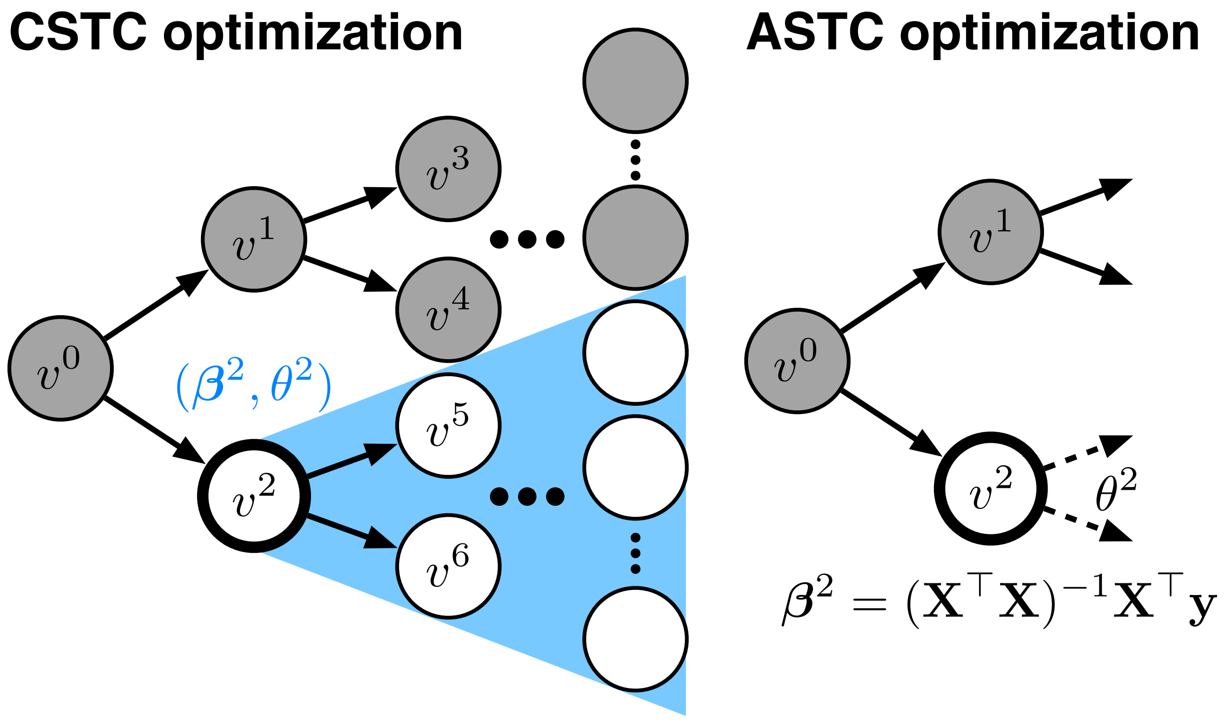

CSTC optimization
ASTC optimization
v^0
v^1
v^2
v^3
v^4
v^5
v^6
$(\boldsymbol{\beta}^2, \theta^2)$
θ^2
$\boldsymbol{\beta}^2 = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$

Figure 2.3: The optimization schemes of CSTC and ASTC. *Left:* When optimizing the classifier and threshold of node v^2, $(\boldsymbol{\beta}^2, \theta^2)$ in CSTC, it affects all of the descendant nodes (highlighted in blue). If the depth of the tree is large (*i.e.*, larger than 3), this results in a complex and expensive gradient computation. *Right:* ASTC on the other hand optimizes each node greedily using the familiar ordinary least squares closed form solution (shown above). θ^2 is set by binary search to send half of the inputs to each child node.

2.1.5 Greedy Optimization

In this section we demonstrate that optimizing a single CSTC node, and hence the CSTC tree, greedily is *approximately submodular*. We begin by introducing a modification to the set function (2.9). We then connect this to the approximation ratio of the greedy cost-aware algorithm eq. (2.11), demonstrating that it produces near-optimal solutions. The resulting optimization is very simple to implement and is described in Algorithm 1.

Approximate Submodularity. To make ℓ_k amenable to resource-constrained set function optimization (2.10) we convert the loss minimization problem into an equivalent label 'fit' maximization problem. Define the set function z_k,

$$z_k(A) = \frac{\mathrm{Var}(\mathbf{y}; p^k) - \ell_k(A)}{\mathrm{Var}(\mathbf{y}; p^k)} \tag{2.12}$$

where $\mathrm{Var}(\mathbf{y}; p^k) = \sum_i p_i^k (y^{(i)} - \bar{y})^2$ is the variance of the training label vector $\mathbf{y}$ multiplied by 0/1 probabilities p_i^k ($\bar{y}$ is the mean predictor). It is straightforward to show that maximizing $z_k(\cdot)$ is equivalent to minimizing ℓ_k. In fact, the following approximation guarantees hold for $z_k(\cdot)$ constructed from a wide range of loss functions (via a modification of [77]). As we are interested in developing a new method for CSTC training, we focus purely on the squared loss. Note that $z_k(\cdot)$ is always non-negative (as the mean predictor is a worse training set predictor than a predictor using any one feature, assuming that the feature takes on more than one value). To see that it is also non-decreasing note that $z_k(\cdot)$ is precisely the squared multiple correlation R^2 [49], [98], which is known to be non-decreasing.

If the features are orthogonal then $z_k(\cdot)$ is submodular [108]. However, if this is not the case it can be shown that $z_k(\cdot)$ is *approximately submodular* and has a *submodularity ratio*, defined as such:

Definition 3 ([43, 77])**.** *Any non-negative set function $z(\cdot)$ has a* ***submodularity ratio*** *γ as follows,*

$$\sum_{s \in S} \Big[z(L \cup \{s\}) - z(L) \Big] \geq \gamma \Big[z(L \cup S) - z(L) \Big],$$

The submodularity ratio ranges from 0 ($z(\cdot)$ is not submodular) to 1 ($z(\cdot)$ is submodular) and measures how close a function $z(\cdot)$ is to being submodular.

The submodularity ratio in general is non-trivial to compute. However, we can take advantage of prior work [43] which shows that the submodularity ratio of z_k (2.12) is further bounded. Define $\mathcal{C}_A^k$ as the covariance matrix of $\tilde{\mathbf{X}}$, where $\tilde{\mathbf{x}}^{(i)} = p_i^k \delta_A(\mathbf{x}^{(i)})$ (inputs weighted by the probability of reaching v^k, using only the features in A). It has been shown [43] that for $z_k(\cdot)$, it holds that $\gamma \geq \lambda_{min}(\mathcal{C}_A^k)$, where $\lambda_{min}(\mathcal{C}_A^k)$ is the minimum eigenvalue of $\mathcal{C}_A^k$.

Approximation Ratio. As in the submodular case, we can optimize $z_k(\cdot)$ subject to the resource constraint that the cost of selected features must total less than a resource budget B. This optimization can be done greedily using the rule described in eq. (2.11). The following theorem—which is proved for any non-decreasing, non-negative, approximately submodular set function [77]—gives an approximation ratio for this greedy rule.

Theorem 2 ([77])**.** *The greedy algorithm selects an ordering G such that,*

$$z_k(G_{\langle L \rangle}) > (1 - e^{-\gamma}) z_k(S^*_{\langle L \rangle})$$

where $G_{\langle L \rangle} = (g_1, g_2, \ldots, g_T)$ is the greedy sequence truncated at cost L, such that $\sum_{i=1}^{T} c(g_i) = L \leq B$ and $S^*_{\langle L \rangle}$ is the set of optimal features having cost L.

Thus, the approximation ratio depends directly on the submodularity ratio of $z_k(\cdot)$. For each node in the CSTC tree we greedily select features using the rule described in (2.11). If we are not at the root node, we set the cost of features used by the parent of v^k to 0, and select them immediately (as we have already paid their cost). We fix a new-feature budget B—identical for each node in the tree—and then greedily select new features up to cost B for each node. By setting probabilities p_i^k to 0 or 1 depending on if $\mathbf{x}^{(i)}$ traverses to v^k, learning each node is like solving a unique approximately submodular optimization problem, using only the inputs sent to that node. Finally, we set node thresholds θ^k to send half of the training inputs to each child node.

We call our approach Approximately Submodular Tree of Classifiers (ASTC), which is shown in Algorithm 1. The optimization is much simpler than CSTC.

Algorithm 1 ASTC in pseudo-code.

```
1:  Inputs: $\{\mathbf{X}, \mathbf{y}\}$; tree depth $D$; node budget $B$, costs $c$
2:  Set the initial costs $c^1 = c$
3:  for $k = 1$ to $2^D - 1$ nodes do
4:      $G = \emptyset$
5:      while budget not exceeded: $\sum_{g \in G} c^k(g) \leq B$ do
6:          Select feature $j \in [d]$ via eq. (2.11)
7:          Add to node-specific features: $G = G \cup \{j\}$
8:      end while
9:      Solve $\boldsymbol{\beta}^k$ using weighted ordinary least squares
10:     if $v^k$ is not a leaf node, with children $v^l$ and $v^u$ then
11:         Set child probabilities:
            $$p_i^u = \begin{cases} 1 & \text{if } p_i^k > \theta^k \\ 0 & \text{otherwise} \end{cases} \qquad p_i^l = \begin{cases} 1 & \text{if } p_i^k \leq \theta^k \\ 0 & \text{otherwise} \end{cases}$$
12:         Set child feature costs: $c^u = c^l = c^k$
13:         Free used features: $c^u(G) = c^l(G) = 0$
14:     end if
15: end for
16: Return $\{\boldsymbol{\beta}^1, \boldsymbol{\beta}^2, \ldots \boldsymbol{\beta}^{2^D-1}\}$
```

2.1.6 Fast Selection via QR-Decomposition

Equation (2.11) requires solving an ordinary least squares problem, eq. (2.9), when selecting the feature that improves $z_k(\cdot)$ the most. This requires a matrix inversion which typically takes $\mathrm{O}(d^3)$ time. However, because we only consider selecting one feature at a time we can avoid the inversion for $z_k(\cdot)$ altogether using the QR decomposition. Let $G_t = (g_1, g_2, \ldots, g_t)$

be our current set of greedily-selected features. For simplicity let $\mathbf{X}_{G_t} = \delta_{G_t}(\mathbf{X})$, the data masked so that only features in G_t are non-zero. Computing $z_k(\cdot)$ requires computing the weighted squared loss, eq. (2.9), which, after the QR decomposition requires no inverse. Redefine $\mathbf{x}^{(i)} = \sqrt{\frac{p_i^k}{n}}\mathbf{x}^{(i)}$ and $y^{(i)} = \sqrt{\frac{p_i^k}{n}}y^{(i)}$, then we have,

$$\ell_k(G_t) = \min_{\boldsymbol{\beta}^k}(\mathbf{y} - \mathbf{X}_{G_t}\boldsymbol{\beta}^k)^\top(\mathbf{y} - \mathbf{X}_{G_t}\boldsymbol{\beta}^k). \tag{2.13}$$

Let $\mathbf{X}_{G_t} = \mathbf{QR}$ be the QR decomposition of $\mathbf{X}_{G_t}$. Plugging in this decomposition, taking the gradient of $\ell_k(G_t)$ with respect to $\boldsymbol{\beta}^k$, and solving at 0 yields [82],

$$\boldsymbol{\beta}^k = \mathbf{R}^{-1}\mathbf{Q}^\top\mathbf{y}$$

The squared loss for the optimal $\boldsymbol{\beta}^k$ is,

$$\begin{aligned}\ell_k(G_t) &= (\mathbf{y} - \mathbf{QRR}^{-1}\mathbf{Q}^\top\mathbf{y})^\top(\mathbf{y} - \mathbf{QRR}^{-1}\mathbf{Q}^\top\mathbf{y})\\ &= (\mathbf{y} - \mathbf{QQ}^\top\mathbf{y})^\top(\mathbf{y} - \mathbf{QQ}^\top\mathbf{y})\\ &= \mathbf{y}^\top\mathbf{y} - \mathbf{y}^\top\mathbf{QQ}^\top\mathbf{y}. \end{aligned} \tag{2.14}$$

Imagine we have extracted t features and we are considering selecting a new feature a. The immediate approach would be to recompute $\mathbf{Q}$ including this feature and then recompute the squared loss (2.14). However, computing $\mathbf{q}_{t+1}$ (the column corresponding to feature a) can be done incrementally using the Gram–Schmidt process:

$$\begin{aligned}\mathbf{q}_{t+1} =& \frac{\mathbf{X}_a - \sum_{j=1}^t(\mathbf{X}_a^\top\mathbf{q}_j)\mathbf{q}_j}{\|\mathbf{X}_a - \sum_{j=1}^t(\mathbf{X}_a^\top\mathbf{q}_j)\mathbf{q}_j\|_2}\\ =& \frac{\mathbf{X}_a - \mathbf{QQ}^\top\mathbf{X}_a}{\|\mathbf{X}_a - \mathbf{QQ}^\top\mathbf{X}_a\|_2}\end{aligned}$$

where $\mathbf{q}_1 = \mathbf{X}_{g_1}/\|\mathbf{X}_{g_1}\|_2$ (recall g_1 is the first greedily-selected feature). Finally, in order to select the best next feature using eq. (2.11), for each feature a we must compute,

$$\begin{aligned}\frac{z_k(G_t \cup a) - z_k(G_t)}{c(a)} &= \frac{-\ell_k(G_t \cup a) + \ell_k(G_t)}{\mathrm{Var}(\mathbf{y};\mathbf{p}^k)c(a)} \\ &= \frac{\mathbf{y}^\top \mathbf{Q}_{1:t+1}\mathbf{Q}_{1:t+1}^\top \mathbf{y} - \mathbf{y}^\top \mathbf{Q}\mathbf{Q}^\top \mathbf{y}}{\mathrm{Var}(\mathbf{y};\mathbf{p}^k)c(a)} \\ &= \frac{(\mathbf{q}_{t+1}^\top \mathbf{y})^2}{\mathrm{Var}(\mathbf{y};\mathbf{p}^k)c(a)} \end{aligned} \tag{2.15}$$

where $\mathbf{Q}_{1:t+1} = \big[\mathbf{Q}, \mathbf{q}_{t+1}\big]$. The first two equalities follow from the definitions of $z_k(\cdot)$ and $\ell_k(\cdot)$. The third equality follows because $\mathbf{Q}$ and $\mathbf{Q}_{1:t+1}$ are orthogonal matrices.

We can compute all of the possible $\mathbf{q}_{t+1}$ columns, corresponding to all of the remaining features a in parallel, call this matrix $\mathbf{Q}_{\text{remain}}$. Then we can compute eq. (2.15) vector-wise on $\mathbf{Q}_{\text{remain}}$ and select the feature with the largest corresponding value of $z_k(\cdot)$.

Complexity. Computing the ordinary least squares solution the naive way for the $(t+1)^{th}$ feature: $(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$ requires $\mathrm{O}\big(n(t+1)^2 + (t+1)^3\big)$ for the covariance multiplication and inversion. This must be done $d-t$ times to compute $z_k(\cdot)$ for every remaining feature. Using the QR decomposition, computing $\mathbf{q}_{t+1}$ requires $\mathrm{O}\big(nt\big)$ time and computing eq. (2.15) takes $\mathrm{O}\big(n\big)$ time. As before, this must be done $d-t$ times for all remaining features, but as mentioned above both steps can be done in parallel.

2.1.7 Experimental Results

In this section, we evaluate our approach on a real-world feature-cost sensitive ranking dataset: the Yahoo! Learning to Rank Challenge dataset. We begin by describing the

Table 2.2: Training speed-up of ASTC over CSTC as a function of tree budgets on Yahoo! and Forest datasets.

	Yahoo!							**Forest**					
Cost Budgets	10	52	86	169	468	800	1495	3	5	8	13	23	50
ASTC	119x	52x	41x	21x	15x	9.2x	6.6x	8.4x	7.0x	6.3x	4.9x	3.1x	1.4x
ASTC, soft	121x	48x	46x	18x	15x	8.2x	6.4x	8.0x	6.4x	5.7x	4.5x	2.8x	1.5x

Table 2.3: Training speed-up of ASTC over CSTC for CIFAR and MiniBooNE datasets.

	CIFAR					**MiniBooNE**						
Cost Budgets	9	24	76	180	239	4	5	12	14	18	33	47
ASTC	5.6x	2.3x	0.68x	0.25x	0.14x	7.4x	7.9x	5.5x	5.2x	4.1x	3.1x	2.0x
ASTC, soft	5.3x	2.3x	0.62x	0.27x	0.13x	7.2x	6.2x	5.9x	4.2x	4.3x	2.5x	1.7x

dataset and show Precision@5 per cost compared against CSTC [185] and another cost-sensitive baseline. We then present results on a diverse set of non-cost sensitive datasets, demonstrating the flexibility of our approach. For all datasets we evaluate the training times of our approach compared to CSTC for varying tree budgets.

Yahoo! Learning to Rank. To judge how well our approach performs in a particular real-world setting, we test ASTC on the Yahoo! Learning to Rank Challenge data set [35]. The dataset consists of $473,134$ web documents and $19,944$ queries. Each input $\mathbf{x}^{(i)}$ is a query-document pair containing 519 features, each with extraction costs in the set $\{1,5,20,50,100,150,200\}$. The unit of cost is in weak-learner evaluations (i.e., the most expensive feature takes time equivalent to 200 weak-learner evaluations). We remove the mean and normalize the features by their ℓ_2 norm, as is assumed by the submodularity ratio bound analysis. We use the Precision@5 metric, which is often used for binary ranking datasets.

Figure 2.4 compares the test Precision@5 of CSTC with the greedy algorithm described in Algorithm 1 (*ASTC*). For both algorithms we set a maximum tree depth of 5. We also compare against setting the probabilities p_i^k using the sigmoid function $\sigma(\mathbf{x}^\top \boldsymbol{\beta}^k) = 1/(1+$

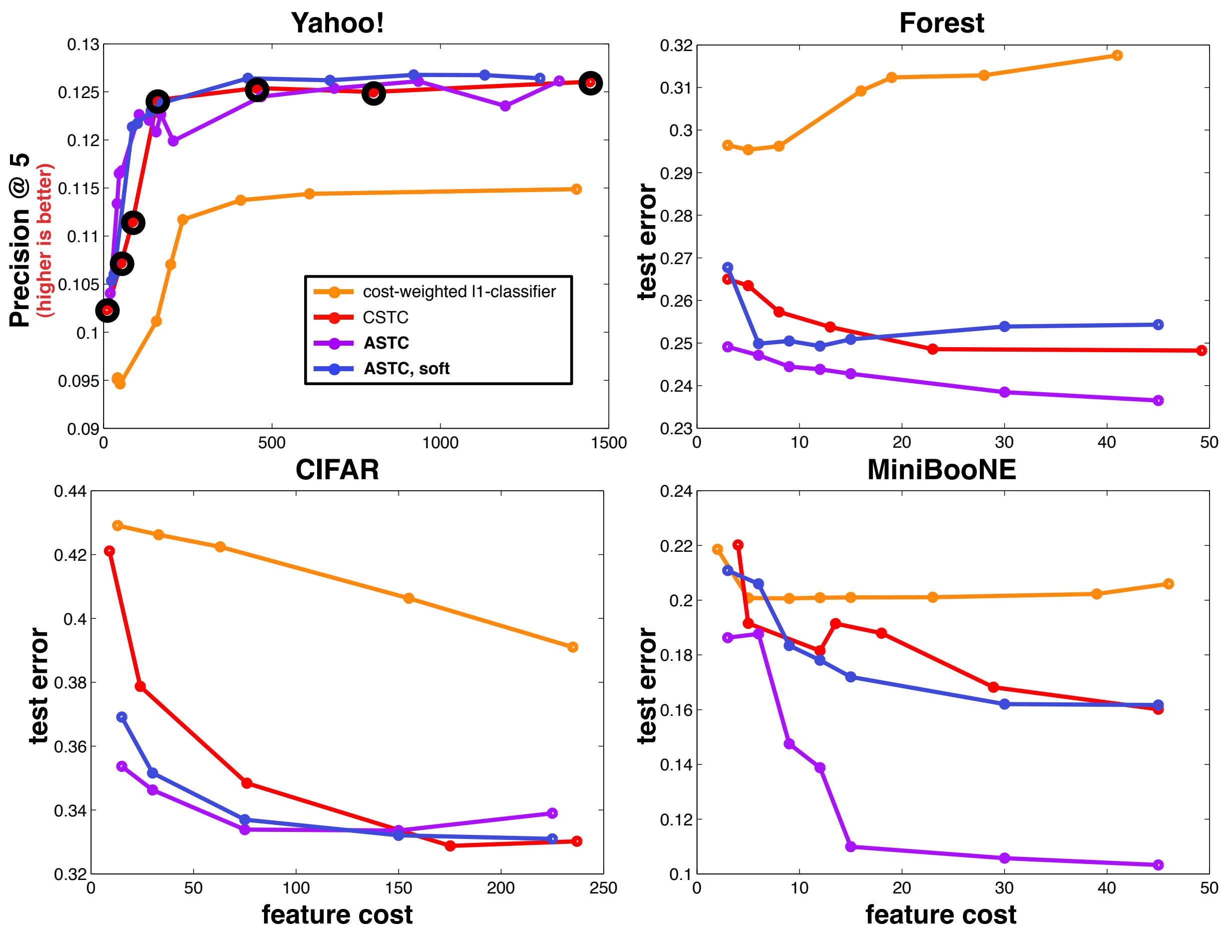


Figure 2.4: Plot of ASTC, CSTC, and a cost-sensitive baseline on on real-world feature-cost sensitive dataset (Yahoo!) and three non-cost sensitive datasets (Forest, CIFAR, MiniBooNE). ASTC demonstrates roughly the same error/cost trade-off as CSTC, sometime improving upon CSTC. For Yahoo! circles mark the CSTC points that are used for training time comparison, otherwise, all points are compared.

$\exp(-\mathbf{x}^\top\boldsymbol{\beta}^k))$ on the node predictions as is done by CSTC (*ASTC, soft*). Specifically, the probability of an input $\mathbf{x}$ traversing from parent node v^k to its upper child v^u is $\sigma(\mathbf{x}^\top\boldsymbol{\beta}^k-\theta^k)$ and to its lower child v^l is $1-\sigma(\mathbf{x}^\top\boldsymbol{\beta}^k-\theta^k)$. Thus, the probability of $\mathbf{x}$ reaching node v^k *from the root* is the product of all such parent-child probabilities from the root to v^k. Unlike CSTC, we disregard the effect $\boldsymbol{\beta}^k$ has on descendant node probabilities (see Figure 2). Finally, we also compare against a single cost-weighted ℓ_1-regularized classifier.

We note that the ASTC methods perform just as good, and sometimes slightly better, than state-of-the-art CSTC. All of the techniques perform better than the single ℓ_1 classifier, as it must extract features that perform well for all instances. CSTC and ASTC instead may select a small number of expert features to classify small subsets of test inputs.

