2018

Automatic Selection of MapReduce Machine Learning Algorithms: A Model Building Approach

Fraggle Franklin
Michigan Technological University, bmfrankl@mtu.edu

Automatic Selection of MapReduce Machine Learning Algorithms: A Model Building Approach

By

Bryan M. Franklin

A Dissertation

Submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In Computer Science

Michigan Technological University

2018

This dissertation has been approved in partial fulfillment of the requirements for the Degree of DOCTOR OF PHILOSOPHY in Computer Science.

Department of Computer Science

Dissertation Advisor: *Laura Brown*

Committee Member: *Timothy Havens*

Committee Member: *Benjamin Ong*

Committee Member: *Thomas Oommen*

Department Chair: *Min Song*

for steve

Contents

List of Figures

List of Tables

Preface

All of the work presented in this dissertation is original and has not been previously published anywhere else.

Acknowledgements

As the epic saga of this project comes to a close, there are several people and organizations that have helped make this work possible.

First and foremost Alexandria Guth [1, 2] who provided invaluable moral, as well as financial support.

Steven Seidel, who had a huge influence on this project from the very beginning. Even though he was unable to see the final results of my efforts, the knowledge that he would have liked to, served as an inspiration to continue, even when the project seemed hopeless.

Laura Brown [3], who helped me navigate the many roadblocks that I encountered in this project.

Isaac, my feline housemate, who was always there to remind me that there are more important things than poking buttons, such as food and warms.

Also, the countless faculty and students that have helped me along the way who are too numerous to mention by name.

Additionally, this project would not have been possible without the datasets I used as inputs. Those datasets were provided by the UCI Machine Learning Repository [4], Georgia Tech [5], and the Computational Intelligence Group at the Basque University.

Abstract

As the amount of information available for data mining grows larger, the amount of time needed to train models on those huge volumes of data also grows longer. Techniques such as sub-sampling and parallel algorithms have been employed to deal with this growth. Some studies have shown that sub-sampling can have adverse effects on the quality of models produced, and the degree to which it affects different types of learning algorithms varies. Parallel algorithms perform well when enough computing resources (e.g. cores, memory) are available, however for a limited sized cluster the growth in data will still cause an unacceptable growth in model training time. In addition to the data size mitigation problem, picking which algorithms are well suited to a particular dataset, can be a challenge. While some studies have looked at selection criteria for picking a learning algorithm based on the properties of the dataset, the additional complexity of parallel learners or possible run time limitations has not been considered. This study explores run time and model quality results of various techniques for dealing with large datasets, including using different numbers of compute cores, sub-sampling the datasets, and exploiting the iterative anytime nature of the training algorithms. The algorithms were studied using MapReduce implementations of four supervised learning algorithms, logistic regression, tree induction, bagged trees, and boosted stumps for binary classification using probabilistic models. Evaluation of these techniques was done using a modified form of learning curves which has a temporal component. Finally, the data collected was used to train a set of models to predict which type of parallel learner best suits a particular dataset, given run time limitations and the number of compute cores to be used. The predictions of those models were then compared to the actual results of running the algorithms on the datasets they were attempting to predict.

Chapter 1

Background

1.1 Introduction

As more and more data is being collected every day, we are experiencing what some have dubbed the information explosion [6]. This has lead to a rise in large-scale parallel systems to help process these mountains of data. However as the amount of data grows larger, it will likely reach a point where even extremely large clusters will be unable to complete all of the iterations of many machine learning (ML) algorithms in a timely manner. Thus it is important to investigate which of these many techniques can build the best models in the least amount of time, even if the algorithm isn't run to completion. The purpose of this research is to develop a technique, based on learning curves [7], that can aid in the selection of parallel ML algorithms for a given domain or problem size.

1.2 Motivation

In the field of machine learning, there are many learning algorithms to choose from, and depending on the dataset on which a model is being trained, some algorithms will work better than others. Unfortunately as a consequence of the no free lunch theorem (NFL) [8, 9] for optimization problems such as machine learning, there can not be a single algorithm that works best for all problems. While many metrics exist to compare machine learning algorithms, as discussed in Section 1.3.10, none of them take into consideration the execution time of the algorithms. Likewise, parallel machine learning algorithms are often shown to produce models that are comparable to their non-parallel counterpart, but are seldom compared to other parallel algorithms in terms of model quality and speed.

Most studies therefore focus on one of these two problems:

1. ML evaluation metrics (e.g. AUC, accuracy, etc.) but rarely account for total run-time.
2. Parallel Algorithm metrics (e.g. speed-up) do not account for differences in model quality produced by different algorithms.

Therefore it is often difficult to pick a suitable algorithm when confronted with a large-scale machine learning task that is well suited for a parallel algorithm. As a result, it would be nice if there were a straightforward way to compare two parallel machine learning algorithms to easily determine if one can both provide better models, and in a more timely manner by examining the characteristics of the dataset.

Many comparisons of sequential machine learning algorithms have been done over the last several decades [10, 11, 7, 12, 13] and typically compare the algorithms across a wide variety of domains

using metrics such as AUC [14, 15, 16], accuracy, learning curves [7], recall, F-score, precision, Youden's index, and Discriminant Power [16].

Also, many parallel machine algorithms for classification problems have been proposed [17, 18, 19, 20, 21, 22, 23]. However parallel algorithms are usually analyzed using metrics such as speedup or efficiency. This can show that a parallel algorithm makes good use of available parallelism, but is not well suited for comparing quality of approximations given by algorithms for otherwise intractable problems such as machine learning algorithms. Speedup as reported is seldom computed correctly, in that most research compares the parallel run time to the run time of the parallel algorithm run on one node, instead of the best sequential algorithm available [24]. It is also becoming increasingly difficult to compute speedup as problem sizes start to exceed the limits that a single node can reasonably handle. It may take days or even weeks to do a single node run needed to compute speedup. Parallel performance of an algorithm is also heavily influenced by the characteristics of the problem instance it is being run on. Larger problems often have more available parallelism, but different algorithms may scale better with respect to certain aspects of the problem. So, a method that takes into account both the total run time and the quality of the output is needed for comparing parallel machine learning algorithms.

The goal of this research is to compare parallel machine learning algorithms. The primary focus will be a modification of learning curves [7]. In traditional learning curves the horizontal axis is the number of samples used to train the model and the vertical axis is a measure of model quality (i.e. accuracy, AUC, etc.). The proposed *temporal learning curves* (TLC) use the same vertical axis to express model quality but replace the horizontal axis with a temporal component. Specifically, the horizontal axis will be the amount of time it took to produce a particular model.

1.3 Machine Learning

While Machine Learning (ML) can refer to many things, as used in this work it refers to a class of algorithms that to attempt to 'learn' mathematical models from a set of multidimensional input samples such that these models can then be used to make predictions about previously unseen samples that are described as points in the same multidimensional attribute space as the training set.

Machine learning techniques fall into two major categories, *supervised* and *unsupervised*. In supervised learning, each sample in the dataset the model is being learned has an associated label, and the model that is produced will assign new samples a label. In unsupervised learning, the samples that are used to train the model do not have assigned classes or labels, but instead are assigned to unlabeled groups. This work focuses exclusively on the problem of supervised learning. A model for supervised learning can then be used to assign labels or value to unlabeled samples. An example dataset for supervised learning is given in Table 1.1. Typically within supervised learning, for classification tasks, the label is a class, while for regression tasks the label is a numeric value. However for this work, the supervised learning problem being explored produces models that give probability estimates that a given sample belongs to a specified class. Such models are probability estimation models. In this case models will only produce a probability that a sample should have a particular label. This research only examines supervised learning techniques that produce probabilistic models.

There are of course limits to the labeling power of ML algorithms. In some cases errors in the training data make it impossible to learn a perfect model. In other cases, the ML model being used is unable to adequately express the relationship between the attributes and the classes. The No Free

Lunch theorem (see Section 1.3.12) also tells us that for any learner, there will be some datasets for which a random guessing will outperform it. So, while the popularity of ML algorithms change over time, each popular algorithm will eventually run into problems it can't solve. *Neural networks* (*deep learning*) are the fashionable algorithm of today, but there is growing concern that even deep learning is approaching its limits [25].

1.3.1 Example Dataset

A dataset for machine learning, X, is composed of a set of samples $(\vec{x}_i, y_i) \in X$ for $i = 1, \ldots, n$, where n is the number of samples, $\vec{x}_i$ is a D-tuple of attributes, and y_i is a label. Each element of the vector $\vec{x}_i = \langle x_{i,0}, x_{i,1}, \ldots, x_{i,D-1} \rangle$ represents one of D different measurable aspects, or attributes, of the sample. The attributes can be of different types. These types are: *nominal* (or categorical) where the value is a member of a discrete unordered set of values, *binary* which is a special case of nominal where the set of values only contains two values, *discrete ordered* where the value is a single item from a sequence of discrete values that have an overall total ordering, and *continuous* which can be any value from a continuum of values, typically a real number.

To illustrate several important concepts in machine learning, I'll use a synthetic dataset to build a model that will try to decide if a student should walk or drive to campus based on weather conditions. This dataset contains a total of 1,000 samples, each with four attributes and a label. The four attributes are weather conditions, the label indicates if the student walked or drove in those conditions. A model can then be trained from this dataset to predict if the student will walk or drive given the current conditions.

The first attribute in the synthetic data is *precipitation*, which is binary and can have the values *yes* or *no*. The second attribute is *wind direction*, which is nominal and can have the values *north*, *south*, *east*, or *west*. The third attribute is *wind speed* which is denoted using the Beaufort scale [26] that assigns a numeric value from zero to twelve to different wind speeds. This attribute is therefore discrete ordered. The final attribute is *temperature* in degrees Fahrenheit, is treated as a continuous attribute. Each sample is also assigned a label (or class) that indicates the appropriate decision *walk* or *drive* for those conditions. A base model was created as shown in Figure 1.1. This model was then used to assign labels to one-thousand randomly generated samples. To make the original model harder to learn from the dataset, 5% of the samples chosen at random were changed to the opposite class. A portion of the resulting set of samples is shown in Table 1.1.

1.3.2 Dataset Splitting

In machine learning, an input dataset is usually partitioned into two or three subsets. These subsets are the trainining set, the test set, and a vaidation set. The training set is used during the primary learning portions of the ML algorithm. The test set which is not used in training is used in scoring the model produced by the algorithm. The validation set is used during training, but functions much like a test set as it is often used to score candidate models to help make decisions during the training process. Splitting the data in this manor is often referred to as hold out testing, since there is a test set that is held out of the training. Assignment to these three sets is usually done randomly and the dataset is usually split multiple ways using different random seeds. The models are then trained on each of the resulting training subsets and the scores produced using each of the corresponding test subsets are averaged to get a final score.

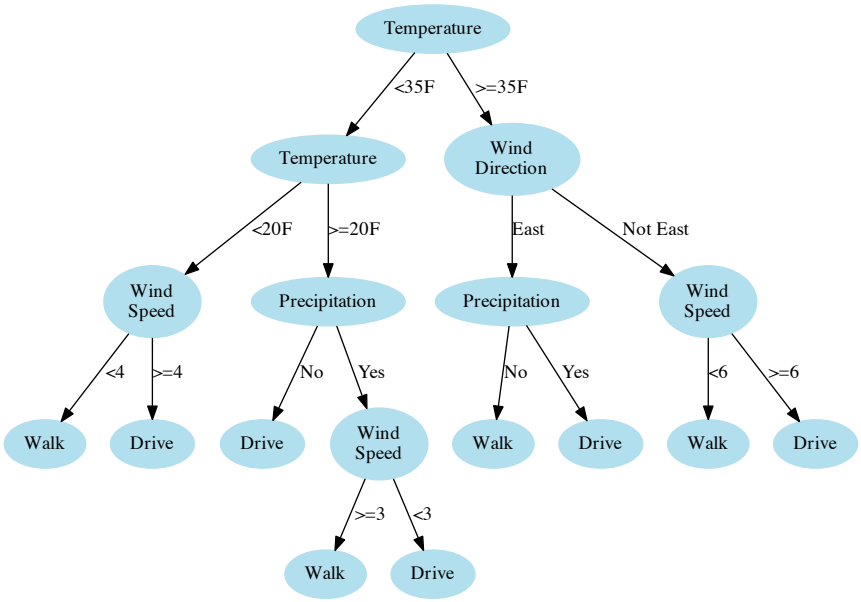

Figure 1.1: The decision tree that was used to generate data shown in Table 1.1.

1	2	3	4	5
Precipitation	Wind	Beaufort number	Temperature (°F)	Class
yes	north	4	62.69	walk
yes	south	1	63.02	walk
yes	west	1	4.72	walk
yes	west	0	47.98	walk
yes	south	8	17.12	drive
no	east	6	6.17	drive
no	east	8	-7.88	drive
yes	west	2	16.31	walk
no	north	1	5.07	walk

Table 1.1: Portion of an Example Dataset

An alternative way of splitting the data, called n-fold cross validation, splits the dataset randomly into n equally sized subsets. Then n models are trained where each of the subsets serves as a test set, and the rest of the subsets are used as training or validation sets. When computing a model score, the average over all n models if given.

For the purposes of this study, only hold-out testing was used with five or ten different random splits of the datasets.

1.3.3 Dataset Conversion

Some machine learning algorithms are unable to handle certain types of attributes directly, so transformations of the data are necessary as a pre-processing step. In logistic regression for example, the input values must all be numeric. While any range of numeric values is allowable, typically values are restricted to the ranges -1 to 1 or 0 to 1, or even the discrete values -1 and 1 or 0 and 1. For the purposes of this work, logistic regression will used inputs in the range zero to one. Binary values are typically mapped to zero and one in such a system. Other types of attributes need additional processing. Nominal attributes may be split into multiple new binary attributes where exactly one of the new attributes (the one corresponding to the original values) will be set to one for each sample. Discrete ordered attributes can either be handled as unordered discrete values (just like nominal attributes) which destroys any ordering information which might be relevant to the system being modeled, or the discrete values may be mapped to values between zero and one maintaining their original ordering. For continuous attributes, the range may be normalized to the range zero to one. It may also be split into multiple bins representing non-overlapping regions with binary attributes to indicate which of the regions the original value belongs to.

To illustrate the transformations needed, Table 1.2 was generated in the same way as the data in Table 1.1 but was transformed to **bi**nary and **norm**alized values (which I refer to as *binorm*). The wind attribute which was nominal with values *north*, *south*, *east*, and *west*, was split into four binary attributes, one for each possible direction. The wind speed attribute is scaled linearly such that 12 maps to 1. The temperature attribute is shifted and scaled so the range -20°F to 75°F maps fits in the range zero to one.

