THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF COMPUTER SCIENCE

PAIRBOOST: GRADIENT BOOSTED CLASSIFICATION FROM PAIRWISE DATA

NEIL ASHTEKAR
SPRING 2021

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Computer Science
with honors in Computer Science

Reviewed and approved* by the following:

Mehrdad Mahdavi
Professor of Computer Science and Engineering
Thesis Supervisor

Rebecca Passonneau
Professor of Computer Science and Engineering
Honors Advisor

*Electronic approvals on file

Abstract

Supervised binary classification requires access to a fully labeled dataset. In many applications, gathering labels can be costly and difficult, and may even be impossible due to privacy concerns. However, it is often feasible to obtain alternative feedback such as pairwise comparisons. This thesis proposes PairBoost – a gradient boosted binary classification algorithm capable of learning from pairwise comparisons and unlabeled data. Specifically, we consider instance pairs with labels indicating which of the two instances is more likely to be positive. Our algorithm consists of two decoupled steps: first, learn a pairwise ranker to transfer the knowledge from pairwise comparisons to unlabeled instances, and second, learn a boosted binary classifier where the labels are adaptively assigned based on the discrepancy between the current classifier's predictions and the confidence of the pairwise ranker. We evaluate PairBoost on several real-world datasets, showing the practical usefulness of our approach and demonstrating significant performance improvements over existing methods.

Table of Contents

List of Figures

List of Tables

Acknowledgements

This thesis is dedicated to Isabel and Ruckus, whose unwavering support made it possible.

Chapter 1

Introduction

In this chapter, we provide relevant background material and introduce the PairBoost problem setting. We begin by stating the supervised and unsupervised learning settings, then discuss limitations of each approach. Finally, we state the goal of our algorithm and describe two real-world use cases.

1.1 Supervised Learning

Machine learning is a subset of artificial intelligence describing algorithms which perform tasks without being explicitly programmed [1]. Machine learning is generally divided into three main categories: supervised learning, unsupervised learning, and reinforcement learning. Supervised learning algorithms first learn from labeled data (training data) and then make predictions on unlabeled data (test data) [2]. Classification refers to supervised learning with categorical labels, while regression refers to supervised learning with continuous labels. Formally, given feature vectors $x \in \mathcal{X}$ and scalar labels $y \in \mathcal{Y}$, the goal of supervised learning is to fit a model:

$$f : \mathcal{X} \mapsto \mathcal{Y}$$

This is typically accomplished by minimizing a cost function which represents the model's prediction error with respect to its parameters. In addition, a regularization term is often included to penalize model complexity. This prevents overfitting to the training data and allows the model to generalize to new data. Commonly used cost functions include mean-squared error for regression and logistic loss for classification [2]:

$$\frac{1}{m}\sum_{i=1}^{m}(y_i - f(x_i))^2$$

$$\frac{1}{m}\sum_{i=1}^{m} -y_i \log(f(x_i)) - (1-y_i)\log(1-f(x_i))$$

Examples of supervised learning include predicting cancer diagnoses from radiology scans, predicting if an email will be spam given its content, and predicting how much a customer will spend when online shopping given their past spending habits. In all of these examples, a supervised learning algorithm learns from labeled historical data (i.e. radiology scans of past patients known to have cancer or be cancer-free) to make predictions on future, unlabeled data (i.e. new patients). The use of labeled data is the defining characteristic of supervised learning.

1.2 Unsupervised Learning

This contrasts with unsupervised learning, in which the goal is to learn structure from unlabeled data $x \in \mathcal{X}$ [3]. Unsupervised learning encompasses a variety of settings including clustering, dimensionality reduction, and synthetic data generation. An example of unsupervised learning is grouping online shopping users together based on similar purchasing habits in order to make more accurate targeted recommendations. In this setting, the goal is to learn the underlying structure of online user data in order to understand the similarities and differences between types of users.

PairBoost, our proposed algorithm, makes use of ideas from supervised and unsupervised learning to handle both pairwise comparisons and unlabeled data.

1.3 Data Limitations

Supervised learning can be applied to a huge number of use cases, ranging from medicine to security to advertising. However, gathering labeled data can challenging due to monetary constraints, time constraints, privacy concerns, and more. In these situations, it is often possible to gather unlabeled data or alternative feedback. Using unlabeled data and unsupervised methods alone often results in poor performance for supervised learning tasks. In such settings, alternative feedback (such as comparisons) can help. For example, subjects in a survey may be hesitant to explicitly state their political or religious beliefs, but may be open to more implicit questions such as “Who do you share similar beliefs with?” or “Do you agree more with statement A or statement B?”. Learning from alternative feedback presents unique challenges, as typical supervised learning algorithms are unusable in such settings.

1.4 Goal and Use Cases

In this thesis, we investigate the binary classification setting with one form of alternative feedback – pairwise comparisons. Namely, we consider instance pairs x_i^1, x_i^2 with label y_i^p indicating which of the two instances is more likely to be positive. Our proposed algorithm learns a classifier from both pairwise comparison data as well as unlabeled data, as illustrated in Figure 1.1. Knowledge of a class prior π_+, the proportion of true positives in the data, can also be used to improve performance in some settings.

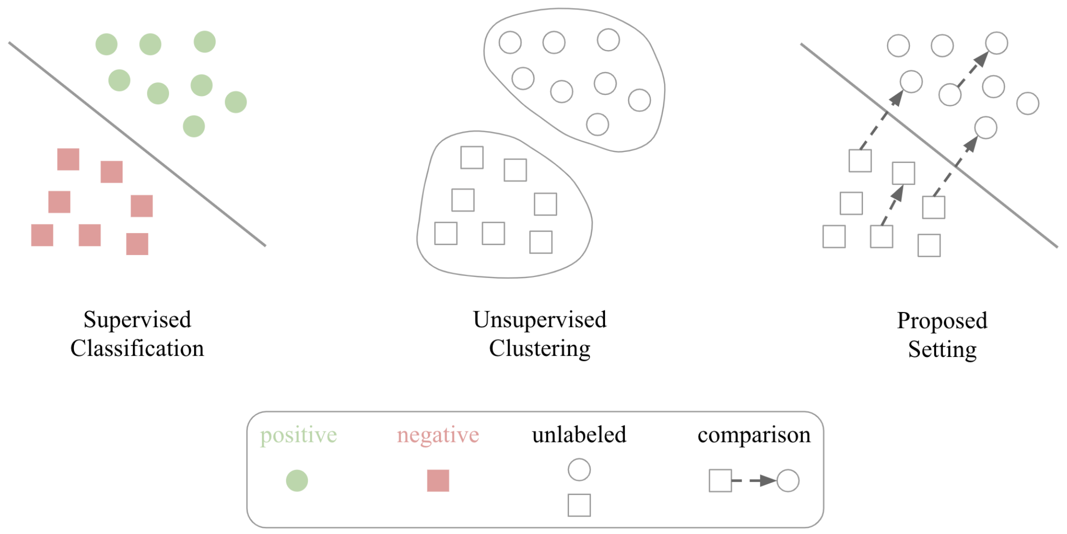


Figure 1.1: Illustration of our proposed setting versus typical supervised and unsupervised settings. Arrows point to the instance in each pair which is more likely to be positive.