Forest, CIFAR, MiniBooNE. We evaluate ASTC on three very different non-cost sensitive datasets in tree type and image classification (*Forest*, *CIFAR*), as well as particle identification (*MiniBooNE*). As the feature extraction costs are unknown we set the cost of each feature α to $c_\alpha=1$. As before, ASTC is able to improve upon the performance of CSTC.

Training Time Speed-up. Tables 2.2 and 2.3 show the speed-up of our approaches *over* CSTC for various tree budgets. For a fair speed comparison, we first learn a CSTC tree for different values of λ, which controls the allowed feature extraction cost (the timed settings on the Yahoo! dataset are marked with black circles on Figure 2.4, whereas all points are timed for the other datasets). We then determine the cost of unique features extracted at each node in the learned CSTC tree. We set these unique feature costs as individual node budgets B^k for ASTC methods and greedily learn tree features until reaching the budget for each node. We note that on the real-world feature-cost sensitive dataset Yahoo! the ASTC methods are consistently faster than CSTC. Of the remaining datasets ASTC is faster in all settings except for three parameter settings on CIFAR. One possible explanation for the reduced speed-ups is that the training set of these datasets are much smaller (Forest: $n = 36,603$ $d = 54$; CIFAR: $n = 19,761$ $d = 400$; MiniBooNE: $n = 45,523$ $d = 50$) than Yahoo! ($n = 141,397$ and $d = 519$). Thus, the speed-ups are not as pronounced and the small, higher dimensionality CIFAR dataset trains slightly slower than CSTC.

2.2 Related Work

2.2.1 Cost-Sensitive Regularization

Prior to CSTC [185], a natural approach to controlling feature resource cost is to use ℓ_1-regularization to obtain a sparse set of features [60]. One downside of these approaches is that certain inputs may only require a small number of cheap features to compute, while other inputs may require a number of expensive features.

2.2.2 Cascades

Perhaps the most famous resource efficient model is the Adaboost [63] cascade by [174]. The cascade consists of stages of Adaboost classifiers that either 'reject' or 'pass on' inputs for further classification. The face detection cascade is designed so that earlier stages are inexpensive and can eliminate the majority of inputs as 'non-face' image patches. [26] design SoftCascade which uses information from multiple previous stages, and how well an instance passes each stage to decide whether to eliminate inputs from the cascade. After these works, there was an explosing of interest in cascade classifiers [112, 145, 138, 35]. In all of these works, earlier classifiers are cheap and simple classifiers (designed to easily eliminate inputs) while later classifiers are expensive and complex (designed to carefully sift out negative inputs to discard). As powerful as the cascade is, its primary strength is dealing with imbalanced datasets (in which negative examples vastly outnumber positive examples). However, it is non-trivial to design cascades for multi-class classification.

2.2.3 Tree-Based Models

This scenario motivated the development of CSTC [185]. Prior to CSTC, there has been a number of works towards efficient classification within tree-based models, including [46], who speed up training and testing time required for label trees [17] for fast object detection. [52] construct resource-efficient decision trees by combining feature cost with its mutual information with the classification label. Recently, [129] construct feature cost sensitive random forests based on minimizing the worst-case (maximum) cost-per-impurity of each split in the tree.

2.2.4 Decision-Making Schemes

There are a number of models that use decision-making schemes to speed-up test-time classification. [30] uses a Markov decision process (MDP), trained with on-policy reinforcement learning to adaptively select features for each instance. [172] consider learning a set of sequential multi-class decisions, inspired by the solution of an MDP. Follow-up work [176] derives a tight convex surrogate to the original optimization and derives a linear program to solve the optimization efficiently. On the other hand there are a number of works that formulate the problem as a partially-observable Markov decision process (POMDP) [151, 97, 99, 65] and select features based on their information gain. [83] use imitation learning from a coach to reduce the regret for selecting features online that are both cost-effective and accurate. [101] design an object detection system that maximizes average precision per cost using a reinforcement learning strategy. Weiss & Taskar design a feature cost sensitive model for structured prediction tasks such as articulated pose estimation and optical character recognition [182]. Their method uses Q-learning [178] to predict the value of individual features at test time. Finally, Wang et al., formulate the feature selection decision problem as a directed acyclic graph structure and use dynamic programming to solve the budgeted learning problem [177].

2.2.5 Submodularity

Feature selection has been tackled by a number of submodular optimization papers [108, 43, 42, 109]. Surprisingly, until recently, there were relatively few papers addressing resource-efficient learning. Recently [77] introduce SpeedBoost which greedily learns weak learners that are cost-effective using (orthogonal) matching pursuit. Work last year [190] considers an online setting in which a learner can purchase features in 'rounds'. Perhaps most similar to our work is work [74] which learns a policy to adaptively select features to optimize a set function. Differently, their work assumes the set function is fully submodular and every policy action only selects a single element (feature). To our knowledge, this work is the first tree-based model to tackle resource-efficient learning using approximate submodularity.

2.3 Conclusion

We have introduced Approximately Submodular Tree of Classifiers (ASTC), making use of recent developments in approximate submodular optimization to develop a practical near-optimal greedy method for feature-cost sensitive learning. The resulting optimization yields an efficient objective update scheme that allows one to train ASTC up to 120 times faster than CSTC.

One limitation of this approach is that the approximation guarantee does not hold if features are preprocessed. Specifically, for web-search ranking, it is common to first perform gradient boosting to generate a set of limited-depth decision trees. The predictions of these decision trees can then be used as features (this is demonstrated in the non-linear version of CSTC [184]). Despite the lack of approximation guarantees, adding this non-linearity may improve the accuracy of ASTC.

Additionally, the cost of a set of features may be less than the sum of their individual costs. Instead, groups of features may be 'discounted'. One common example are feature descriptors for object detection (i.e., HOG [41] and SIFT [117] features). Each descriptor can be thought of as a group of features. Once a single feature from the group is selected for making a classification, the remaining features in the group become 'free', as they were already computed for the descriptor. Extending ASTC to model these features would notably widen the scope of the approach.

Overall, by presenting a simple, efficient, near-optimal method for feature-cost sensitive learning we hope to bridge the gap between machine learning models *designed* for real-world industrial settings and those *implemented* in such settings. Without the need for specialized tuning and with faster training we truly believe our approach can be rapidly incorporated into the ever-increasing number of large-scale machine learning applications that could benefit the most.

Chapter 3

Space: A model for compressing the k-nearest neighbor rule

Low-memory computing devices are pervasive. Indeed, mobile phones, tablets, electronic-reading devices, smart watches, augmented and virtual reality glasses, are increasingly computationally-capable. The ability to collect and learn from data tailored to such devices is an exciting frontier for machine learning.

It is easy to imagine potential applications for machine learning in these contexts:

1. In wildlife mapping it could be extremely useful to have multiple unmanned aerial vehicles (UAVs) that are able to classify whether or not they may have taken images of wildlife in a nature reserve.

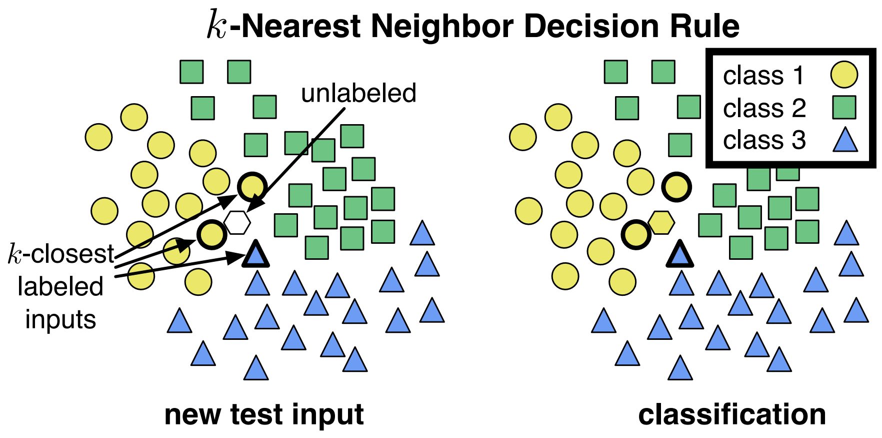
2. In developing countries, in places where doctors are scarce and internet is non-existent, it could be useful to have health workers go door-to-door with tablets to collect health information and classify individuals as likely having an illness or not. This also has the benefit that once the classification is made, the sensitive personal data can be deleted (more on this in the next chapter).

3. As civic infrastructure and buildings age, it may prove useful to attach small computing devices that measure various aspects of material stress and wear. These devices could also classify whether a structure is in danger of collapsing in the near future.

In all of these scenarios, running classification *on the device*, as opposed to repeatedly sending data back to a centralized server and retrieving a classification, is crucial. This is because communication often requires significant power (e.g., communicating imagery in example 1 may severely limit search time, and in example 3 may require sensors to be replaced often or even overheat) or communication may not even be possible (as in example 2).

If we wish to run machine learning algorithms on mobile devices, it is imperative to address the primary constraint of such devices: often, to improve mobility, they have highly-restricted memory sizes. Thus the question is: How can we design machine learning models that are as accurate as they are compact? Critically, there is a natural trade-off that arises as shrinking a model usually comes at the price of model expressibility. The goal of this chapter is to devise a technique to directly optimize this trade-off. One surprising observation we will make is that compressing a machine learning model can sometimes improve its generalization accuracy. This is because model-shrinking can act as a form of regularization, biasing the model at the expense of flexibility.

We will begin by considering the k-nearest neighbor (kNN) model; a classification technique that is widely used in practice, but naturally requires a non-trivial amount of memory (i.e., it must store the entire training set). To reduce the model size of kNN we propose to learn an entirely new 'compressed' training set that is optimized to closely approximate the original training set. We formalize this compression procedure as an optimization problem using a continuous relaxation of the 1-nearest neighbor rule first introduced by Hinton and Roweis, 2002 [86]. This allows us to directly learn a new training set using gradient descent. We



k-Nearest Neighbor Decision Rule
unlabeled
k-closest labeled inputs
new test input
classification
class 1
class 2
class 3

Figure 3.1: The k-nearest neighbor rule first described by Cover & Hart [40], in which a test point is classified by the majority class of its nearest neighbor in the training set.

show that the learned compressed training sets can achieve the same (and sometimes even better) generalization error as the full training set, at a fraction of the size.

3.1 Stochastic Neighbor Compression

The k-nearest neighbors (kNN) decision rule classifies an unlabeled input by the majority label of its k nearest training inputs. Figure 3.1 shows an example k-nearest neighbor classification. It is one of the oldest and most intuitive classification algorithms [40]. Nevertheless, when paired with domain knowledge [16, 153] or learned distance metrics [73, 44, 181], it is highly competitive in many machine learning applications [171]. As machine learning algorithms are increasingly used in application settings, *e.g.* recommender systems [147], the kNN rule is particularly attractive because its predictions are easily explained.

An important drawback of kNN is its slow test-time performance. Since it must compute the distances between the test input and all elements in the training set, it takes $O(dn)$ with respect to the data dimensionality d and the training set size n. Similarly, space requirements

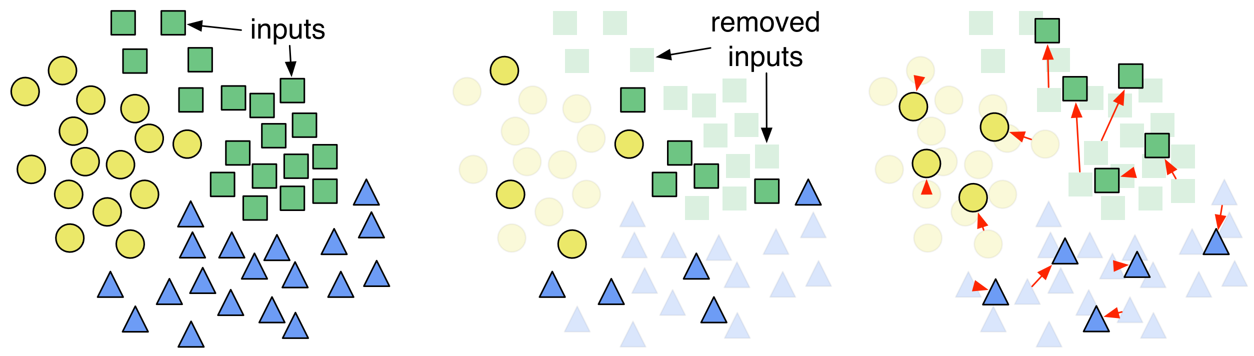


Figure 3.2: An illustration of the individual stages of SNC. The input data (*left*) is first subsampled uniformly (*middle*) and then optimized to minimize leave-one-out nearest neighbor error (*right*).

are also $O(dn)$, as the entire training set needs to be stored. This high time and space complexity makes computing the decision rule impracticable for time critical applications and large-scale datasets—a problem that is likely to remain relevant as datasets continue to grow.

There are three high-level approaches for speeding up the testing. First is to reduce the number of distance computations to some polylogarithmic function in n, through clever tree data structures, such as cover/ball trees [21, 133], or hashing functions [72, 4]. Although they often yield impressive speed ups, these methods still store the entire training set and their performance tends to deteriorate with increasing (intrinsic) data dimensionality. The second approach is to reduce the data dimensionality d through supervised dimensionality reduction, *e.g.* large margin nearest neighbors (LMNN) [180], which is particularly effective in combination with tree data structures. The third approach is to compress the training set by reducing the number of data inputs n. Prior works often involve data set *condensing* (or *thinning*) [81, 5], which subsample the training data according to clever rules and remove redundant inputs. Alternative algorithms shrink the data to few cluster centers [32, 103], which can be optimized with multi-phase initialization procedures [45, 115]. Others learn prototypes by 'softening' the kNN decision rule at test-time [20], preventing the use of tree data structures.

In this chapter, we introduce a novel approach for data set compression, *Stochastic Neighbor Compression (SNC)*, which falls into the third category of algorithms. SNC compresses the training data by learning a new set of m synthetic reference vectors, where $m \ll n$. Figure 3.2 illustrates our algorithm schematically. We initialize our compressed set with a small subset of the training set, sampled uniformly at random. We then optimize the position of these inputs directly to minimize the classification error on the training set. To this end, we relax the kNN rule into a stochastic neighborhood framework [73, 86], which allows us to approximate the classification error of the training set with a continuous and differentiable function.

We are making four novel contributions: 1. we introduce and derive SNC, a novel data compression algorithm for kNN; 2. we demonstrate the efficacy of SNC on seven real world data sets and show that on all tasks it outperforms existing algorithms for data set reduction and on 4/7 data sets kNN on the full training set obtains even *higher* error rates than kNN with SNC—at a staggeringly low compression ratio of only 4%; 3. We conjecture and observe empirically that SNC substantially increases robustness of kNN to (label) noise; 4. We demonstrate that SNC works well alongside existing algorithms — such as ball trees, hashing, and dimensionality reduction — that speed up nearest neighbor classification. In fact, it adds impressive speed ups of one order of magnitude on top of the existing state-of-the-art.

3.1.1 The Stochastic Neighborhood

We denote the training data to be a set of input vectors $\{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}\} \subset \mathcal{R}^d$, arranged as columns in matrix $\mathbf{X} \in \mathcal{R}^{d \times n}$, and corresponding labels $\{y^{(1)}, \ldots, y^{(n)}\} \subseteq \mathcal{Y}$, where $\mathcal{Y}$ contains some finite number of classes.[¶]

Our approach draws from two ideas in machine learning that use stochastic neighborhood distributions: stochastic neighborhood embeddings [86], and neighborhood components analysis [73]. Here, we describe both in some detail.

Stochastic Neighborhood Embedding (SNE) [86] is an algorithm to visualize a given data set by learning a low-dimensional embedding in 2d or 3d. For two points $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$, we define the *dissimilarity measure* d_{ij}^2; it is commonly an element of the Gaussian kernel $d_{ij}^2 = \gamma_i^2 \|\mathbf{x}^{(i)} - \mathbf{x}^{(j)}\|^2$, where γ_i^2 is the precision of the Gaussian distribution. The authors define a stochastic neighborhood, which captures the neighborhood relation between inputs $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$ through probability p_{ij} of the event that $\mathbf{x}^{(i)}$ is assigned $\mathbf{x}^{(j)}$ as its nearest neighbor,

$$p_{ij} = \frac{\exp(-d_{ij}^2)}{\sum_{k=1}^{n} \exp\left(-d_{ik}^2\right)}. \tag{3.1}$$

The low dimensional embedding is optimized to approximately preserve the stochastic neighborhood distribution of the input data. More precisely, SNE minimizes the KL-divergence between the original (high dimensional) stochastic neighborhoods and the induced neighborhoods in the low dimensional space. This approach was recently further refined [173] to yield improved visualizations by substituting the local Gaussian distributions with Student t-distributions in the input space.

[¶]Throughout this manuscript we will abuse notation slightly and treat $\mathbf{X}$ as a set of column vectors or matrix interchangeably (*i.e.* we allow the notation $\mathbf{x}^{(i)} \in \mathbf{X}$ but also $\mathbf{X}^\top \mathbf{y}$).

Neighborhood Components Analysis (NCA) [73] is an algorithm that uses stochastic neighborhoods to learn a Mahalanobis pseudo-metric, $d_{ij} = \|\mathbf{A}(\mathbf{x}^{(i)} - \mathbf{x}^{(j)})\|$. This metric is parameterized by a matrix $\mathbf{A}$ and is incorporated into the stochastic neighborhood in (3.1). In contrast to SNE, NCA is a supervised learning algorithm and optimizes this metric explicitly for kNN. To improve the kNN accuracy, it maximizes an approximation of the leave-one-out (LOO) training accuracy of the 1 stochastic neighbor rule. Under this rule, an input $\mathbf{x}^{(i)}$ with label $y^{(i)}$ is classified correctly if its nearest neighbor is any $\mathbf{x}^{(j)} \neq \mathbf{x}^{(i)}$ from the same class ($y^{(j)} = y^{(i)}$). The probability of this event can be stated as

$$p_i = \sum_{j: y^{(j)} = y^{(i)}} p_{ij}, \tag{3.2}$$

where we define $p_{ii} = 0$. NCA learns $\mathbf{A}$ by maximizing (3.2) over all inputs $\mathbf{x}^{(i)} \in \mathbf{X}$.

3.1.2 How to Compress a Dataset

In this section, we describe our approach, called *Stochastic Neighbor Compression (SNC).* SNC is inspired by the seminal works from Section 3.1.1 and uses a stochastic neighborhood distribution to reduce the training set, with n data inputs, into a compressed set with m vectors, where $m \ll n$. This much smaller compressed set is then used as a reference set during kNN testing. For standard kNN implementations, the test time and space complexity reduce from $\mathrm{O}\big(nd\big)$ to only $\mathrm{O}\big(md\big)$.

Stochastic Reference. We learn a new compressed set of reference vectors $\mathbf{Z} = [\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}]$ with labels $\hat{y}^{(1)}, \ldots, \hat{y}^{(m)}$. This data set is initialized by uniformly subsampling m vectors from $\mathbf{X}$, while maintaining their exact labels. The labels $\hat{y}^{(i)}$ will be fixed throughout, whereas the vectors $\mathbf{Z}$ will be optimized. Let us define the probability that input $\mathbf{x}^{(i)}$ is assigned $\mathbf{z}^{(j)}$ as nearest reference vector as

$$p_{ij} = \frac{\exp(-\gamma^2 \|\mathbf{x}^{(i)} - \mathbf{z}^{(j)}\|^2)}{\sum_{k=1}^{m} \exp(-\gamma^2 \|\mathbf{x}^{(i)} - \mathbf{z}^{(k)}\|^2)}. \tag{3.3}$$

Input $\mathbf{x}^{(i)}$ is classified correctly if and only if it is paired with a reference vector from the same class $\hat{y}^{(j)} = y^{(i)}$. The probability of this event is precisely given by (3.2).

Objective. Ideally, we want $p_i = 1$ for all $\mathbf{x}^{(i)} \in \mathbf{X}$, corresponding to 100% classification accuracy of $\mathbf{X}$ on $\mathbf{Z}$. It is straight-forward to see that the KL-divergence [73] between this "perfect" distribution and p_i is

$$KL(1||p_i) = -\log(p_i). \tag{3.4}$$

Our goal is to position the compressed set $\mathbf{Z}$ such that as many training inputs as possible are classified correctly. In other words, we need p_i to be close to 1 for all inputs $\mathbf{x}^{(i)} \in \mathbf{X}$. Hence, we define our loss function to sum over the KL-divergences (3.4) for all inputs in $\mathbf{X}$,

$$\mathcal{L}(\mathbf{Z}) = -\sum_{i=1}^{n} \log(p_i) \tag{3.5}$$

Gradient with respect to Z. In order to state the gradients in simpler form, we first define two additional matrices $\mathbf{Q}, \mathbf{P} \in \mathcal{R}^{n \times m}$ as

$$[\mathbf{Q}]_{ij} = (\delta_{y^{(i)},y^{(j)}} - p_i), \;\; [\mathbf{P}]_{ij} = \frac{p_{ij}}{p_i}.$$

Here, $\delta_{y^{(i)},y^{(j)}} \in \{0,1\}$ denotes the Dirac Delta function and takes on value 1 if and only if $y^{(i)} = y^{(j)}$. Although we omit the details of the derivation, this notation allows us to state

Algorithm 2 SNC in pseudo-code.

```
1: Inputs: $\{\mathbf{X}, \mathbf{y}\}$; new (compressed) data set size $m$
2: Initialize $\mathbf{Z}$ by class-based sampling $m$ inputs from $\mathbf{X}$
3: Learn $\mathbf{Z}$ with conj. gradient descent, eq. (3.6)
4: Return $\mathbf{Z}$
```

the gradient of $\mathcal{L}$ with respect to the compressed set $\mathbf{Z}$ entirely in matrix operations,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{Z}} = 2\Bigg(\mathbf{X}\Big(\mathbf{Q} \circ \mathbf{P}\Big) - \mathbf{Z}\Delta\Big(\Big(\mathbf{Q} \circ \mathbf{P}\Big)^{\top}\mathbf{1}_n\Big)\Bigg), \tag{3.6}$$

where $\circ$ is the Hadamard (element-wise) product, $\mathbf{1}_n$ is the $n\times 1$ vector of all ones, and $\Delta(\cdot)$ signifies placing a vector along the diagonal of an otherwise 0 matrix.

Computational complexity. The computational complexity of each gradient descent iteration with respect to $\mathbf{Z}$ costs $\mathrm{O}\big(nm\big)$ to compute $(\mathbf{Q}\circ\mathbf{P})$, $\mathrm{O}\big(dnm\big)$ to compute $\mathbf{X}(\mathbf{Q}\circ\mathbf{P})$, and $\mathrm{O}\big(dm^2\big)$ to compute $\mathbf{Z}\Delta((\mathbf{Q}\circ\mathbf{P})^{\top}\mathbf{1}_n)$, resulting in $\mathrm{O}\big(dmn\big)$ overall complexity.