Since the data in Table 1.1 and Table 1.2 are both generated using a tree based model, it seems likely that a tree based learning system would likely outperform other learners. To examine this, possibility, a second dataset based on a logistic regression model where $P(Y = \mathrm{w}alk) = \frac{e^v}{1+e^v}$ and $v = 50 + (-20)(\mathrm{p}recip) + 2.5 * (\mathrm{w}indIsNorth) + 5 * (\mathrm{w}indIsSouth) + (-40) * (\mathrm{w}indIsEast) + 5 * (\mathrm{w}indIsWest) + 2.5 * (\mathrm{w}indSpeed) + 20 * (\mathrm{t}emp)$ was also constructed. The 50 term at the beginning of the model expression is the *bias factor*, which is analogous to the y-intercept in a linear regression model. This model can be expressed as a vector $\vec{\beta} = \langle 50, -20, 2.5, 5, -40, 5, 2.5, 20\rangle$. Full details of how $\vec{\beta}$ represents that model are given in Section 1.3.8.

1.3.4 Tree Induction

One popular machine learning method is *tree induction* [27] where a *decision tree* (sometimes called a *classification tree*, depending on the type of data used for training) is built from the training dataset. One reason for the popularity of decision trees is their relative ease of understanding. Part of this ease of understanding is that decision trees can be represented in an easy to use graphical form, such as the one in Figure 1.1. A decision tree consists of a set of nodes connected by directed edges. Each internal node in a decision tree contains a test that can be performed on a given sample

1	2	3	4	5	6	7	8
	Wind Dir.				Beaufort #	Temperature	
Precip.	N	S	E	W	(normalized)	(normalized)	Class
1	1	0	0	0	0.3333	0.8705	walk
1	0	1	0	0	0.0833	0.8739	walk
1	0	0	0	1	0.0833	0.2602	walk
1	0	0	0	1	0.0000	0.7155	walk
1	0	1	0	0	0.6667	0.3907	drive
0	0	0	1	0	0.5000	0.2755	drive
0	0	0	1	0	0.6667	0.1276	drive
1	0	0	0	1	0.1667	0.3822	walk
0	1	0	0	0	0.0833	0.2639	walk

Table 1.2: Example Training Dataset for use in Logistic Regression

and the edge to a child node is chosen depending on the outcome of that test. Each leaf node of the tree gives a possible outcome from the model. Figure 1.1 shows a decision tree that can be used to decide between walking or driving depending on current weather conditions. In this model each internal node is labeled with a weather condition being examined and the edges are labeled with the possible outcomes of that test. The leaf nodes are labeled with the actual decision to walk or drive.

To use a decision tree, a traversal starting from the root is performed, applying the test at each node to the sample being evaluated, and following the appropriate edge to the next node. When a leaf node is reached, that node gives the class that the tree assigns to the sample. A complete example of this is given later in this section.

The tree shown in Figure 1.1 has only two possible output classes (*walk* and *drive*), however this need not be the case. It is possible to have more than two outcomes, for instance some of the leaves could be changed to *bike* and it would still be a valid decision tree. It is also possible for internal nodes to have more than two children, for example when evaluating wind direction there could be four possible children, one for each direction (north, south, east and west) in the dataset. This research however focuses specifically on binary classification (i.e. exactly two output classes) using binary trees (i.e. exactly two outcomes at each internal node).

A binary decision tree is built from the training data iteratively. Starting with an empty model, which is a tree that contains a single node. Every attribute in the dataset is examined to pick the best way to split the training samples on that attribute, producing a set of candidate split points. The best candidate split point is then chosen using some *goodness of split measure* or *selection measure* such as information gain, information gain ratio or variance minimization. The chosen split point is assigned to the tree node and new child nodes are added for each possible outcome of the split point criteria. In a binary tree the split point criteria will always be binary, thus every internal node will have exactly two children. Each leaf node in the tree is assigned an output label based on the samples that reach that leaf node. Once the children are added all of the training samples are passed through the new model. Each leaf node is then treated as an empty model and a split point is chosen based only on the training samples that reached it. The process of sending the training samples through the tree and splitting the leaves repeats until some stopping criteria is met, such as every leaf only receives samples of a single class, or a leaf gets fewer than a minimum number of samples. When the tree growing process ceases, each leaf is assigned a label. Typically the label will be the most common label among the training samples that reach that node.

Algorithm 1 Tree Induction

```
1: procedure TreeInduction(X)                          ▷ Build tree from samples in dataset X
2:     V ← {v_Root}
3:     E ← ∅
4:     T ← (V, E)
5:     repeat
6:         for all x_i ∈ X do                          ▷ Assign each sample to a leaf node
7:             k ←TreeTraverse(T, x_i)
8:             LeafSamples_k ← LeafSamples_k ∪ x_i
9:         end for
10:        for all n_k ∈ Leafs(T) do
11:            SplitNode(n_k, LeafSamples_k)           ▷ Determine test for leaf node k
12:        end for
13:    until no nodes split
14:    return T                                        ▷ The finished tree
15: end procedure
```

Algorithm 2 Tree Traversal

```
1: procedure TreeTraverse(T, x_i)                      ▷ Find leaf node to put sample into
2:     n ← v_Root
3:     while ¬ Leaf(n) do
4:         if PassTest(n, x_i) then                    ▷ PassTest is either x_{i,d} < n.t or x_{i,d} = n.t
5:             n' ← v_k s.t. (n, v_k, true) ∈ E
6:         else
7:             n' ← v_k s.t. (n, v_k, false) ∈ E
8:         end if
9:         n ← n'
10:    end while
11:    return n                                        ▷ A single leaf node
12: end procedure
```

An alternative to decision trees called Probability Estimation Trees (PETs) label the leaf nodes with a probability that a sample should have a particular label [28]. The probability assigned is simply the ratio of positive samples reaching the leaf to the total number of samples. The probability found when using the tree on unseen samples is then the conditional probability $P(x_i \text{ has label } Y|C)$ where C is the conjunction of all of the tests performed on the sample along the path from the root to the leaf. The trees in this research are PETs. Pseudo-code for tree-induction is given in Algorithm 1 through Algorithm 3.

A tree is a graph $T = (V, E)$ where V is a set of nodes, and E is a set of directed edges. Each node $v_i \in V$ in the tree consists of a tuple (d, t), where d is the attribute id, t is the value for the split point. The node will then separate samples by applying a test to each sample. For numeric or ordered attributes, the test be $x_{i,d} < t$. For categorical attributes the test will be $x_{i,d} = t$. Each edge $e_i \in E$ in the tree consist of tuples (p, c, b) where p is the parent node, c is the child node, and b is a flag to mark is the edge is to be taken on a pass or a failure of the test in the parent node. This differs slightly from the model depicted in Figure 1.1, which shows the edges as having the split point values. In practice, the split point values are stored in the nodes, and not the edges.

Algorithm 3 Node Spliting

```
1: procedure TreeSplitNode(T, n, X)                          ▷ Pick split for samples in X
2:     gain_best ← −∞
3:     for all attributes d in X do
4:         for all values z ∈ Attribute_d do
5:             gain_d ← info_gain(d, z, X)
6:             if gain_d > gain_best then                    ▷ Pick largest information gain
7:                 gain_best ← gain_d
8:                 d_best ← d
9:                 z_best ← z
10:            end if
11:        end for
12:    end for
13:    v_pass(n) ← (d_best, z_best)                          ▷ Add new leaf nodes and attach them to the tree
14:    v_fail(n) ← (d_best, z_best)
15:    V ← V ∪ {v_pass(n), v_fail(n)}
16:    E ← E ∪ {(n, v_pass(n), true), (n, v_fail(n), false)}
17:    return
18: end procedure
```

Once a tree is built, it can be used to process a previously unseen sample. To do this, start at the root of the tree, examine the attribute the root node uses to split samples, then follow the edge that matches the sample being classified. Repeat this process at each node down the tree until a leaf node is reached. Once a leaf node is reached, look at the leaf's label to determine the sample's label.

Assume we have a sample <precipitation, wind direction, wind speed, temperature> = <no, north, 3, 45> and we wish to classify it using the tree in Figure 1.1. We start at the root which examines the temperature. For this sample, the temperature is 45°F, which is greater than 35°F, so we follow the right branch of the tree. The next node looks at the wind direction which is 'north' for this sample, so we follows the 'not east' branch. This leads to a wind speed node, this sample has a speed of 3, so we follow the less than 6 branch. This leaves us at a leaf node with a label of 'walk'. So we assign the sample <no,north,3,45> a label of 'walk'.

1.3.5 Node Splitting Methods

In Algorithm 3, the splitting criteria used is information gain [29], however there are other methods such as variance minimization [21] and minimum square expected error (MSEE) [30] for choosing split points. Mingers [31] showed that how splitting of nodes is chosen does not affect the quality of of the resulting model in terms of its accuracy, however it does affect the size of the resulting tree. Since this study will be focusing not only on the quality of the resulting models but also how long it takes to produce them, the splitting methods to be examined were chosen either for their tendency to produce smaller trees (information gain ratio [31]), because they were specifically designed for probability estimation trees (MSEE) or it was used in the original MapReduce algorithm for tree induction (variance minimization) [21].

The goal of each splitting method is to quantify the purity of each portion of the split. For each of the methods described, v_i is a unique node within the tree, for a split S, v_p is the parent node, and $v_i \in V_S$ are the children, x_{ij} is sample j that takes the path from the root to node v_i, N is the total

number of samples, N_i is the number of samples that reach node v_i, N_{ic} is the number of samples belonging to class c that reach node v_i, L is the set of class labels, C is the total number of classes, c_i is an individual class label, y_i is the label assigned to sample i (if the sample is from the class being learned, $y_i = 1$, otherwise $y_i = 0$).

Information Gain

The information gain splitting metric is based on the concept of entropy, and seeks to reduce the total randomness across both new child nodes.

In general entropy for a node v_i ($ent(v_i)$) is defined as

$$ent(v_i) = -\sum_{c \in L} \frac{N_{ic}}{N_i} \lg \frac{N_{ic}}{N_i} \tag{1.1}$$

for node v_i, where N_{ic} is the number of elements belonging to class c and N_i is the total number of samples across all classes within the node.

Information gain is then the difference between the entropy within a node prior to the split (i.e. $ent(v_p)$) minus the entropy across all child nodes after the split weighted by the number of samples incident upon each child node. The the information gain for a split S for a parent node v_p is

$$gain(S) = N_p \cdot ent(v_p) - \sum_{v_i \in V_S} N_i \cdot ent(v_i) \tag{1.2}$$

where V_S is the set of child nodes of v_p.

While Mingers [31] showed that information gain ratio produces the smallest trees. The method Mingers used, which was proposed by Quinlan [29], requires an initial selection by information gain and a secondary selection by information gain ratio is not well suited to the MapReduce framework used in this study as the reduce and combine phases require applying a simple total ordering to the splits.

Variance Minimization

With variance minimization [21], negative samples are assigned the value zero ($y_i = 0$) and positive samples are assigned the value one ($y_i = 1$). The goal is to minimize the total variance across all children. The variance for node v_i is

$$var(v_i) = \frac{\sum_{j=1}^{N_i} (c_{ij} - \bar{c}_i)^2}{N_i} \tag{1.3}$$

where N_i is the number of samples that reach node v_i, c_{ij} is a class assignment for sample j in node i, and $\bar{c}_i$ is

$$\bar{c}_i = \frac{\sum_{j=1}^{N_i} c_{ij}}{N_i} \tag{1.4}$$

.

So, if within a node all samples belong to the same class, the variance will be zero. Thus, minimizing the variance within a leaf node is desirable.

To account for uneven splits, a weighted variance is used, such that the weighted variance of split S_i for samples reaching node v_i is

$$weighted\_var(S_i) = \sum_{k=1}^{K} N_k \cdot var(v_k) \tag{1.5}$$

where K is the number of child nodes, N_k is the number of samples that reach child node node v_k. Since only binary trees are considered in this study, K will always be 2.

Minimum Squared Expected Error

The tree based models in this study use probability estimation trees (PETs), where the output when applying the model to a sample is a probability that the sample belongs to the class that the model was trained on (i.e. $P(y_i \in L)$). Most node splitting methods are based on the assumption that the output will be a discrete class that is either right or wrong for a given sample. Ferri et al. [30] presented the minimum squared expected error (MSEE) splitting method that is based on probabilities and expected error, which is intended for building probability estimation trees. This splitting criteria works by trying to minimize the square of the expected error based on each split's predictions for each sample in the training set.

Assuming that the quantity p_{ij} is both the probability of a sample reaching node child v_i from its parent v_p, and the probability of being assigned the label for class c_j. The value p_{ij} is therefore the number of samples belonging to class c_j that reach v_i divided by the total number of samples that reach v_p.

Consider for each class c, each child node v_i of parent v_p has a probability of p_i. Thus the probability of a sample being missclassified is given by

$$P(x_i \text{ is missclassified}) = p_i \cdot \sum_{i \neq j} p_j = p_i \cdot (1 - p_i) \tag{1.6}$$

. Thus the total error over all possible missclassifcations that the model can make for a single sample is given by

$$Error_i = p_i \cdot (1 - p_i) \cdot \left((1 - p_i)^2 \sum_{i \neq j} p_j^2 \right) \tag{1.7}$$

. The error for each sample can then be summed over all children resulting from a possible split. This gives an overall quality of split measure for MSEE is

$$MSEE(S_i) = \sum_{c=1}^{C} N_c \left(\sum_{i=1}^{K} Error_i \right) \tag{1.8}$$

.

For weighted samples the MSEE measure requires some modifications. Since the weighting is required for boosting as described in Section 1.3.7, the weights are such that all weights zero to one and sum to one, (i.e., w_i s.t. $0 \leq w_i \leq 1$ and $\sum_i w_i = 1$). Under this assumption, Equation 1.7 can be rewritten as Equation 1.9, where w_i is the weight for sample x_i.

$$P(x_i \text{ is missclassified}) = w_i \cdot p_i \cdot \sum_{i \neq j} w_j \cdot p_j = w_i \cdot p_i \cdot (1 - w_i \cdot p_i) \tag{1.9}$$

Equation 1.7 then is updated to be Equation 1.10.

$$Error_i = w_i \cdot p_i \cdot (1 - w_i \cdot p_i) \cdot \left((1 - w_i \cdot p_i)^2 \sum_{i \neq j} w_i \cdot p_j^2 \right) \tag{1.10}$$

The resulting $Error_i$ given by Equation 1.10 can then be used in Equation 1.8 to get a weighted MSEE split measure.

1.3.6 Bagging

Bagging (short for **b**ootstrap **agg**regat**ing**) [32] is an ensemble technique which uses many models each trained on a portion of the dataset which together can produce better overall results than a single model trained on a full dataset.

To produce a bagged model with k sub-models, a series of k independent models is computed. If the original training set has N samples, the training set passed to each of the models is a subset chosen randomly from the overall training set such that each sub-model gets N/k samples. This sampling is often done with replacement, but is performed without replacement in this study for reasons that are discussed in Section 1.4.1. Each sub-model is then trained using any technique, that could be used on the full dataset, typically tree induction.

Once all k sub-models are computed, the overall model is applied to new samples by first collecting the output of each sub-model individually, then combining the results. If the bagged model is for a classification problem, the most common output over all of the sub-models is reported. If the bagged model is for a probability estimation problem, the arithmetic mean of the probabilities can be used.