Listed below are example use cases and descriptions of each data type:

- Predicting political beliefs
  - Pairwise comparisons: voter x_i^1 is more likely than voter x_i^2 to support a given candidate
  - Unlabeled data: voter demographic information
  - Class prior (optional): actual or predicted proportion of votes for given candidate
- Predicting medical diagnoses
  - Pairwise comparisons: patient x_i^1 is more likely than patient x_i^2 to have a given disease
  - Unlabeled data: patient medical records
  - Class prior (optional): estimated proportion of general population with given disease

Note that the features should have the same format for both the pairwise comparison data as well as the unlabeled data. These example use cases illustrate how gathering pairwise comparison labels can be much easier than gathering true labels. In the medical diagnosis example, it may be easier for a doctor to determine which of two patients is more likely to have a given disease as opposed to definitively diagnosing both patients.

In addition, the use of a class prior may only be appropriate in some settings. Using a class prior makes sense when the proportion of positives in the data is known or can be estimated, yet the true label for each individual instance is unknown. This setting is similar to that of uncoupled regression [4], in which both unlabeled data and labels are given, but the correspondence between them is unknown. When the learning from a class imbalanced dataset, including class prior information can significantly improve performance. These findings are discussed in greater detail in Chapter 3 and Chapter 4.

Chapter 2

Related Works

In this chapter, we discuss previous research done on related topics. First, we discuss approaches to learn from limited labeled data and alternative feedback. Next, we discuss techniques used in PairBoost, including gradient boosting and ranking.

2.1 Active Learning

A variety of methods have been proposed to perform classification or regression with little to no labeled training data. One category of related work trains classifiers or regressors using a combination of pairwise comparisons and labeled data. This is often framed as an active learning problem with comparison and/or labelling oracles. These works show that the addition of pairwise comparisons improves model performance [5] and in some settings allows learning with exponentially fewer labels [6, 7, 8, 9].

[5] considers the classification setting with limited labeled data and similar/dissimilar pairwise constraints. (In this context, similar pairs have the same true labels, while dissimilar pairs do not.) These constraints are incorporated within a maximum-margin framework to improve performance, implemented as a multi-class pairwise comparison support vector machine. [6] and [8] tackle a related setting with relative pairwise comparisons rather than similar/dissimilar pairs. These work shows that pairwise comparisons can be used to rank instances, then direct labels can be queried using binary search in order to determine a classification threshold. This reduces the direct label sample complexity to the logarithm of the desired error rate. [9] extends this work to "Reliable and Probably Useful" (RPU) learning (sometimes called "perfect selective classification"), showing that the addition of pairwise comparisons makes RPU learning feasible. The pairwise comparison and direct label setting in regression is discussed in [7]. It is shown that the use of pairwise comparisons removes the exponential relationship between a model's error rate and the dimensionality of the data, avoiding the infamous "curse of dimensionality". Unlike our proposed algorithm, the focus of these works is to supplement labeled data with pairwise comparisons rather than to learn from pairwise comparisons alone.

2.2 Pairwise Comparisons

An additional category of approaches is to use an unsupervised model with additional constraints. Such approaches include constrained K-means clustering [10], classification from spectral clustering [11], and information-theoretic metric learning [12]. All approaches take advantage of similar/dissimilar pairwise information to improve performance. Constrained K-means clustering uses similar/dissimilar pairs as must-link and cannot-link constraints during each iteration of cluster assignment [10]. Alternatively, spectral clustering groups data based on connectivity rather than compactness. This is accomplished by clustering after performing dimensionality reduction, then using pairwise constraints for affinity propagation [11]. Finally, information-theoretic metric learning learns a distance function under the constraint that similar instances must be sufficiently close together and dissimilar instance must be sufficiently far apart [12]. These methods are limited given their reliance on assumptions such as the cluster hypothesis and manifold assumption.

Other methods focus on learning explicitly from pairwise comparisons rather than modifying unsupervised techniques. [13] learns a classifier from similar pairs and unlabeled data, while [14] and [15] also incorporate dissimilar pairs in training data. These methods express classification risk using only similar, dissimilar, and unlabeled data, then learn a model through empirical risk minimization. [13] and [14] derive model error estimates and convergence rates, and provide implementations of several pairwise loss functions. Experiments on benchmark datasets reveal that similar-dissimilar-unlabeled classification outperforms existing methods when the data has a class imbalance. The problem setting in [14] is the most closely related to that of our proposed algorithm, with training data including both similar and dissimilar pairs as well as unlabeled data. However, while [14] requires pairs to be either definitely similar or definitely dissimilar, we only require relative pairwise comparisons indicating which of two instances is more likely to be positive.

2.3 Gradient Boosting

Many machine learning algorithms optimize a cost function using gradient descent [16]. This involves taking the gradient of a cost function with respect to model parameters in order to determine the direction of steepest decrease, then adjusting the parameters accordingly. For a model with parameters w, cost function C, and learning rate α, this process is described by:

$$w := w - \alpha \nabla C(w)$$

When properly tuned, gradient descent converges to a local minimum of $C(w)$. Gradient descent optimizes a cost function with respect to model parameters, though it is also possible to optimize a cost function with respect to model predictions. This is the technique used in gradient boosting [17]. Here, we optimize over a function space, and learn a new model at each iteration of gradient descent. The first model is trained on the original data, while each successive model is trained on the gradient of the current model's predictions. This process is outlined below:

$$\begin{gathered}
\text{Train } f_1 \text{ on } \{(x_1, y_1)...(x_m, y_m)\} \\
\hat{y}^{(1)} = f_1(x) \\
z^{(1)} = \nabla C(\hat{y}^{(1)}) \\
\text{Train } f_2 \text{ on } \{(x_1, z_1)...(x_m, z_m)\} \\
\hat{y}^{(2)} = f_1(x) + f_2(x) \\
z^{(2)} = \nabla C(\hat{y}^{(2)}) \\
\vdots \\
\text{Final model } f(x) = f_1(x) + f_2(x) + \ldots + f_t(x)
\end{gathered}$$

Gradient boosting is an ensemble method. Ensemble methods often outperform individual models, as combining different models (each with some degree of randomness) adds stability to predictions. Gradient boosting typically makes use of nonlinear models, as the final model is represented as a linear combination. Decision trees [18] are often used due to their predictive power and low time complexity. PairBoost uses gradient boosting to adaptively assign labels during training.