Implementation. We optimize $\mathbf{Z}$ by minimizing (3.5) with conjugate gradient descent (we use a freely-available Matlab implementation[*]) and provide our implementation of SNC as open source available for download at `http://tinyurl.com/msovcfu`. The individual steps of the SNC approach are described in Algorithm 2.

[*]`http://tinyurl.com/minimize-m`

3.1.3 Metric Learning Extension

Drawing directly on ideas proposed in NCA [73], for additional flexibility, we can extend (3.3) with an affine feature transformation matrix $\mathbf{A}$,

$$p_i = \sum_{j:\hat{y}^{(j)}=y^{(i)}} \frac{\exp\left(-\|\mathbf{A}(\mathbf{x}^{(i)} - \mathbf{z}^{(j)})\|^2\right)}{\sum_{k=1}^{m} \exp(-\|\mathbf{A}(\mathbf{x}^{(i)} - \mathbf{z}^{(k)})\|^2)}. \tag{3.7}$$

Let us denote the corresponding loss function as $\mathcal{L}^{\mathbf{A}}$. The resulting objective can be minimized with respect to $\mathbf{A}$ and $\mathbf{Z}$. This extension allows us to automatically optimize the feature scale γ^2 by setting $\mathbf{A}=\gamma^2\mathbf{I}$; rescale features with a diagonal matrix $\mathbf{A} = \Delta$; or induce dimensionality reduction with a rectangular matrix, *i.e.* $\mathbf{A}\in\mathcal{R}^{r\times d}$.

Gradients w.r.t. A and Z. The gradient of $\mathcal{L}$ w.r.t. $\mathbf{A}$ is similar to the NCA gradient [73],

$$\frac{\partial \mathcal{L}^{\mathbf{A}}}{\partial \mathbf{A}} = -2\mathbf{A} \sum_{i=1}^{n} \sum_{j=1}^{m} \frac{p_{ij}}{p_i} q_{ij} \mathbf{v}_{ij} \mathbf{v}_{ij}^{\top}, \tag{3.8}$$

where we abbreviate $\mathbf{v}_{ij} = (\mathbf{x}^{(i)} - \mathbf{z}^{(j)})$ and $q_{ij} = [\mathbf{Q}]_{ij}$. The gradient of $\mathcal{L}^{\mathbf{A}}$ w.r.t. $\mathbf{Z}$ results in a modification of (3.6):

$$\frac{\partial \mathcal{L}^{\mathbf{A}}}{\partial \mathbf{Z}} = 2\mathbf{A}^{\top}\mathbf{A}\frac{\partial \mathcal{L}}{\partial \mathbf{Z}}. \tag{3.9}$$

Due to additional multiplications by $\mathbf{A}$, the time complexity of each gradient iteration increases to $\mathrm{O}\left(d^2mn + d^3\right)$. (The cubic term drops if $\mathbf{A}$ is diagonal or of the form $\gamma^2\mathbf{I}$.)

Table 3.1: Characteristics of datasets used in evaluation.

Dataset Statistics			
NAME	n	$\lvert\mathcal{Y}\rvert$	d (d_L)
YALE-FACES	1961	38	8064 (100)
ISOLET	3898	26	617 (172)
LETTERS	16000	26	16 (16)
ADULT	32562	2	123 (50)
W8A	49749	2	300 (100)
MNIST	60000	10	784 (164)
FOREST	100000	7	54 (54)

Practical aspects. We find that the form $\mathbf{A} = \gamma^2\mathbf{I}$ leads to comparable results as the diagonal or full matrix. It has the added advantage that it is substantially faster and that it alleviates the need to multiply the test data with $\mathbf{A}$, as the kNN decision rule is invariant to uniform feature scaling. Optimizing the scaling factor γ^2 does however affect the compressed set $\mathbf{Z}$. Also, optimizing γ^2 with conjugate gradient descent prior to optimizing $\mathbf{Z}$ (instead of jointly) leads to similar results and may be preferred in practice due to its improved running time. In our experiments, we initialize γ^2 with cross-validation and optimize it prior to learning. We pick the initialization that yields minimal training error.

3.1.4 Experimental Results

We evaluate the efficacy of SNC on seven benchmark data sets. We begin with a brief description of the individual learning tasks and then evaluate the compression ratio and test error, training time, sensitivity to noise and finally visualize the SNC decision boundary and reference vectors.

Dataset descriptions. We evaluate SNC and other training set reduction baselines on seven classification datasets detailed in Table 3.1. *YaleFaces* [71] consists of gray-scale face

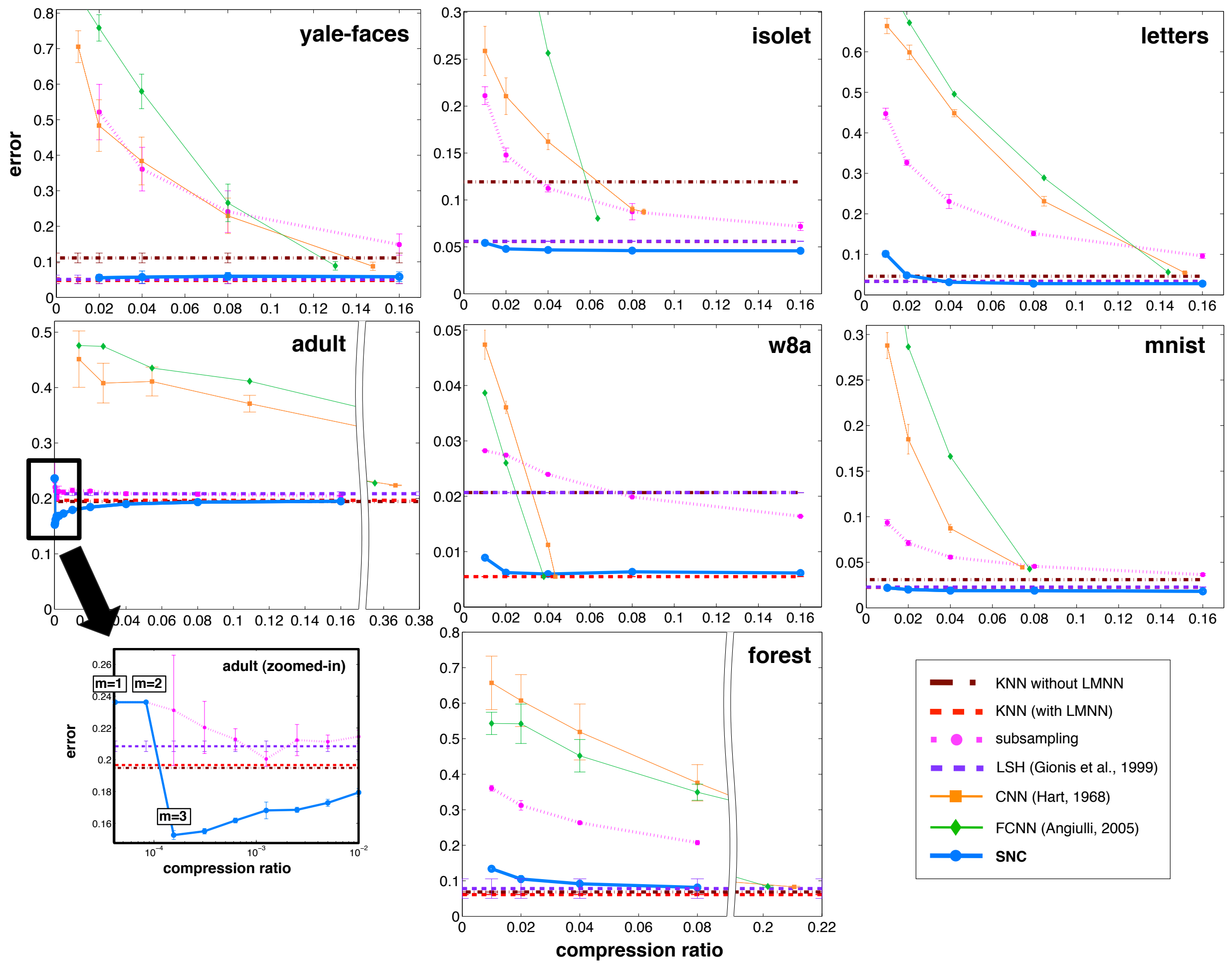


Figure 3.3: kNN test error rates after training set compression obtained by various algorithms. See text for details.

images of 38 individuals under varying (label invariant) illumination conditions. The task is to identify the individual from the image pixel values. *Isolet** is a collection of audio feature vectors of spoken letters from the English alphabet. The task is to identify which letter is spoken based on the recorded (and pre-processed) audio signal. *Letters** is derived from images of English capital letters, the learning task is to identify the letter type based on font specific features. *Adult** contains U.S. census income and personal statistics, the task

*`http://tinyurl.com/uci-ml-data`

is to predict if a household has an income over $\$50,000$. *W8a*[†] contains keyword attributes extracted from web pages and the task is to categorize a web page into a one of a set of predefined categories. *MNIST*[‡] is a set of gray-scale handwritten digit images; the task is to identify the digit value from the image pixels. *Forest*[*] contains geological and map-based data, and the task is to identify the type of ground cover (e.g. tree type) in a given area of a map.

In addition to these data sets, we also used the *USPS*[§] handwritten digits data set as a development set. As we evaluated SNC multiple times on its test portion, and we want to clearly separate development and evaluation data, we are not including it in this result section. The results are comparable to the benchmark sets included in this section.

Preprocessing. Large Margin Nearest [180] Neighbors (LMNN) is an effective method to speed up kNN search through dimensionality reduction, that is, by reducing the parameter d in the running time $O(nd)$. LMNN learns a projection into a lower dimensional space that speeds up kNN while maintaining (or improving) the classification error. We can validate this observation on our benchmark tasks and therefore pre-process all datasets with LMNN, which improves the kNN speed and accuracy for nearly all datasets.

For Isolet and MNIST, the dimensionality is reduced as described in the original paper [181]. For the remaining datasets, if the input dimensionality d is ≥ 200 it is reduced to 100 with LMNN, and if it is between 100 and 200, it is reduced to 50. For the YaleFaces data set, we follow prior work [181] and first rescale the images to 48x42 pixels, then reduce the dimensionality with PCA (to 200) while omitting the leading 5 principal components (which capture large variations in image brightness). Finally, we apply LMNN to reduce the

[†]`http://tinyurl.com/libsvm-data`
[‡]`http://tinyurl.com/mnist-data`
[§]`http://tinyurl.com/usps-data`

dimensionality further to $d = 100$. Table 3.1 lists the dimensionality of each dataset before (d) and after (d_L) LMNN preprocessing. For Forest, we use the same procedure as previous work [5] who subsample uniformly.

Implementation. For purposes of this evaluation, SNC is initialized by subsampling inputs based on the class distribution up to the desired compression rate. For testing, we use the 1NN (1 Nearest Neighbor) rule for all of the algorithms. Results of SNC and of any initialization-dependent baselines are reported with the average and standard deviation over 5 runs. Neither YaleFaces nor Forest have predefined test sets and so we report the average and standard deviations in performance over 5 and 10 splits, respectively.

Baselines. Figure 3.3 shows the test error of kNN evaluated on a compressed training set generated by SNC (solid blue line) with $\mathbf{A} = \gamma^2\mathbf{I}$. We depict varying rates of compression, and compare against the following related baselines: 1. kNN without compression both before (brown dotted line) and after LMNN dimensionality reduction (red dotted line), 2. kNN on a reference set subsampled uniformly from the training set based on the class balance (pink dotted line), 3. Approximate kNN via locality-sensitive hashing *LSH* [72], using a previous implementation [3] (purple dotted line) 4. *CNN* [81] (orange line), and 5. *FCNN* [5] (green line). CNN and FCNN select training-consistent subsets and are arguably the most popular training set reduction algorithms for kNN. Both methods are briefly described in Section 3.2.

Both SNC and subsampling can be performed at varying rates of compression and are plotted at compression ratios $\{1\%, 2\%, 4\%, 8\%, 16\%\}$. (We omit the 16% compression rate for forest, due to its large size.) kNN with and without LMNN does not do any compression and for better readability both methods are depicted as horizontal lines. For LSH we cross-validate over the number of tables and hash functions and select the fastest setting that has equal or less leave-one-out error compared to kNN without LSH (for larger datasets, we performed the LSH cross-validation on class-balanced subsamples of the training set: 10% subsamples of Adult, W8a and MNIST, and 5% of Forest). Identical to SNC, we plot average LSH test error and standard deviation for multiple random initializations. CNN and FCNN do not have a parameter for compression ratio. However both algorithms incrementally add inputs to the reference set and for comparison to our method with variable compression rate we have depicted the errors of partial compressed sets. For CNN, we also plot standard deviations as the algorithm is order dependent. In full disclosure, we want to point out that both CNN and FCNN as intended by the authors would only output a single compressed set (the rightmost point of the respective plot lines).

Error and Compression. We observe several general trends from the results in Figure 3.3. Simply subsampling the training set yields high error rates, showing that *optimization* of the compressed data set is crucial to obtain good compression/error trade-offs. SNC performs extremely well on all data sets even with a compression ratio as low as 2%. In fact, SNC clearly outperforms all other compression methods in terms of compression/error trade-off across *all* data sets—often yielding significantly lower test error rates than CNN and FCNN under only a fraction of their final compression ratio. SNC at $\geq 4\%$ matches (up to significance) or outperforms LSH error on every dataset. Further, on almost all data sets (except W8a and Forest), kNN with SNC can match the test error rates (with and without LMNN) using the full training data even at very high compression ratios ($2-4\%$). In fact, on 4/7 data sets, kNN with SNC at a compression ratio of 4% achieves even *lower* test error than kNN using the full training set.

Table 3.2: SNC training times.

Training Times					
Dataset	Compression Ratio				
	1%	2%	4%	8%	16%
YALE-FACES	–	4s	6s	9s	15s
ISOLET	11s	17s	28s	50s	1m 26s
LETTERS	41s	1m 18s	2m 44s	4m 34s	8m 13s
ADULT	2m 27s	4m 1s	7m 39s	12m 51s	23m 18s
W8A	6m 5s	10m 19s	19m 26s	39m 12s	1h 12m
MNIST	17m 18s	36m 43s	1h 13m	2h 17m	4h 57m
FOREST	17m 38s	33m	55m 44s	1h 45m	–

The last observation is particularly surprising, as one would expect an increase in error due to compression, rather than a reduction. However, one explanation for this effect is that SNC optimizes the compressed data especially to do well with kNN classification. A good example is the Adult data set, which has a strong class imbalance with 78% of the data belonging to one class. In other words, this is a data set in which kNN barely outperforms predicting the most common label. With high compression, SNC can position its learned reference vectors in a way to learn a simpler decision boundary and outperform kNN drastically with 0.15 vs. 0.20 error (zoomed in portion of the graph).

Time complexity and training time. Training times for SNC, averaged across 5 runs, are given in Table 3.2. SNC (with $\mathbf{A}=\gamma^2\mathbf{I}$) is expected to scale with complexity $O(dmn)$ per iteration. As the size of the compressed set m doubles between columns, training times roughly double as well. Variations in dimensionality and training set sizes among datasets make comparisons along the columns less precise, but training times do not seem to exceed theoretical expectations. All experiments were performed on an 8-core Intel L5520 CPU with 2.27GHz clock frequency.

Speed-up at test time. At testing time, standard implementations of kNN testing will compute the distances between each test point and *all* reference set points. However, dimensionality reduction [180] or clever structures, such as ball trees [133] or hash tables [72], can vastly reduce test time. Table 3.3 (*Left*) shows the test time speed-up obtained through kNN with an SNC compressed reference set versus kNN with the full training set (after dimensionality reduction with LMNN). The table depicts the speed-up with the standard exhaustive neighbor search (in black), and accelerated versions with ball-trees (in teal) and LSH (in purple), each applied before and after SNC compression.

With a compression ratio $\geq 4\%$ the error rates on all data sets are lower or very close to those obtained with kNN without compression. However, we highlight settings that match or outperform the uncompressed kNN error in bold. For the standard exhaustive implementation in Table 3.3, speed-ups achieved by SNC generally exceed those expected at the given compression ratio (e.g $>100\times$ speed-up at 1% compression). This may be due to favorable cache effects from using a smaller reference set. This table shows that SNC compression can lead to notable speed-ups even when using ball-trees and hashing, demonstrating that SNC can be used in conjunction both methods for even greater speed-ups. The results with ball-trees are particularly impressive, as all inputs have undergone dimensionality reduction, which is known to significantly improve ball-tree speed-up itself [180].

Table 3.3 (*Right*) compares the number of distance computations required for kNN search with SNC at 4% compression versus kNN search with ball-trees or LSH using the full training set. The implementations of kNN, ball-trees, and LSH may not be directly comparable, so we use distance computations as a proxy for speed. SNC requires fewer distance computations than either method on all datasets except Adult (with LSH) and Forest.

Table 3.3: *Left:* Speed-up of kNN testing through SNC compression without a data structure (in black) *on top* of ball-trees (in teal) and LSH (in purple). Results where SNC matches or exceeds the accuracy of full kNN (up to statistical significance) are in **bold**. *Right:* Speed-up of SNC at 4% compression *versus* ball-trees and LSH on the full dataset. **Bold** text indicates matched or exceeded accuracy.

	Speed-up															**SNC 4% Comparison**	
	Compression Ratio															Distance Comps.	
Dataset	1%			2%			4%			8%			16%			Ball-Trees	LSH
yale-faces	–	–	–	**28**	**17**	**3.6**	**19**	**11**	**3.5**	**12**	**7.3**	**3.2**	**6.5**	4.2	**2.8**	**7.1**	**21**
isolet	**76**	**23**	**13**	**47**	**13**	**13**	**26**	**6.8**	**13**	**14**	**3.7**	**13**	**7.0**	**2.0**	**13**	**13**	**14**
letters	143	9.3	100	73	6.3	61	**34**	**3.6**	**34**	**16**	**2.0**	**17**	**7.6**	1.1	**8.4**	**3.3**	**23**
adult	**156**	**56**	**3.5**	**75**	**28**	**3.4**	**36**	**15**	**3.3**	**17**	**7.3**	**3.1**	**7.8**	**3.8**	**3.0**	**17**	**0.7**
w8a	146	68	39	71	36	35	33	19	26	15	10	18	7.3	5.5	11	13	**2.1**
mnist	**136**	**54**	**84**	**66**	**29**	**75**	**32**	**16**	**57**	**15**	**8.4**	**37**	**7.1**	**3.6**	**17**	**11**	**8.5**
forest	146	3.1	12	70	1.6	11	32	0.90	10	15	1.1	7.0	–	–	–	0.15	**0.35**

In summary, our results give strong indication that 1. SNC obtains drastic speed-ups during test-time while only marginally increasing or, at times, decreasing kNN error rates; and 2. it is an effective complement and competitor to existing state-of-the-art strategies for speeding up kNN.

Compressed synthetic faces. Figure 3.4 visualizes synthetic SNC reference vectors learned on the YaleFaces data. Here, we preprocess the data using PCA and learn a compressed data set using the first 100 principal components. The figure shows (reconstructed) input faces that are initially subsampled to be in the compressed set (left columns) and the resulting optimized (synthetic) faces after SNC. It is interesting to observe that SNC is easily able to identify and emphasize distinguishing characteristics (*e.g.* mustaches) while ignoring noisy qualities (*e.g.* lighting).

Label noise. An interesting observation from the results in Figure 3.3 is that SNC compression at times *improves* the kNN test error. We conjectured earlier that one explanation may be that kNN with SNC can yield a smoother decision boundary. This effect may be particularly beneficial in scenarios with label noise. We test this conjecture in Figure 3.5

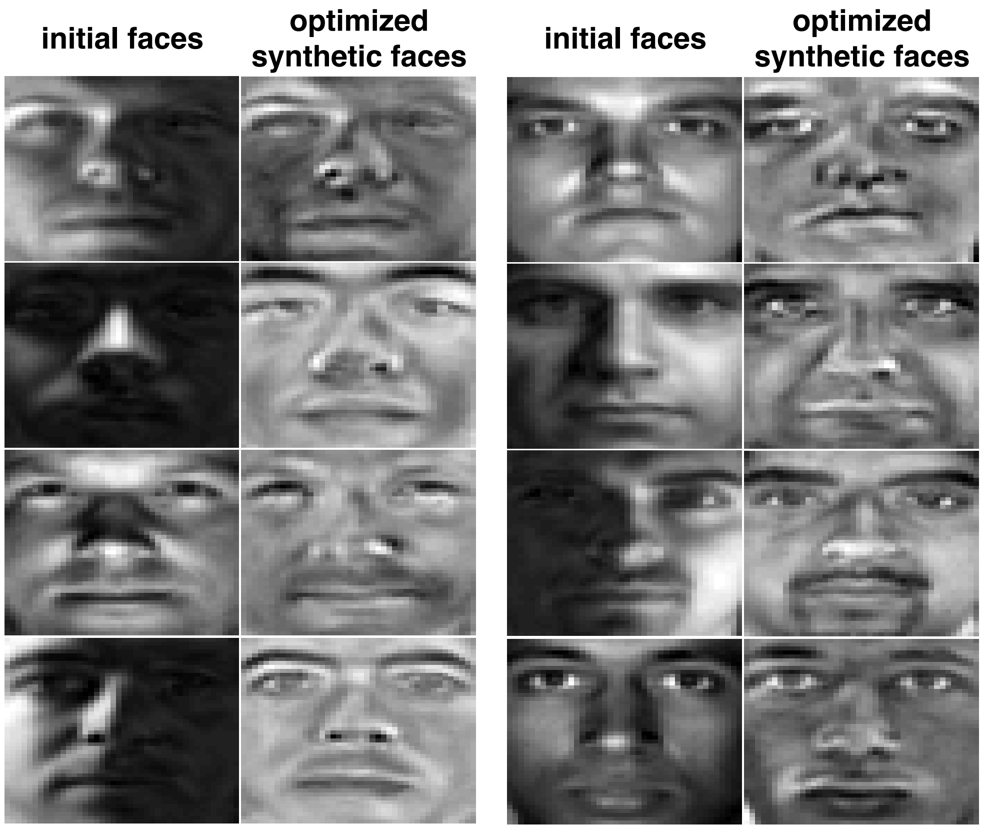


Figure 3.4: YaleFaces before and after compression.

(*Right*), where we examine the kNN error on the Letters dataset under increasing random label corruption (for $k = 1$ and $k = 3$). The figure shows clearly that the kNN error increases approximately linearly with label noise. SNC with 2%, 4%, 8% compression seems to smooth out mislabeled inputs and yields a significantly more robust kNN classifier. In contrast, CNN, FCNN and also subsampling (not shown in the figure to reduce clutter) do not mitigate the effect of label noise and at times tend to even amplify the test error. It is worth noting out that, for this experiment, CNN and FCNN were run to convergence and had significantly higher compression ratios than SNC's fixed ratios. For instance, at 0.32 label noise, CNN and FCNN both use more than 65% of the data in their compressed set.