1.3.7 Boosting

Boosting [33] is another ensemble method that uses many sub-models in conjunction to produce a model than can outperform a single simple model. However unlike in bagging, the sub-models with boosting are given different weights. The samples are also assigned new weights before the training of each sub-model.

To produce a boosted model, all N of the samples in the training set are initially assigned a weight of $1/N$. The training samples are then used to produce a single model using any machine learning technique, however *weak learners* [33] (i.e., learners that by themselves can only achieve an error rate slightly below 50%). Very short trees, or *stumps*, which are height limited to only two or three levels have been found to be good weak learners for boosting. Each of the training samples s_i are then passed through the new sub-model m_j. The weights of the incorrectly classified samples (i.e., false positives or false negatives) are then summed. This sum, ϵ_j, can then be used to update the individual sample weights and compute a weight for sub-model m_j. The weights of all samples are updated such that the misclassified samples will have a total weight of 0.5. To accomplish this each sample weight w_i is updated to be w_i' using Equation 1.11 [33].

$$w_i' = w_i \div \begin{cases} 2\epsilon_j & : s_i \text{ misclassified by } m_j \\ 2(1 - \epsilon_j) & : otherwise \end{cases} \tag{1.11}$$

The reweighted samples are then used to train a new sub-model m_{j+1}. This process of reweighting and training continues until either ϵ_j reaches zero (i.e. the model is perfect on the training set) or $\epsilon_j \geq 0.5$ (i.e the incorrectly classifies samples are at least half the weight).

To apply the model to previously unseen samples first each sub-model m_j is assigned a weight α_j based on its ϵ_j value as shown in Equation 1.12.

$$\alpha_j = \frac{1}{2} \ln \frac{1 - \epsilon_j}{\epsilon_j} \tag{1.12}$$

Unseen samples are then passed though each sub-model. The prediction for a boosted classification model is the class for which the sum of the weights of the models predicting that class is the largest. For probability estimation models, the weighted average of the sub-model probability estimates is computed.

Since the sub-model weight $\lim_{\epsilon_j \to 0} \alpha_j = \infty$, special care must be taken in this case. To prevent the weight from going to ∞, smoothing can be used [34]. The model weighting equation then becomes

$$\alpha_j = \frac{1}{2} \ln \frac{1 - \epsilon_j + \mu}{\epsilon_j + \mu} \tag{1.13}$$

where μ is a very small positive value. For this study, μ was 10^{-5}.

1.3.8 Logistic Regression

In a logistic regression model [20] the prediction is a real number in the range zero to one, which is the probability that the sample is a member of the set the target class.

Starting with the probability (p_i) that the label y_i for a sample x_i belongs to a particular class c_k, denoted by $y_i = c_k$.

$$p = P(y_i = c_k). \tag{1.14}$$

The odds ratio r, which is the ratio of $P(Y = 1)$ to $P(Y \neq 1)$, can be expressed as

$$r = \frac{P(Y = 1)}{P(Y \neq 1)} = \frac{p}{1 - p}. \tag{1.15}$$

Log odds, which is the log of the odds ratio, is then

$$\ln r = \ln \frac{p}{1 - p}. \tag{1.16}$$

Solving for p in Equation 1.16 gives

$$p = \frac{e^r}{1 + e^r} \tag{1.17}$$

which has the property that $p \in (0, 1) \forall r \in (-\infty, \infty)$.

To apply the model to a sample to get a prediction, each sample is expressed as a vector $\vec{x}$, where each dimension of the vector is an attribute. The vector, must then be mapped onto a real number $v \in (-\infty, \infty)$ such that as v approaches $-\infty$, p approaches 0 and as v approaches ∞, p approaches 1. To do this v is computed using a linear combination of each sample's attributes

$$r = \beta_0 + x_1 \cdot \beta_1 + x_2 \cdot \beta_2 + \cdots + x_n \cdot \beta_n = \vec{x} \cdot \vec{\beta} \tag{1.18}$$

where $\vec{x}$ represents the sample and $\vec{\beta}$ is the model. The vectors $\vec{\beta}$ and $\vec{x}$ each have a length that is one greater than the number of attributes in the dataset being learned. The additional element x_0 in $\vec{x}$ is always set to 1 and is needed to compute the additional element β_0 in $\vec{\beta}$ which is the *bias factor*, which is analogous to the y-intercept in a linear regression model. Thus Equation 1.17 can be rewritten as

$$p = f(\vec{\beta}, \vec{x}) = \frac{e^{\vec{x}\cdot\vec{\beta}}}{1 + e^{\vec{x}\cdot\vec{\beta}}}, \tag{1.19}$$

and the problem of learning the model is reduced to picking the values of the vector $\vec{\beta}$ based on the values in the training set such that p is as close to the actual Y value for all samples in the training set.

To find the value of $\vec{\beta}$ for a particular training set, an initial empty model is used as a starting point (i.e. $|\vec{\beta}| = 0$) and attributes are added to the model one at a time in a process referred to as *forward feature selection*.

Since all unused elements in $\vec{\beta}$ are zero, the equation for p can be rewritten to include the candidate feature

$$p = P(Y = 1) = f_d(\vec{x}, \vec{\beta}, \beta'_d) = \frac{e^{\vec{x}\cdot\vec{\beta} + x_d \cdot \beta'_d}}{1 + e^{\vec{x}\cdot\vec{\beta} + x_d \cdot \beta'_d}} \tag{1.20}$$

where β'_d is element of $\vec{\beta}$ being optimized.

The log likelihood for a sample i described by $\vec{x}_i$ and has label y_i, given a model vector $\vec{\beta}$ is

$$L(\vec{x}_i, \vec{\beta}, y_i) = \begin{cases} \ln f(\vec{x}_i, \vec{\beta}) & : y_i = 0 \\ \ln(1 - f(\vec{x}_i, \vec{\beta})) & : y_i = 1 \end{cases} \tag{1.21}$$

The optimal value for each element of $\vec{\beta}$ depends on all samples in the training set, thus the total log likelihood for all samples

$$E(X, \vec{\beta}) = \sum_{i=1}^{n} L(\vec{x}_i, \vec{\beta}, y_i) \tag{1.22}$$

needs to be maximized. Since $y_i \in \{0, 1\}$, the composite function for $L(\vec{x}_i, \vec{\beta})$ can be rewritten in non-composite form as

$$L(\vec{x}_i, \vec{\beta}, y_i) = (1 - y_i) \cdot \ln\left(f(\vec{x}_i, \vec{\beta})\right) + y_i \cdot \ln\left(1 - f(\vec{x}_i, \vec{\beta})\right) \tag{1.23}$$

since either y_i or $(1 - y_i)$ will always be zero, one term of the addition will always be zero.

The goal is then to maximize $E(X, \vec{\beta})$ by carefully choosing values for $\vec{\beta}$. Many methods exist for finding the values $\vec{\beta}$, for example iteratively reweighted least squares (IRLS) [35], gradient descent, and single feature optimization (SFO) [20]. While in IRLS and gradient descent all of the elements of $\vec{\beta}$ are optimized simultaneously, in SFO, only one element of $\vec{\beta}$ is optimized at a time.

To optimize a single element β_d of $\vec{\beta}$, Newton's method is employed. In Newton's method, an initial guess of the solution (zero in this case) is refined through multiple iterations until it converges on a solution. Since the optimization problem to be solved here is a maximization problem, this is the same as finding a root of the first derivative. This will require the first and second derivatives of L.

The updating formula used each iteration to compute the new value for dimension d of β, β'_d from β_d is

$$\beta'_d = \beta_d - \frac{\frac{\delta L}{\delta \beta_d}}{\frac{\delta^2 L}{\delta \beta_d^2}} \tag{1.24}$$

where

$$\frac{\delta L}{\delta \beta_d} = \sum_{i=1}^{n} x_{i,d} \cdot \left(y_i - f_d(\vec{x}_i, \vec{\beta}, \beta_d) \right) \tag{1.25}$$

and

$$\frac{\delta^2 L}{\delta \beta_d^2} = \sum_{i=1}^{n} (x_{i,d})^2 \cdot f_d(\vec{x}, \vec{\beta}, \beta_d) \cdot \left(1 - f_d(\vec{x}, \vec{\beta}, \beta_d) \right). \tag{1.26}$$

The first element of β to be found is always the bias factor β_0. Once an initial bias factor is found, all other elements in the $\vec{\beta}$ vector are chosen to maximize log likelihood independently. The models resulting from adding each newly computed element of β are evaluated using AUC as described in Section 1.3.10. The new element with the best score is chosen and added to the model. All non-zero coefficient in the model so far can then re-optimized repeatedly until all of them converge, in a process referred to as *full regression*. This step is time consuming, but can lead to much better models overall. Due to the time involved in full regression, it can also be done once at the very end, instead of after each feature is added.

After full regression, the process of finding optimal values for each zero element in the $\vec{\beta}$ vector, scoring the resulting models and doing full regression repeats until the score obtained by adding new features to the model stops improving. Typically this will stop before all features are added to the model.

The computation of $\frac{\delta L}{\delta \beta_d}$ and $\frac{\delta^2 L}{\delta \beta_d^2}$ require a summation over all samples, that can be time consuming to compute. To greatly increase the speed of this computation, histograms can be used [20]. To do this two histograms with B bins are built for each attribute. The bins equally subdivide the range $[0, 1]$. Samples are mapped into bins based on the probability computed by the current model β when applied to that sample. The first histogram, H, contains a count of all samples where $x_d = 1$ that the current model predicts will be in that each bin. The second histogram, H^+ counts only the samples for which $x_{i,d} = 1$ and $y_i = 1$.

To compute these histogram, we first rewrite the equations for building the model to be built based on the aggregate value for each bin. The log likelihood for each bin b is

$$a_b = \ln \left(\frac{p_b}{1 - p_b} \right) \tag{1.27}$$

and the probability of samples belonging to bin b is

$$p'_b = \frac{e^{a_b + \beta_d}}{1 + e^{a_b + \beta_d}}. \tag{1.28}$$

Once the histograms H and H^+ are built, $\frac{\delta L}{\delta \beta_d}$ and $\frac{\delta^2 L}{\delta \beta_d^2}$ can be computed as

$$\frac{\delta L}{\delta \beta_d} = \sum_{b=1}^{B} \left(H_b^+ - p'_b \cdot H_b \right) \tag{1.29}$$

and

$$\frac{\delta^2 L}{\delta \beta_d^2} = \sum_{b=1}^{B} H_b \cdot p'_b \cdot (1 - p'_b)\,. \tag{1.30}$$

Using this heuristic, each iteration of Newton's method will run in $O(B)$ time, which is much less than the original $O(N)$ when $B << N$.

1.3.9 Non-binary Arttributes

The histogram technique described by Singh et al. [20] only works with binary attributes. However, it can be adapted to work with non-binary attributes by using two dimensional histograms, $\hat{H}_{b,d}$ and $\hat{H}^+_{b,d}$. Both dimensions of these histograms contain bins that evenly subdivide the range $[0, 1]$. The first dimension is used to map the samples based on the model's predicted probability. The second dimension is based on the value of $x_{i,d}$.

Using these modified histograms $\frac{\delta L}{\delta \beta_d}$ and $\frac{\delta^2 L}{\delta \beta_d^2}$ can be computed using

$$\frac{\delta L}{\delta \beta_d} = \sum_{b=1}^{B} \sum_{d=1}^{B} \hat{x}_{b,d} \cdot \left(\hat{H}^+_b - p'_b \cdot \hat{H}_b \right) \tag{1.31}$$

and

$$\frac{\delta^2 L}{\delta \beta_d^2} = \sum_{b=1}^{B} \sum_{d=1}^{B} \hat{x}^2_{b,d} \cdot \hat{H}_b \cdot p'_b \cdot (1 - p'_b) \tag{1.32}$$

where $\hat{x}_{b,d}$ is the midpoint of each bin in the $x_{i,d}$ dimension this will of course take more time than the binary histogram method but still runs in $O(B^2)$ time, which is much less than the original $O(N)$ when $B << \sqrt{N}$. Pseudo-code for SFO is given in Algorithm 4.

If we again use the sample <precipitation,wind direction,wind speed,temperature> = <no,north,3,45>. To classify this sample using a logistic regression model, it must first be converted to a form that is compatible with logistic regression. The first attribute in the sample vector is the bias factor, which is always 1. The precipitation attribute is binary (yes→1, no→0), so no becomes 0. The wind direction is nominal, so it will become four binary attributes (one for each direction) where only the attribute for north will be set to 1. For numerical values such as the Beaufort number (wind speed) and temperature each value in a range [lower, upper] can be mapped onto the range [0,1]

$$value_{\text{normalized}} = \frac{value - \text{lower}}{\text{upper} - \text{lower}}. \tag{1.33}$$

The wind speed attribute has a lower bound of zero and an upper bound of 12. So, 3 is converted using $(3-0)/(12-0) = 0.25$. The temperature attribute in this dataset has a lower bound of -20.0°F and an upper bound of 75°F. So, 45°F is converted using $(45-(-20))/(75-(-20)) \approx 0.68421$. So, the sample after conversion is the vector $\vec{x}$ =<precipitation, north wind, south wind, east wind, west wind, wind speed, temperture> = <1, 1, 1, 0, 0, 0, 0.25, 0.68421>.

Once the sample has been converted, it can be used directly as $\vec{x}$ in the logistic regression model. Putting this vector and the example model vector $\beta =< 50, -20, 2.5, 5, -40, 5, 2.5, 20 >$ into Equation 1.19 gives a probability that the sample belongs to the $w$alk class $p \approx 1$.

Algorithm 4 Single Feature Optimization

procedure LOGISTICREGRESSION(X) ▷ Build logistic regression model
  $\vec{\beta} \leftarrow < \beta_0, \beta_1, \ldots, \beta_D > = < 0, 0, \ldots, 0 >$ ▷ where D is the number of attributes
  $\beta'_0 \leftarrow$ OptimizeCoef$(X, \beta, 0)$ ▷ Compute bias factor
  $AUC_{best} \leftarrow$ ScoreCoef$(X, \vec{\beta}, \vec{\beta'}, 0)$
  repeat
    $\vec{\beta'} \leftarrow < \beta'_0, \beta'_1, \ldots, \beta'_D > = < 0, 0, \ldots, 0 >$
    for all attributes $d \in X$ **do**
      $\beta'_d \leftarrow$OptimizeCoef(X, β, d)
    end for
    $AUC_{max} \leftarrow 0$
    for all $d \in D$ **do**
      $AUC_{new} \leftarrow$ScoreCoef$(X, \vec{\beta}, \vec{\beta'}, d)$ ▷ Compute AUC with β_d replaced by β'_d
      if $AUC_{new} > AUC_{max}$ **then**
        $AUC_{max} \leftarrow AUC_{new}$
        $d_{max} \leftarrow d$
      end if
    end for
    $\beta_d \leftarrow \beta'_{d_{max}}$
  until $AUC_{max} \leq AUC_{best}$
  return β ▷ The model vector
end procedure

Algorithm 5 Optimize Coeffcient

procedure OPTIMIZECOEF(X, β, d) ▷ Optimize a Single Model Coefficient
  $\beta_d \leftarrow 0$
  while $\frac{\delta L}{\delta \beta_d} > \epsilon$ **do**
    $\beta'_d \leftarrow \beta_d - \left(\frac{\delta L}{\delta \beta_d} / \frac{\delta^2 L}{\delta \beta_d^2} \right)$
    $\beta_d \leftarrow \beta'_d$
  end while
  return β_d ▷ Single element of model vector
end procedure

1.3.10 Model Evaluation

When evaluating machine learning models, a variety of quantitative metrics are commonly used, each with their strengths and weaknesses [36, 16]. Among these are accuracy, recall, F-score, precision, AUC [37], Youden's index, likelihood values, and Discriminant Power.