2.4 Learning to Rank

Both steps of our proposed algorithm rely on approaches used in learning-to-rank. The learning-to-rank problem setting is as follows. Given a set of instances and some partial ordering defined between instances, we seek to learn a model to optimally order new instances [19]. Partial orderings could include pointwise scores indicating the quality of each instance, pairwise comparisons between instances, or binary labels indicating if an instance is relevant. The model's ordering is represented as a permutation of instances.

The first step of PairBoost trains a simple pairwise ranker, while the second step computes weights similarly to that of LambdaMART [20, 21]. Given cost function C, ranking model f, and score $s_i = f(x_i)$, [20] defines:

$$\lambda_{i,j} \equiv \frac{\partial C(s_i - s_j)}{\partial s_i} \tag{2.1}$$

This $\lambda_{i,j}$ can be thought of as a pairwise gradient between two instances in a ranking. In addition, for each set of index pairs $\{i, j\} \in I$ such that x_i and x_j should be ranked differently, [20] introduces:

$$\lambda_i = \sum_{j:\{i,j\}\in I} \lambda_{i,j} - \sum_{j:\{j,i\}\in I} \lambda_{i,j} \tag{2.2}$$

Here, λ_i indicates how instance x_i should be moved in a given ranking. Its sign indicates direction, while its magnitude indicates how much it should move. These ideas are similar those used to order instances in PairBoost. The weight w_i described in Section 3.5 is analogous to λ_i discussed above. Additionally, gradient boosted decision trees are particularly apt for this setting, as used in LambdaMART.

Chapter 3

Proposed Algorithm

In this chapter, we formally describe our problem setting and proposed algorithm. We provide details regarding PairBoost's inputs, outputs, and learning process. Finally, we discuss how each of the two main steps of our algorithm (knowledge transfer and label estimation) were derived.

3.1 Problem Setting

PairBoost is a gradient boosted binary classification algorithm capable of learning from pairwise comparisons and unlabeled data. The input features are represented as a set of vectors $\mathcal{X} = \{x \in \mathbb{R}^d\}$ while the true labels are represented as $\mathcal{Y} = \{+1, -1\}$ indicating instances from the positive and negative classes. Pairwise comparisons are represented by the set $\mathcal{P} = \{(x_i^1, x_i^2, y_i^p)\}_{i=1}^m$. Here, x_i^1, x_i^2 is a pair of instances with pairwise label:

$$y_i^p = \begin{cases} +1 & \text{if } x_i^1 \text{ is more likely to be positive than } x_i^2 \\ -1 & \text{if } x_i^2 \text{ is more likely to be positive than } x_i^1 \end{cases} \tag{3.1}$$

Note that only pairwise labels, not true labels, are available to PairBoost at training. Unlabeled data is represented as the set $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$ which includes all instances in $\mathcal{P}$ as well as instances with no pairwise labels. Knowledge of a class prior $\pi_+ = p(y = +1)$, indicating the proportion of true positives in the data, can be used to improve performance in some settings. Given pairwise labeled data, unlabeled data, and optional class prior information, PairBoost learns a function $f : \mathcal{X} \mapsto \mathcal{Y}$ which map features to predicted true labels.

3.2 Overview

Our algorithm consists of two decoupled steps. First, a pairwise ranker is learned to transfer the knowledge from pairwise comparisons to unlabeled instances. Second, a gradient boosted binary classifier is learned, with labels adaptively assigned based on the the discrepancy between the current classifier's predictions and the confidence of the pairwise ranker. This second step is accomplished by learning an ordering of the training instances from most likely to be negative to most likely to be positive, then determining a classification threshold. This process is illustrated by Figure 3.1 and described in further detail in Sections 3.4 and 3.5.

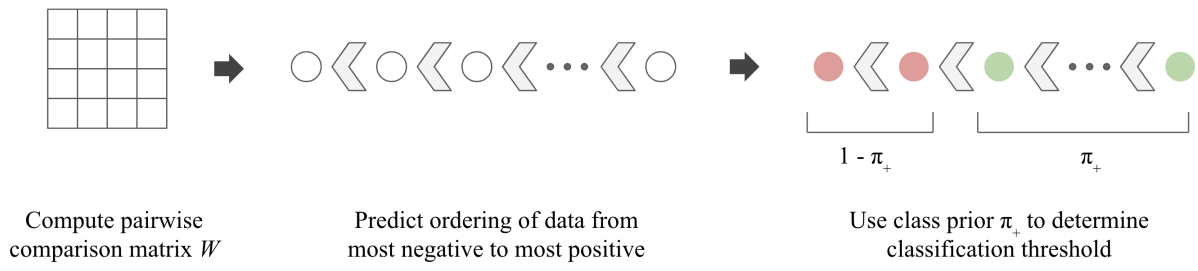

1 - π+
π+
Compute pairwise comparison matrix W
Predict ordering of data from most negative to most positive
Use class prior π+ to determine classification threshold