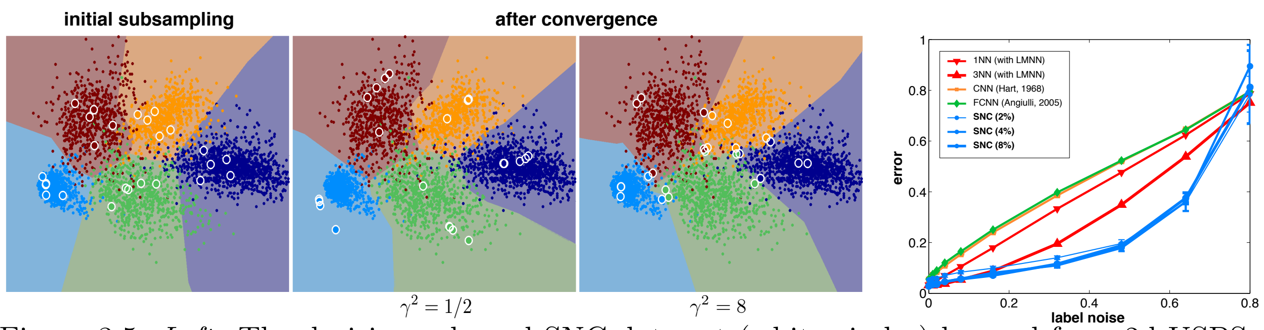


Figure 3.5: *Left:* The decision rule and SNC data set (white circles) learned from 2d USPS digits under varying γ^2. *Right:* kNN test error rates with various data set reduction methods on the Letters dataset under artificial label noise.

Visualized decision boundary. Figure 3.5 (*Left*) shows the reference set (white circles) and decision rule (colored shading) before and after SNC optimization. The data set consists of USPS handwritten digits $\{0, 1, 2, 3, 4\}$ after projection onto 2D with LMNN (class membership is indicated by color). The left plot shows the (randomly subsampled) initialization of the reference set and the decision boundaries generated by this set. The SNC vectors are learned with 4% compression ratio and different scaling factors ($\gamma^2 = 1/2$ and $\gamma^2 = 8$, respectively). The decision regions for each class are notably erroneous in several regions prior to reference set optimization. For both scale factors γ^2, optimizing the reference set with SNC improves the decision boundary over the random sampling.

With a small γ^2 (middle pane) the corresponding large variance of the stochastic neighborhood encourages reference set vectors to produce results resembling a mixture model [23], where groups of compressed vectors act as mixture component centers. The cluster centers are not at the expected locations (i.e. nested within a dense set of vectors), but are pushed outwards to accommodate the (possibly too) small γ^2. Larger values of γ^2 (right pane), converge to the decision boundaries between classes. Indeed, for every compressed input close to the boundary there are one or more representing the neighboring classes.

It is interesting to observe that by controlling γ^2, SNC can learn very different compressed sets. For naturally clustered data sets, larger values of γ^2 may be preferred, to make reference vectors represent dense regions in the data distribution. For data without such structure, smaller values of γ^2 may result in lower errors, as SNC can model the decision boundary more accurately. As γ^2 is an important hyperparameter that changes the characteristics of the compressed set, its initial value should be set via cross-validation prior to potential further optimization.

3.2 Related Work

Research on speeding up kNN is almost as old as the kNN rule itself. A big fraction concentrates on developing clever data structures in order to reduce the number of test time comparisons; examples include KD trees [18], cover- and ball-trees [21, 133] and hashing [72]. In this section we review prior research that takes the complementary approach of reducing the size of training data. We group these approaches under three general categories: *training set consistent sampling*, *prototype generation*, and *prototype positioning*. (A complete survey is [170].)

3.2.1 Training Set Consistent Sampling

The earliest work, called Condensed Nearest Neighbors (CNN) [81], starts by randomly selecting a single input and creating a 'reference' set, which it will use to classify the training data. It adds misclassified training inputs sequentially to this reference set until the full training set is correctly classified. There have been multiple extensions to CNN including post-processing methods [67] and stricter selection rules [169, 47]. Recently Fast CNN [5] makes CNN sub-quadratic in n to train (as opposed to $O(n^3)$ naïvely for CNN), with empirically better test generalization. All these methods retain a set of inputs that are a subset of the original training set.

3.2.2 Prototype Generation

Prototype generation methods create 'prototypes' for the training data, which allow the inclusion of new, artificial instances in the reduced data set [14]. These generated inputs are typically found via clustering. Prior work has proposed to repeatedly merge nearest neighbors within a class while the training error is unaffected [32]. Another idea is to merge clusters of inputs, until the LOO training error increases over a pre-determined threshold [123].

3.2.3 Prototype Positioning

There has also been work on learning the positions of this prototype subset. Proximity graphs have been used to generate a reduced set [170]. It is even possible to determine the best set of prototypes to exactly reproduce any decision boundary requested [146]. There has been a body of work on using learning vector quantization (LVQ) [104] for designing kNN prototypes as well. In general, all of these methods consider local properties to optimize the reference set whereas SNC incorporates global information from the entire dataset through the stochastic neighborhoods.

3.2.4 Gaussian Methods

Most similar to SNC are methods which optimize prototypes to maximize Gaussian mixtures (which can be interpreted as a stochastic neighborhood). The stochastic neighborhood, described in Section 3.1.1, smoothly models the probability that each prototype is the nearest neighbor of a given training point using a Gaussian likelihood. The primary differences between SNC and these methods in initial prototype selection and how inputs are classified during test-time. Two works ([45] and [115]) use a three-phase search for initial prototypes that involves k-means and two different elimination rules. A variation of kNN called 'soft'-kNN uses the R nearest prototypes to classify inputs [20]. This work then maximizes the probability of a correct prediction using the R closest prototypes. In effect, if $R = n$ their objective is similar to ours. However, because soft-kNN always considers all of the R prototypes to make a classification this setting cannot be sped up with ball trees, so the authors cross validate the soft-kNN error over R. This cross-validation does not consider the objective of reducing the size of R, which may result in a model that requires significantly more computation than SNC.

3.3 Conclusions

We have introduced SNC, which is a simple and efficient algorithm to compress the training set for kNN. Our experiments indicate that SNC reference set almost always provides comparable or lower test errors than the training set with compression rates of 2-4%, leading to an order of magnitude improvement in testing time.

One important direction for future work is extending SNC to neighborhoods of higher cardinality (i.e. $k = 3, 5$). Recent work [166] provides a rigorous and efficient approach for extending the stochastic neighborhood framework to larger neighborhoods. Another consideration is that, as nearest-neighbor search in non-Euclidean spaces becomes more popular (e.g. covariance matrices [31]) it is valuable to consider if SNC can be extended to this setting. Another interesting direction is if one can learn compressed reference inputs while simultaneously optimizing ball-tree structures.

In summary, we believe that SNC is a robust and highly effective algorithm that is based on straight-forward gradient descent optimization. As it (a) seems to consistently improve kNN *speed, accuracy* and *robustness*, and (b) can be combined with existing algorithms to improve kNN, we hope it will be useful to researchers and practitioners in machine learning and its application domains.

Chapter 4

Privacy: Protecting individual privacy in causal inference

Data is now being collected about every aspect of our personal lives. From where we work and live, who we contact, what our past and family history of illness is, data is informing search results, social media, and doctor prescriptions, among many other services. This data can be useful for personalizing such services; for example when searching for restaurants it should be clear one is usually interested in restaurants within a few miles of where they live or work. It also may be critical in emergency situations; for instance if an emergency medical technician (EMT) wishes to treat someone in pain, but that person is allergic to morphine (a common initially-used pain reliever), the EMT can opt to use a different pain-reliever. As machine learning improves, personal data will increasingly become digitized for these purposes and more.

Nearly all of this data however, is sensitive information that a person may reasonably want to keep private. Indeed, even seemingly innocuous data has been used to identify individuals in an anonymized dataset, leading to personally-harmful consequences:

1. **The AOL search release.** In 2006, the web-search company AOL released a dataset of search results for more than 650,000 users during a 3-month window, for the purpose of improving search. Usernames were replaced by an anonymous numerical identifier designed to disassociate the search results from AOL accounts. Upon releasing the data however, it was discovered that the search queries contained uniquely-identifying information for many of the users in the dataset. For example, after only a few searches including "landscapers in Lilburn, Ga" and "homes sold in shadow lake subdivision gwinnett county georgia", Thelma Arnold was identified by the New York Times as user #4417749 [1]. Other users were also identified based on the fact that a common search query is one's own name.

2. **The Neflix prize.** Netflix announced a \$1 million dollar competition to improve its movie recommendation system in September 2006. They released a dataset that included movie ratings, when the ratings occurred and, similar to the AOL dataset, a unique ID secretly assigned to each user. Soon after, work was published that matched Netflix reviews to those posted on the Internet Movie Database (IMDB) to de-anonymize users [130]. This work was followed by a lawsuit against Netflix by a woman who believes that her movie ratings made her private sexual orientation public knowledge.

These are only a few examples of the impact of releasing sensitive data. Even more insidious attacks on seemingly "more anonymized" data are possible, such as using genetic summary statistics called single-nucleotide polymorphisms (SNPs) to determine if an individual has a disease or not [57] (for more examples see Chapter 1 of Dwork & Roth, 2014 [57]). In the same way, predictions from a machine learning algorithm trained on sensitive data can leak this information. Consider the example (slightly modified) from [92]: Imagine you are company A that buys advertising space on an web-search engine. The web-search company has an algorithm that ranks ads in a list based on the search query Q *and* the profile information of the person who made the search. Imagine that each time a person clicks on an ad, the web-search engine updates its recommendation algorithm based on this profile information. Now, as company A you would like to know what sort of people click on your advertisement. To know this, you can create two profiles that are identical except for say, their age: one ≤ 50 the other > 50. You can search for query Q and see whether your ad is ranked higher for the younger profile or the older profile. What's more, you can determine this information for an individual! Say a user with IP address X clicks on your company A advertisement. You can create the age-specific profiles again after this click and see how the ad ranking changes before and after the click. Say that the ranking for the ≤ 50 profile goes down and the ranking for the > 50 profile goes up after the click. Then you can infer that the person with IP address X is older than 50, and you have just inferred information that someone may wish to keep private.

4.0.1 Motivation

As machine learning reaches into the fields of health care, finance, and legal work, it can only be practical if it is able to adequately protect the privacy of sensitive data. One prime example of a machine learning task that should be adequately privatized is predicting if a random variable X *causes* another random variable Y, or vice-versa, or not at all. This problem is termed "bivariate causal inference". For example, let X be the amount a person uses Facebook and Y be the extent to which that person is suffering from depression. One possible relationship between these random variables is that increased Facebook usage *causes* one to be more/less depressed. Equally, it could be that the more/less one is depressed the more one uses Facebook, perhaps to communicate this to others. Finally, it could be that there is no relationship at all between these variables. State-of-the-art work in bivariate causal inference first asks individuals about their Facebook usage and their level of depression. It then uses this data to determine if a causal link exists between X and Y, and if so, in what direction [125]. However, publishing this causal inference can possibly leak individual information about their Facebook usage and depression status (e.g., it is easy to determine how frequently a person uses Facebook, which reveals information about their depression level, via the published result).

4.1 Private Causal Inference

In this chapter we devise a scheme to prevent the leak of sensitive information in a popular causal inference model, called the additive noise model (ANM) [89]. We make use of the robust framework of differential privacy [55], which has been applied to other machine learning problems including empirical risk minimization [33], online learning [92], and principal components analysis [59]. In essence, the technique works by adding cleverly calibrated random noise to the causal inference result such that it is impossible to tell if fluctuations in the result are caused by an individual or the random noise. We show that it is possible to release causal inference results that are simultaneously private and nearly always the same as the non-private result.

4.1.1 Prior Art

Causal identification allows one to reason about how manipulations of certain random variables (the causes) affect the outcomes of others (the effects). Uncovering these causal structures has implications ranging from creating government policies to informing health-care practices.

The gold-standard in discovering causal relations is the randomized intervention experiment: a researcher fixes a random variable to take on values uniformly from its domain and observes the outcomes of all other random variables. It can be shown that any observed correlations result directly from causal relationships (whereas without such randomization they may not). While such interventions are conceptually simple, they are in many cases cost-impractical, technically-impossible, or even more seriously morally-questionable. As an extreme example, implementing an intervention to answer whether diet X causes cancer Y would require making individuals consume different diets for a period of time and observing their cancer outcomes to determine if $X \rightarrow Y$. In the same way, if one wanted to identify if $Y \rightarrow X$ one would have to induce different cancer outcomes and observe dietary outcomes. Therefore, there has been a wealth of research towards determining causal structure without having to resort to interventions.

One initial alternative to randomized experiments is conditional independence testing [159, 135], in which one works to test whether two random variables X, Y are independent given another Z. Imagine one wants to know about the causal relationships between a set of random variables, often represented by a causal graphical model. Using the results of many such conditional independence tests for all random variables of interes Pearl et al. showed that one could find a certain set of causal graphical models that were consistent with these test results (such a set is referred to as 'Markov-equivalent'). There are two immediate difficulties with this approach: 1. Often, these conditional independence tests result in many possible consistent causal structures. When this happens, we are not able to distinguish the causal relationships between certain pairs of random variables, and more testing or other methods are required to determine these. 2. Conditional independence testing cannot be used to determine the causal relationship between just two random variables, as no such conditional independence test exists.

In the absence of interventions, the field of *causal inference* attempts to discover the underlying causal relationships of a set of random variables entirely based on samples from their joint distribution, without requiring conditional independence assumptions. The field of causal inference is now a mature research area, covering learning topics as diverse as supervised batch inference [116, 125, 137], time-series causal prediction [69], and linear dynamical systems [149]. Many inference methods require only a regression technique and a way to compute the independence between two distributions given samples [89, 95], and so thus are extrememly practical.

One would hope that researchers could publicly release their causal inference findings to inform individuals and policy makers. One of the primary roadblocks to doing so is that often causal inference is performed on data that individuals may wish to keep private, such as data in the fields of medical diagnosis, fraud detection, and risk analysis. Currently, no causal inference method has formal guarantees about the privacy of individual data, which may be able to be inferred via attacks such as reconstruction attacks [50].

Arguably one of the best notion of privacy is differential privacy, introduced by [55] and since used throughout machine learning [54, 92, 119, 33, 56]. Differential privacy guarantees that the outcome of an algorithm only reveals aggregate information about the entire dataset and never about the individual. An individual who is considering to participate in a study can be reassured that his/her personal information cannot be recovered with extremely high probability.

To our knowledge, this paper is the first to investigate private causal inference. We show that it is possible to privately release the quantities produced by the highly-successful additive noise model (ANM) framework by adding small amounts of noise, as dictated by differential privacy. Furthermore, these private quantities, with high probability, do not change the causal inference result, so long as it is confident enough. We demonstrate on a set of real-world causal inference datasets how our privacy-preserving methods can be readily and usefully applied.

4.1.2 Causal Inference & Privacy

Our aim is to protect the privacy of individuals who submit personal information about two random variables of interest X and Y. Their information should remain private when it is used to infer whether X causes Y ($X \rightarrow Y$), or Y causes X ($Y \rightarrow X$) using the ANM framework. This personal information comes in the form of i.i.d. samples $\{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ from the joint distribution $\mathbb{P}_{X,Y}$. We will assume that, 1. There is no confounding variable Z that commonly causes or is a common effect of X and Y. 2. X and Y do not simultaneously cause each other.

4.1.3 Additive Noise Model

Deciding on the causal direction between two variables X and Y from a finite sample set has motivated an array of research [64, 100, 162, 89, 189, 124, 95, 107, 116]. Perhaps one of the most popular results is the Additive Noise Model (ANM) proposed by [89]. The ANM framework assumption is defined as follows.

Definition 4. *Two random variables X, Y with joint density $p(x, y)$ are said to 'satisfy an ANM' $X \rightarrow Y$ if there exists a non-linear function $f : \mathcal{R} \rightarrow \mathcal{R}$ and a random noise variable N_Y, independent from X, i.e. $X \perp\!\!\!\perp N_Y$, such that*

$$Y = f(X) + N_Y.$$

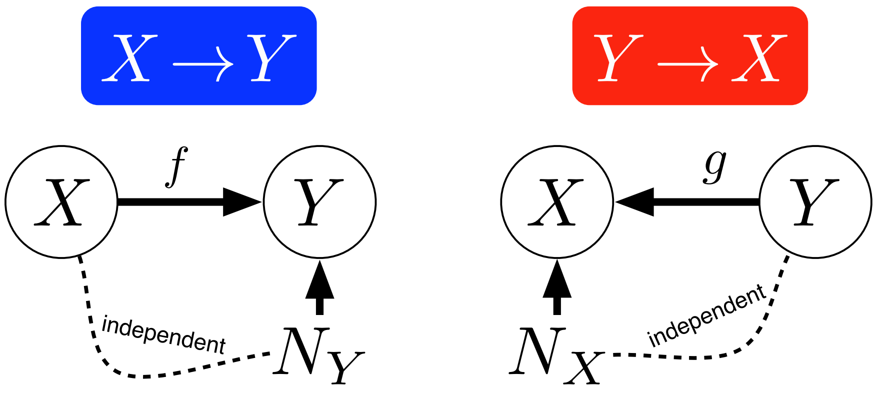


Figure 4.1: The graphical model representations for both possible additive noise models (ANMs) [89]: $X \to Y$ and $Y \to X$. In this model if a random variable X causes another Y, then Y is a function (e.g., f) of X *plus* random noise N_Y. Importantly, it is assumed that this noise is independent from the input X, which will help us identify the causal direction from samples of X and Y. See text for details.

As defined, an ANM $X \to Y$ implies a functional relationship mapping X to Y, alongside independent noise, as shown in Figure 4.1. In order for this model to be useful for causal inference we would like the induced joint distribution $\mathbb{P}_{X,Y}$ for this ANM to be somehow identifiably different from the one induced by the ANM $Y \to X$. If so, we say that the causal direction is *identifiable* [125]. If not, we have no hope of recovering the causal direction purely from samples under the ANM.

[89] showed that ANMs are generically identifiable from i.i.d. samples from $\mathbb{P}_{X,Y}$ (except for a few special cases of non-linear functions f and noise distributions). The intuition behind this is for the $X \to Y$ ANM, consider for most non-linear f and (for simplicity) 0-mean N_Y, the density $p(y|x)$ has mean $f(x)$ with distribution given by N_Y. This implies that $p(y - f(x)|x)$ has distribution N_Y that is independent of X. However, $p(x - f^{-1}(y)|y)$ is for many choices of f and N_Y not independent of y.

Algorithm 3 ANM Causal Inference [125]

```
1: Input: train/test data $\{x^{(i)}, y^{(i)}\}_{i=1}^{n}$, $\{x^{(i)\prime}, y^{(i)\prime}\}_{i=1}^{m}$
2: Regress on training data, to yield $\hat{f}, \hat{g}$, such that:
3: $\hat{f}(x^{(i)}) \approx y^{(i)}, \quad \hat{g}(y^{(i)}) \approx x^{(i)}, \ \forall i$
4: Define the test vectors:
5: $\mathbf{x}' = [x^{(1)\prime}, \ldots, x^{(m)\prime}]^\top, \quad \mathbf{y}' = [y^{(1)\prime}, \ldots, y^{(m)\prime}]^\top$
6: Compute residuals on test data:
7: $\mathbf{r}'_Y := \mathbf{y}' - \hat{f}(\mathbf{x}'), \quad \mathbf{r}'_X := \mathbf{x}' - \hat{g}(\mathbf{y}')$
8: Calculate dependence scores:
9: $s_{X \to Y} := s(\mathbf{x}', \mathbf{r}'_Y), \quad s_{Y \to X} := s(\mathbf{y}', \mathbf{r}'_X)$
10: Return: $s_{X \to Y}$, $s_{X \to Y}$, and $D$, where
11: $\mathrm{D} = \begin{cases} X \to Y & \text{if } s_{X \to Y} < s_{Y \to X} \\ Y \to X & \text{if } s_{X \to Y} > s_{Y \to X} \end{cases}$
```

4.1.4 Inferring Causality

[125] give a practical algorithm for determining the causal relationship between X and Y (i.e., either $X \to Y$ or $Y \to X$), as shown in Algorithm 3. The first step is to partition the i.i.d. samples into a training and a testing set. We use the training set to train the regression functions $\hat{f} : X \to Y$ and $\hat{g} : Y \to X$. We use the testing set to compute the residuals $\mathbf{r}'_Y = \mathbf{y}' - \hat{f}(\mathbf{x}')$ and $\mathbf{r}'_X := \mathbf{x}' - \hat{g}(\mathbf{y}')$. If we have an ANM $X \to Y$ then the residual $\mathbf{r}'_Y$ is an estimate of the noise N_Y which is assumed to be independent of X. Therefore, we calculate the dependence between the residual $\mathbf{r}'_Y$ and the input $\mathbf{x}'$, $s_{X \to Y} := s(\mathbf{x}', \mathbf{r}'_Y)$, and $s_{Y \to X} := s(\mathbf{y}', \mathbf{r}'_X)$, using a dependence score $s(\cdot, \cdot)$. If $s_{X \to Y}$ is less than $s_{Y \to X}$, then we declare $X \to Y$, otherwise $Y \to X$. Figure 4.2 gives a visual illustration of this ANM causal inference algorithm.

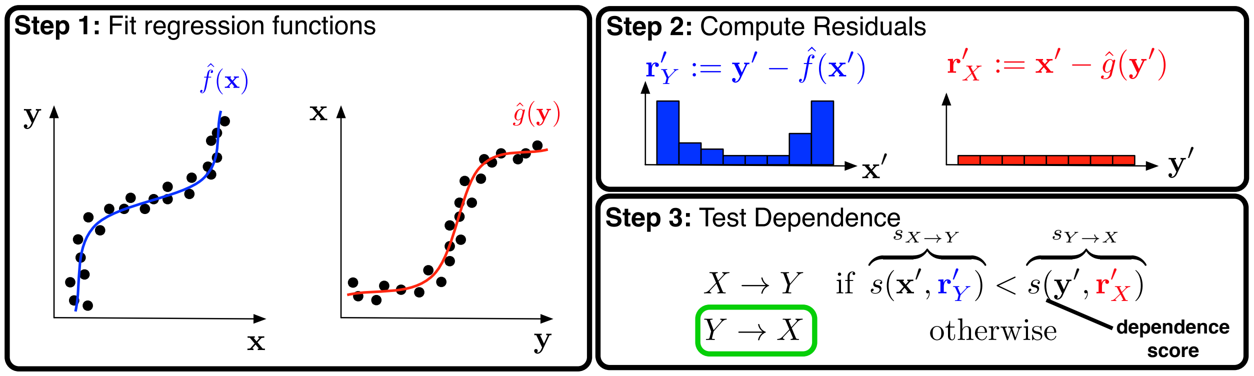

Step 1: Fit regression functions
$\hat{f}(\mathbf{x})$
$\mathbf{y}$
$\mathbf{x}$
$\hat{g}(\mathbf{y})$
$\mathbf{x}$
$\mathbf{y}$
Step 2: Compute Residuals
$\mathbf{r}'_Y := \mathbf{y}' - \hat{f}(\mathbf{x}')$
$\mathbf{x}'$
$\mathbf{r}'_X := \mathbf{x}' - \hat{g}(\mathbf{y}')$
$\mathbf{y}'$
Step 3: Test Dependence
$X \to Y$ if $\overbrace{s(\mathbf{x}', \mathbf{r}'_Y)}^{s_{X\to Y}} < \overbrace{s(\mathbf{y}', \mathbf{r}'_X)}^{s_{Y\to X}}$
$\boxed{Y \to X}$ otherwise
dependence score

Figure 4.2: An illustration of the high-level steps of the ANM algorithm [89, 125].