Many evaluation metrics are based on the contents of the confusion matrix generated by running a test set of data (of size N_{test}), that was not used in training the model, through the model and comparing each sample's actual label to the label assigned by the model. Assuming a model produces outcomes that are either *positive* (i.e. the sample belongs to the class being learned) or *negative*, (i.e. does not belong to the class), each sample will fall into one of four categories. Samples that a model assigns to the class being learned are said to be positive results, those samples that actually belong to the class are *true-positives* (*TP*), those that do not belong to the class are *false-positives* (*FP*). Likewise samples that a model does not assign to the class are said to be negative results, those samples that actually belong to the class are *false-negatives* (*FN*), those that do not belong to the class are *true-negatives* (*TN*). The number of samples in each of these categories make up the confusion matrix are shown in Table 1.3.

actual \ model	positive	negative
positive	TP	FN
negative	FP	TN

Table 1.3: Confusion matrix

Using the values in the confusion matrix, common evaluation metrics can be computed as follows. *Accuracy* is the ratio of correctly classified samples ($TP + TN$) to the total number of test samples (N_{test})

$$accuracy = \frac{TP + TN}{N_{test}}. \tag{1.34}$$

The *error* rate is the ratio of the incorrectly classified samples ($FP + FN$) to the total number of samples,

$$error = \frac{FP + FN}{N_{test}} = 1 - accuracy. \tag{1.35}$$

Recall (also referred to as *sensitivity*) is the ratio of true positives to the sum of the true positives and the false negatives (that is the ratio of true positives to the total number of actual positives),

$$recall = \frac{TP}{TP + FN}. \tag{1.36}$$

Specificity which gives the percent of negative samples that were correctly classified, is computed as

$$specificity = \frac{TN}{FP + TN}. \tag{1.37}$$

Precision gives the percentage of positive results that were actually correct, is computed as

$$precision = \frac{TP}{TP + FP}. \tag{1.38}$$

F-score (or *F-measure*) uses a scalar parameter β (which has nothing to do with the vector $\vec{\beta}$ in logistic regression). The β parameter is used to tune the preference for precision ($\beta > 1$) or recall

($\beta < 1$). The F-score is computed using

$$\textit{F-score} = \frac{(\beta^2) \cdot \textit{precision} \cdot \textit{recall}}{\beta^2 \cdot \textit{precision} + \textit{recall}}. \tag{1.39}$$

A slightly more obscure metric *Youden's Index* [16] focuses on a classifier's ability to avoid failures, and is computed as

$$\gamma = \textit{recall} - (1 - \textit{specificity}). \tag{1.40}$$

Higher γ values are better. *Discriminant Power* [16] is typically used only on feature selection, and measures a model's ability to separate positive and negative samples. It is computed using

$$\begin{aligned} DP &= \tfrac{\sqrt{3}}{\pi}(\log X + \log Y) \\ X &= \textit{recall}/(1 - \textit{recall}) \\ Y &= \textit{specificity}/(1 - \textit{specificity}). \end{aligned} \tag{1.41}$$

The Area Under the ROC Curve (AUC) is a measure that compensates for uneven ratios of positive and negative samples in the test dataset. To compute AUC, the receiver operator characteristic (ROC) curve is constructed, the AUC is then the area under that curve,

$$AUC = \int_0^1 ROC(X_{\text{test}}), \tag{1.42}$$

where $ROC(x)$ represents the Receiver operating characteristics (ROC) curve [38]. To construct an ROC curve, all of the samples in the test dataset are run through the mode to get a predicted probability. Each of the unique probabilities are then used as thresholds for *specificity* for determining the positive versus negative classification. If the prediction is greater than of equal to the threshold, the sample is considered a positive, if it is below the threshold, it it considered to be negative. This will lead to some negative samples being false positives at lower thresholds, and some positive samples being false negatives at higher thresholds. The *sensitivity* at each threshold is then equivilant to the recall value (Equation 1.36). The samples in Table 1.4 were assigned predicted values using a logistic regression model $\beta =< 1.4583, -0.2797, 0.1567, 0.0548, -0.5301, 0.3188, -4.8885, 0.4227 >$ computed using Weka [39]. Using this test dataset, a receiver operator characteristic (ROC) curve can be constructed. Using each of the unique values in the predicted probability column (P(Class=walk)) as specificity values, a set of points in ROC space ($1 - \textit{specificity}$, $\textit{sensitivity}$) can be found, as shown in Figure 1.2. These points approximate the ROC curve, the area under this curve can easily be obtained using the trapezoidal method giving an AUC of 0.631.

AUC has been criticized for being overused by researchers [15], specifically that reporting only the AUC obscures the shape of the ROC curve which is needed to pick an appropriate specificity for using the model.

To apply many of the other common metrics, a threshold for determining a positive prediction must be chosen. For simplicity, a threshold of 0.5 will be used. That is any sample where the model says $P(Y = \text{walk}) \geq 0.5$ will be assigned to the class 'walk'. Using this threshold gives 7 true positives, 1 false positives, 0 false negatives, and 2 true negatives. This can be represented as a confusion matrix as shown in Table 1.5.

Using these numbers several metrics can easily be computed.

- Accuracy is $(TP + TN)/N_{test} = (8 + 7)/20 = 0.750$
- Error is $(FP + FN)/N_{test} = (3 + 2)/20 = 0.250$

1	2	3	4	5	6
Precipitation	Wind	Beaufort number	Temperature (°F)	Actual Class	Prediction P(Class=walk)
no	north	0	19.65	walk	0.857
yes	south	8	19.71	drive	0.136
yes	west	4	-14.65	drive	0.473
yes	north	8	31.01	walk	0.155
no	east	4	44.10	walk	0.397
no	east	4	-16.45	drive	0.335
no	west	6	1.83	drive	0.361
yes	west	2	-3.18	walk	0.681
no	south	0	20.32	drive	0.845
no	west	1	0.45	walk	0.812
no	east	2	37.99	walk	0.592
no	west	6	4.24	drive	0.364
yes	west	1	63.48	walk	0.812
no	north	4	54.43	walk	0.579
no	west	0	53.35	walk	0.891
no	south	0	10.82	drive	0.839
no	east	1	27.10	drive	0.675
yes	south	2	39.76	walk	0.665
no	north	6	49.85	drive	0.373
no	south	6	-1.79	drive	0.299

Table 1.4: Example Test Dataset

actual \ model	positive	negative
positive	8	2
negative	3	7

Table 1.5: Confusion matrix for test data using a threshold of 0.5

Spec.	TP	FN	Sens.
0.000	10	0	1.000
0.136	10	0	1.000
0.155	9	1	0.900
0.299	9	1	0.900
0.335	9	1	0.900
0.361	9	1	0.900
0.364	9	1	0.900
0.373	9	1	0.900
0.397	9	1	0.900
0.473	8	2	0.800
0.579	7	3	0.700
0.592	6	4	0.600
0.665	5	5	0.500
0.675	5	5	0.500
0.681	4	6	0.400
0.812	2	8	0.200
0.812	2	8	0.200
0.839	2	8	0.200
0.845	2	8	0.200
0.857	2	8	0.200
0.891	1	9	0.100
1.000	0	10	0.000

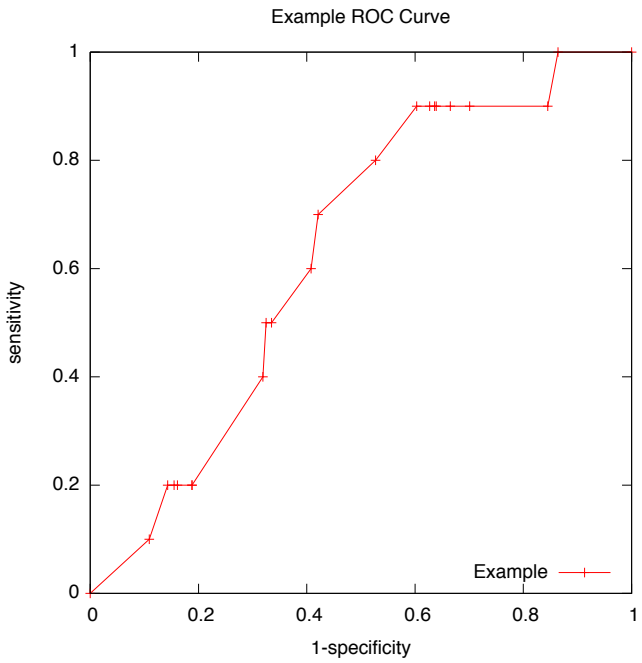

Figure 1.2: Specificity and sensitivity for example data (left) and a plot of the ROC curve (right).

- Recall is $TP/(TP+FN) = 8/(8+2) = 0.800$
- Precision is $TP/(TP+FP) = 8/(8+3) = 0.727$
- Specificity is $TN/(FP+TN) = 7/(3+7) = 0.7$
- Youden's Index is $recall - (1 - specificity) = 0.8 - (1 - 0.7) = 0.5$

Computing the discriminant power is bit more complicated. First X and Y must be found, $X = 0.8/(1-0.8) = 4$ and $Y = 0.7/(1-0.7) \approx 2.333$. The discriminant power is then $\frac{\sqrt{3}}{\pi}(\log 4 + \log 2.333) = 0.5348$.

Due to the numeric nature of probability estimates, metrics such as root mean squared error (RMSE) can be used to give an estimate of how far off the estimates are from actual. To compute RMSE all of the positive samples are assigned the value one ($c_i = 1$), and all of the negative samples are assigned the value zero ($c_i = 0$). RMSE can then be computed as shown in Equation 1.43, where p_i is the model's prediction, and N is the total number of samples in the test dataset.

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - c_i)^2} \tag{1.43}$$

Since RMSE for a perfect model (i.e., one that guesses 1 for all positive samples, and 0 or all negative samples) is zero, $1 - RMSE$ is used to for consistency with other metrics where higher values are better.

In much the same way that accuracy and error measure the same quantity ($error = 1 - accuracy$) a similar measure can also be devised using the per sample accuracy (i.e., $1 - error$), in the same way RMSE uses the per sample error. This gives a measure I refer to as root mean squared accuracy (RMSA). At first glance this seems like it should just be $1 - RMSE$, however due to the squaring of the accuracy which is at most one $accuracy^2 \leq accuracy$, small decreases in accuracy for a sample can have larger impacts on the overall RMSA. To compute RMSA all of the positive samples are assigned the value one ($c_i = 1$), and all of the negative samples are assigned the value zero ($c_i = 0$), as in RMSE. RMSA can then be computed as shown in Equation 1.44, where p_i is the model's prediction, and N is the total number of samples in the test dataset.

$$RMSA = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(1 - |p_i - c_i|)^2} \tag{1.44}$$

$RMSA$ is also the square root of the Brier score [40].

This is by no means an exhaustive list of metrics, however this sampling of them indicates that collecting enough data to create a confusion matrix for each model is sufficient to compute most of them. In particular, only non-parametric model quality metrics are considered in this study. This eliminates measures such as F-score, which is dependent on the β parameter.

1.3.11 Overfitting

A common problem in machine learning is *overfitting*. Overfitting occurs when the model produced during the training phase learns the dataset itself and not the underlying relationships within the dataset. As a result, the model does not predict unseen samples as well as it should. To deal with this, machine learning methods employ different methods.

L_2 regularization

In logistic regression, as in many other machine learning techniques, overfitting is reduced by adding a penalty to the loss function. One such penalization mechanism is L_2-regularization, which was used in the original logistic regression algorithm [20] used in this study. In particular L_2-regularization penalizes coefficients of β with large absolute values by adding a penalty that is proportional to the the length of the model vector $\vec{\beta}$ [41]. This is accomplished by adjusting the log-likelihood function $L(\vec{x}_i, \vec{\beta}, y_i)$ to be

$$L_\lambda(\vec{x}_i, \vec{\beta}, y_i) = L(\vec{x}_i, \vec{\beta}, y_i) - \lambda||\beta||^2 \tag{1.45}$$

The first and second derivatives of the log-likelihood become,

$$\frac{\delta L_\lambda}{\delta \beta_d} = \left(\sum_{i=1}^{n} x_{i,d} \cdot \Big(y_i - f_d(\vec{x}_i, \vec{\beta}, \beta_d)\Big)\right) - 2\lambda\beta_d \tag{1.46}$$

and

$$\frac{\delta^2 L_\lambda}{\delta \beta_d^2} = \left(\sum_{i=1}^{n} (x_{i,d})^2 \cdot f_d(\vec{x}, \vec{\beta}, \beta_d) \cdot \Big(1 - f_d(\vec{x}, \vec{\beta}, \beta_d)\Big)\right) - 2\lambda \tag{1.47}$$

.

Pruning

In tree induction, pruning is employed [42, 43] to remove parts of the tree that do not classify samples well. Pruning is done by applying the model to the validation set, then finding places where removing leaves gives a tree that scores higher on the chosen metric on the vlaidation set than the model did with the leaves included. *Reduced error pruning* [44] is employed in this study to counter overfitting. However, as is evident from model score vs time graphs in Chapter 4, the pruning may greatly decreases the model quality for unseen samples in certain domains.

1.3.12 No Free Lunch Theorem

A No Free Lunch Theorem (NFL) generally says that there is no universally optimal way to solve all problems within a domain. For supervised machine learning there is an NFL that essentially states that no single algorithm can be optimal for all datasets [8]. More specifically, the average performance of any machines learning algorithm over all possible learning tasks is the same as any other machine learning algorithm [45]. At first glance this would indicate that attempting a study of this sort is doomed to fail. However, the NFL for supervised learning only applies *a priori*, so if the dataset can first be evaluated, it may be possible to determine from its properties which machine learning algorithms will work best. Several previous studies such as Michie et al. [46], Ali and Smith [47], and to a lesser extent Perlich et al. [7] all indicate that there are learnable properties of datasets that can inform which algorithm(s) should work well for a given dataset.

Additionally the concept of meta-learning has been proposed [48, 45] which goes beyond simply recommending the best algorithm for a task and turns the problem into a sort of planning task for choosing different aspects of the learning process and fine tuning of hyper-parameters. Many of the decisions made during meta-learning are done based on the properties of the datasets themselves. This study however will stop short of planning based meta-learning, and will instead focus on recommending the best parallel algorithm for a dataset given a cluster and a CPU-time budget.