Figure 3.1: Illustration of the training and prediction process in PairBoost

3.3 Description

Algorithm 1 `PairBoost`

```
1: inputs: $\mathcal{P} = \{(x_i^1, x_i^2, y_i^p)\}_{i=1}^m$, $\mathcal{S} = \{x_1, x_2, \ldots, x_n\}$, $\pi_+$
2: output: Binary classifier $f : \mathcal{X} \mapsto \mathcal{Y}$
3: Learn a pairwise ranker $r$ by training a supervised probabilistic classifier on $\mathcal{P}$
4: Compute $W_{n \times n}$ using predictions from $r$ over all instance pairs:
       $$W_{i,j} = P(y_{i,j}^p = +1 | (x_i, x_j))$$
5: Initialize $f(x) = 0$ for all instances, $t = 1$, and $\alpha_1 = 0$
6: while $t \leq T$ and $\alpha_t \geq 0$ do
7:     Compute $\xi_{i,j}$ for each instance pair using Eq. (3.5)
8:     Compute the weight of each instance $w_i$ using Eq. (3.6)
9:     if $\pi_+$ is known then
10:        Sort instances by weight $w_i$ in ascending order
11:        Assign label $y_i = +1$ to top $\pi_+$ proportion of weights
12:        Assign label $y_i = -1$ to bottom $1 - \pi_+$ proportion of weights
13:    else
14:        Assign labels using weights:
               $$y_i = \text{sign}(w_i)$$
15:    Create training data $\mathcal{D}_t = \{(x_i, y_i), \ldots, (x_n, y_n)\}$ by sampling based on $|w_i|$
16:    Learn a binary classifier $f_t$ minimizing the weighted empirical loss over $\mathcal{D}_t$
17:    Predict the label of each instance $\hat{y}_i = f_t(x_i)$
18:    Compute the weight of classifier $f_t$ as:
           $$\alpha_t = \frac{1}{2} \log \frac{\sum_{i,j=1}^n \xi_{i,j} \mathbb{I}[\hat{y}_i = 1] \mathbb{I}[\hat{y}_j = -1]}{\sum_{i,j=1}^n \xi_{i,j} \mathbb{I}[\hat{y}_i = -1] \mathbb{I}[\hat{y}_j = 1]}$$
19:    Update the final classifier:
           $$f(x) \leftarrow f(x) + \alpha_t f_t(x)$$
20:    end while
21:    return $f(x)$
```

3.4 Knowledge Transfer to Unlabeled Instances

The first part of the algorithm transfers knowledge from labeled instance pairs to unlabeled instances in order to utilize all of the available data for training. This is accomplished by learning a supervised model on pairwise data $\mathcal{P}$. Next, this supervised model is used to make probabilistic predictions for all $n \times n$ pairs in order to fill in the W matrix, with entry $W_{i,j}$ indicating the predicted probability that instance x_i is more likely to be positive than instance x_j. This corresponds to steps 3 and 4 in Algorithm 1. These steps are purposefully left open ended, and allow PairBoost to be model-agnostic. This means that these steps could be performed with any model (support vector machine, neural network, etc.) as long as the model includes functionality to make probabilistic predictions. Additionally, this allows for a variety of pairwise feature representations (x_i, x_j concatenated, $x_i - x_j$ elementwise difference, etc.).

3.5 Boosting with Adaptive Label Estimation

Next, we attempt to learn an ordering of the training data which is consistent with the information in matrix W. To do so, we introduce cost C as a function of the model f:

$$C(f) = \sum_{i,j=1}^{n} W_{i,j} \mathbb{I}[f(x_j) > f(x_i)] \tag{3.2}$$

where $\mathbb{I}$ is the indicator function. This essentially sums the probability that the model is incorrect over each pair of instances. However, this function is difficult to optimize as it is step-like and discontinuous. To fix this problem, we replace the indicator function with a smooth surrogate such as exponential loss, and the resulting function becomes:

$$C(f) = \sum_{i,j=1}^{n} W_{i,j} \exp(f(x_j) - f(x_i)) \tag{3.3}$$

To optimize this function, we first compute its gradient. This gives us a direction to adjust the model's weights through gradient descent. For a linear model $f(x) = w^\top x$ and exponential loss function, the gradient with respect to the model's weights w is:

$$\nabla_w C(w) = \sum_{i,j=1}^{n} W_{i,j} \exp(w^\top x_j - w^\top x_i)(x_j - x_i) \tag{3.4}$$

However, what if the model is nonlinear? To handle different types of models, we need a more general approach. Instead of taking the gradient of the cost function with respect to the model's weights, we can take the derivative of the cost function with respect to the model's pairwise predictions. Formally, this quantity is $\frac{\partial C}{\partial s_{i,j}}$ with pairwise prediction $s_{i,j} = f(x_j) - f(x_i)$. This is easy to calculate because the derivative of the exponential function is itself. For a single instance pair x_i, x_j, we get the derivative:

$$\xi_{i,j} = W_{i,j} \exp(f(x_j) - f(x_i)) \tag{3.5}$$

This quantity represents the discrepancy between the current model and the learned ranker for a single instance pair. Next, we compute the importance of each instance x_i by summing over pairs:

$$w_i = \sum_{j=1}^{n} \xi_{i,j} - \xi_{j,i} \tag{3.6}$$

These weights w_i indicate how each instance should move in the predicted ordering. Much like λ_i in LambdaMART [20], the sign of w_i determines if x_i should move up or down in the ordering, while the magnitude of w_i determines how much x_i should move. Weights w_i are used to adaptively assign labels at each iteration of gradient boosting. This corrects for errors previously made by the model, reducing the discrepancy with the learned pairwise ranker.

At each iteration, a new binary classifier is trained to predict this discrepancy, thus improving the predicted ordering. (This is step 16 in Algorithm 1 – note that this step is model-agnostic similar to step 3.) Each classifier is weighted based on its performance and added to the final model. This process learns an instance ordering, but cannot perform classification without a threshold to make predictions. This is where the class prior π_+ comes into play. If π_+ is known, its value can be used to determine a threshold as illustrated in Figure 3.1. If π_+ is unknown, a threshold is estimated based on the sign of the weights. Access to π_+ generally improves performance when the data has a large class imbalance (corresponding to π_+ close to either 0 or 1), though PairBoost often performs well on balanced data even when π_+ is unknown. These findings are discussed in greater detail in Chapter 4.

Chapter 4

Empirical Results

In this chapter, we evaluate the performance of PairBoost on real-world data. We use nine binary classification datasets from the UCI Machine Learning Repository [22] and LIBSVM [23]. These datasets are commonly used to benchmark machine learning algorithms. We analyze PairBoost performance against supervised and unsupervised baselines as well as current state-of-the-art pairwise methods across several settings.

4.1 Baseline Methods

We use K-Means clustering [24] as a simple unsupervised baseline. K-Means clustering is an iterative, two step algorithm which assigns instances to clusters based their distance to mean centroid points. As a supervised baseline, we consider XGBoost [25]. XGBoost is an ensemble method which uses parallelized gradient boosted decision trees [18]. PairBoost is compared to XGBoost because of their similar structures.

As for pairwise methods, we compare PairBoost to SDU classification [14] which which learns from similar/dissimilar pairwise data and unlabeled data. As discussed in Section 2.2, this problem setting is most closely related to that of our proposed method. The SDU classifier described in [14] is a linear models with weights learned through empirical risk minimization. Specifically, the similar-dissimilar and dissimilar-unlabeled (SDDU) risk formulation is shown to achieve state-of-the-art classification performance, thus we compare PairBoost to this implementation.