4.1.5 Dependence Scores

Crucially, the ANM approach hinges on the choice of dependence score $s(\cdot, \cdot)$. There have been many proposals, and we give a quick review of the most popular methods (for a detailed review see [125]).

Spearman's ρ is a rank correlation coefficient that describes the extent to which one random variable is a monotonic function of the other. Specifically, imagine independently sorting the observations $\{a^{(1)}, \ldots, a^{(m)}\}$ and $\{b^{(1)}, \ldots, b^{(m)}\}$ by value in increasing order. Let $o_a^{(i)}$ be the rank of $a^{(i)}$ in the a-ordering, and similarly, $o_b^{(i)}$ for $b^{(i)}$ in the b-ordering. Then Spearman's ρ is,

$$s(\mathbf{a}, \mathbf{b}) := \left| 1 - \frac{6 \sum_{i=1}^{m} {d^{(i)}}^2}{m(m^2 - 1)} \right|$$

where $d^{(i)} := (o_a^{(i)} - o_b^{(i)})$ are the rank differences for $\mathbf{a}, \mathbf{b}$.

Kendall's τ. Similar to Spearman's ρ, the Kendall τ rank score calls a pair of indices (i, j) *concordant* if it is the case that $a^{(i)} > a^{(j)}$ and $b^{(i)} > b^{(j)}$. Otherwise (i, j) is called *discordant*.

Then the dependence score is defined as

$$s(\mathbf{a}, \mathbf{b}) := \frac{|C - D|}{\frac{1}{2}m(m-1)}$$

where C is the number of concordant pairs and D is the number of discordant pairs.

HSIC Score. The first proposed score for the ANM causal inference is based on the Hilbert-Schmidt Independence Criterion (HSIC) [76], which was used by [89]. Let $\mathbf{a} = [a^{(1)}, \ldots, a^{(m)}]^\top$ and $\mathbf{b} = [b^{(1)}, \ldots, b^{(m)}]^\top$. They compute an estimate of the p-value of the HSIC under the null hypothesis of independence, selecting the causal direction having the lower p-value. Alternatively, one can use an estimator to the HSIC value itself:

$$s(\mathbf{a}, \mathbf{b}) := \widehat{\mathrm{HSIC}}_{k_{\theta(\mathbf{a})}, k_{\theta(\mathbf{b})}}(\mathbf{a}, \mathbf{b}) \tag{4.1}$$

where k_θ is a kernel with parameters θ. [125] show that under certain assumptions the algorithm in section 3 with the HSIC dependence score is consistent for estimating the causal direction in an ANM.

Variance Score. When the noise variables in the ANM are Gaussian, the variance score was proposed in [29], and defined as $s(\mathbf{a}, \mathbf{b}) := \log \mathbb{V}(\mathbf{a}) + \log \mathbb{V}(\mathbf{b})$. Changes to a single input value can induce arbitrarily large changes to this score, which makes the variance score ill suited to preserve differential privacy.

IQR Score. We introduce a robust version of this score by replacing the variance of the random variables with their interquartile range (IQR). The IQR is the difference between the third and first quartiles of the distribution and can be estimated empirically. We defined the following IQR-based score:

$$s(\mathbf{a}, \mathbf{b}) := \log \mathrm{IQR}(\mathbf{a}) + \log \mathrm{IQR}(\mathbf{b}). \tag{4.2}$$

4.1.6 Differential Privacy

We assume that the data set $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{n}$ contains sensitive data that should not be inferred from the release of the dependence scores. One of the most widely accepted mechanisms for private data release is *differential privacy* [55]. In a nutshell it ensures that the released scores can only be used to infer aggregate information about the data set and never about an individual datum $(x^{(i)}, y^{(i)})$.

Let us define the distance $d(\mathcal{D}, \tilde{\mathcal{D}})$ between two data sets $\mathcal{D}$ and $\tilde{\mathcal{D}}$ as the number of instances in which these two sets differ. If a data set $\mathcal{D}$ is changed to $\tilde{\mathcal{D}}$, a distance $d(\mathcal{D}, \tilde{\mathcal{D}}) \leq 1$ implies that at most one element was substituted.

Definition 5. *A randomized algorithm $\mathcal{A}$ is (ϵ, δ)-**differentially private** for $\epsilon, \delta \geq 0$ if for all $\mathcal{O} \in \mathrm{Range}(\mathcal{A})$ and for all neighboring datasets $\mathcal{D}, \tilde{\mathcal{D}}$ with $d(\mathcal{D}, \tilde{\mathcal{D}}) \leq 1$ we have that*

$$\Pr\big[\mathcal{A}(\mathcal{D}) = \mathcal{O}\big] \leq e^{\epsilon} \Pr\big[\mathcal{A}(\tilde{\mathcal{D}}) = \mathcal{O}\big] + \delta. \tag{4.3}$$

One of the most popular methods for making an algorithm $(\epsilon, 0)$-differentially private is the Laplace mechanism [55]. For this mechanism we must define an intermediate quantity called the **global sensitivity**, $\Delta_{\mathcal{A}}$ describing how much $\mathcal{A}$ changes when $\mathcal{D}$ changes,

$$\Delta_{\mathcal{A}} := \max_{\mathcal{D}, \tilde{\mathcal{D}} \subseteq \mathcal{X} \text{ s.t. } d(\mathcal{D}, \tilde{\mathcal{D}}) \leq 1} |\mathcal{A}(\mathcal{D}) - \mathcal{A}(\tilde{\mathcal{D}})|.$$

The Laplace mechanism hides the output of $\mathcal{A}$ with a small amount of additive random noise, large enough to hide the impact of any *single* datum $(x^{(i)}, y^{(i)})$.

Definition 6. *Given a dataset $\mathcal{D}$ and an algorithm $\mathcal{A}$, the* ***Laplace mechanism*** *returns $\mathcal{A}(\mathcal{D}) + \omega$, where ω is a noise variable drawn from $\mathrm{Lap}(0, \Delta_{\mathcal{A}}/\epsilon)$, the Laplace distribution with scale parameter $\Delta_{\mathcal{A}}/\epsilon$.*

It may be that the global sensitivity of an algorithm $\mathcal{A}$ is unbounded in general, but can be bounded in the context of a specific data set $\mathcal{D}$ over all neighbors $\tilde{\mathcal{D}}$. For such datasets we can bound the **local sensitivity**

$$\Delta(\mathcal{D})_{\mathcal{A}} := \max_{\tilde{\mathcal{D}} \subseteq \mathcal{X} \text{ s.t. } d(\mathcal{D}, \tilde{\mathcal{D}}) \leq 1} |\mathcal{A}(\mathcal{D}) - \mathcal{A}(\tilde{\mathcal{D}})|.$$

If an algorithm has bounded global sensitivity it certainly has bounded local sensitivity. [132, 54, 94] show how to use the local sensitivity to cleverly produce private quantities for datasets with bounded local sensitivity.

4.1.7 Test Set Privacy

The data is partitioned into training and test sets, which are used in different ways. We therefore introduce mechanisms to preserve training and test set privacy respectively, which can be used jointly. Specifically, we show how to privatize the dependence scores $s_{X \to Y}, s_{Y \to X}$. The reason for this is three-fold: 1. Privatizing the dependence score immediately privatizes the causal direction D, because operations on differentially private outputs preserve privacy (so long as they do not again touch the data). 2. Releasing the scores indicates how confident the ANM method is about the causal direction, which is absent from the binary output D. 3. It is unclear which dependence score is best for a particular dataset, so we privatize multiple

Table 4.1: Dependence scores and their privacy. A checkmark indicates that there exist meaningful bounds on either the global or local sensitivity.

	Test		**Training**	
Score	GLOBAL SENSE.	LOCAL SENSE.	GLOBAL SENSE.	LOCAL SENSE.
SPEARMAN'S ρ	✓	✓	-	✓
KENDALL'S τ	✓	✓	-	✓
HSIC	✓	✓	✓	✓
IQR	-	✓	-	✓

scores and leave this choice to the practitioner. In this section we begin with test set privacy and describe training set privacy in Section 4.1.8. Table 4.1 gives an overview of test and training set privacy results for the dependence scores that we consider.

Let $(\mathbf{x}', \mathbf{y}')$ be the initial test data and $(\tilde{\mathbf{x}}', \tilde{\mathbf{y}}')$ be the test data after a single change in the dataset. Let $\tilde{\mathbf{x}}' = [x^{(1)'}, \ldots, x^{(k-1)'}, \tilde{x}^{(k)}, x^{(k+1)'}, \ldots, x^{(m)'}]^\top$ and similarly for $\tilde{\mathbf{y}}$ so that this single change occurs at some index k. The key to preserving privacy is to show that the selected dependence score $s(\cdot, \cdot)$ can be privatized. We show that if our dependence score is a rank correlation coefficient (Spearman's ρ, Kendall's τ) or the HSIC score [76], we can readily bound its test set global sensitivity when applied to $(\mathbf{x}', \mathbf{y}')$ versus $(\tilde{\mathbf{x}}', \tilde{\mathbf{y}}')$. As the IQR score has bounded test set local sensitivity we can apply the algorithm of [54] for privacy.

Rank Correlation Coefficients

We first demonstrate global sensitivity for the two rank correlation scores in Section 4.1.2.

Theorem 3. *The rank correlation coefficients have the following global sensitivities,*

1. *Let $\rho(\cdot, \cdot)$ be Spearman's ρ score, then*

$$|\rho(\mathbf{x}', \mathbf{r}'_Y) - \rho(\tilde{\mathbf{x}}', \tilde{\mathbf{r}}'_Y)| \leq \frac{30}{m}$$

2. Let $\tau(\cdot,\cdot)$ be Kendall's τ score, then

$$|\tau(\mathbf{x}', \mathbf{r}'_Y) - \tau(\tilde{\mathbf{x}}', \tilde{\mathbf{r}}'_Y)| \leq \frac{4}{m}$$

Proof. Our goal is to bound the following global sensitivity in both scores: $|s(\mathbf{x}', \mathbf{r}'_Y) - s(\tilde{\mathbf{x}}', \tilde{\mathbf{r}}'_Y)|$. For Spearman's ρ, suppose the change is on $a^{(k)}$ and $b^{(k)}$, it is easy to verify that (1) $d^{(i)}$ changes by at most 2, for $i \neq k$; (2) $d^{(k)}$ changes by at most $m-1$; (3) $d_i \leq m-1$ for all i. Since ${d^{(i)}}^2 - (d^{(i)} - 2)^2 = 4(d^{(i)} - 1) \leq 4(m-2)$ for $i \neq k$, the maximum change inside the summation of the Spearman's ρ score is upper bounded by $(m-1)(4m-8) + (m-1)^2$. Therefore, global sensitivity of ρ is bounded by

$$\frac{6(m-1)(5m-3)}{m(m^2-1)} \leq \frac{30}{m}$$

.

For Kendall's τ we can affect at most $(m-1)$ pairs by moving a single element of $\mathbf{x}'$, as well as $(m-1)$ pairs for changing $\mathbf{r}'_Y$ (either from concordant pairs to discordant pairs, or vice versa). Therefore, the global sensitivity of Kendall's τ is

$$|s(\mathbf{x}', \mathbf{r}'_Y) - s(\tilde{\mathbf{x}}', \tilde{\mathbf{r}}'_Y)| \leq \frac{2(m-1)}{\frac{1}{2}m(m-1)} \leq \frac{4}{m}$$

■

The bound on the global sensitivity Δ of our scores enables us to apply the Laplace mechanism [55] to produce $(2\epsilon, 0)$-differentially private scores: $p_{X\to Y}, p_{Y\to X}$. Specifically, we add Laplace noise $\mathrm{Lap}(0, \Delta/\epsilon)$ to our Spearman's ρ and Kendall's τ scores to preserve privacy w.r.t. the test set. Moreover, as a general property of differential privacy we can compute any functions on these private scores and, so long as they do not touch the data, the outputs of these functions are also private. This means that we can compute the inequality $p_{X\to Y} < p_{Y\to X}$ to decide if X causes Y or vice-versa privately.

An important consideration is to what degree the addition of noise affects the true decision: $s_{X\to Y} < s_{Y\to X}$. Importantly, we can prove that, in certain cases, the addition of Laplace noise required by the mechanism is small enough to not change the direction of causal inference. These are cases in which there is a large 'margin' between the scores $s_{X\to Y}$ and $s_{Y\to X}$. So long as this margin is large enough and in the correct order the addition of Laplace noise has no effect on the inference with high probability.

Theorem 4. *Given two random variables X, Y who have w.l.o.g. the causal relationship $X \to Y$, assume that they produce correctly-ordered scores: $s_{X\to Y} < s_{Y\to X}$, with margin $\gamma = s_{Y\to X} - s_{X\to Y}$. Let $p_{X\to Y}, p_{Y\to X}$ be these scores after applying the Laplace mechanism [55] with scale $\sigma = \Delta/\epsilon$ then the probability of correct inference with these private scores is,*

$$\mathbb{P}(p_{X\to Y} < p_{Y\to X}) = 1 - \frac{\gamma + 2\sigma}{4\sigma} e^{-\frac{\gamma}{\sigma}}.$$

We leave the proof to the appendix. Note that the probability of incorrect inference decreases nearly exponentially as the margin γ increases. This is a particularly nice property as the margin essentially describes the confidence of the (non-private) causal inference prediction: large γ corresponds to high confidence in the inference. Additionally, there is an exponential decrease as m and ϵ grow. In Section 4.1.9, we show on real-world causal inference data that we can accurately recover the true causal direction for a variety ϵ settings.

HSIC Score

We begin by defining the empirical estimate of the HSIC score given kernels k, l:

$$\widehat{\mathrm{HSIC}}_{k,l}(\mathbf{x}', \mathbf{r}'_Y) := \frac{1}{(m-1)^2}\mathrm{trace}(KHLH) \tag{4.4}$$

where $K_{ij} = k(x'_i, x'_j)$, $L_{ij} = l(r'_{Y,i}, r_{Y,j})$ and $H_{ij} = \delta_{\{i=j\}} - 1/m$. We assume k, l are bounded above by 1 (e.g., the squared exponential kernel, the Matern kernel [139]). Our goal is to show that when we replace $(\mathbf{x}', \mathbf{y}')$ with $(\tilde{\mathbf{x}}', \tilde{\mathbf{y}}')$ the global sensitivity is small. Specifically we prove the following theorem.

Theorem 5. *The score in eq. (4.4) has a global sensitivity of at most* $\frac{16m-8}{(m-1)^2}$. *Specifically,*

$$|\widehat{HSIC}_{k,l}(\mathbf{x}', \mathbf{r}'_Y) - \widehat{HSIC}_{k,l}(\tilde{\mathbf{x}}', \tilde{\mathbf{r}}'_Y)| \leq \frac{16m-8}{(m-1)^2}$$

Proof. For simplicity define $\mathcal{H}(\cdot,\cdot) := \widehat{\mathrm{HSIC}}_{k,l}(\cdot,\cdot)$. Note that, as the trace is cyclic: $\mathrm{trace}(KHLH) = \mathrm{trace}(HKHL)$. Further, let $\tilde{K}, \tilde{L}$ be the kernels defined on the modified data $(\tilde{\mathbf{x}}', \tilde{\mathbf{y}}')$. Then as the data is represented purely through the kernel matrices and the trace is Lipschitz w.r.t. these matrices, we can apply the triangle inequality to yield,

$$|\mathcal{H}(\mathbf{x}', \mathbf{r}'_Y) - \mathcal{H}(\tilde{\mathbf{x}}', \tilde{\mathbf{r}}'_Y)| \leq \frac{\|HLH\|_\infty \|K - \tilde{K}\|_1}{(m-1)^2} + \frac{\|HKH\|_\infty \|L - \tilde{L}\|_1}{(m-1)^2}$$

To bound the infinity norms, let $\overline{L} = HLH$, then

$$|\overline{L}_{ij}| = \left| L_{ij} - \frac{\sum_{a=1}^m L_{aj}}{m} - \frac{\sum_{b=1}^m L_{ib}}{m} + \frac{\sum_{a,b=1}^m L_{ab}}{m^2} \right|$$
$$\leq 4$$

as $L_{ij} \leq 1$ (this inequality also holds for HKH). Finally, note that as there is only a single-element difference between $(\mathbf{x}', \mathbf{r}'_Y)$ and $(\tilde{\mathbf{x}}', \tilde{\mathbf{r}}'_Y)$, we have that $\|K - \tilde{K}\|_1 \leq 2m - 1$ (and also for $L, \tilde{L}$). ■

In fact, we can improve this bound to $\frac{12m-11}{(m-1)^2}$ using trace identities, as described in the appendix. Given this global sensitivity bound we can use Theorem 4 to guarantee that under certain conditions the Laplace mechanism w.h.p. does not change the direction of causal influence.

IQR Score

Unfortunately the IQR does not have a bounded global sensitivity, as there exist datasets for which the IQR can change by an unbounded amount. Instead, [54] offer an efficient technique to privately release the IQR. We give a slightly modified version of their approach in Algorithm 4.

Algorithm 4 IQR Propose-Test-Release [54]

```
1: **Input:** data $\mathbf{X} = \{x^{(1)}, \ldots, x^{(m)}\}$, privacy $\epsilon, \delta > 0$
2: $k = \lfloor \log \mathrm{IQR}(\mathbf{X}) \rfloor$
3: $B_1 = [e^k, e^{k+1})$
4: $B_2 = [e^{k-0.5}, e^{k+0.5})$
5: **for** j = 1,2 **do**
6:     $A_j$ := number of data-points to modify to move $\mathrm{IQR}(\mathbf{X})$ out of interval $B_j$
7:     $R_j = A_j + z$, where $z \sim \mathrm{Lap}(0, \frac{1}{\epsilon})$
8:     **if** $R_j > 1 + \log(1/\delta)$ **then**
9:         **return** $\log \mathrm{IQR}(\mathbf{X}) + z$, where $z \sim \mathrm{Lap}(0, \frac{1}{\epsilon})$
10:    **end if**
11: **end for**
12: **return** $\perp$
```

$$\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\int_{z_1+z_2-z_3}^{\infty} p(z_1 \mid a,s)p(z_2 \mid b,s)p(z_3 \mid c,s)p(z_4 \mid d,s)dz_4dz_3dz_2dz_1 \tag{4.5}$$

First the algorithm defines two intervals B_1 and B_2 which both contain $\mathrm{IQR}(\mathbf{X})$. If the IQR were to be pushed out of both of these intervals it would imply that the IQR changed by a factor of e. Therefore we loop over both intervals and calculate the number of points A_j that an adversary would need to change to push the IQR out of B_1 or B_2. Note that A_j is itself a data-sensitive query and so, to preserve privacy of this query, we can add Laplace noise to it. Then, if one of these noisy estimates $R_j = A_j + z$, where $z \sim \mathrm{Lap}(0, 1/\epsilon)$ is larger than some threshold, it implies that with high probability (exactly $1-\delta$), that the $\mathrm{IQR}(\mathbf{X})$ has multiplicative sensitivity of at most e, *for the specific dataset* $\mathbf{X}$. Note that this is precisely the local sensitivity as defined in Section 4.1.2, as it is specific to $\mathbf{X}$. This means that we can add Laplace noise z to $\log \mathrm{IQR}(\mathbf{X})$. If neither of the R_j are above the threshold then the algorithm returns null: $\perp$. This algorithm was shown to be $(3\epsilon, \delta)$-differentially private.

In our case we would like to release four private IQR scores. Note that we must look at $\mathbf{x}'$ three separate times: for $\mathrm{IQR}(\mathbf{x}'), \mathrm{IQR}(\mathbf{r}'_Y)$, and $\mathrm{IQR}(\mathbf{r}'_X)$ (and three times as well for $\mathbf{y}'$). Therefore for both $\mathbf{x}'$ and $\mathbf{y}'$ we are composing three differentially private outputs. Under simple composition this would lead to $(9\epsilon, 3\delta)$ differential privacy for both $\mathbf{x}'$ and $\mathbf{y}'$. However, we can make use of Corollary 3.21 in [56] to give $(\epsilon', 3\delta+\delta')$-differential privacy, for $0 < \epsilon' < 1$ and $\delta' > 0$, over three repeated mechanisms by ensuring each private mechanism is $(3\epsilon, \delta)$-private, where $3\epsilon = \epsilon'/(2\sqrt{6\log(1/\delta')})$.

The remaining question is whether this noise addition causes one to infer the incorrect causal direction. Again, as long as there is a significant margin between the scores, we can preserve the correct causal inference with high probability as follows.

Theorem 6. *Let $Q_{\mathbf{x}'} = \log IQR(\mathbf{x}')$, and similarly for $Q_{\mathbf{y}'}, Q_{\mathbf{r}'_X}, Q_{\mathbf{r}'_Y}$, be the true log-IQR scores. As well let $P_{\mathbf{x}'}, P_{\mathbf{y}'}, P_{\mathbf{r}'_X}, P_{\mathbf{r}'_Y}$ be the private versions, multiplied by e^z noise where $z \sim Lap(0, 1/\epsilon)$. The the following results hold:*

1. *[54] If the number of data-points needed to significantly change the IQR, A_j, is less than e then, the probability that any one of the private IQR P_* is released is small:*

$$\mathbb{P}\left[P_* \neq \perp |A_1 \text{ or } A_2 \leq e\right] \leq \frac{3\delta}{2}.$$

2. *If all private log-IQR scores are released, and the relationship between the true scores holds $Q_{\mathbf{x}'} + Q_{\mathbf{r}'_Y} < Q_{\mathbf{y}'} + Q_{\mathbf{r}'_X}$ (which implies $X \to Y$), then the probability that we make the correct causal inference from the private scores is large,*

$$\mathbb{P}[P_{\mathbf{x}'} + P_{\mathbf{r}'_Y} < P_{\mathbf{y}'} + P_{\mathbf{r}'_X}] =$$
$$1 - \frac{e^{\frac{-\gamma}{\sigma}}}{96\sigma^3}\Big(48\sigma^3 + 33\sigma^2\gamma + 9\sigma\gamma^2 + \gamma^3\Big)$$

where $\gamma = Q_{\mathbf{y}'} + Q_{\mathbf{r}'_X} - Q_{\mathbf{x}'} + Q_{\mathbf{r}'_Y}$, and $\sigma = 1/\epsilon$.