1.4 MapReduce

The MapReduce [49] programming model simplifies many aspects of writing programs for large clusters by simplifying synchronization and communication, hiding most of it from from the programmer entirely To accomplish this simplification, a MapReduce program consists of multiple pieces of sequential code that communicate by passing key/value pairs between each other. The run-time system handles the passing of key/value pairs between the correct pieces of sequential code based on the keys, as well as scheduling when to start each sequential chunk. MapReduce run-time systems also typically provide a distributed filesystem or a mechanism for distributing files across the nodes to provide the initial input for the MapReduce jobs.

A MapReduce program consists of at least three sequential pieces of code, a main driver, a mapper and a reducer. The main driver performs basic setup of the runtime system and launches the individual MapReduce job. A MapReduce job is launched from the main driver and consists of a map phase and a reduce phase. The main driver typically also has a mechanism to broadcast configuration information, before a job is initiated, to all of the mappers, reducers, or other pieces of code. Once a MapReduce job is started, the main driver typically sits idle until the final output is available.

The mapper's main task is to split the input into individual independent sub-problems. In the map phase the input is read from disk, processed and turned into a set of key/value pairs which are sent back to the run-time system, which in turn passes them to reducers.

The reducer's task is to solve those sub problems and pass their results back to the main driver. In the reduce phase a reducer is given all of the values for a given key and processes them further. Each reduce task then emits new key/value pairs that are sent back to a central collection point where they can be dealt with by the main driver. If there are more keys than reducers, each reducer may have to process multiple independent keys.

1.4.1 Tree Induction via MapReduce

In the PLANET tree induction algorithms [21], there are two possible MapReduce jobs that divide the problem into independent sub-problems in different ways.

The first MapReduce job which is used early in the execution is used for picking a splitting criteria for a single node in the tree. In this MapReduce job, the mapper reads each individual sample, passes it down the current tree model until it reaches a leaf node. If the sample reaches the node that is being split, a key/value pair is created per attribute. In each pair, the key is an attribute identifier and the value is the element of the sample for the corresponding the attribute along with the sample's class. In the reduce phase, each reducer searches for the best split point for the key's attribute and emits a key/value pair where the key is the attribute identifier and the value contains a description of the chosen split along with a score for that split. The main driver then takes the splits from all of the reducers and picks the best attribute split point based on the returned scores.

The second MapReduce job which is used later in the execution is capable of splitting all of the leaves in the tree as a single job. In this MapReduce job, the mapper again reads each individual sample and passes it down the current tree until it reaches a leaf node. However, all of the samples are converted into key/value pairs where the key is the leaf node's identifier and the value is the entire sample including its class. The reducer then picks the best split point for all of the samples that reach the given node. In this case, each reducer must examine all of the features in all of the samples it's given, instead of just a single feature as in the first MapReduce job. The main driver then updates the model by converting the newly split leaves into internal nodes and adding new leaf nodes to the tree.

The first splitting method has the most available parallelism when there is a large number of features in the dataset, but performance degrades as the fraction of total samples that reach the node being split decreases. The second method has the most available parallelism when there are a large number of leaves in the tree.

The main driver program starts with an empty model, and iteratively calls one of the MapReduce jobs until no split points are returned or all split points are rejected by the main driver. This can occur due to height limits, insufficient samples reaching a leaf, or all of the samples belonging to the same class. Picking which MapReduce job to use each iteration is therefore the responsibility of the main driver program.

Bagging via MapReduce

When doing bagging, as described in Section 1.3.6, each sample is assigned to a given tree for all rounds of training. This is a tricky thing to do in MapReduce, especially when running with variable numbers of mappers. To accomplish this, each sample was tagged with a unique numeric identifier

when the dataset was split across the mapper nodes. For simplicity, the identifier is based on the order of the samples in the respective train, validation, or test input files. Since the identifiers are based on their position in the files after being split into train, validation and test sets, the assignment to trees is slightly more random than if the pre-split positions were used. This unique identifier is then used to assign the sample to the correct tree in each iteration of training.

Typically when bagging is done in serial, the samples are assigned to individual trees for training by randomly sampling the training set with replacement. Thus a sample may be used by multiple trees for training. Due to the structure of MapReduce algorithms, sampling with replacement was deemed too complicated for this study. Instead the samples are assigned to the trees in a round-robin fashion. Sample j is assigned to tree i according the the formula $i = j \mod N$, where N is the number of trees.

Each round of training a bagged model uses the second method for splitting nodes as described in Section 1.4.1, where every leaf is potentially split each iteration. In the first iteration, the root of each tree (which is also that trees only leaf) is split using all of the samples assigned to that tree. In subsequent iterations all of the samples are run through their respective trees until they reach a leaf, at which point they are mapped using a key that includes both the tree identifier and the leaf node identifier. As in regular tree induction, each reducer then finds the best split point for each leaf and passes the result to the main driver. Splitting continues until no nodes are split.

Boosting via MapReduce

In boosting, the samples are assigned weights that are adjusted after each stump is trained. For a MapReduce algorithm this presents a difficult challenge as there is no means to retain weights across iterations without using additional hacks such as a central database or recording the weights to the local disk. Using a central database would introduce more complexity than I was willing to consider, and recording the weights to the local disk would be problematic if advanced features such as data replication were enabled, as each iteration, the mapper could switch from one node to another that has the same samples, but not the weight data.

To solve the sample reweighting problem, each iteration all of the weights are recomputed from scratch. Everytime a mapper loads a sample from the training file, it is assigned a weight of $1/N$ where N is the number of training samples. It is then passed through each stump in the current model, after each model, the score is updated according to Equation 1.11. At first glance this seems like it would be a prohibitively expensive operation, however the time required for each sample is $O(h \cdot k)$ where h is the maximum height of a stump, k is the number of stumps in the current model. If one considers the latency involved in loading a weight from disk or over the network, each of which are typically on the order of milliseconds, the current model would have to be extremely large in either height or number of stumps to exceed 1ms (a typical network latency) to reweight a sample.

To build a boosted stump model using MapReduce, an additional step must be added. In order the determine the initial weights of the samples ($1/N$), the number of samples N must be determined. So before the first stump can be trained there is a preliminary MapReduce phase that simply counts all of the samples. Once the number of samples is know, training of the first stump progresses using only the first splitting method described in Section 1.4.1. Since boosted stumps typically have very few leaf nodes and the entire training set is used each iteration, this should provide more parallelism than the other splitting method. When the first stump is fully trained, typically by hitting its height limit, a reweighting MapReduce job is used to compute the total error in the training set (ϵ in Equation 1.11). Using the ϵ value, the new stump is assigned a weight and added to the

boosted model. Additional stumps are added to the model in the same way until either a maximum number of stumps is reached or ϵ reaches 0 (perfectly classifies test training dataset) or 0.5 (further improvement is impossible).

1.4.2 Logistic Regression via MapReduce

In the Single Feature Optimization MapReduce logistic regression algorithm [20], there are again two types of MapReduce jobs.

The first type of MapReduce job computes coefficients of $\vec{\beta}$ using Newton's method. In this MapReduce job, the mapper reads samples from the training dataset, computes the predicted value p for the sample and produces key/value pairs where the key is a attribute identifier and the value is the sample's class along with information additional information that is needed in the Newton's method computations (either histogram info or the value of $\vec{\beta} \cdot \vec{x}$). In the reducer, which gets all of the value for a particular attribute, a new coefficient for that element in $\vec{\beta}$ is computed using Newton's method as described in Section 1.3.8. The reducer then emits a key/value pair where the key is again the attribute identifier and the value is the new coefficient value.

The second type of MapReduce job scores the candidate coefficients computed in the first MapReduce. In this MapReduce job, the mapper reads samples from the validation dataset and computes the new predicted value using each of the models that would result by adding one of the candidate coefficients computed in the first MapReduce job. The mapper then emits a set of key/value pairs where the keys are attribute identifiers and the values are the actual class and the values predicted by the model with the corresponding candidate coefficient added. The reducer then takes all of the predicted values and actual values and computes the AUC for each candidate model, as described in Section 1.3.10. The reducer then emits a key/value pair where the key is the attribute identifier and the value is the candidate element of $\vec{\beta}$ and the AUC. The main drive then adds the coefficient that gives the best AUC to the current model.

The main driver uses both types of MapReduce jobs in sequence to generate sets of candidate coefficients, score those coefficients and update the model until the scores no longer improve.

1.5 Evaluated Run-time Systems

Several MapReduce run-time systems were considered. These systems were informally compared based on their relative run times of an implementation of an iterative machine learning algorithm (logistic regression) discussed in Section 2.2.1.

1.5.1 Hadoop

The first run-time system considered was Hadoop [50] (versions 0.20.2 and 0.21.0), which attempts to provides similar features to the MapReduce system used by Google [49], and is very popular due to its open source nature [51]. However, using a MapReduce job that only collected timestamp information during the map and reduce tasks, it was determined that Hadoop required at least 28 seconds to start each MapReduce phase, meaning it was not well suited for iterative algorithms such as the logistic regression method being considered which has two MapReduce phases per iteration an may need to run many iterations.

There is a machine learning library Mahout [52] which implements several machine learning algorithms using the Hadoop run-time system. However an examination of the machine learning algorithms provided indicated they are insufficient for the purposes of this study. While Mahout does have implementations of logistic regression and tree induction, the focus of this research is a comparison of the SFO [20] and PLANET [21] algorithms with particular emphasis on the intermediate models produced during training. Additionally, a comparison of SFO and PLANET with Mahout would be interesting, but it beyond the scope of this research.

1.5.2 HaLoop

The second run-time system investigated was HaLoop [53], which is a variant of Hadoop with extensions to facilitate faster iterative execution speed. However in testing the speed improvements of HaLoop over Hadoop were at best 20% for small training sets and HaLoop was far slower than Hadoop for larger training sets and for very large inputs HaLoop failed to finish. The main issue was related to HaLoop implementation of a local on-disk cache of key-value pairs, which for large input sets ran the compute nodes out of disk space.

1.5.3 Twister

A third MapReduce run-time system considered was Twister [54] (version 0.9). Based on a published report it was comparable to MPI in terms of execution speed [55] for iterative machine learning algorithms such as k-means. Twister also seemed like a good options for iterative algorithms such as SFO and tree induction. Testing using the logistic regression implementation showed that Twister was in fact much faster than Hadoop and HaLoop for iterative algorithms. Twister has been criticized for not having fault tolerance like Hadoop [51]. However, Twister can rerun an iteration if it fails. While this is not as fine grained as Hadoop's per task restarting, the length of a typical iteration in tree induction and logistic regression in Twister can be less than the 28 seconds of start up time per job (with several job per iteration) needed when using Hadoop. Thus rerunning an entire iteration is not a huge penalty compared to the start up costs of a Hadoop job.

1.5.4 GraphLab

Another run-time system is GraphLab [56], was specifically developed for machine learning on clusters. Since GraphLab uses an even more restrictive programming model than MapReduce, implementing algorithms found in the literature such as SFO and PLANET, which are specifically MapReduce, would have added a layer of complexity. It was not readily apparent if GraphLab allows for the implementation of anytime algorithms as needed for the proposed research.

1.5.5 Final Choice

Given the speed advantages of Twister over Hadoop and HaLoop, Twister was the obvious choice. GraphLab already has many algorithms already implemented, however it is not readily apparent if GraphLab could be instrumented and made to report intermediate models that this research required. Therefore, Twister was chosen as the fastest option that offered all of flexibility required.

1.6 Literature Review

1.6.1 Empirical Studies

Many empirical studies of machine learning techniques have been done using a wide variety of data sets over the last several decades. This section will discuss the more important ones in chronological order.

In 1989, Mingers examined the effects of different feature selection [31] and pruning [42] methods in tree induction. To examine the different feature selection methods four datasets were used. In that study, a total of 11 feature selection methods were tested, ten actual methods and a random one. All of the trees where pruned using Breiman's error complexity method. Each datasets was split into 70% training and 30% test nine different ways, the results over those nine unique splits where then combined. The results reported included the size (number of leaves) and accuracy of both the pruned and unpruned trees. One notable result is that the different selection measures didn't have any measurable effect on the accuracy of the trees, before or after pruning. The accuracy was also not significantly different than selecting attributes randomly. The selection measures did have an impact on the size of unpruned trees however.

To examine the efficacy of different pruning methods and possible interactions with goodness of split measures, Mingers [42] examined five pruning methods with four different goodness of split measures, for a total of 20 configurations. Each of five datasets were split nine different ways into 60% training, 20% validation (used in certain pruning algorithms), and 20% test. The results are reported as tree size (number of leaves) and accuracy. Critical value, error-complexity, and reduced-error pruning methods were reported to be among the best, and pruning typically improved accuracy by $\approx 25\%$. One particularly interesting comment on the topic of interactions between goodness of split measures and pruning method, "Equally noteworthy are those differences and interactions which were not significant." Specifically, the goodness of split measure doesn't affect the accuracy improvements during pruning. There was also no apparent interaction between goodness of split measure and the datasets, with respect to model accuracy. Both of these non-results are consistent with Minger's other study which indicated that the goodness of split measure only significantly affects the size of the tree.

While Mingers's results show the importance of pruning, and insignificance of feature selection during the training process, the small number of datasets used makes it hard to tell if these results are a valid generalization of the methods studied. It is also uncertain if these results will hold for much larger datasets.

Later in 1995, the StatLog project by King et al. [10] compared 17 different machine learning algorithms using twelve datasets. While this was a very ambitious project and lead to interesting results, those results are now quite dated and two decades of further improvements in machine learning require such studies to periodically be carried out. Also, compared to later studies twelve datasets is fairly small. The primary results of this study are that no one algorithm was all around better than the rest. However, depending on the metric used and certain properties of the datasets, some of the algorithms tended to do better than others.

Quinlan [57] primarily examined the effects of bagging and boosting as model improvement techniques. As such, it only used three algorithms, C4.5, C4.5 with bagging, and C4.5 with boosting. It did however use a larger number of datasets – 27 – in the comparison, many more than in previous studies. The only metric used for comparison was percent error. However a win-loss ratio over a 10-fold cross validation is also reported.

Provost and Fawcett [37] in 1997 pointed out that accuracy isn't the best metric to use in some cases, and proposed the use of ROC curves to express the quality of models. While ROC curves are useful, much of the machine learning community has focused on reporting only AUC, which is not as informative. In recent years this trend has been criticized by some researchers [58, 15].

Esposito et al. [59] examined various methods of pruning decision tree to reduce overfitting, six methods in all. These methods are: reduced error pruning (which was used in this study), pessimistic error pruning, minimum error pruning, critical value pruning, cost-complexity pruning, and error-based pruning. To compare the pruning methods, 14 datasets were used. The metric used to compare pruning methods was percent error.