4.2 Setting

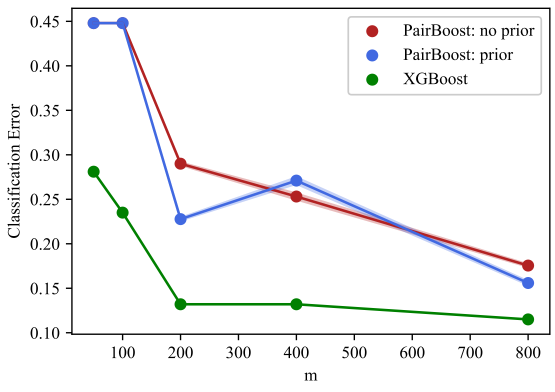
In all experiments, we implement PairBoost using decision trees [18] (step 16, Algorithm 1) with a maximum depth of 3 for $T = 20$ iterations of boosting. For each setting, we evaluate two models of PairBoost: one with the class prior π_+ known, and one with the class prior π_+ unknown. We randomly sample m pairwise comparisons across n total instances. Performance is reported on a test set disjoint from the training set. Our experiments seek to answer three key questions:

1. How is performance affected by the number of pairwise comparisons m?
2. How does our proposed method compare to standard supervised and unsupervised baselines?
3. How does our proposed method compare to state-of-the art pairwise methods?

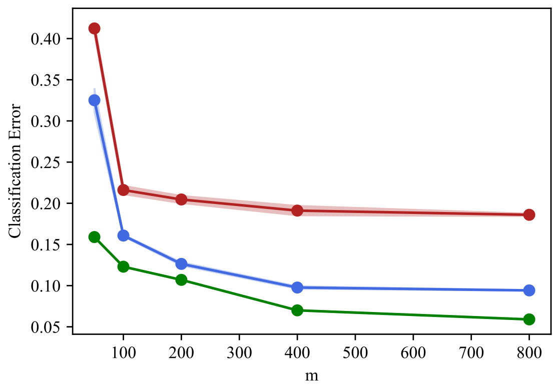
4.3 Improvement by Pairwise Comparisons

We answer the first question through Figure 4.1 by plotting performance as m varies. Performance generally improves as m increases, though it levels off for larger values. Performance approaches that of supervised baseline XGBoost [25].

For the banana dataset, PairBoost performance is similar with and without class prior π_+ known. For the cod-rna dataset, the model with π_+ known clearly outperforms the model without. We hypothesize that this is because π_+ is closer to 0.5 for the banana dataset, while the cod-rna dataset is more heavily class-imbalanced. The value of π_+ and its effect on performance is presented in Table 4.1 and discussed in further in Section 4.4.



(a) banana



(b) cod-rna

Figure 4.1: Mean classification error versus the number of pairwise comparisons m over 10 trials. The number of additional unlabeled instances is fixed at 500.

4.4 Performance versus Baseline Methods

		PairBoost		SDDU		Baselines	
Dataset	m	π_+ unknown	π_+ known	Squared	Double-Hinge	K-Means	XGBoost
adult	50	65.0 (0.0)	**79.4 (0.0)**	61.9 (0.9)	**77.7 (0.6)**	69.4 (0.0)	77.4 (0.0)
$\pi_+ = 24.0$	200	66.6 (0.0)	**82.2 (0.0)**	71.4 (0.7)	**82.5 (0.3)**	69.4 (0.0)	78.6 (0.0)
banana	50	**68.3 (0.1)**	67.2 (0.3)	63.9 (1.2)	63.5 (1.1)	54.4 (0.0)	72.4 (0.0)
$\pi_+ = 44.8$	200	**72.2 (0.1)**	**72.3 (0.1)**	66.5 (0.8)	66.9 (0.7)	54.4 (0.0)	85.2 (0.0)
codrna	50	74.1 (0.1)	**81.5 (0.2)**	**78.1 (1.1)**	68.5 (0.8)	70.6 (0.0)	83.2 (0.0)
$\pi_+ = 33.3$	200	76.4 (0.2)	**86.6 (0.0)**	**87.7 (0.6)**	72.7 (0.7)	70.8 (0.0)	90.2 (0.0)
ijcnn1	50	77.6 (0.1)	**88.5 (0.1)**	64.7 (0.8)	68.8 (0.9)	73.7 (0.8)	89.0 (0.0)
$\pi_+ = 9.50$	200	78.4 (0.1)	**90.8 (0.1)**	75.1 (0.7)	76.3 (0.5)	72.2 (0.9)	92.6 (0.0)
magic	50	73.3 (0.1)	**74.3 (0.0)**	65.5 (0.9)	65.1 (1.0)	61.3 (0.0)	77.4 (0.0)
$\pi_+ = 35.2$	200	**72.5 (0.1)**	**74.8 (0.1)**	**73.0 (0.6)**	**71.4 (0.7)**	60.0 (0.0)	82.6 (0.0)
phishing	50	**86.9 (0.0)**	**86.7 (0.1)**	69.4 (0.8)	80.5 (0.9)	56.9 (0.3)	89.2 (0.0)
$\pi_+ = 55.7$	200	**89.0 (0.0)**	88.6 (0.1)	81.7 (0.7)	87.0 (0.4)	52.9 (0.0)	90.2 (0.0)
phoneme	50	63.7 (0.1)	**73.1 (0.1)**	67.9 (0.9)	69.2 (0.9)	76.8 (0.0)	75.6 (0.0)
$\pi_+ = 29.3$	200	69.9 (0.1)	**77.6 (0.1)**	73.5 (0.5)	74.4 (0.4)	76.2 (0.0)	81.6 (0.0)
spambase	50	**84.4 (0.1)**	**84.2 (0.1)**	66.7 (0.8)	**82.9 (0.6)**	80.2 (0.0)	89.2 (0.0)
$\pi_+ = 39.4$	200	**86.7 (0.2)**	**87.6 (0.2)**	77.9 (0.7)	**87.5 (0.3)**	80.3 (0.0)	92.2 (0.0)
w8a	50	59.3 (0.0)	**97.8 (0.0)**	60.8 (0.9)	73.2 (0.9)	78.3 (0.1)	97.4 (0.0)
$\pi_+ = 3.30$	200	63.8 (0.2)	**97.8 (0.0)**	64.1 (0.7)	80.2 (0.7)	78.8 (0.0)	97.6 (0.0)

Table 4.1: Performance of PairBoost versus competing methods. Mean classification accuracy and standard error are listed for each setting over 20 trials. Significantly outperforming methods are bolded, computed using a t-test with $\alpha = 0.05$ across the PairBoost and SDDU columns.