Proof. Given a set of Laplace random variables: $z_1 \sim \text{Lap}(a, s)$, $z_2 \sim \text{Lap}(b, s)$, $z_3 \sim \text{Lap}(c, s)$, $z_4 \sim \text{Lap}(d, s)$, where a, b, c, d are the means of the Laplace random variables and s are the identical scale parameters. Furthermore, given that $a + b < c + d$, we would like to compute the probability that $z_1 + z_2 < z_3 + z_4$. Let $p(x \mid \mu, \sigma)$ be the pdf of the Laplace distribution $\text{Lap}(\mu, \sigma)$. Computing the above probability requires evaluating the expression in eq. (4.5). Similar to the above proof, we can compute this integral by enumerating all of the possible cases, which gives the stated result. ■

The first result says that the probability that we release an IQR score just because too much noise was added to A_j is small. The second result says that with high probability we recover the true causal direction, depending on the size of the dataset.

4.1.8 Training Set Privacy

Let $(\mathbf{x}, \mathbf{y})$ be the initial training data and $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$ be the training data after a change in the dataset. Note that $\mathbf{x}$ and $\tilde{\mathbf{x}}$ differ in at most one element (similarly for $\mathbf{y}$ and $\tilde{\mathbf{y}}$). The length of both training datasets is n. From Algorithm 3, the only way the training set can affect the dependency scores $s_{X\to Y}, s_{Y\to X}$ is through the regression functions $\hat{f}, \hat{g}$, used to compute test set residuals $\mathbf{r}'_Y, \mathbf{r}'_X$. We use the kernel ridge regression method and so the functions $\hat{f}$ (and $\hat{g}$) can be written in the form: $\hat{f}(\mathbf{w}, x) = \mathbf{w}^\top \phi(x)$, where $\phi(x)$ is a (possibly infinite) feature space mapping to the Hilbert space corresponding to the kernel function used. Similar to other work on private regression [164] we assume that $|x|, |y| \leq 1$. The ridge regression algorithm can now be written as:

$$\mathbf{w} = \underset{\mathbf{w}\in\mathcal{H}}{\operatorname{argmin}} \frac{\lambda}{2}\|\mathbf{w}\|^2_{\mathcal{H}} + \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^\top\phi(x^{(i)}) - y^{(i)})^2, \tag{4.6}$$

where $\mathcal{H}$ is the corresponding Hilbert space. Practically speaking, even though $\mathbf{w}$ may be infinite-dimensional, because it always appears in an inner product with the feature mapping $\phi(x)$ we can utilize the 'kernel trick': $k(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^\top\phi(x^{(j)})$ to avoid having to represent $\mathbf{w}$ explicitly.

Let $\hat{f}(\mathbf{w}^*, \cdot)$ and $\hat{f}(\tilde{\mathbf{w}}^*, \cdot)$ be the classifiers resulting from the optimization problem in eq. (4.6) when trained on $(\mathbf{x}, \mathbf{y})$ and $(\tilde{\mathbf{x}}, \tilde{\mathbf{y}})$, respectively (and similarly for $\hat{g}$). We show that the residuals in Algorithm 3 are bounded.

Theorem 7. *Given that the classifiers $\hat{f}(\mathbf{w}^*, \cdot), \hat{f}(\tilde{\mathbf{w}}^*, \cdot)$ are the result of the optimization problem in eq. (4.6), the residuals of these functions $\mathbf{r}'_Y, \tilde{\mathbf{r}}'_Y$ are bounded as,*

$$|r_Y^{(i)'} - \tilde{r}_Y^{(i)}| \leq \frac{8}{n\lambda^{3/2}} \tag{4.7}$$

for all i, where $r_Y^{(i)'}, \tilde{r}_Y^{(i)}$ are the ith elements of $\mathbf{r}'_Y, \tilde{\mathbf{r}}'_Y$ and m is the size of the test set.

See the appendix for the proof. This bound holds equally for $\mathbf{r}'_X, \tilde{\mathbf{r}}'_X$. The proof of the above is inspired by the work of [150] and [94]. We place the proof in the appendix for the interested reader. As far as we are aware this is the tightest bound for the optimization problem in eq. (4.6), with a non-Lipschitz loss. In the following, we use this bound to preserve training set privacy for the dependence scores considered in the previous section.

Rank Correlation Coefficients

Note that the bound in Theorem 7 directly implies that the ranking dependence scores have global sensitivity 1 (equal to the size of their ranges). To see this note that we can consider an adversarial situation in which the rank of every element of the residual $\mathbf{r}'_Y$ changes when the training set is altered in one element (as all the residual elements may change). This means that the Laplace mechanism cannot guarantee useful privacy.

Instead, note that both ranking scores may still have reasonably bounded local sensitivity. Specifically, if we consider the list of sorted residuals, it may be that there are large gaps between neighboring residuals. If this is the case then changing the training set by one point may not change the residual rankings. Thus, the ranking scores are in some sense stable to changes in the training set (for certain sets).

Definition 7. *We call a function f **k-stable on dataset** $\mathcal{D}$ if modifying any k elements in $\mathcal{D}$ does not change the value of f. Specifically, $f(\mathcal{D}) = f(\mathcal{D}^*)$ for all $\mathcal{D}^*$ such that $\mathcal{D}$ can be transformed into $\mathcal{D}^*$ with a minimum of k element substitutions. We say f is **unstable** on $\mathcal{D}$ if it is not even 1-stable on $\mathcal{D}$. The **distance to instability** of a dataset $\mathcal{D}$ w.r.t. a function f is the number of elements that must be changed to reach an unstable dataset.*

With these definitions, we will use a modification of the Propose-Test-Release framework that makes use of this stability as described in Algorithm 13 in [56].

Theorem 8 ([56])**.** *Algorithm 13 [56] is (ϵ, δ)-differentially private. Further, for all $\beta > 0$ if $s(\mathbf{x}', \mathbf{r}'_Y)$ is $\frac{\log(1/\delta)+\log(1/\beta)}{\epsilon}$-stable on $\mathbf{r}'_Y$, then Algorithm 13 releases $s(\mathbf{x}', \mathbf{r}'_Y)$ w.p. at least $1 - \beta$.*

A lower bound on the distance to instability d is easily given by noting that $s(\mathbf{x}', \mathbf{r}'_Y)$ always outputs the same result as long as none of the ranks of $\mathbf{r}'_Y$ change. Let γ be the smallest absolute distance between any two ranks. Then a lower bound on d is, $d > \lfloor n\gamma\lambda^{3/2}/16 \rfloor$. This is the largest number of training points that may change so that the closest ranks moving towards each other do not overlap (given that they change by at most the amount in eq. 4.7). This lower-bound is sufficient to use Algorithm 13 [56] to privatize the ranking dependence scores.

HSIC Score

Theorem 9. *For $m \geq 2$, with kernels $k, l \leq 1$ where l is L_l-Lipschitz, the HSIC score has a training set sensitivity as follows,*

$$\left| \widehat{HSIC}_{k,l}(\mathbf{x}', \mathbf{r}'_Y) - \widehat{HSIC}_{k,l}(\mathbf{x}', \tilde{\mathbf{r}}'_Y) \right| \leq R \frac{32 L_l \sqrt{m}}{n}$$

where $R = \frac{8}{\lambda^{3/2}}$.

The proof follows directly from Theorem 7 and Lemma 16 in [125]. Thus, the Laplace mechanism gives us $(\epsilon, 0)$-differential privacy and Theorem 4 gives us our utility guarantee.

IQR Score

Similar to the test set privacy section we will use propose-test-release to give a useful, private IQR score. In fact, we will use the IQR algorithm almost identically, except that we will define A_j as the number of *training points* required to move the IQR out of an interval. Note that a lower bound on A_j is simply the number of points required to move every input less than the median to the left and every input larger than the median to the right (or the reverse of these), using the bound on $\mathbf{r}$ in eq. (4.7). The aforementioned privacy and utility results of the IQR propose-test-release framework apply here. The only difference is we just need to add noise to the IQR scores computed on the residuals, which implies $(6\epsilon, 2\delta)$-privacy and that the results of Theorem 6 can be tightened.

4.1.9 Experimental Results

We test our methods for private release of causal inference statistics on a small subsets from the Cause-Effect Pairs Competition collection [78]. Specifically, we randomly select 10 of the largest 25 datasets that have a causal direction either $X \to Y$ or $Y \to X$. We average over 10 random 50/50 train/test splits of the data. Table 4.2 shows the non-private accuracy of the four dependence scores over these datasets. We show the probability of correct causal inference changes as these scores are made private w.r.t. the test set. Note that these scores

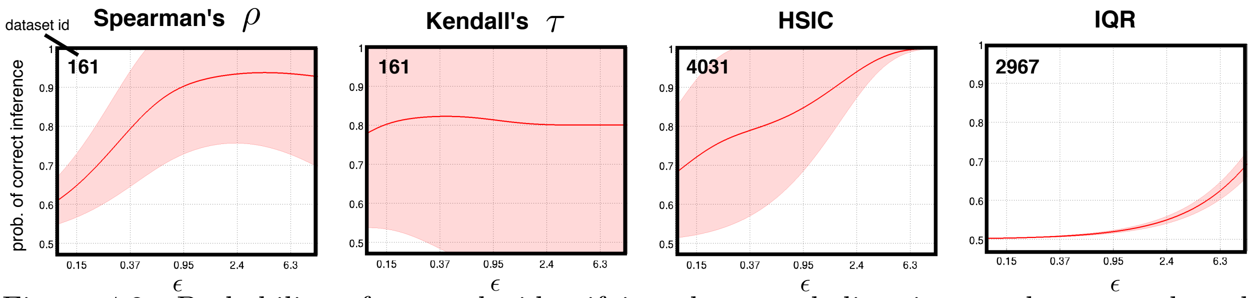


Figure 4.3: Probability of correctly identifying the causal direction on datasets selected from the Cause-Effect Pairs Challenge [78]. Datasets for which the scores perform well were selected in order to isolate the effect of privatization on the scores.

are often complementary, with the ranking-based scores performing well on datasets in which HSIC does worse, and vice-versa.

Figure 4.3 shows the effect of privatization of the *test set* on the dependence scores. Note that, for low ϵ (increased privacy), the probability of correct influence is lower as the amount of noise required blurs the true dependence scores. However, as ϵ increases, so does this probability, in some cases drastically. For the IQR score, recall that there is a probability that the algorithm returns null: $\perp$, if R_j is less than a threshold controlled by δ. We investigated this probability, by varying $\delta \in [10^{-5}, 10^{-2}]$ and sampling $10,000$ points from the appropriate Laplace distribution. We found that, for the IQR dataset in figure 4.3 *every sample* did not move R_j below the null threshold. Therefore, the probability of null is essentially 0.

The three left-most plots in Figure 4.4 demonstrate how λ, which has a large effect on the *training set sensitivity* (as described in eq. 4.7) affects the probability of correct inference. We perform this experiment for different settings of ϵ, and each one produces a distinctive 'hump' shape. This is because for small λ the sensitivity bound (4.7) is too large to produce meaningful causal inference. Similarly, for large λ the kernelized regression algorithm (4.6) is overly-regularized, which produces a poor regressor and poor dependence scores. Only when λ is within a certain range do we balance the size of the sensitivity bound with the size

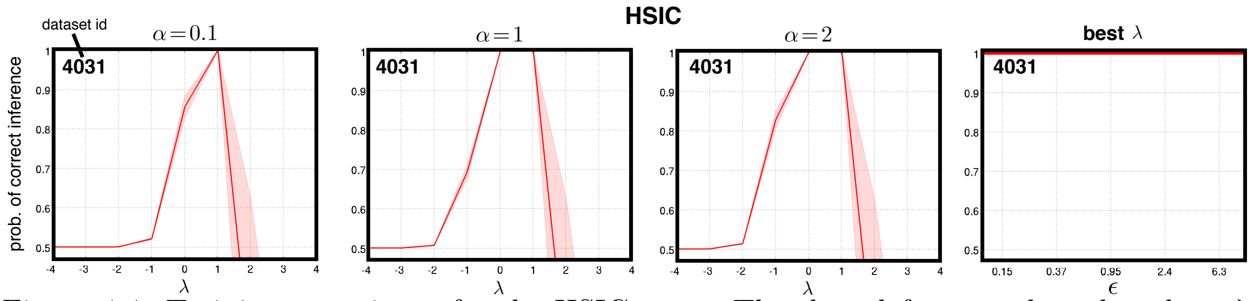


Figure 4.4: Training set privacy for the HSIC score. The three left-most plots show how λ affects the probability of correctly inferring the causal direction, while the right-most plot depicts this probability when the best λ is selected over a $\epsilon \in [0.1, 10]$. See text for more details.

Table 4.2: The non-private accuracies of the ANM model on a subset of the Cause-Effect Pairs Challenge [78], as well as the probability of correct causal inference after privatization.

DATASET IDS	4031	597	2209	2967	161	2132	1656	901	3484	1627
SIZE	7713	7748	7766	7771	7782	7784	7803	7820	7853	7862
	$\epsilon = \infty$ (NON-PRIVATE ACCURACIES)									
Spearman's ρ	0.50 ± 0.53	0.00 ± 0.00	0.00 ± 0.00	0.70 ± 0.48	0.90 ± 0.32	1.00 ± 0.00	0.00 ± 0.00	0.30 ± 0.48	0.00 ± 0.00	1.00 ± 0.00
Kendall's τ	0.50 ± 0.53	0.00 ± 0.00	0.00 ± 0.00	0.70 ± 0.48	0.80 ± 0.42	1.00 ± 0.00	0.00 ± 0.00	0.80 ± 0.42	0.00 ± 0.00	1.00 ± 0.00
HSIC [76]	1.00 ± 0.00	0.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.70 ± 0.48	0.60 ± 0.52	1.00 ± 0.00	0.40 ± 0.52	1.00 ± 0.00	0.10 ± 0.32
IQR [29]	0.50 ± 0.53	0.00 ± 0.00	0.10 ± 0.32	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	0.00 ± 0.00	0.90 ± 0.32	0.00 ± 0.00	1.00 ± 0.00
	$\epsilon = 0.1$									
Spearman's ρ	0.56 ± 0.45	0.03 ± 0.00	0.20 ± 0.02	0.57 ± 0.10	0.61 ± 0.06	0.92 ± 0.02	0.40 ± 0.06	0.34 ± 0.21	0.01 ± 0.00	0.82 ± 0.02
Kendall's τ	0.54 ± 0.48	0.00 ± 0.00	0.00 ± 0.00	0.69 ± 0.38	0.78 ± 0.24	1.00 ± 0.00	0.12 ± 0.09	0.76 ± 0.41	0.00 ± 0.00	1.00 ± 0.00
HSIC [76]	0.68 ± 0.17	0.49 ± 0.00	0.60 ± 0.01	0.50 ± 0.00	0.50 ± 0.01	0.50 ± 0.00	0.52 ± 0.00	0.43 ± 0.06	0.66 ± 0.03	0.50 ± 0.00
IQR [29]	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.51 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00	0.50 ± 0.00
	$\epsilon = 1$									
Spearman's ρ	0.50 ± 0.53	0.00 ± 0.00	0.00 ± 0.00	0.69 ± 0.43	0.91 ± 0.17	1.00 ± 0.00	0.06 ± 0.07	0.30 ± 0.41	0.00 ± 0.00	1.00 ± 0.00
Kendall's τ	0.50 ± 0.53	0.00 ± 0.00	0.00 ± 0.00	0.70 ± 0.48	0.81 ± 0.40	1.00 ± 0.00	0.00 ± 0.00	0.80 ± 0.42	0.00 ± 0.00	1.00 ± 0.00
HSIC [76]	0.85 ± 0.16	0.39 ± 0.03	0.98 ± 0.00	0.52 ± 0.01	0.55 ± 0.06	0.50 ± 0.01	0.66 ± 0.02	0.21 ± 0.25	1.00 ± 0.01	0.49 ± 0.01
IQR [29]	0.54 ± 0.04	0.48 ± 0.00	0.49 ± 0.00	0.52 ± 0.00	0.58 ± 0.01	0.51 ± 0.01	0.48 ± 0.00	0.50 ± 0.00	0.47 ± 0.01	0.51 ± 0.00
	$\epsilon = 2$									
Spearman's ρ	0.50 ± 0.53	0.00 ± 0.00	0.00 ± 0.00	0.69 ± 0.47	0.93 ± 0.17	1.00 ± 0.00	0.01 ± 0.02	0.31 ± 0.45	0.00 ± 0.00	1.00 ± 0.00
Kendall's τ	0.50 ± 0.53	0.00 ± 0.00	0.00 ± 0.00	0.70 ± 0.48	0.80 ± 0.42	1.00 ± 0.00	0.00 ± 0.00	0.80 ± 0.42	0.00 ± 0.00	1.00 ± 0.00
HSIC [76]	0.92 ± 0.09	0.29 ± 0.04	1.00 ± 0.00	0.55 ± 0.01	0.59 ± 0.11	0.51 ± 0.01	0.78 ± 0.02	0.20 ± 0.26	1.00 ± 0.00	0.48 ± 0.02
IQR [29]	0.58 ± 0.09	0.46 ± 0.01	0.49 ± 0.01	0.54 ± 0.01	0.65 ± 0.02	0.52 ± 0.02	0.47 ± 0.01	0.51 ± 0.01	0.45 ± 0.01	0.52 ± 0.01

of the regularization. This range grows larger as ϵ increases as the privacy setting becomes less strict (requiring less noise). The right-most plot shows the correct inference probability using the best λ for a range of $\epsilon \in [0.1, 10]$. With proper selection of λ we can achieve high-quality causal inference that maintains privacy w.r.t. the training set.

4.2 Related Work

Discovering the causal nature between random events has captivated researchers and philosophers long before the formal developments of statistics. This interest was formalized by [141] who argued that *all* statistical correlations in data arise from underlying causal structures between the concerned random variables. For example, the correlation between smoking and lung cancer was found to arise from a direct causal link [61].

4.2.1 Bivariate Causal Inference

One of the most popular causal inference alternatives to conditional independence testing is the Additive Noise Model (ANM) approach developed by [89] and used in many recent works [189, 126, 107, 29]. ANMs, originally designed for inferring whether $X \rightarrow Y$ or $Y \rightarrow X$ and later extended to large numbers of random variables, work under the assumption that the effect is a non-linear function of the cause plus independent noise. ANMs are one of many proposed causal inference methods in recent literature [95, 68, 116, 148]

4.2.2 Classical Methods

Work by [159, 135] shows how to determine if $X \rightarrow Y$ when these variables are a part of a larger 'causal network', via conditional independence testing. One downside to conditional independence based approaches is that inherently they cannot distinguish between Markov-equivalent graphs. Thus it may be possible that a certain set of conditional independences imply both $X \rightarrow Y$ and $Y \rightarrow X$. Furthermore, if X and Y are the only variables in the causal network there is no conditional independence test to determine whether $X \to Y$ or $Y \to X$.

4.3 Conclusion

We have presented, to the best of our knowledge, the first work towards differentially private causal inference. There are numerous directions of future work including privatizing other causal inference frameworks (e.g. IGCI [95]), analyzing the ANM algorithm without train/test splits, as well as other dependence scores. As there is significant overlap in the applications of causal inference and private learning we believe this work constitutes an important step towards making causal inference practical.

Chapter 5

Discussion and Future Directions

The goal of this thesis was to directly address three roadblocks that frequently prevent state-of-the-art machine learning models from being used in real-world application settings.

The first is the time required to evaluate a machine learning model, or its **time-cost**. In chapter 2 we introduced this problem in the context of web-search ranking: a user will likely be more dissatisfied by a ranking algorithm that is slightly more accurate and requires 10 seconds to evaluate, than a worse ranking that evaluates in 0.01 seconds. We formulated this accuracy/time trade-off as a constrained set function optimization problem. We noted that this optimization problem is approximately submodular, which allows us to obtain near-optimal solutions to the NP-hard optimization problem via a simple greedy procedure. Recursively solving these problems within a classifier tree allows us to refine the features we select to rank each web-page, instead of simply selecting the same set of features for all inputs. We demonstrated that this matches, and often outperforms, a much more complicated technique that requires clever initialization and an arduous alternating block coordinate descent procedure to optimize [184]. In the future, we would like to design algorithms for more complicated cost functions (for example, it may be that 'purchasing' a set of features 'in bulk' may be cheaper than paying for them individually).

The second practical roadblock for machine learning models is any requirement on the model size, or **space**. If we want to run models on any sort of low-memory computing device (smart phones, watches, tablets, virtual reality devices, etc.) we need to design algorithms that can flexibly construct models of various sizes, to fit the memory budget at hand. In chapter 3 we took a close look at the prolific, yet memory-intensive k-nearest neighbor classifier [40], which requires that we store the entire training set for classification. We designed a continuous relaxation to the 1-nearest neighbor rule using the stochastic neighborhood [86]. Directly minimizing this surrogate for nearest neighbor classification error allowed our method to achieve the same generalization error as the full training dataset with a small fraction of the data points (in some cases as low as 4% of the full dataset). Important future work would be theoretical results that provide guarantees about trade-offs between compression and generalization accuracy of the approach.

The third fact that prevents machine learning from weighing in on practical problems is data **privacy**. One huge field that stands to benefit from machine learning techniques is health care. Unfortunately, obtaining patient data is notoriously difficult due to its extremely sensitive nature. Indeed most countries require an arduous review process for gathering and/or collecting health records (e.g., as dictated by Health Insurance Portability and Accountability Act in the United States [6]). In chapter 4 we addressed the problem of determining the causal relationship between two random variables of interest while maintaining data privacy. We showed how privatizing a recently popular framework for causal inference called the additive noise model (ANM) [89] leads to inferences that with high probability match the non-private causal inferences. To the best of our knowledge there is no prior work that considers how to perform causal inference on sensitive data while protecting data privacy. There is much space for future research. An important next step is to consider privatizing multivariate causal inference methods such as those described by [159] and [135].

Machine learning has recently begun to address the gap between research and practice, alongside the work in this thesis. There has been an exciting amount of work on resource-efficient or budgeted or cost-sensitive learning in the last few years. This includes work in supervised learning [112, 83, 35, 186, 185, 177], structured prediction [182, 24], Bayesian optimization [158, 66], and bandit algorithms [80], among others. There are however many necessary avenues for future research. (1) In nearly all of the prior work, we assume that costs are known ahead of time; that they can be estimated from a training set. There are many possible scenarios in which the costs may vary wildly such that a training estimate may only provide a rough guess of the true cost at test time. One example is the cost of a medical procedure: while simple, routine examinations have clearly-standardized costs, more complicated procedures such as a coronary artery bypass surgery can vary in cost due to many factors. First, the location of the operation itself can change depending on the person. Second, the time to complete the operation can change. Third, where the operation takes place (e.g., at a new medical facility versus an older medical facility) affects the cost as well. Future algorithms need to carefully consider how to address these types of fluctuating costs. (2) Significant effort needs to be put towards obtaining real-world resource-efficient benchmark datasets (e.g., the equivalent of MNIST for object recognition). In general, there are two primary resource-efficient datasets that have been used multiple times: the Yahoo Learning to Rank datasets, originally from [35], and the Scene 15 recognition dataset, originally described with feature costs in [186]. The majority of follow-up work in supervised learning has used either of these datasets [184, 129] or used UCI datasets [114] and consider cost as simply the number of used features, or have random costs [83, 184, 176, 129]. Because these datasets are somewhat specialized (one is web-search ranking and the other is scene recognition), have multiple versions (there are at least two different versions of the Yahoo dataset: binary vs. non-binary) and some are not even truly cost-sensitive, it makes it difficult to compare different algorithms. The remaining work that does not use these datasets uses specialized datasets that are either not public or have non-public costs, which also makes comparison difficult. For the field to flourish (in a similar way that deep learning has flourished using MNIST, CIFAR, and ImageNet datasets), there should be an effort to construct standard benchmark datasets, as well as evaluation metrics for comparing different resource-efficient algorithms.