Domingos and Pazzani [60] focused primarily on the circumstances in which bayesian classifiers are optimal, but also compared bayes to four other learners, including C4.5, using 28 datasets, but only using accuracy as a metric.

In 2000, Lim et al. [11] examined 33 algorithms using 16 different datasets. The 16 datasets were used both as is, and with additional noise added, giving a total of 32 total datasets. This study is notable in that it is one of the rare machine learning studies that reports both the quality of models produced, but also the time it took to train the models, including how the training time varies with training set size. It also presents a plot of model error rate versus model training time. The algorithm that overall performed the best in this study was C4.5.

Perlich et al. [7], in 2003, compared C4.5 and logistic regression using learning curves with 36 different datasets. To generate learning curves they varied the number of samples used for training. As the number of samples increased, the quality of the model typically improved. What is interesting in their results is that for some datasets, the learning curves for logistic regression and C4.5 cross. This indicates that for small numbers of samples, one algorithm is better, but for larger numbers of samples, the other one is better. So, picking one algorithm based on a limited number of samples may not be the best approach when the actual dataset is much larger. Unfortunately, this study did not report any run time information in their comparison other than to say that logistic regression sometimes took "an excessively long time to run even on moderately large datasets."

Provost and Domingos [28] examined a variant of decision trees called probability estimation trees (PETs), and found that many of the techniques used to improve accuracy in decision trees actually harms the probability estimations made by PETs. Instead they found Laplace correction is a better method for improving the accuracy of PETs. In their study they used 25 datasets to look at two different tree induction methods which then had bagging or Laplace correction applied to them for a total of five different tree building methods. The trees were then compared using AUC.

Caruana and Niculescu-Mizil [13], in 2006, performed one of the more recent comprehensive studies of machine learning algorithms. They looked at ten algorithms: support vector machines, neural nets, logistic regression, naive bayes, memory-based learning, random forests, decision trees, bagged trees, boosted trees, and boosted stumps. Many of the models were also calibrated using either Platt Scaling or Isotonic regression to improve their predictions by mapping model output probabilities to different values. To compare the algorithms they normalized and averaged 8 different performance metrics over 11 different datasets. The metrics used were accuracy, f-score, lift-score, AUC, average precision, precision/recall break-even point, root-mean squared error, and cross-entropy [61].

Ali and Smith [47], also in 2006, used eight algorithms over 100 datasets and used the results to build a set of rules based on measured properties of the datasets that can be used to help pick an algorithm for a given dataset. The algorithms used were all part of the [39] suite of learning algorithms using default settings.

Author	Year	Model Types	Data Sets	Train/Validate/Test	Unique Splits
Mingers [31]	1989	Decision Trees (11)	4	70%/0/30%	9
Mingers [42]	1989	Decision Trees (20)	5	60%/20%/20%	9
King [10]	1995	12 algorithms	12	dataset dependent	
Quinlan [57]	1996	Decision Trees (3)	27	90%/0/10%	10
Esposito [59]	1997	Decision Trees (6)	14	49%/21% (prune)/30%	25
Domingos [60]	1997	5 algorithms	28	67%/0/33%	20
Lim [11]	2000	33 algorithms	16×2	90%/0/10%	10
Perlich [7]	2003	Decision Trees (3) and Logistic Regression (5)	36	variable/0/25%-33%	10
Provost [28]	2003	Probability Estimation Trees (5)	25	67%/0/33%	20
Caruana [13]	2006	10 algorithms	11	5000/1000/rest	5
Ali [47]	2006	8 algorithms	100	10-fold cross validation	10

Table 1.6: Summary of empirical studies

Table 1.6 provides a summary of the various empirical studies. The first column lists the first author of the study along with a citation. The second column gives the algorithms used, or how may if there are too many to list. If the second column has a number in parenthesis, it indicates the total number of variations of the algorithm used. Many other empirical studies have also been performed, but have not been included here as they were too narrow in scope in terms of datasets used (e.g. LeCun *et al.* [62] and Cooper *et al* [63]).

All of these studies, with the exceptions of Lim *et al.* [11] and StatLog [10], only looked at the predictive abilities of models, and not how long it takes to train the models themselves. Very few looked at properties of the datasets that may have allowed certain algorithms to outperform other. In addition, none of them considered the quality of intermediate models produced during each iteration of training.

1.6.2 Meta-Learning

While there have been many attempts to used meta-learning to automate, or at least assist in the selection of machine learning algorithms based on the characteristics of datasets, most of them have focused on development of new algorithms or only providing general guidelines. In addition to these limitations of previous meta-learning studies, all of them focused on serial algorithms, and most did not consider the time it takes to build models.

The StatLog [10] project in the mid-1990s was one of the first large scale comparisons of machine learning algorithms undertaken. It compared 17 algorithms using 12 datasets. It concluded that while no single algorithm is best on all datasets, it did provide some general guidelines for when certain algorithms might do well. It did however stop short of attempting to build a model to do so.

MetaL [64] was a meta-learning project undertaken in Europe in the last 1990s and early 2000s. However instead of building models to do algorithm selection, data mining on learning data with the goal of generating better algorithms seemed to be the goal.

Kalousis [65], in 2002, attempted to use meta-learning to perform algorithm selection and provided

much of the basis for the dataset characteristic measures used in this study.

Ali and Smith [47], in 2006, used 100 datasets and eight algorithms, along with a variety of dataset characteristics to try to derive rules for when each algorithm might perform well.

Grąbczewski [45] provided a survey of meta-learning over the years, which mostly has focused on the design of new algorithms that are tailored for an individual dataset than picking from a set of existing algorithms.

1.6.3 Dealing with Large Datasets

There are a number methods that have been developed to deal with large datasets. One method is to sub-sample the dataset (i.e. use just a portion of the total dataset) and build a model from that sub-sample, methods based on this technique are described in Section 1.6.3. Alternatively parallel techniques, as described in Section 1.6.3, can be employed to speed up processing of large datasets. Additionally, a combination of sub-sampling and parallel techniques can be used, but that is outside the scope of this research.

Subsampling

Provost and Kolluri [66], in 1999, presented a comprehensive survey of techniques being employed at the time to deal with larger datasets for classification problems. One of their main conclusions was that improving sampling methods was an important direction for future research. They also concluded that there was little evidence to support the need for very large datasets for training. While both of these conclusions may have been valid at the time, the rapid increase in data collection over the last decade has produced huge datasets to be mined. Also, the work of Perlich *et al.* [7] in 2003, showed that sub-sampling datasets can adversely affect the quality of model produced by various algorithms. This suggests that if possible, using an entire dataset would be preferable.

Parallel Techniques

Chu *et al.* [18] investigated the speed of ten MapReduce machine learning algorithms on a 'multi-core' machine (they actually used a symmetric multiprocessor (SMP) system) with up to 16 cores, using ten different datasets. Since the purpose of this study was primarily to show speedup, they do not report any model quality metrics. The speedup results reported are somewhat mixed. Most algorithms had an average speedup around 14 on 16 nodes. However, for some dataset algorithm combinations only showed speedups of 4 to 6 on 16 nodes. These speedup numbers may not be indicative of performance on a distributed MapReduce environment, since and SMP machine will have much lower latency when moving data between cores over a network. Also, due to the non-distributed nature of this paper's methods, it would be limited to dataset sizes that fit onto a single compute node limiting its scale-ability.

Singh *et al.* [20] presented a MapReduce algorithm for logistic regression that uses forward feature selection to build a model one attribute at a time, which the authors call Single Feature Optimization (SFO), as described in Section 1.3.8. The SFO algorithm was compared to both IRLS and gradient descent (GD) on relatively small datasets to validate that it chooses features in approximately the same order as the other commonly used methods for feature selection. Strangely, model quality metrics (e.g. accuracy, AUC, error rate) are not reported in any of the comparisons. SFO was then run on much larger synthetic datasets to demonstrate the speedup that is achievable. However, by

using different datasets for verifying correctness and showing speedup, the speedup numbers are somewhat dubious. In fact the dimensions of the synthetic dataset were near optimal for showing speedup, meaning the speedup reported should be taken as an upper limit, and not a typical expected value. The logistic regression algorithm used in this study is based on the SFO algorithm, however the version used in this research included an extension for non-binary attributes described in Section 1.3.9.

Panda *et al.* [21] presented a MapReduce for top down Tree Induction that can also train ensembles (e.g., bagging), which the authors call PLANET. The single tree induction algorithm is described in Section 1.3.4. Much like the SFO paper, the results presented for PLANET focuses primarily on total execution time and does not report model quality metrics. No speedup numbers are reported due to not running on a single node, which is a common problem with large scale datasets, as it is sometimes impossible to fit larger datasets onto single compute node. This suggests that in the future metrics other than classic speedup will be needed to show the scale-ability of algorithms that can only be run in a parallel context. The execution times reported do indicate that the algorithm should demonstrate reasonable speedup if the proper measurements are taken. However they only used one dataset, the Ads dataset from the University of California Irvine (UCI) database, in their testing, leaving it unclear if/how the run-time is affected by different datasets. The tree induction algorithm used in this research is based on the PLANET algorithm, but will be used to build probability estimation trees instead of classification trees.

Low *et al.* [56] present a system GraphLab which is a C++ API that uses graph based memory model along with update functions that have limited access to data in the graph to train models in a highly parallel fashion. The data model consists of a static graph where program state can be stored either in nodes or on edges. However the update function will be passed a single node, and can only access data on edges incident on that node and nodes on the other end of the edge. This allows for the update function to be applied to any node that isn't adjacent to another node that is currently being updated. GraphLab was compared to Hadoop and MPI implementations for solving three common machine learning problems: Netflix Movie Recommendation, Video Cosegmentation (CoSeg), and Name Entity Recognition (NER). These tests showed that GraphLab's run times are comparable to highly optimized MPI code, and 40 to 60 times faster than Hadoop/Mahout in some tests. The speedup numbers were mixed. Using 64 nodes, NER only achieved a speedup of 3. Netflix recommendation was better, but only about 4.5. The CoSeg algorithm got the best speedup, but was still only about 10. In addition to the run time and speedup results, price/performance curves were also generated for running GraphLab on Amazon's EC2 clusters, which show that GraphLab is far better than Hadoop in terms of cost to run these algorithms. While this paper doesn't discuss logistic regression or tree induction, those algorithms have been implemented as part of the GraphLab project as well.

1.6.4 Algorithm Selection and Hyper-parameter Optimization

The algorithm selection problem [67] has been studied for the last four decades [68]. In recent years, there has been a lot of work in the area of algorithm selection and hyper-parameter optimization, often referred to as meta-learning. This section provides a summary of recent work in these areas.

In the area of algorithm selection a number of approaches have been examined. In many cases the characteristics of datasets are measured and used as features to build a meta-model. In other cases the problem is treated as a search problem where the performance of the individual algorithms on a dataset are compared to that of other datasets. Abdulrahman *et al.* [74, 75, 76] proposed an iterative system (A3R) that selects algorithms by training models on small datasets to determine

which algorithms might perform better than the previously selected best algorithm. Nural *et al.* [77] used 114 datasets to build models to predict which regression algorithm among 15 tested would perform well on the task given the characteristics of the datasets.

In the area of hyper-parameter optimization, much of the work has been focused on ways to find optimal hyper-parameters more quickly, either by reducing the search space, or reducing the time needed to sample points in the search space. Wistuba *et al.* [72] proposed an improvement to sequential model-based optimization [73] for hyper-parameter optimization that prunes the search space to avoid searching areas that likely will not yield improvements. Sanders *et al.* [78] examined the problem of if hyper-parameter optimization is even necessary in some cases and found that unless the models produced by default parameters is perfect, hyper-parameter optimization is probably needed. Eggensperger *et al.* [80], created a set of benchmark datasets that can be used in place of more computationally intensive benchmark datasets for hyper-parameter optimization. Sousa *et al.* [81], similarly examined the problem of creating new datasets, however instead of speeding up the hyper-parameter optimization process are modifications of existing datasets that give better coverage of the dataset characteristic space to produce better algorithm selection models. Joy *et al.* [82] presented an approach for hyper-parameter optimization where the data is split into small chunks and Bayesian optimization is done on each chunk, in parallel. The results are then combined into hyper-parameters that should work for the overall dataset. Horn *et al.* [83] explored a means by which hyper-parameters can be optimized towards multiple goal simultaneously. van Rijn *et al.* [84] empirically examined the effects of various hyper-parameters on the performance of algorithms.

Additionally there has been some work in the area of combining algorithm selection and hyper-parameter optimization. Thornton *et al.* [69] provided an extension to the popular Weka [39] utility that uses Bayesian optimization [70] techniques to both select an algorithm among the ones provided by Weka and optimize the hyper-parameters. Smith *et al.* [71] proposed a system to select algorithms and hyper-parameters using collaborative filtering using a landmark-like approach to predicting similarity of algorithms instead of measured dataset characteristics.

Smith *et al.* [79] created the *machine learning results repository* (MLRR), which provides meta-datasets for use in reproducible meta-learning research.

Brazdil *et al.* [48, 85] provide a more detailed overview of many of the recent developments discussed above.

However most of this recent work in algorithm selection and hyper-parameter optimization has focused on sequential algorithms. The hyper-parameter space for parallel algorithms can be much larger with many more trade-offs in terms of model quality, reproducible, and speed.

Chapter 2

Overall Experimental Design

2.1 Goal of this work

The primary goal of this research was to train a model to predict which algorithm will produce a high quality model given a dataset's characteristics, a number of nodes, and a time limit. To answer this question, a variety of datasets are needed and models must be trained on a varying number of nodes recording the time needed to train each model. The datasets chosen for a preliminary proof of concept were chosen based on the results of Perlich *et al.* [7] such that some of them should favor logistic regression and other will favor tree induction.

To examine the differences in parallel algorithms for machine learning, two MapReduce algorithms for supervised machine learning, SFO [20] and PLANET [21], found in the literature, were implemented. The MapReduce run-time systems that were considered are discussed in Section 1.5 and the algorithms that have been implemented are discussed in Section 2.2.

2.2 Implemented Algorithms

All of the algorithms used in this study were implemented using the Twister [54] MapReduce API and run-time system.

2.2.1 Logistic Regression

The first algorithm implemented is a MapReduce logistic regression algorithm single feature optimization (SFO) [20], which uses forward feature selection. This algorithm was further enhanced to allow non-binary attributes by using a two-dimensional set of bins, as described in Section 1.3.9.

2.2.2 Tree Induction

The tree-induction algorithm PLANET [21] was also implemented. The tree inducer also implements Reduced Error Pruning [43, 42] to reduce overfitting. Boosting [33] and bagging [32] were also implemented, but do not allow for pruning.