Next, we compare performance against supervised and unsupervised baselines as well as state-of-the-art pairwise methods. K-means clustering [24] is used as a simple unsupervised baseline, while XGBoost [25] is again used as a supervised baseline. SDU clasification [14] with similar-dissimilar and dissimilar-unlabeled (SDDU) risk is used as a pairwise baseline.

We evaluate performance with $m = \{50, 200\}$ randomly sampled pairwise comparisons and 500 additional unlabeled instances. This is almost identical to the setting in [14], however the SDDU results are reported with a fixed class prior $\pi_+ = 0.7$. (This is because SDU classification performs poorly when the class prior is near 0.5.) Since the author's exact implementation is unavailable, we compare our results to those published in [14] in the SDDU column. The results are listed in Table 4.1.

PairBoost significantly outperforms or matches SDDU classification on all nine datasets tested. When π_+ is known, PairBoost outperforms K-means clustering and approaches XGBoost accuracy on most datasets. Additionally, the difference in PairBoost performance when π_+ is known versus unknown is greatest for datasets with skewed π_+ values (i.e. π_+ near 0 or 1). For datasets with balanced π_+ values (i.e. π_+ near 0.5), the performance difference is generally small.

Chapter 5

Conclusion

5.1 Summary

We introduced PairBoost, a novel gradient boosted binary classification algorithm capable of learning from pairwise comparisons and unlabeled data. We discussed how our algorithm differs from existing methods in its ability to learn from relative pairwise comparisons rather than requiring supplemental labeled data or similar/dissimilar pairs. We derived PairBoost using ideas from learning-to-rank and gradient boosting, and described its implementation in detail. Finally, we showed that our approach significantly outperforms existing pairwise methods through experiments on real-world datasets.

5.2 Future Work

We described PairBoost and evaluated its performance on benchmark classification datasets. However, we did not focus on specific use cases within the pairwise setting. Perhaps our approach will be well suited for situations in which user privacy must be preserved. Or maybe PairBoost will be particularly useful for applications involving survey data with preference relations. It would be interesting to explore such use cases in future works. Additionally, we did not complete a theoretical analysis of PairBoost. Understanding our method's sample complexity and error bounds (particularly under noisy data) could be valuable when considering potential applications.

Chapter 6

Appendix

6.1 Experiment Details

Unless otherwise specified, default hyperparameter values were used in PairBoost decision tree submodels, K-Means clustering, and XGBoost as described in [25, 26]. When computing performance for K-Means clustering, we set K = 2 and took $\max(a, 1-a)$ with a being classification accuracy on out-of-sample test data.

6.2 Simulated Pairwise Setting

We faced a unique challenge when conducting our experiments, as we sought to simulate the pairwise comparison setting using fully labeled datasets. Given labeled data in a typical supervised setting:

$$\{(x_i, y_i)\}_{i=1}^n$$

with true label $y_i = \pm 1$ indicating that instance x_i belongs to the positive or negative class, we needed some way to get pairwise comparisons of the form:

$$\{(x_i^1, x_i^2, y_i^p)\}_{i=1}^m$$

with pairwise label $y_i^p = \pm 1$ indicating if instance x_i^1 is more likely to be positive than instance x_i^2. An obvious solution would be to use dissimilar pairs in which one instance's true label is positive and the other instance's true label is negative.

However, this approach is flawed in that it is essentially the same as the supervised setting. Any instance x_i^1 with a positive pairwise label clearly has a positive true label, and any instance x_i^1 with a negative pairwise label clearly has a negative true label (the opposite is true for x_i^2). A simple algorithm could easily recover true labels when this approach is used – in fact, PairBoost reduces to a supervised learning algorithm in this context.

Additionally, this approach ignores an extremely important consideration. Namely, this approach does not include *same-label comparisons*, referring to pairwise comparisons in which one instance is more likely to be positive than another, though both instances have the same true label. Including same-label comparisons differentiates the pairwise setting from the typical supervised setting.

To incorporate same-label comparisons, we used a less obvious approach. We started by training a supervised, probabilistic classifier on fully labeled data $\{(x_i, y_i)\}_{i=1}^n$. Next, we randomly selected m pairs of instances and used our classifier to make probabilistic predictions of the form:

$$\hat{y}_i = P(y_i = +1 | x_i)$$

Then, we compared $\hat{y}$ values across instance pairs and assigned pairwise labels based on which instance was more likely to be positive:

$$y_i^p = \begin{cases} +1 & \text{if } \hat{y_i}^1 > \hat{y_i}^2 \\ -1 & \text{if } \hat{y_i}^1 < \hat{y_i}^2 \end{cases}$$

This approach incorporates both same-label and different-label comparisons. The proportion of each type of comparison is determined by the class prior π_+. This distinction is very important in order to accurately simulate how PairBoost would operate in real-world applications.