Techniques to shrink the size of machine learning models have also recently received much attention. In 2006, Bucilu et al. [28], were the first to consider *model compression*; they compressed large ensembles of machine learning classifiers by training small neural networks to match their performance [28]. Nearly 10 years later Ba & Caruana [9] were also able to compress deep neural networks to small ones. Inspired by this work Hinton et al., developed 'distillation' in which the costly deep neural network's predictions are properly smoothed to allow the smaller compressed network to accurately learn the complex decision function of the large network [87]. This work has inspired many follow-up deep learning papers including works in compressing deep reinforcement learning models [144], hashed deep networks [36, 79], and Bayesian posterior distillation [105], and even word embeddings [127]. There has also been work towards compressing neural models using kernel tricks [187] and using binary-valued weight vectors (as opposed to double-precision floating-point) [102, 140]. However, there are still many unexplored areas in model shrinking. (1) In large part, most of the current work is focused on compressing deep neural networks. While such models are popular, they are in large part *parametric* models, and thus the number of parameters they learn is fixed. Thus, as long as we can learn a model with few parameters that approximates a large model (e.g, as is done in distillation [87]) we have successfully compressed the model. In contrast, *non-parametric* models grow as the size of the training set grows, rendering these models impossible for small devices. Research into how to compress such models is still sparse [11], but necessary if machine learning is to be successfully used on small embedded devices. (2) Current models have mostly considered compressing models during training time so that models fit during test time, when they are evaluated. However, models may need to be retrained on devices and so it may be crucial to consider how to learn compressed datasets, that can be modified and used for retraining as needed.

Differentially private machine learning models first began appearing around 2008, when Chaudhuri et al. first derived methods for performing differentially private empirical risk minimization [33] (the first version appeared in NIPS 2008). It was followed by work in private robust statistics [54], private boosting [58], private online learning [93], private principal components analysis [34, 59], private LASSO [156, 165], private Gaussian processes [157], and private deep learning [152, 2]. At the same time there has been recent exciting work in encrypted machine learning [134, 75, 183, 7, 51]. Despite the proliferation of private machine learning techniques there are still fundamental problems that need solving. (1) In differential privacy, there is always a hyperparameter that fixes the level of privacy ϵ (and, if applicable δ). For the Laplace mechanism [55], this controls the scale of Laplace noise added to the quantity we wish to keep private. In theory, setting ϵ gives practitioners a flexible way to control how much private information is leaked to the public, regardless of the tools or side-information any adversary has at his/her disposal. In practice, it is not clear how to set this hyperparameter to ensure that data is kept 'sufficiently private'. One recent work considers setting ϵ by introducing two other hyperparameters which describes (a) the probability that the true parameter lies (b) within an interval [128]. Upon choosing these parameters, ϵ can be automatically decided. In general, differential privacy needs to derive an intuitive way to set ϵ for the task at hand. (2) For cryptographic methods, there is a dire need for methods that are simultaneously private, require reasonable memory usage, and are computationally efficient. Current machine learning methods use variants of fully homomorphic encryption [70], which usually require memory many times larger than the dataset size and can take hours to perform simple operations. The recent promising work of [51] achieved a surprising amount of parallelism, enabling practical classification of the MNIST with deep networks. One of the drawbacks however was that most of the non-linearities were replaced with quadratic approximations as multiplication depth is the most costly part of the encryption process.

In total, machine learning has come a long way towards addressing practical problems. The hope of this thesis is to motivate the ML community to think about how to solve such problems that arise when applying machine learning in the real world; by designing principled techniques, given objectives and constraints of the task at hand.

Appendix A

Privacy Proofs of Chapter 4

Proof of Theorem 4. Let $x_1 \sim \mathrm{Lap}(\mu_1, \sigma)$ and $x_2 \sim \mathrm{Lap}(\mu_2, \sigma)$ be two independent Laplace random variables with $\mu_1 < \mu_2$, then the probability of failure is $\Pr(x_1 > x_2)$. We would like to compute the probability of failure in closed form. We know that by independence, the joint probability is equal to the product of marginal probabilites. We also know that the Laplace cdf. is

$$F(x;\mu,\sigma) = \begin{cases} F_1(x;\mu,\sigma) = \frac{1}{2}\exp(\frac{x-\mu}{\sigma}) & \text{if } x \le \mu \\ F_2(x;\mu,\sigma) = 1 - \frac{1}{2}\exp(-\frac{x-\mu}{\sigma}) & \text{if } x > \mu \end{cases}$$

where F_1 and F_2 are only defined on the specified domains. There are six mutually exclusive and collective exhaustive ways for which a failure could happen:

$$\textcircled{1}\mu_1 < x_2 < x_1 < \mu_2$$

$$\textcircled{2}x_2 < \mu_1 < \mu_2 < x_1$$

$$\textcircled{3}\mu_1 < x_2 < \mu_2 < x_1$$

$$\textcircled{4}x_2 < \mu_1 < x_1 < \mu_2$$

$$⑤\mu_1 < \mu_2 < x_2 < x_1$$

$$⑥x_2 < x_1 < \mu_1 < \mu_2$$

By symmetry of the Laplace distribtuion, we know that $\Pr(③) = \Pr(④)$ and $\Pr(⑤) = \Pr(⑥)$. Thus we only need to calculate $\Pr(①), \Pr(②), \Pr(③), \text{and} \Pr(⑤)$.

$$\Pr(①) = \int_{\mu_1}^{\mu_2}\int_{x_2}^{\mu_2} p(x_1)p(x_2)dx_1dx_2 = \int_{\mu_1}^{\mu_2}[F_2(\mu_2;\mu_1,\sigma) - F_2(x_2;\mu_1,\sigma)]p(x_2)dx_2$$

Now consider the quantity being integrated, which is equal to

$$-\frac{1}{2}\exp(-\frac{\mu_2-\mu_1}{\sigma})p(x_2) + \underbrace{\frac{1}{2}\exp(-\frac{x_2-\mu_1}{\sigma})p(x_2)}_{\star}$$

The right-hand term is,

$$\star = \frac{1}{2}\exp(-\frac{x_2-\mu_1}{\sigma})\frac{1}{2\sigma}\exp(-\frac{\mu_2-x_2}{\sigma}) = \frac{1}{4\sigma}\exp(-\frac{\mu_2-\mu_1}{\sigma})$$

since $x_2 < \mu_2$. So we have that,

$$\begin{aligned}\Pr(①) &= \frac{\mu_2-\mu_1}{4\sigma}\exp(-\frac{\mu_2-\mu_1}{\sigma}) - \frac{1}{2}\exp(-\frac{\mu_2-\mu_1}{\sigma})\int_{\mu_1}^{\mu_2} p(x_2)\\ &= \frac{\mu_2-\mu_1}{4\sigma}\exp(-\frac{\mu_2-\mu_1}{\sigma}) - \frac{1}{2}\exp(-\frac{\mu_2-\mu_1}{\sigma})[\frac{1}{2} - F_1(\mu_1;\mu_2,\sigma)]\end{aligned}$$

Next, we have that

$$\begin{aligned}\Pr(②) &= \Pr(x_1 > \mu_2)\Pr(x_2 < \mu_1)\\ &= (1 - F_2(\mu_2;\mu_1,\sigma))F_1(\mu_1;\mu_2,\sigma)\\ &= \frac{1}{2}\exp(-\frac{\mu_2-\mu_1}{\sigma})F_1(\mu_1;\mu_2,\sigma)\end{aligned}$$

And similarly,

$$\begin{aligned}
\Pr(\textcircled{3}) &= \Pr(x_1 > \mu_2)\Pr(\mu_1 < x_2 < \mu_2) \\
&= (1 - F_2(\mu_2; \mu_1, \sigma))[\frac{1}{2} - F_1(\mu_1; \mu_2, \sigma)] \\
&= \frac{1}{2}\exp(-\frac{\mu_2 - \mu_1}{\sigma})[\frac{1}{2} - F_1(\mu_1; \mu_2, \sigma)]
\end{aligned}$$

and as stated, $\Pr(\textcircled{4})$ is the same. Moving on,

$$\begin{aligned}
\Pr(\textcircled{5}) &= \int_{\mu_2}^{\infty}\int_{x_2}^{\infty} p(x_1)p(x_2)dx_1dx_2 \\
&= \int_{\mu_2}^{\infty}[1 - F_2(x_2; \mu_1, \sigma)]p(x_2)dx_2 \\
&= \int_{\mu_2}^{\infty}\frac{1}{2}\exp(-\frac{x_2 - \mu_1}{\sigma})\frac{1}{2\sigma}\exp(-\frac{x_2 - \mu_2}{\sigma})dx_2 \\
&= \frac{1}{4}\int_{\mu_2}^{\infty}\frac{1}{2(\sigma/2)}\exp(-\frac{x_2 - (\mu_1 + \mu_2)/2}{\sigma/2})dx_2 \\
&= \frac{1}{4}(1 - F_2(\mu_2; \mu', \sigma')) \\
&= \frac{1}{8}\exp(-\frac{\mu_2 - (\mu_1 + \mu_2)/2}{\sigma/2}) = \frac{1}{8}\exp(-\frac{\mu_2 - \mu_1}{\sigma})
\end{aligned}$$

and as stated, $\Pr(\textcircled{6})$ is the same. So lastly,

$$\Pr(x_1 > x_2) = 2\Pr(\textcircled{5}) + 2\Pr(\textcircled{3}) + \Pr(\textcircled{2}) + \Pr(\textcircled{1}) = \frac{\mu_2 - \mu_1 + 2\sigma}{4\sigma}\exp(-\frac{\mu_2 - \mu_1}{\sigma})$$

This completes the derivation. ■

Proof of Tighter HSIC bound (described after Theorem 5). For simpler notation, let D be the original dataset and D' be the dataset with one column modified. We subscript $HSIC$

with l and k implicitly. This is the quantity of interest

$$|H\hat{S}IC(D) - H\hat{S}IC(D')| = \frac{1}{(N-1)^2}|tr(K'HL'H) - tr(KHLH)|$$

Pulling out the constant, we have

$$\begin{aligned}
&(N-1)^2|H\hat{S}IC(D) - H\hat{S}IC(D')| \\
&= |tr(K'HL'H) - tr(KHLH)| \\
&= |tr((K'HL' - KHL)H)| && \text{linearity of trace} \\
&= |tr(H(K'HL' - KHL))| && \text{cyclicity of trace}
\end{aligned}$$

Let $\mathbf{1}$ be the square matrix of $ones(N)$. We know that since $H = I - \frac{1}{N}\mathbf{1}$ by definition,

$$H(K'HL' - KHL) = (K'HL' - KHL) - \frac{1}{N}\mathbf{1}(K'HL' - KHL)$$

so we have that

$$(N-1)^2|H\hat{S}IC(D) - H\hat{S}IC(D')| = |tr(K'HL' - KHL) - \frac{1}{N}tr(\mathbf{1}(K'HL' - KHL))| \tag{A.1}$$

Next, we need three identities. Let $sum(A) = \sum_{i,j} A_{ij}$, then

Identity 1: $$tr(\mathbf{1}A) = sum(A)$$

Identity 2: $$tr(\mathbf{1}A\mathbf{1}B) = sum(A)sum(B)$$

Identity 3: $$sum(AB) = sum(BA)$$

Where Identity 3 holds only for symmetric matrices. Identity 3 is obvious since $AB = (BA)^T$ and $sum(C) = sum(C^T)$, while the first two can be proven by expanding out the matrices and using the row-column rule, or just trying random matrices on MATLAB until you believe that it works. I did both, they are sure to be correct.

And again from the definition of H, we know that

$$KHL = K(L - \frac{1}{N}\mathbf{1}L) = KL - \frac{1}{N}K\mathbf{1}L$$

so

$$K'HL' - KHL = (K'L' - \frac{1}{N}K'\mathbf{1}L') - (KL - \frac{1}{N}K\mathbf{1}L)$$

Now we continue our derivation of eq. (A.1)

$$(N-1)^2|H\hat{S}IC(D) - H\hat{S}IC(D')| = |\underbrace{tr(K'HL' - KHL)}_{\star} - \underbrace{\frac{1}{N}sum(K'HL' - KHL)|}_{\diamond}$$

We can rewrite each term $\star$ and $\diamond$ using our traces identities as follows,

$$\begin{aligned}
\star &= [tr(K'L') - \frac{1}{N}sum(L'K')] - [tr(KL) - \frac{1}{N}sum(LK)] \\
\diamond &= \frac{1}{N}[sum(K'L' - \frac{1}{N}K'\mathbf{1}L') - sum(KL - \frac{1}{N}K\mathbf{1}L)] \\
&= \frac{1}{N}[sum(K'L') - \frac{1}{N}sum(K'\mathbf{1}L')] \\
&- \frac{1}{N}[sum(KL) - \frac{1}{N}sum(K\mathbf{1}L)]
\end{aligned}$$

By identity 3, we see that the $sum(KL)$ and $sum(LK)$ as well as the $sum(K'L')$ and $sum(L'K')$ terms in $\star$ and $\diamond$ are identical. Thus we are left with

$$\begin{aligned}
&(N-1)^2|H\hat{S}IC(D) - H\hat{S}IC(D')| = \star - \diamond \\
= &|[tr(K'L') - tr(KL)] \\
&- \frac{2}{N}[sum(K'L') - sum(KL)] \\
&+ \frac{1}{N}[\frac{1}{N}sum(K'\mathbf{1}L') - \frac{1}{N}sum(K\mathbf{1}L)]| \\
= &|[sum(K'.*L') - sum(K.*L)] \\
&- \frac{2}{N}[sum(K'L') - sum(KL)] \\
&+ \frac{1}{N^2}[sum(K')sum(L') - sum(K)sum(L)]|
\end{aligned} \tag{A.2}$$

where the last line comes from applying Identity 1 backwards so we have, for example, $tr(\mathbf{1}K\mathbf{1}L)$, then applying Identity 2. We use MATLAB© notation .* for the element-wise product of two matrices.

We bound eq. (A.2) by the triangle inequality,

$$\begin{aligned}
&(N-1)^2|H\hat{S}IC(D) - H\hat{S}IC(D')| \\
&\leq |sum(K'.*L') - sum(K.*L)| && \text{①} \\
&+ \frac{2}{N}|sum(K'L') - sum(KL)| && \text{②} \\
&+ \frac{1}{N^2}|sum(K')sum(L') - sum(K)sum(L)| && \text{③}
\end{aligned}$$

$$
\begin{aligned}
&(N-1)^2|H\hat{S}IC(D) - H\hat{S}IC(D')| \\
&\leq [\text{①} + \text{②} + \text{③}] \leq \max_{K,K',L,L'} \text{①} + \max_{K,K',L,L'} \text{②} + \max_{K,K',L,L'} \text{③}
\end{aligned}
$$

Recall that the kernels k and l are bounded by 1. And that the kernel pairs K, K' and L, L' differ in at most one row and column. Thus, for ①, it is clear that the maximum occurs when a row and column c (no matter what c is) is changed from all 0 to 1 in both L and K, so $\max\limits_{K,K',L,L'} \text{①} = 2N - 1$

For ③, the maximum occurs at exactly same the point as ①, and the value achieved is

$$
\frac{1}{N^2}[N^4 - (N^2 - 2N + 1)^2] \leq 4N - 5
$$

for $N > 3$. For ②, applying the row-column rule and reasoning on small matrices inductively suggest that the maximum is also achieved when we change one row and column of L and K from all 0 to 1, and is thus

$$
\frac{2}{N}[N^2 + (N-1)(2N-1)] \leq 6N - 5
$$

for $N \geq 2$. As the argmax of all three terms coincide, we have that

$$
\max_C \, [\text{①} + \text{②} + \text{③}] = \max_C \text{①} + \max_C \text{②} + \max_C \text{③}
$$

Therefore, we have derived that for all practical purposes, the overall bound is $\frac{12N-11}{N^2-1}$. ■

Proof of Theorem 7.

$$
|r_Y^{(i)\prime} - \tilde{r}_Y^{(i)}| = \left|\mathbf{w}^\top \phi(X) - \tilde{\mathbf{w}}^\top \phi(X)\right| \leq \|\mathbf{w} - \tilde{\mathbf{w}}\|_{\mathcal{H}} \|\phi(X)\|_{\mathcal{H}} \leq \|\mathbf{w} - \tilde{\mathbf{w}}\|_{\mathcal{H}} \tag{A.3}
$$

In the above we used the fact that $\|\phi(X)\|_{\mathcal{H}} = \sqrt{K(x,x)} \leq 1$. On the other hand note that $\mathbf{w}$ is the minimizer of regularized objective on data set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})$ and $\tilde{\mathbf{w}}$ is the minimizer on set $(x^{(1)}, y^{(1)}), \ldots, (x^{(n-1)}, y^{(n-1)}), (x^{(n)'}, y^{(n)'})$ (we assume the last coordinate is the one that is changes w.l.o.g.). By strong convexity of the regularized objective we have,

$$
\begin{aligned}
\frac{\lambda}{2}\|\mathbf{w}-\tilde{\mathbf{w}}\|^2 &\leq \frac{\lambda}{2}\|\tilde{\mathbf{w}}\|_{\mathcal{H}}^2 + \frac{1}{n}\sum_{i=1}^{n}(\tilde{\mathbf{w}}^\top \phi(x^{(i)}) - y^{(i)})^2 - \frac{\lambda}{2}\|\mathbf{w}\|_{\mathcal{H}}^2 - \frac{1}{n}\sum_{i=1}^{n}(\mathbf{w}^\top \phi(x^{(i)}) - y^{(i)})^2 \\
&\leq \frac{\lambda}{2}\|\tilde{\mathbf{w}}\|_{\mathcal{H}}^2 + \frac{1}{n}\sum_{i=1}^{n-1}(\tilde{\mathbf{w}}^\top \phi(x^{(i)}) - y^{(i)})^2 + (\tilde{\mathbf{w}}^\top \phi(\tilde{x}^{(n)}) - \tilde{y}^{(n)})^2 \\
&\quad - \frac{\lambda}{2}\|\mathbf{w}\|_{\mathcal{H}}^2 - \frac{1}{n}\sum_{i=1}^{n-1}(\mathbf{w}^\top \phi(x^{(i)}) - y^{(i)})^2 - (\mathbf{w}^\top \phi(\tilde{x}^{(n)}) - \tilde{y}^{(n)})^2 \\
&\quad + \frac{1}{n}(\tilde{\mathbf{w}}^\top \phi(x^{(n)}) - y^{(n)})^2 - (\mathbf{w}^\top \phi(x^{(n)}) - y^{(n)})^2 \\
&\quad - \frac{1}{n}(\tilde{\mathbf{w}}^\top \phi(\tilde{x}^{(n)}) - \tilde{y}^{(n)})^2 + (\mathbf{w}^\top \phi(\tilde{x}^{(n)}) - \tilde{y}^{(n)})^2 \\
&\leq \frac{2}{n}\sup_{x,y\in[-1,1]}\Big((\tilde{\mathbf{w}}^\top \phi(x) - y)^2 - (\mathbf{w}^\top \phi(x) - y)^2\Big) \\
&\leq \frac{2}{n}\|\tilde{\mathbf{w}}-\mathbf{w}\| \times (\|\tilde{\mathbf{w}}\| + \|\mathbf{w}\| + 2)
\end{aligned}
$$

Now note that since $0 \in \mathcal{H}$ we can conclude that,

$$
\|\mathbf{w}\| \leq \frac{1}{\sqrt{\lambda}}
$$

(The above is got by plugging in the 0 in the regularized objective which yields a value of 1 and since loss is non-negative, we can conclude that the norm of the minimizer of the regularized objective is at most $1/\sqrt{\lambda}$. Plugging this in yields:

$$
\|\mathbf{w}-\tilde{\mathbf{w}}\| \leq \frac{8}{\lambda^{3/2} n}
$$

Plugging this in Eq. A.3 yields the theorem. ■

Appendix B

Grants

I would like to thank and acknowledge the grants that have funded me throughout my graduate study including: NIH grant U01 1U01NS073457-01, NSF grants IIS-1149882, EFRI-1137211, IIA-1355406, III-1526012, and III-1618134.

References

[1] A face is exposed for aol searcher no. 4417749. `http://search-id.com/user/4417749-thelma_arnold`. Accessed: 2016-05-28.

[2] Martín Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. *arXiv preprint arXiv:1607.00133*, 2016.

[3] Mohamed Aly, Mario Munich, and Pietro Perona. Indexing in large scale image collections: Scaling properties and benchmark. In *WACV*, pages 418–425. IEEE, 2011.

[4] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468. IEEE, 2006.

[5] F. Angiulli. Fast condensed nearest neighbor rule. In *ICML*, pages 25–32, 2005.

[6] George J Annas. Hipaa regulations-a new era of medical-record privacy? *New England Journal of Medicine*, 348(15):1486–1490, 2003.

[7] Louis JM Aslett, Pedro M Esperança, and Chris C Holmes. Encrypted statistical machine learning: new privacy preserving methods. *arXiv preprint arXiv:1508.06845*, 2015.

[8] Mordecai Avriel. *Nonlinear programming: analysis and methods*. Courier Corporation, 2003.

[9] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *NIPS*, pages 2654–2662, 2014.

[10] Francis Bach. Learning with submodular functions: A convex optimization perspective. *arXiv preprint arXiv:1111.6453*, 2011.

[11] Olivier Bachem, Mario Lucic, and Andreas Krause. Coresets for nonparametric estimationthe case of dp-means. In *ICML*, 2015.

[12] Wenruo Bai, Rishabh Iyer, Kai Wei, and Jeff Bilmes. Algorithms for optimizing the ratio of submodular functions. In *International Conference on Machine Learning (ICML)*, New York, NY, July 2016.

[13] Sergey Bakin. Adaptive regression and model selection in data mining problems. 1999.

[14] S. Bandyopadhyay and U. Maulik. Efficient prototype reordering in nearest neighbor classification. *Pattern Recognition*, 35(12):2791–2799, 2002.

[15] Mikhail Belkin, Partha Niyogi, and Vikas Sindhwani. Manifold regularization: A geometric framework for learning from labeled and unlabeled examples. *Journal of machine learning research*, 7(Nov):2399–2434, 2006.