2.3 Runtime Infrastructure

2.3.1 The `mpi_wrapper`

The Twister MapReduce run-time system makes several assumptions about a user's access to the nodes on which it will run. Among these assumption is that the user will be able to both log directly into each compute node via secure shell (`ssh`) and copy files directly to each node via secure copy (`scp`) [86]. One of the clusters that was used does not allow this type of direct access to the compute nodes by regular users. As a workaround for the lack of direct access to the compute nodes, the `mpi_wrapper` was created. The `mpi_wrapper` is a C program that uses the MPI [87] API, MPI runtime, and a set of shell scripts, that together manage the transfer of files as well as starting and stopping of both the Twister runtime system and individual MapReduce jobs. In addition to starting the individual MapReduce jobs, the `mpi_wrapper` also monitored the progress of MapReduce jobs via a heartbeat file that is periodically updated within each MapReduce job.

Before any jobs can be run, the `mpi_wrapper` needs to install and start the run-time system. The run-time system consists of two components, ActiveMQ [88] which provides an overlay network that is used by the other component, Twister, which provides the MapReduce environment and uses the overlay network for communication.

Once the runtime system is started, it checks an input directory, for learning task files. So long as task files are found in the specified directory they are executed in chronological order based on the files' timestamps. It is possible that a given task may produce more tasks. If any tasks fail to produce valid output, the runtime system is restarted and the timestamp of the failed task is updated to move it to the end of the line. A detailed description of what happens within each job is provided in Section 2.3.5. If sufficient consecutive tasks fail, or all of the task files are exhausted, the Twister runtime system is shut down and the `mpi_wrapper` exits.

2.3.2 Open Grid Scheduler

The cluster that required the creation of the `mpi_wrapper` used the Open Grid Scheduler (OGS) [89] to queue, start, and stop jobs. OGS works by dividing the cluster into a set of slots (in this case a slot was a single compute core), and assigning jobs to slots. Whenever there are sufficient free slots for a waiting job, a job that can fit onto the system is assigned to those slots and started. If a job completes, or is otherwise terminated, the slots it had been using are freed up for any waiting jobs to use. It is also possible in OGS to have job dependencies where one job cannot start until another job (or set of jobs) has finished.

2.3.3 SHIELD versus Hydra

Unfortunately, there is a bug that appears to be in an OGS component named hydra, that caused random crashes, taking the MPI job it had launched with it. To counter this random crashing, a system called SHIELD was developed (named for the fictional organization that fights against Hydra in the Marvel superhero universe [90]). SHIELD is a script, that is run once when the `mpi_wrapper` starts, that checks to see if there are remaining learning tasks to be completed. If there are any remaining tasks, SHIELD makes use of the dependency system and queues a new identical MPI job that will be allowed to start when the job that ran SHIELD terminates. That way, if the current

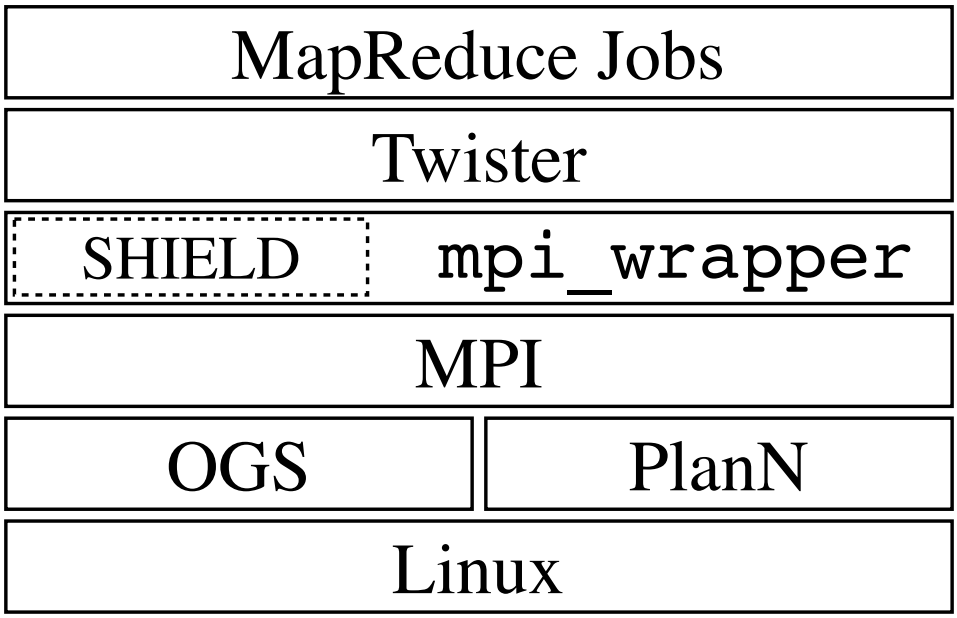

Figure 2.1: Software layers involved in running a MapReduce job.

job dies for any reason, the other job will be in the queue waiting to take over. If there is no work to be done, SHIELD does nothing, and the `mpi_wrapper` terminates immediately.

2.3.4 Plan N

In order to make more efficient use of a second cluster that did not have a queuing system capable of assigning jobs to nodes, a custom job scheduler was needed. The Plan N scheduler is a very rudimentary scheduler that subdivides the cluster into chunks of equal numbers of cores. MPI jobs can be assigned to individual chunks, then as jobs complete, start new jobs to replace them. Between runs, the number of cores in the chunks can be changed to meet the needs of the experiments being run.

Figure 2.1 gives an overall picture of how `mpi_wrapper` fits into the software stack used. Each node is running Linux as the operating system. The clusters used for this project either ran Open Grid Scheduler (OGS) [89] or the custom built PlanN scheduler, which were responsible for scheduling, starting, and stopping `mpi_wrapper` jobs running on the cluster. The `mpi_wrapper` would then manage the Twister runtime system and work through the set of learning tasks it had been given.

2.3.5 Life-Cycle of a Model Training Job

A learning task file (or sub-job file) is a shell script file that can perform four essential tasks. Which task is performed on a given run is determined by command line options. These four tasks are

prelaunch, *launch*, *score*, and *retire*. Within each sub-job there is a unique set of variables that specify the hyper-parameters to be used for that particular training task.

The prelaunch task prepares the input dataset by splitting it, if necessary, and copying it to the appropriate directories on the compute node. In stock Twister, the *prelaunch* task would be handled by a provided shell script that uses `scp` to copy the files. The *launch* task starts up the actual MapReduce job with whatever hyper-parameters are needed for that sub-job. The *score* task then goes through all of the models produced during training and scores them using the test portion of the dataset. Finally, the *retire* task examines the output of the scoring task, and if it has finished correctly, moves the unscored models file, the scored models file, and the sub-job file to a safe location in a shared NFS directory.

2.4 Datasets

To train a model that will use the characteristics of datasets as attributes, a collection of datasets is needed for training. This section describes both the datasets that were used for training the algorithm selection model and the datasets that were used to test that model.

Table 2.1 summarizes the datasets used and gives their most basic characteristics. For each dataset the total number of attributes, samples, and classes, as described in Section 1.3.1, are given, as well as the target class for positive samples and the portions of this study the dataset was used. For datasets that have non-numeric (e.g. nominal) attributes, the number of attributes shows both the number with the non-numeric attributes and the number after conversion to all numeric (except the class) attributes, if the number changes in the conversion. The class is counted as one of the attributes. The "uses" column indicates which portions of this project the dataset was used in. A full listing of measured characteristics for each dataset can be found in Appendix B.

2.5 Algorithm Selection Training Datasets

The datasets listed in this section are the datasets that were used to train the algorithm selection model.

2.5.1 Ads

The Ads dataset is intended to train models to detect advertisements on websites. The attributes describe the dimensions of an image, as well was words that appear in the link associated with the image. The problem to be learned is whether or not the image is an ad. There are 3,279 samples comprised of 1,558 attributes, three real attributes that describe the size and aspect ratio of the image, the rest are binary and indicate if certain words or phrases appear in the link. The classes for this dataset are "ad" and "noad", the positive class used was "ad". The dataset was included primarily due to its number of attributes. This dataset is part of the UCI database [4] under the name "Internet Advertisements".

Name	Total Attributes	Samples	Classes	Target Class	Uses
Abalone	9 / 11	4177	29	9	test
Ads	1559	3279	2	ad	train
Adult	15 / 105	32561	2	$>$ 50k	train
Botswana	146	3248	14	9	test
Covertype	55	495141	2	1	train
Credit	16 / 44	653	2	+	train
Example	5 / 8	1000	2	1	train
Example2	5 / 8	1000	2	1	train
German	21 / 62	1000	2	2	test
Heart	14 /16	270	2	1	test
IndianPines	201	10249	16	11	test
IntCensor	75 / 467	10108	4	1	train
IntShopping	104 / 258	7219	10	Never	train
Intrusion	42 / 123	4898431	23	normal	train
KSC	177	5211	13	13	test
Mushrooms	22 / 116	8124	2	e	$2^k r$, train
PaviaU	104	42776	9	2	test
Pima	9	768	2	1	$2^k r$, train
SalinasA	205	5348	6	12	test
Thrombin	139352	1909	2	A	train

Table 2.1: Datasets used

2.5.2 Adult

For the Adult dataset, the problem is to determine if a person's income is >$50k per year. The 14 attributes are taken from census data with a total of 48,842 samples. There are two classes, "≤50k" and ">50k", the positive class used was ">50k". This dataset was included due to unusual results reported by Perlich *et al.* [7], where AUC and accuracy disagreed on which algorithm was better. This dataset is part of the UCI database [4] under the name "Adult".

2.5.3 Covertype

The Covertype dataset describes the types of trees that can be seen at the top of a forest based on cartographic data. The 54 attributes mostly contain information about how the sun will hit each sampled point and soil type data. There are seven classes in the original dataset, however two of the classes make up 85% of the samples. So, just as Perlich *et al.* [7] did, all but those two classes were filtered out. This reduced the number of samples from 581,012 to 495,141, and the two classes, "Spruce-Fir" and "Lodgepole Pine". This dataset was included due to its large number of samples and the fact that Perlich *et al.* reported tree induction and logistic regression each outperforming the other under certain circumstances. This dataset is part of the UCI database [4] under the name "Covertype".

2.5.4 Credit

The Credit dataset represents a problem relating to credit card applications. The 15 attributes have had their original meanings obscured by meaningless symbols, the classes are simple "+" and "-". With only 690 samples, which becomes 653 after samples with missing values are removed, this is one of the smaller datasets used. It was included due to having a variety of attribute types. This dataset is part of the UCI database [4] under the name "Adult".

2.5.5 Example

The Example dataset is a synthetic dataset created from the tree model shown in Figure 1.1, with $\sim 5\%$ of the class labels randomly reversed. The learning problem in this dataset is to build a model that can predict if a student should walk (positive) or drive (negative) to campus on a given day based on the weather conditions that day. There are 1000 samples and four attributes that describe the wind speed, wind direction, temperature, and precipitation. This problem was inspired by an example problem described by Quinlan [29] and was created primarily to provide examples for Chapter 1. It was included as part of the actual experiments to test a hypothesis that synthetic data from a tree model will be more easily learned by a tree based learner than logistic regression. This dataset is part of the digital appendices for this dissertation.

2.5.6 Example2

The Example2 dataset is similar to the Example dataset. The attributes are identical to Example. The randomly assigned values for each sample are different and the labels are assigned based on the logistic regression model given in Section 1.3.3. As in Example, $\sim 5\%$ of the samples have had their class labels reversed. This dataset is part of the digital appendices for this dissertation.

2.5.7 IntCensor

The IntCensor dataset is based on a 1997 survey of internet users, conducted by Georgia Tech [5], regarding their opinions on censorship on the internet. There are 10,108 samples, where the 75 attributes represent the answers to questions on the survey. There are four classes, which represent the options for "Opinions on Censorship". The class chosen to be the positive class was "1". This dataset is available from Georgia Tech at `https://www.cc.gatech.edu/gvu/user_surveys/survey-1997-10/datasets/final_general.repl`.

2.5.8 IntShopping

The IntShopping dataset is based on the same survey as IntCensor, however IntShopping focuses on the online shopping habits of those surveyed. The 7,219 samples consist of 104 attributes which represent the answers to the survey questions. The positive class for model training was "Never". This dataset is available from Georgia Tech at `https://www.cc.gatech.edu/gvu/user_surveys/survey-1997-10/datasets/final_use.repl`.

2.5.9 Intrusion

The Intrusion dataset is from the 1999 KDD Cup[1]. Each sample represents a network connection, where the 42 attributes describe aspects of the connection that may be useful in determining if the connection is malicious. The class labels for the samples are either "normal", or the name of a type of malicious attack, for a total of 23 classes. The models trained on this dataset were attempting to identify "normal" connections. Overall, this dataset has 4,898,431 samples. There is also an additional test dataset with around 2 million samples that is intended to test models trained on this dataset, however that testing data was not used. This dataset was chosen because of its larger number of samples. This dataset is part of the UCI database [4] under the name "KDD Cup 1999".

2.5.10 Mushrooms

The Mushrooms dataset represents the problem of determining if a mushroom is edible or poisonous. The 22 attributes describe the color, smell, and structural components of the mushroom, with a total of 8,124 samples. The classes are "e" for edible and "p" for poisonous. This dataset was chosen for inclusion because it is well known to be an easy to learn dataset. This dataset is part of the UCI database [4] under the name "Mushroom".

2.5.11 Pima

The Pima dataset (sometimes referred to as the Pima Indians or Diabetes dataset) consists of 768 samples, each with eight attributes. The attributes describe medical information regarding a patient, and the problem is to predict if that patient has diabetes or not. The classes are "1" (diabetic) and "0" (not diabetic). Missing values in this dataset were represented with the value "0". Rather than remove the missing values or replace them with median or mean values, they were left as zero, a common mistakes [91], that was discovered after all data collection was completed. This dataset was

[1] `http://www.kdd.org/kdd-cup/view/kdd-cup-1999/`

included due to its small size, and because Perlich *et al.* reported that logistic regression produces better models for it than tree induction. This dataset had previously been available from the UCI database [4] under the name "Pima Indians Diabetes", but has been removed.

2.5.12 Thrombin

The Thrombin dataset comes from the 2001 KDD Cup[2], and consists of 1,909 samples, with 139,351 binary attributes and two classes. Each sample represents a chemical, and the attributes describe the structure of the chemical. The two classes, "active" and "inactive", indicate if that chemical is able to bind to molecules involved in the clotting of blood. Of the 1,909 chemicals in the dataset, only 42 are active. The original purpose of this dataset was not to identify which chemicals could bind, but instead to identify which attributes (i.e., chemical structures) are important in the binding process. This dataset was included due to its very large number of attributes.

2.6 Algorithm Selection Test Datasets

The datasets listed in this section are the datasets that were used to test the algorithm selection model.

2.6.1 Abalone

The Abalone dataset's goal is to simplify estimating the age of abalone. The eight attributes, for each of the 4,177 samples, represent easily measured dimensions of an abalone. The class is the number of rings in each abalone sampled, which can be used to calculate the age. The number of rings chosen as the positive class was "9", as it was the largest class. This dataset is part of the UCI database [4] under the name "Abalone".