6.3 Implementation in Python

```
""" PairBoost Implementation """

import numpy as np
from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.datasets import load_svmlight_file


###########################################################################
# Simulate Pairwise Setting
###########################################################################
def all_diff_pairs(X_train, y_train):
    """ Get every pair and true pairwise label from original data """

    n = X_train.shape[0]

    n_rows = (n * (n + 1)) // 2
    n_cols = 3 * X_train.shape[1]

    X_diff = np.full(shape=(n_rows, n_cols), fill_value=np.NaN)
    y_diff = np.full(shape=n_rows, fill_value=np.NaN)

    k = 0

    for i in range(n):

        for j in range(n):

            X_diff[k] = np.hstack((X_train[i], X_train[j],
                                   X_train[i] - X_train[j]))

            # Can't use same label comparisons at this point
            if y_train[i] == y_train[j]:
                continue

            elif y_train[i] > y_train[j]:
                y_diff[k] = 1.0

            elif y_train[i] < y_train[j]:
                y_diff[k] = -1.0

            k += 1

    # Cut NaN rows
    return X_diff[~np.isnan(y_diff)], y_diff[~np.isnan(y_diff)]
```

```
def predict_pair_labels(X_train, y_train, X_diff, y_diff, m,
                        random_seed=None):
    """ Use XGBoost to predict pairwise labels """

    np.random.seed(random_seed)

    # Randomly choose m pairs to predict labels
    idx_pair = np.hstack((
                np.random.choice(X_train.shape[0], m).reshape(-1, 1),
                np.random.choice(X_train.shape[0], m).reshape(-1, 1)
                ))

    # Get features for predictions
    X_pair = np.full(shape=(m, 3 * X_train.shape[1]), fill_value=np.NaN)

    k = 0

    for (i, j) in idx_pair:

        X_pair[k] = np.hstack((X_train[i], X_train[j],
                               X_train[i] - X_train[j]))

        k += 1

    # Learn XGBoost on all pairs
    xgb = XGBClassifier(n_estimators=30)
    xgb.fit(X_diff, y_diff)
    print('PRE-PAIRBOOST')
    print('Accuracy on train set: %.3f\n' % xgb.score(X_diff, y_diff))

    # Make predictions
    y_pair = xgb.predict(X_pair)

    return X_pair, y_pair


###########################################################################
# Helper Functions for PairBoost
###########################################################################
def cal_uncertainty(y, W):
    """
    Computes uncertaintity matrix xi
    """
    pair_dist = np.exp(-y).dot(np.exp(y).T)
    return W * pair_dist


def cal_weights(xi):
    """
    Computes importance (weight) of each instance, wi
    """
    return np.sum(xi - xi.T, axis=1)
```

```
def cal_alpha(y, xi):
    """
    Calculates the weight of classifier t, alpha
    """
    y_p = (y > 0).astype(float).reshape(-1, 1)
    y_n = (y < 0).astype(float).reshape(-1, 1)
    I_1 = xi * y_p.dot(y_n.T)
    I_2 = xi * y_n.dot(y_p.T)
    return 0.5 * np.log((np.sum(I_1) + 1e-6) / (np.sum(I_2) + 1e-6))


###########################################################################
# Full PairBoost Algorithm
###########################################################################
def pairboost(X_train, y_train, X_test, y_test, X_pair, y_pair, prior,
              T=20, sample_prop=1, max_depth=3, random_seed=None,
              verbose=True):
    """ Learns PairBoost classifier with decision tree submodels """

    if verbose:

        if prior:
            print('WITH CLASS PRIOR %.2f' % prior)
        else:
            print('NO CLASS PRIOR GIVEN')

        if random_seed:
            print('Random seed %d', random_seed)

        print('Using %d classifiers and sample proportion of %d'
              % (T, sample_prop))

    # Constants
    m = X_train.shape[0]
    n = X_train.shape[1]
    np.random.seed(random_seed)

    # Step 3: learn a pairwise ranker
    xgb = XGBClassifier(n_estimators=30)
    xgb.fit(X_pair, y_pair)

    # Step 4: compute W using probabilistic predictions
    W = np.full(shape=(m, m), fill_value=np.nan)

    for i in range(m):

        for j in range(i, m):

            X_pred = np.hstack((X_train[i], X_train[j],
                                X_train[i] - X_train[j]))
            y_pred = xgb.predict_proba(X_pred.reshape(1, -1))[:, 1]
            # 'Mirror' predictions across diagonal
            W[i, j] = y_pred
            W[j, i] = 1 - y_pred

        # Set diagonal
        W[i, i] = 0.5

    # Step 5: initialize counters
    t = 1
    alpha_t = 0
    acc_train_ls = []
    acc_test_ls  = []

    # Instantiate models and weights
    f = []
    alpha = []

    # Step 6: training
    while t <= T and alpha_t >= 0:

        # Step 7: compute xi
        if t == 1:
            curr_pred = np.zeros(y_train.shape)
        else:
            curr_pred = sum([alpha[i] * f[i].predict_proba(X_train)[:, 1]
                             for i in range(t - 1)])

        xi = cal_uncertainty(curr_pred, W)

        # Step 8: compute weights
        weight = cal_weights(xi)

        # Steps 9 - 14: extract labels
        weight_idx = np.argsort(weight)
        weight = np.sort(weight)

        X_new = X_train[weight_idx]

        if prior:
            y_new = np.concatenate((-np.ones(m - int(m * prior)),
                                    np.ones(int(m * prior))))
        else:
            y_new = np.sign(weight)

        # Step 15: create training (sample) data by sampling based on weights
        p_weight = np.abs(weight)
        p_weight /= np.sum(p_weight)
        sample = np.random.choice(m, size=m*sample_prop,
                                  replace=True, p=p_weight)
        X_sample = X_new[sample]
        y_sample = y_new[sample]

        # Step 16: learn binary classifier on training (sample) data
        clf = DecisionTreeClassifier(max_depth=max_depth)
        clf.fit(X_sample, y_sample)

        # Step 17: predict labels using current classifier
        y_prob = clf.predict_proba(X_train)[:, 1]
        y_pred = np.round(y_prob) * 2 - 1

        # Step 18: compute weight of current classifier
        xi_t = cal_uncertainty(y_prob, W)
        alpha_t = cal_alpha(y_pred, xi_t)

        # Make sure alpha is valid
        if np.isnan(alpha_t) or np.isinf(alpha_t):
            print('Alpha invalid, terminated')
            break

        # Step 19: update final classifier, iteration
        f.append(clf)
        alpha.append(alpha_t)
        t += 1

        # Evaluation
        y_train_pred = sum(
                    [(alpha[i] * f[i].predict_proba(X_train)[:, 1] * 2 - 1)
                     for i in range(t - 1)]
                     )
        y_test_pred = sum(
                    [(alpha[i] * f[i].predict_proba(X_test)[:, 1] * 2 - 1)
                     for i in range(t - 1)]
                     )
        y_train_pred = np.sign(y_train_pred)
        y_test_pred = np.sign(y_test_pred)

        acc_train_ls.append(accuracy_score(y_train, y_train_pred))
        acc_test_ls.append(accuracy_score(y_test, y_test_pred))

        if verbose:
            if t == 2:
                print('t\tPrior\t\tTrain\t\tTest')
            print('%d\t%.2f\t\t%.2f\t\t%.2f'
            % (t - 1, (np.sum(y_sample == 1) / y_sample.size),
               acc_train_ls[-1], acc_test_ls[-1]))
            if alpha_t < 0:
                print('Alpha %.2f, terminated' % alpha_t)

    # Step 20: end while loop, print summary
    if verbose:
        print('\nProportion of positive predictions: %.2f'
              % (np.sum(y_train_pred == 1) / y_train_pred.size))
        print('\nTrain: boost diff %.2f, final acc %.2f'
              % (acc_train_ls[-1] - acc_train_ls[0], acc_train_ls[-1]))
        print('Test:  boost diff %.2f, final acc %.2f\n'
              % (acc_test_ls[-1] - acc_test_ls[0], acc_test_ls[-1]))

    # Step 21: return components of classifier
    return f, alpha


############################################################################
# Example
############################################################################
if __name__ == "__main__":

    # Load adult dataset, located:
    # www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html#a5a
    X_train, y_train = load_svmlight_file('a5a.txt', n_features=123)
    X_test, y_test = load_svmlight_file('a5a.t')
    X_train, X_test = X_train.toarray(), X_test.toarray()
    X_train, y_train, X_test, y_test = (X_train[:500], y_train[:500],
                                        X_test[:500], y_test[:500])

    # Simulate pairwise setting
    X_diff, y_diff = all_diff_pairs(X_train, y_train)
    X_pair, y_pair = predict_pair_labels(X_train, y_train,
                                         X_diff, y_diff, m=200)

    # Get class prior
    prior = np.sum(y_train == 1) / y_train.size

    # Run PairBoost
    pairboost(X_train, y_train, X_test, y_test, X_pair, y_pair, prior, T=20,
              sample_prop=1, max_depth=3, random_seed=None, verbose=True)
```

Bibliography

[1] Daniel Kifer. Introduction to machine learning. *CMPSC 448: Machine Learning and AI*, January 2020.