[16] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, 24(4):509–522, 2002.

[17] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. *NIPS*, 23:163–171, 2010.

[18] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

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

[20] S. Bermejo and J. Cabestany. Adaptive soft k-nearest-neighbor classifiers. *Pattern Recognition*, 32:2077–2979, 1999.

[21] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104, 2006.

[22] Yatao Bian, Baharan Mirzasoleiman, Joachim M Buhmann, and Andreas Krause. Guaranteed non-convex optimization: Submodular maximization over continuous domains. *arXiv preprint arXiv:1606.05615*, 2016.

[23] C.M. Bishop. *Pattern recognition and machine learning*. Springer New York, 2006.

[24] Tolga Bolukbasi, Kai-Wei Chang, Joseph Wang, and Venkatesh Saligrama. Resource constrained structured prediction. *arXiv preprint arXiv:1602.08761*, 2016.

[25] Léon Bottou. Stochastic learning. In *Advanced lectures on machine learning*, pages 146–168. Springer, 2004.

[26] Lubomir Bourdev and Jonathan Brandt. Robust object detection via soft cascade. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 236–243. IEEE, 2005.

[27] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[28] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *KDD*, pages 535–541. ACM, 2006.

[29] Peter Bühlmann, Jonas Peters, Jan Ernest, et al. Cam: Causal additive models, high-dimensional order search and penalized regression. *The Annals of Statistics*, 42(6):2526–2556, 2014.

[30] R. Busa-Fekete, D. Benbouzid, B. Kégl, et al. Fast classification using sparse decision dags. In *ICML*, 2012.

[31] Vijay Chandrasekhar, Gabriel Takacs, David Chen, Sam S Tsai, Jatinder Singh, and Bernd Girod. Transform coding of image feature descriptors. In *IS&T/SPIE Electronic Imaging*, 2009.

[32] C.L. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, 100(11):1179–1184, 1974.

[33] Kamalika Chaudhuri, Claire Monteleoni, and Anand D Sarwate. Differentially private empirical risk minimization. *JMLR*, 12:1069–1109, 2011.

[34] Kamalika Chaudhuri, Anand Sarwate, and Kaushik Sinha. Near-optimal differentially private principal components. In *Advances in Neural Information Processing Systems*, pages 989–997, 2012.

[35] M. Chen, K. Q. Weinberger, O. Chapelle, D. Kedem, and Z. Xu. Classifier cascade for minimizing feature evaluation cost. In *AISTATS*, pages 218–226, 2012.

[36] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

[37] M. M. Chong, A. Abraham, and M. Paprzycki. Traffic accident analysis using machine learning paradigms. *Informatica (Slovenia)*, 29(1):89–98, 2005.

[38] Stephan Clémençon, Igor Colin, and Aurélien Bellet. Scaling-up empirical risk minimization: Optimization of incomplete u-statistics. *Journal of Machine Learning Research*, 17(76):1–36, 2016.

[39] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[40] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[41] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, volume 1, pages 886–893, 2005.

[42] Abhimanyu Das, Anirban Dasgupta, and Ravi Kumar. Selecting diverse features via spectral regularization. In *Advances in Neural Information Processing Systems 25*, pages 1592–1600, 2012.

[43] Abhimanyu Das and David Kempe. Submodular meets spectral: Greedy algorithms for subset selection, sparse approximation and dictionary selection. *arXiv preprint arXiv:1102.3975*, 2011.

[44] J.V. Davis, B. Kulis, P. Jain, S. Sra, and I.S. Dhillon. Information-theoretic metric learning. In *ICML*, pages 209–216, 2007.

[45] C. Decaestecker. Finding prototypes for nearest neighbour classification by means of gradient descent and deterministic annealing. *Pattern Recognition*, 30(2):281–288, 1997.

[46] J. Deng, S. Satheesh, A.C. Berg, and L. Fei-Fei. Fast and balanced: Efficient label tree learning for large scale object recognition. In *NIPS*, 2011.

[47] V.S. Devi and M.N. Murty. An incremental prototype set building technique. *Pattern Recognition*, 35(2):505–513, 2002.

[48] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Kernel k-means: spectral clustering and normalized cuts. In *KDD*, pages 551–556. ACM, 2004.

[49] George Diekhoff. *Statistics for the social and behavioral sciences: Univariate, bivariate, multivariate*. Wm. C. Brown Publishers Dubuque, IA, 1992.

[50] Irit Dinur and Kobbi Nissim. Revealing information while preserving privacy. In *Proceedings of the SIGMOD-SIGACT-SIGART symposium on principles of database systems*, pages 202–210. ACM, 2003.

[51] Nathan Dowlin, Ran Gilad-Bachrach, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 201–210, 2016.

[52] M. Dredze, R. Gevaryahu, and A. Elias-Bachrach. Learning fast classifiers for image spam. In *proceedings of the Conference on Email and Anti-Spam (CEAS)*, 2007.

[53] George H Dunteman. *Principal components analysis*. Number 69. Sage, 1989.

[54] Cynthia Dwork and Jing Lei. Differential privacy and robust statistics. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*, pages 371–380. ACM, 2009.

[55] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography*, pages 265–284. Springer, 2006.

[56] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 9(3-4):211–407, 2013.

[57] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.

[58] Cynthia Dwork, Guy N Rothblum, and Salil Vadhan. Boosting and differential privacy. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 51–60. IEEE, 2010.

[59] Cynthia Dwork, Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Analyze gauss: Optimal bounds for privacy-preserving principal component analysis. In *STOC*, 2014.

[60] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004.

[61] Centers for Disease Control, Prevention, et al. *How tobacco smoke causes disease: The biology and behavioral basis for smoking-attributable disease: A report of the surgeon general*. Centers for Disease Control and Prevention (US), 2010.

[62] John B Fraleigh and Raymond A Beauregard. Linear algebra. add0ison, 1987.

[63] Yoav Freund and Robert E Schapire. A desicion-theoretic generalization of on-line learning and an application to boosting. In *European conference on computational learning theory*, pages 23–37. Springer, 1995.

[64] Nir Friedman and Iftach Nachman. Gaussian process networks. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 211–219. Morgan Kaufmann Publishers Inc., 2000.

[65] T. Gao and D. Koller. Active classification based on value of classifier. In *NIPS*, pages 1062–1070. 2011.

[66] J. Gardner, M. Kusner, Z. Xu, K. Q. Weinberger, and J. Cunningham. Bayesian optimization with inequality constraints. In *ICML*, pages 937–945, 2014.

[67] G.W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18:431–433, 1972.

[68] Philipp Geiger, Dominik Janzing, and Bernhard Schölkopf. Estimating causal effects by bounding confounding. In *Proceedings of the 30th Conference on Uncertainty in Artificial Intelligence*, pages 240–249, 2014.

[69] Philipp Geiger, Kun Zhang, Bernhard Schoelkopf, Mingming Gong, and Dominik Janzing. Causal inference by identification of vector autoregressive processes with hidden components. In *ICML*, pages 1917–1925, 2015.

[70] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[71] A.S. Georghiades, P.N. Belhumeur, and D.J. Kriegman. From few to many: Illumination cone models for face recognition under variable lighting and pose. *PAMI*, 23(6):643–660, 2001.

[72] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. Similarity search in high dimensions via hashing. In *VLDB*, volume 99, pages 518–529, 1999.

[73] J. Goldberger, G.E. Hinton, S.T. Roweis, and R. Salakhutdinov. Neighbourhood components analysis. In *NIPS*, pages 513–520. 2004.

[74] Daniel Golovin and Andreas Krause. Adaptive submodularity: Theory and applications in active learning and stochastic optimization. *Journal of Artificial Intelligence Research*, 42(1):427–486, 2011.

[75] Thore Graepel, Kristin Lauter, and Michael Naehrig. Ml confidential: Machine learning on encrypted data. In *International Conference on Information Security and Cryptology*, pages 1–21. Springer, 2012.

[76] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Algorithmic learning theory*, pages 63–77. Springer, 2005.

[77] A. Grubb and J. A. Bagnell. Speedboost: Anytime prediction with uniform near-optimality. In *AISTATS*, 2012.

[78] I. Guyon. Cause-effect pairs kaggle competition, 2013.

[79] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015.

[80] M. K. Hanawal, V. Saligrama, M. Valko, and R. Munos. Cheap bandits. In *ICML*, 2015.

[81] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516, 1968.

[82] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. The elements of statistical learning: data mining, inference and prediction. *New York: Springer-Verlag*, 1(8):371–406, 2001.

[83] He He, Jason Eisner, and Hal Daume. Imitation learning by coaching. In *Advances in Neural Information Processing Systems*, pages 3149–3157, 2012.

[84] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[85] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[86] G.E. Hinton and S.T. Roweis. Stochastic neighbor embedding. In *NIPS*, pages 833–840. MIT Press, 2002.

[87] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[88] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[89] Patrik O Hoyer, Dominik Janzing, Joris M Mooij, Jonas Peters, and Bernhard Schölkopf. Nonlinear causal discovery with additive noise models. In *Advances in neural information processing systems*, pages 689–696, 2009.

[90] Peter J Huber et al. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[91] Rishabh Iyer and Jeff Bilmes. Submodular optimization with submodular cover and submodular knapsack constraints. In *Neural Information Processing Society (NIPS)*, Lake Tahoe, CA, December 2013.

[92] Prateek Jain, Pravesh Kothari, and Abhradeep Thakurta. Differentially private online learning. *COLT*, 2012.

[93] Prateek Jain, Pravesh Kothari, and Abhradeep Thakurta. Differentially private online learning. In *COLT*, volume 23, pages 24–1, 2012.

[94] Prateek Jain and Abhradeep Thakurta. Differentially private learning with kernels. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 118–126, 2013.

[95] Dominik Janzing, Joris Mooij, Kun Zhang, Jan Lemeire, Jakob Zscheischler, Povilas Daniušis, Bastian Steudel, and Bernhard Schölkopf. Information-geometric approach to inferring causal directions. *Artificial Intelligence*, 182:1–31, 2012.

[96] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

[97] Shihao Ji and Lawrence Carin. Cost-sensitive feature acquisition and classification. *Pattern Recognition*, 40(5):1474–1485, 2007.

[98] Richard Arnold Johnson and Dean W Wichern. *Applied multivariate statistical analysis*, volume 5. Prentice hall Upper Saddle River, NJ, 2002.

[99] Pallika Kanani and Prem Melville. Prediction-time active feature-value acquisition for cost-effective customer targeting. *Advances In Neural Information Processing Systems (NIPS)*, 2008.

[100] Yutaka Kano and Shohei Shimizu. Causal inference using nonnormality. In *Proceedings of the International Symposium on Science of Modeling, the 30th Anniversary of the Information Criterion*, pages 261–270, 2003.

[101] Sergey Karayev, Tobias Baumgartner, Mario Fritz, and Trevor Darrell. Timely object recognition. In *Advances in Neural Information Processing Systems*, pages 890–898, 2012.

[102] Minje Kim and Paris Smaragdis. Bitwise neural networks. *arXiv preprint arXiv:1601.06071*, 2016.

[103] T. Kohonen. Improved versions of learning vector quantization. In *IJCNN*, pages 545–550. IEEE, 1990.

[104] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[105] A. Korattikara, V. Rathod, K. Murphy, and M. Welling. Bayesian dark knowledge. *arXiv preprint arXiv:1506.04416*, 2015.

[106] M. Kowalski. Sparse regression using mixed norms. *Applied and Computational Harmonic Analysis*, 27(3):303–324, 2009.

[107] Samory Kpotufe, Eleni Sgouritsa, Dominik Janzing, and Bernhard Schölkopf. Consistency of causal inference under the additive noise model. In *ICML*, 2014.

[108] Andreas Krause and Volkan Cevher. Submodular dictionary selection for sparse representation. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 567–574, 2010.

[109] Andreas Krause and Carlos E Guestrin. Near-optimal nonmyopic value of information in graphical models. *arXiv preprint arXiv:1207.1394*, 2012.

[110] Matt Kusner, Stephen Tyree, Kilian Q Weinberger, and Kunal Agrawal. Stochastic neighbor compression. In *ICML*, pages 622–630, 2014.

[111] Su-In Lee, Honglak Lee, Pieter Abbeel, and Andrew Y Ng. Efficient l˜ 1 regularized logistic regression. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 401. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

[112] L. Lefakis and F. Fleuret. Joint cascade optimization using a product of boosted classifiers. In *NIPS*, pages 1315–1323. 2010.

[113] Adrien Marie Legendre. *Nouvelles méthodes pour la détermination des orbites des comètes*. Number 1. F. Didot, 1805.

[114] M. Lichman. UCI machine learning repository, 2013.

[115] C. L. Liu and M. Nakagawa. Prototype learning algorithms for nearest neighbor classifier with application to handwritten character recognition. In *ICDAR*, pages 378–381. IEEE, 1999.

[116] David Lopez-Paz, Krikamol Muandet, Bernhard Schölkopf, and Iliya Tolstikhin. Towards a learning theory of cause-effect inference. In *ICML*, 2015.

[117] David G Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[118] Mario Lucic, Olivier Bachem, Morteza Zadimoghaddam, and Andreas Krause. Horizontally scalable submodular maximization. In *Proc. International Conference on Machine Learning (ICML)*, July 2016.

[119] Frank McSherry and Kunal Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103. IEEE, 2007.

[120] Marina MeilPa and Jianbo Shi. Learning segmentation by random walks. In *Neural Information Processing Systems*, 2001.

[121] Scott Menard. *Applied logistic regression analysis*. Number 106. Sage, 2002.

[122] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[123] R.A. Mollineda, F.J. Ferri, and E. Vidal. An efficient prototype merging strategy for the condensed 1-nn rule through class-conditional hierarchical clustering. *Pattern Recognition*, 35(12):2771–2782, 2002.

[124] Joris M Mooij, Dominik Janzing, Tom Heskes, and Bernhard Schölkopf. On causal discovery with cyclic additive noise models. In *Advances in neural information processing systems*, pages 639–647, 2011.

[125] Joris M Mooij, Jonas Peters, Dominik Janzing, Jakob Zscheischler, and Bernhard Schölkopf. Distinguishing cause from effect using observational data: methods and benchmarks. *arXiv preprint arXiv:1412.3773*, 2014.

[126] Joris M Mooij, Oliver Stegle, Dominik Janzing, Kun Zhang, and Bernhard Schölkopf. Probabilistic latent variable models for distinguishing between cause and effect. In *Advances in Neural Information Processing Systems*, pages 1687–1695, 2010.

[127] Lili Mou, Ge Li, Yan Xu, Lu Zhang, and Zhi Jin. Distilling word embeddings: An encoding approach. *arXiv preprint arXiv:1506.04488*, 2015.

[128] Maurizio Naldi and Giuseppe D'Acquisto. Differential privacy: An estimation theory-based method for choosing epsilon. *arXiv preprint arXiv:1510.00917*, 2015.

[129] Feng Nan, Joseph Wang, and Venkatesh Saligrama. Feature-budgeted random forest. In *ICML*, 2015.

[130] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[131] G. L. Nemhauser, L. A. Wolsey, and M. L. Fisher. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14(1):265–294, 1978.

[132] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. Smooth sensitivity and sampling in private data analysis. In *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing*, pages 75–84. ACM, 2007.

[133] S.M. Omohundro. *Five balltree construction algorithms.* International Computer Science Institute, Berkeley, 1989.

[134] Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*, 2007:18, 2007.

[135] Judea Pearl. Causality: models, reasoning, and inference. 2000.

[136] 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.

[137] Jonas Peters, Joris M Mooij, Dominik Janzing, and Bernhard Schölkopf. Causal discovery with continuous additive noise models. *The Journal of Machine Learning Research*, 15(1):2009–2053, 2014.

[138] J. Pujara, H. Daumé III, and L. Getoor. Using classifier cascades for scalable e-mail classification. In *CEAS*, 2011.

[139] Carl Edward Rasmussen and Christopher K. I. Williams. Gaussian processes for machine learning. 2006.

[140] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. *arXiv preprint arXiv:1603.05279*, 2016.

[141] Hans Reichenbach and Maria Reichenbach. *The direction of time.* Univ of California Press, 1956.

[142] Peter Richtarik and Mark Schmidt. Modern convex optimization methods for large-scale empirical risk minimization. In *ICML*, 2015.

[143] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[144] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[145] M. Saberian and N. Vasconcelos. Boosting classifier cascades. In J. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R.S. Zemel, and A. Culotta, editors, *NIPS*, pages 2047–2055. 2010.

[146] S. Salzberg, A.L. Delcher, D. Heath, and S. Kasif. Best-case results for nearest-neighbor learning. *PAMI*, 17(6):599–608, 1995.

[147] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl. Application of dimensionality reduction in recommender system-a case study. Technical report, DTIC Document, 2000.

[148] Eleni Sgouritsa, Dominik Janzing, Philipp Hennig, and Bernhard Schölkopf. Inference of cause and effect with unsupervised inverse regression. In *AISTATS*, pages 847–855, 2015.

[149] Naji Shajarisales, Dominik Janzing, Bernhard Shoelkopf, and Michel Besserve. Telling cause from effect in deterministic linear dynamical systems. In *ICML*, 2015.

[150] Shai Shalev-Shwartz, Ohad Shamir, Nathan Srebro, and Karthik Sridharan. Stochastic convex optimization. In *COLT*, 2009.

[151] Victor S Sheng and Charles X Ling. Feature value acquisition in testing: a sequential batch test algorithm. In *Proceedings of the 23rd international conference on Machine learning*, pages 809–816. ACM, 2006.

[152] Reza Shokri and Vitaly Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1310–1321. ACM, 2015.

[153] P. Simard, Y. LeCun, and J.S. Denker. Efficient pattern recognition using a new transformation distance. In *NIPS*, pages 50–58, 1992.

[154] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[155] Adish Singla, Sebastian Tschiatschek, and Andreas Krause. Noisy submodular maximization via adaptive sampling with applications to crowdsourced image collection summarization. In *Proc. Conference on Artificial Intelligence (AAAI)*, February 2016.

[156] Adam Smith and Abhradeep Thakurta. Differentially private feature selection via stability arguments, and the robustness of the lasso. In *Proceedings of Conference on Learning Theory*, 2013.

[157] Michael Thomas Smith, Max Zwiessele, and Neil D Lawrence. Differentially private gaussian processes. *arXiv preprint arXiv:1606.00720*, 2016.

[158] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, pages 2951–2959, 2012.

[159] Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, prediction, and search*, volume 81. MIT press, 2000.

[160] Suvrit Sra, Sebastian Nowozin, and Stephen J Wright. *Optimization for machine learning*. Mit Press, 2012.

[161] M. Streeter and D. Golovin. An online algorithm for maximizing submodular functions. Technical report, DTIC Document, 2007.

[162] Xiaohai Sun, Dominik Janzing, and Bernhard Schölkopf. Causal reasoning by evaluating the complexity of conditional densities with kernel methods. *Neurocomputing*, 71(7):1248–1256, 2008.

[163] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[164] Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Private empirical risk minimization beyond the worst case: The effect of the constraint set geometry. *arXiv preprint arXiv:1411.5417*, 2014.

[165] Kunal Talwar, Abhradeep Thakurta, and Li Zhang. Nearly optimal private lasso. In *Advances in Neural Information Processing Systems*, pages 3025–3033, 2015.

[166] D. Tarlow, K. Swersky, L. Charlin, I. Sutskever, and R. Zemel. Stochastic k-neighborhood selection for supervised and unsupervised learning. In *ICML*, pages 199–207, 2013.

[167] C-Y Teng, Y-R Lin, and L. A. Adamic. Recipe recommendation using ingredient networks. In *Proceedings of the 4th Annual ACM Web Science Conference*, pages 298–307. ACM, 2012.

[168] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[169] I. Tomek. Two modifications of cnn. *IEEE Trans. on Systems, Man, and Cybernetics*, (11):769–772, 1976.

[170] G.T. Toussaint. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface*, 34, 2002.

[171] D. Tran and A. Sorokin. Human activity recognition with metric learning. In *ECCV*, pages 548–561. Springer, 2008.

[172] Kirill Trapeznikov and Venkatesh Saligrama. Supervised sequential classification under budget constraints. In *AISTATS*, pages 581–589, 2013.

[173] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *JMLR*, 9(2579-2605):85, 2008.

[174] P. Viola and M.J. Jones. Robust real-time face detection. *IJCV*, 57(2):137–154, 2004.

[175] J. Wang and V. Saligrama. Local supervised learning through space partitioning. In *Advances in Neural Information Processing Systems 25*, pages 91–99, 2012.

[176] Joseph Wang, Kirill Trapeznikov, and Venkatesh Saligrama. An lp for sequential learning under budgets. In *AISTATS*, pages 987–995, 2014.

[177] Joseph Wang, Kirill Trapeznikov, and Venkatesh Saligrama. Efficient learning by directed acyclic graph for resource constrained prediction. In *Advances in Neural Information Processing Systems*, pages 2152–2160, 2015.

[178] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[179] Kilian Q Weinberger, Fei Sha, Qihui Zhu, and Lawrence K Saul. Graph laplacian regularization for large-scale semidefinite programming. In *NIPS*, pages 1489–1496, 2006.

[180] K.Q. Weinberger and L.K. Saul. Fast solvers and efficient implementations for distance metric learning. In *ICML*, pages 1160–1167, 2008.

[181] K.Q. Weinberger and L.K. Saul. Distance metric learning for large margin nearest neighbor classification. *JMLR*, 10:207–244, 2009.

[182] David J Weiss and Ben Taskar. Learning adaptive value of information for structured prediction. In *Advances in Neural Information Processing Systems*, pages 953–961, 2013.

[183] Pengtao Xie, Misha Bilenko, Tom Finley, Ran Gilad-Bachrach, Kristin Lauter, and Michael Naehrig. Crypto-nets: Neural networks over encrypted data. *arXiv preprint arXiv:1412.6181*, 2014.

[184] Z. Xu, M. J. Kusner, M. Chen, and K. Q. Weinberger. Cost-sensitive tree of classifiers. In *ICML*, 2013.

[185] Z. Xu, M. J. Kusner, K. Q. Weinberger, M. Chen, and O. Chapelle. Budgeted learning with trees and cascades. *JMLR*, 2014.

[186] Z. Xu, K. Q. Weinberger, and O. Chapelle. The greedy miser: Learning under test-time budgets. In *ICML*, 2012.

[187] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang. Deep fried convnets. In *CVPR*, pages 1476–1483, 2015.

[188] Ming Yuan and Yi Lin. Model selection and estimation in regression with grouped variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67, 2006.

[189] Kun Zhang and Aapo Hyvärinen. On the identifiability of the post-nonlinear causal model. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 647–655. AUAI Press, 2009.

[190] Navid Zolghadr, Gábor Bartók, Russell Greiner, András György, and Csaba Szepesvári. Online learning with costly features and labels. In *Advances in Neural Information Processing Systems*, pages 1241–1249, 2013.