[2] Daniel Kifer. Fundamentals of supervised learning. *CMPSC 448: Machine Learning and AI*, January 2020.

[3] Daniel Kifer. Unsupervised learning and latent variable models. *CMPSC 448: Machine Learning and AI*, April 2020.

[4] Liyuan Xu, Junya Honda, Gang Niu, and Masashi Sugiyama. Uncoupled regression from pairwise comparison data, 2019.

[5] Nam Nguyen and Rich Caruana. Improving classification with pairwise constraints: A margin-based approach. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 113–124, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[6] Yichong Xu, Hongyang Zhang, Kyle Miller, Aarti Singh, and Artur Dubrawski. Noise-tolerant interactive learning using pairwise comparisons. In *Advances in Neural Information Processing Systems*, pages 2431–2440, 2017.

[7] Yichong Xu, Hariank Muthakana, Sivaraman Balakrishnan, Aarti Singh, and Artur Dubrawski. Nonparametric regression with comparisons: Escaping the curse of dimensionality with ordinal information. *arXiv preprint arXiv:1806.03286*, 2018.

[8] Daniel M. Kane, Shachar Lovett, Shay Moran, and Jiapeng Zhang. Active classification with comparison queries, 2017.

[9] Max Hopkins, Daniel M Kane, and Shachar Lovett. The power of comparisons for actively learning linear classifiers. *arXiv preprint arXiv:1907.03816*, 2019.

[10] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schrödl. Constrained k-means clustering with background knowledge. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 577–584, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[11] Weifu Chen and Guocan Feng. Spectral clustering: A semi-supervised approach. *Neurocomput.*, 77(1):229–242, February 2012.

[12] Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, page 209–216, New York, NY, USA, 2007. Association for Computing Machinery.

[13] Han Bao, Gang Niu, and Masashi Sugiyama. Classification from pairwise similarity and unlabeled data. *arXiv preprint arXiv:1802.04381*, 2018.

[14] Takuya Shimada, Han Bao, Issei Sato, and Masashi Sugiyama. Classification from pairwise similarities/dissimilarities and unlabeled data via empirical risk minimization. *arXiv preprint arXiv:1904.11717*, 2019.

[15] Soham Dan, Han Bao, and Masashi Sugiyama. Learning from noisy similar and dissimilar data. *arXiv preprint arXiv:2002.00995*, 2020.

[16] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Ann. Math. Statist.*, 22(3):400–407, 09 1951.

[17] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Statist.*, 29(5):1189–1232, 10 2001.

[18] J. R. Quinlan. Induction of decision trees. *MACH. LEARN*, 1:81–106, 1986.

[19] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pages 129–136, 2007.

[20] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[21] Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the Learning to Rank Challenge*, pages 1–24, 2011.

[22] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[23] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[24] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[25] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Neil Ashtekar

Education

B.S. with Honors in Computer Science *University Park, PA / Aug 2016 - May 2021*
Minors in Mathematics and Statistics
The Pennsylvania State University -- Schreyer Honors College

Skills

Languages:	Python ● R ● C ● C++ ● Java ● Scala ● MATLAB
Tools:	NumPy ● Pandas ● Scikit-Learn ● TensorFlow ● Spark ● Hadoop ● SQL ● Git
General:	Research ● Data Analysis ● Artificial Intelligence ● Machine Learning ● Deep Learning Distributed Computing ● Data Structures & Algorithms ● Agile Project Management

---

Work Experience

Undergraduate Researcher, PSU Computer Science *University Park, PA / Aug 2019 - Present*
- Working with Professor Mehrdad Mahdavi on fundamental problems in machine learning
- Led projects exploring multiobjective learning, ranking, and algorithmic fairness through Pareto optimality
- Completed honors thesis on novel gradient-boosted classification algorithm for pairwise data

Data Science Intern, Apixio *San Mateo, CA (remote) / May 2020 - Aug 2020*
- Interned remotely with a healthcare AI startup (Series D funded) based in Silicon Valley
- Created distributed feature generation scripts in Spark on AWS for big data in healthcare (~500GB)
- Improved error-detection models, increasing both precision and recall at a fixed review rate

Research Intern, Boeing *St Louis, MO / May 2019 - Aug 2019*
- Designed transfer learning algorithms for fault detection in Aviation Internet of Things (AIoT) applications
- Wrote Python scripts that ran on 777 aircraft at Boeing's ecoDemonstrator Fall '19 event at Frankfurt Airport
- Submitted a patent application and received the Boeing "Meritorious Invention Disclosure" award

Machine Learning Intern, PSU Applied Research Lab *University Park, PA / May 2018 - Aug 2018*
- Spearheaded project investigating object tracking and the evaluation of radar technology
- Replaced thousands of lines of rules-based code with streamlined SVM, improving performance
- Delivered multiple technical presentations to high-level sponsors, resulting in $5M project funding

---

Projects

Co-Founder, LionPad *State College, PA / Jan 2019 - Jan 2020*
- Founded startup to create a comprehensive, AI-powered housing search tool for Penn State students
- Won $25k and placed 2nd overall in a university-wide startup challenge

Project Lead, Nittany Data Labs *University Park, PA / Aug 2017 - Aug 2019*
- Won the Kaggle University Club Winter 2018 Hackathon, an international data science competition
- Led the best-attended student workshop at HackPSU Fall '18, Spring '19, and Fall '19

Extracurriculars

Safety Officer: Outing Club *University Park, PA / Aug 2017 - Present*
- Responsible for student safety, trip planning, club procedures, and emergency preparedness

ACSM Certified Personal Trainer *State College, PA / Jun 2016 - Present*
- Certified through the American College of Sports Medicine