2.6.2 Botswana

The Botswana dataset is based on satellite imagery taken of the Okavango Delta, in Botswana. Each of the 3,248 samples represents a 30 meter by 30 meter pixel. Within each pixel there are 145 attributes that represent a subset of the original 242 bands captured by the NASA EO-1 satellite. The 14 classes are numbered 1 through 14 to indicate different types of land cover. Class "9" was chosen among these, as it was the largest. This dataset is from the Computational Intelligence Group at the Basque University `http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes` as the "Botswana" dataset.

2.6.3 German Credit

The German dataset attempts to determine if applications for credit should be approved or denied. The 20 attributes provide financial and demographic data for 1000 people (samples), and the labels "1" and "2". The good class (1) was chosen as the positive class. This dataset also provides cost matrix (an alternative scoring system for misclassifications), but that was not used. This dataset

[2] `http://www.kdd.org/kdd-cup/view/kdd-cup-2001/`

was included due to its mix of attribute types, an alternative numeric only version was also provided but was not used. Instead, the binorm version of the dataset was converted the same way as all other datasets. This dataset is part of the UCI database [4] under the name "Statlog (German Credit Data)".

2.6.4 Heart Disease

The Heart dataset is intended to detect heart disease using basic demographic data and the results of certain medical tests, for a total of 13 attributes. For the 270 patients sampled, the class labels are either "1" and "2" indicating heart disease and no heart disease respectively. This dataset also provides a cost matrix (an alternative scoring system for mistakes), but that was not used. This dataset was included due to its mix of attribute types. This dataset is part of the UCI database [4] under the name "Statlog (Heart)".

2.6.5 Indian Pines

The IndianPines dataset is based on satellite imagery of a site in northwestern Indiana. Each of the 10,249 sampled pixels contains 200 bands which are represented as numeric attributes. The 16 classes indicate if a given pixel is a type of structure, woods, or a specific type of crop. The positive class was chosen to be class "11" (Soybean-mintill), because it has the most samples. This dataset is from the Computational Intelligence Group at the Basque University `http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes` as the "Indian Pines" dataset.

2.6.6 Kennedy Space Center

The KSC dataset is based on aerial imagery of the Kennedy Space Center. It consists of 5,211 sampled pixels each with 224 bands, of which 176 bands were used to create attributes for each sampled pixel. The samples are each assigned to one of 13 classes, numbered 1 through 13, indicating types of land cover. The positive class chosen was class "13" because it contained the most samples. This dataset is from the Computational Intelligence Group at the Basque University `http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes` as the "KSC" dataset.

2.6.7 Pavia University

The PaviaU dataset is based on aerial imagery of Pavia in northern Italy. The 42,776 sampled pixels each consist of 103 bands. Each sample is assigned to one of nine classes numbered one through nine and indicate what type of building material or vegetation is seen in that pixel. Class "2" was chosen to be the positive class, as it was the largest. This dataset is from the Computational Intelligence Group at the Basque University `http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes` as the "Pavia University" scene.

2.6.8 Salinas-A

The SalinasA dataset is based on aerial imagery taken of the Salinas Valley in California. The 5,348 sampled pixels each consist of 205 bands. Each sample is assigned to one of six classes indicating a type of crop in its corresponding pixel. The class that was chosen for the positive class was class four, because it contained the most samples. This dataset is from the Computational Intelligence Group at the Basque University `http://www.ehu.eus/ccwintco/index.php?title=Hyperspectral_Remote_Sensing_Scenes` as the "Salinas-A" scene.

2.7 Ethical Concerns

In recent years there has been increasing concerns about the use of machine learning techniques in ways that have started to result in serious consequences for real people [92]. To help bring attention to ways in which machine learning can be misused, a recent controversial study attempted to train a model to detect a person's sexual orientation based on a photo of their face [93].

One application of machine learning that has garnered increasing attention has been the use of these techniques in the criminal justice system [94, 95]. It is tempting to think that if human decision makers are replaced by machines, that the implicit biases that humans hold will be eliminated in those decisions. However, this blind faith in machines making decisions has lead to a sort of computational bigotry, where machines are making racist or otherwise prejudiced decisions, that humans accept as legitimate without question. Some of the systems used in the criminal justice system have been shown to predict, incorrectly, lower recidivism rates for whites than non-whites. These errors may not be due to flaws in the algorithms, but may instead be due to either the biases in the training data, or the resulting models being used to answer questions that are subtly different than what the data they were trained were meant to answer [96].

While none of the datasets used in this study tackle such controversial topics as sexual orientation or recidivism, the Pima dataset has become somewhat controversial recently [97]. It has also been removed from the UCI database since originally obtained for this project. The webpage that used to contain the dataset does not offer an explanation for the removal, other than the simple message, "Thank you for your interest in the Pima Indians Diabetes dataset. The dataset is no longer available due to permission restrictions.". This removal makes one wonder what lead to such a decision after nearly three decades of availability. For these reasons, it is unfortunate, but unavoidable, that this work is partially based on datasets (e.g, Adult, German, Pima, Heart) that are intended to classify people.

2.8 Data Collection and Processing

The process of collecting data, building an algorithm selection model, and testing that model, was divided into four phases. The first phase (Chapter 3) examined the hyper-parameters for each learning algorithm to determine their relative effects on learning time and model quality with the goal of eliminating any insignificant hyper-parameters. The second phase (Chapter 4) trained models using a variety of hyper-parameters to find a set of hyper-parameters that produced the highest scoring models. The third phase (Chapter 5) reran select iterations of the model training processes on various numbers of compute cores to estimate the parallel performance of each algorithm on each dataset. The fourth and final phase (Chapter 6) combined all of the data from phases two and

three to build an algorithm selection model. That model's predictions were then compared to actual results from training models on datasets that were not used during phases two and three (i.e., the datasets described in Section 2.6).

Chapter 3

Phase 1: Determining Significant Factors

The first problem to be solved in this study was to determine which hyper-parameters significantly affected the model quality and run-time of each algorithm.

Due to the large number of factors and the resulting exponential growth of experimental complexity, testing the effects of multiple values for all hyper-parameters for even a single dataset is intractable. Therefore doing so for multiple datasets is out of the question. Thus a much smaller experiment was performed, to determine which hyper-parameters warrant further study, to rule out hyper-parameters that have statistically insignificant effects on the model quality and model training times.

In order to determine the significant hyper-parameters $2^k r$ analysis [98] was performed using two smaller datasets. Of the datasets that were chosen, one has a small number of features and samples, but is notorious for being very difficult to build a model for (Pima) and another dataset that is a bit larger but known to be fairly easy to build models for (Mushrooms).

3.1 $2^k r$ Experimental Design

In a 2^k factorial experimental design the goal is to determine how much (as a percentage) of the total variability of an output value (e.g. program run time) is caused by each of the k factors as well as interactions between combinations of those k factors. Additionally, if the same experiments are repeated for a total of r iterations ($r \geq 3$), as in $2^k r$, it is possible to perform significance tests on the variability to rule out insignificant factors.

In a $2^k r$ design it is assumed that the response value that you are interested in is a weighted sum of each of the factors and their interactions. For a three factor design, y_{ij}, with experimental configuration i and repetition j is represented as

$$y_{ij} = q_0 + q_1 x_A + q_2 x_B + q_3 x_C + q_4 x_A x_B + q_5 x_A x_C + q_6 x_B x_C + q_7 x_A x_B x_C + e_{ij} \tag{3.1}$$

where q_0 though q_7 are the variability due to each subset of the three factors, and x_A, x_B, and x_C are -1 or 1 indicating if the high or low level is used in configuration i, and e represents the experimental error.

A sign table as show in Table 3.1 is used to compute the variation due to each factor or combination of factors. In the sign table, the first column, I, is all 1s. After the first column of all ones, the pattern is identical to a little-endian k-bit counter enumerating all values from zero to 2^k, where the zero and one bits correspond to negative and positives ones respectively. The second column, A, alternates -1 and 1 every row. The next column, B, alternates -1 and 1 at half the rate, two -1s followed by two 1s. The third column, C, again halves the rate to four -1s followed by four 1s. This

Sign Table ($\mathbf{X}$)								Measured values		Error
I	A	B	C	AB	AC	BC	ABC	y	$\bar{y}$	e
1	-1	-1	-1	1	1	1	-1	(y_{11}, y_{12}, y_{13})	$\hat{y_1}$	(e_{11}, e_{12}, e_{13})
1	1	-1	-1	-1	-1	1	1	(y_{21}, y_{22}, y_{23})	$\hat{y_2}$	(e_{21}, e_{22}, e_{23})
1	-1	1	-1	-1	1	-1	1	(y_{31}, y_{32}, y_{33})	$\hat{y_3}$	(e_{31}, e_{32}, e_{33})
1	1	1	-1	1	-1	-1	-1	(y_{41}, y_{42}, y_{43})	$\hat{y_4}$	(e_{41}, e_{42}, e_{43})
1	-1	-1	1	1	-1	-1	1	(y_{51}, y_{52}, y_{53})	$\hat{y_5}$	(e_{51}, e_{52}, e_{53})
1	1	-1	1	-1	1	-1	-1	(y_{61}, y_{62}, y_{63})	$\hat{y_6}$	(e_{61}, e_{62}, e_{63})
1	-1	1	1	-1	-1	1	-1	(y_{71}, y_{72}, y_{73})	$\hat{y_7}$	(e_{71}, e_{72}, e_{73})
1	1	1	1	1	1	1	1	(y_{81}, y_{82}, y_{83})	$\hat{y_8}$	(e_{81}, e_{82}, e_{83})
$2^k q_0$	$2^k q_1$	$2^k q_2$	$2^k q_3$	$2^k q_4$	$2^k q_5$	$2^k q_6$	$2^k q_7$	Total		
q_0	q_1	q_2	q_3	q_4	q_5	q_6	q_7	Total/2^k		

Table 3.1: Example sign table for $2^k r$

pattern continues until the final factor, which will always have the first half be -1s and the second half will be 1s.

The columns labels with multiple letters contain the product of the values in the corresponding columns. The column AB contains the products of the values in A and B. Column BC contains the products of columns B and C. Column ABC contains the products of columns A, B and C.

The measured values for each experimental trial go into the y column. In $2^k r$ the r repetitions for the same configuration are averages to give an estimated actual values $\hat{y}$.

The variation due to each factor or combination of factors, the sum of each $\hat{y}$ value multiplied by the corresponding value in the sign table is give by

$$q_j = \frac{1}{2^k} \sum_{i=1}^{2^k} X_{i,j} \cdot \hat{y}_i \tag{3.2}$$

where $X_{i,j}$ is the value at row i and column j in the sign table, and can be rewritten in matrix-vector multiplication notation

$$\frac{1}{2^k} \left(\mathbf{X}^T \cdot \vec{y} \right) = \vec{q} \tag{3.3}$$

.

3.1.1 Percent of Variation

For each factor, q_j, the sum of square variation, SS_{q_j}, is

$$SS_{q_j} = 2^k \cdot r \cdot q_j^2 \tag{3.4}$$

. The sum of squared error (SSE) is

$$SSE = \sum_{i=1}^{2^k} \sum_{j=1}^{r} e_{ij}^2 \tag{3.5}$$

. The total sum of squares

$$SST = \left(\sum_{j=1}^{2^k} SS_{q_j} \right) + SSE \tag{3.6}$$

is the sum of all the sum of square variation terms SS_{q_j} plus an error term SSE.

The percent of variation due to any individual factor or combination of factors is the ratios of the sum of squared variation to the total sum of squares,

$$percent_{q_j} = 100\% \cdot \frac{SS_{q_j}}{SST} \tag{3.7}$$

.

3.1.2 Statistical Significance of Factors

The error for each experimental configuration i and repetition j is

$$e_{ij} = y_{ij} - \hat{y}_i \tag{3.8}$$

.

To compute a confidence interval, the standard deviation of the variability due to random error is needed for each factor. The standard deviation for the errors is

$$s_e = \sqrt{\frac{SSE}{2^k(r-1)}} \tag{3.9}$$

which is used to determine the standard deviation for each factor

$$s_{q_0} = s_{q_1} = q_{q_2} = \cdots = \frac{s_e}{\sqrt{2^k r}} \tag{3.10}$$

.

In addition to the standard deviation, the t-value is needed. Given a normal distribution with a mean of zero and standard deviation of one, such that the total area under that distribution from $-\infty$ to ∞ is equal to one, the t-value gives the point where the fraction of area under the distribution to the left of that point is equal to the confidence level. While t-values are typically looked up in a t-value table, the values needed to compute a t-value, and therefore to look up the correct value in a table are the level of confidence as a percentage (p), and the number of degrees of freedom (n). Some t-value tables use α instead of p, in which case $\alpha = 1 - p$. A typical p value is 0.95, or $\alpha = 0.05$. However for a two sided confidence interval, α is divided by two. For a two sided confidence interval with 95% confidence, $p = 0.975$ or $\alpha = 0.025$. The degrees of freedom is $2^k(r-1)$. For a three factor design with three repetitions with 95% confidence, $p = 0.975$, and $t = 2^3(3-1) = 8 \cdot 2 = 16$, the t-value would then be 2.120.

Using the amount of variation due to a factor (q_i), the standard deviation (s_e), and the t-value (t), the lower and upper bounds of the confidence interval are

$$q_i \pm s_e \cdot t \tag{3.11}$$

. If the confidence interval for a factor includes zero within its range, that is if $s_e \cdot t > q_i$, the factor is considered to be statistically insignificant.

I	A	B	C	AB	AC	BC	ABC	y	$\bar{y}$	error
1	-1	-1	-1	1	1	1	-1	(2.18, 2.33, 2.15)	2.219	(-0.04, 0.11, -0.07)
1	1	-1	-1	-1	-1	1	1	(2.03, 1.87, 1.91)	1.938	(0.09, -0.06, -0.03)
1	-1	1	-1	-1	1	-1	1	(1.79, 2.08, 1.98)	1.949	(-0.16, 0.13, 0.03)
1	1	1	-1	1	-1	-1	-1	(1.85, 2.08, 1.92)	1.951	(-0.10, 0.13, -0.03)
1	-1	-1	1	1	-1	-1	1	(1.96, 2.17, 2.11)	2.079	(-0.12, 0.09, 0.03)
1	1	-1	1	-1	1	-1	-1	(1.96, 2.00, 2.11)	2.023	(-0.06, -0.02, 0.08)
1	-1	1	1	-1	-1	1	-1	(1.95, 2.11, 2.17)	2.076	(-0.13, 0.03, 0.09)
1	1	1	1	1	1	1	1	(2.06, 1.89, 2.04)	2.000	(0.06, -0.11, 0.04)
16.23	-0.41	-0.28	0.12	0.26	0.15	0.23	-0.30	Total		
2.03	-0.05	-0.04	0.02	0.03	0.02	0.03	-0.04	Total/8		
26721%	16.96%	8.14%	1.49%	6.94%	2.18%	5.42%	9.34%	% of variation		

Table 3.2: Sign table with example data.

3.1.3 $2^k r$ Example

To illustrate the $2^k r$ experimental design, this section gives an example based on data collected in this study.

In this example, data from learning a tree induction model for the Mushrooms dataset is used. There are three factors, A) minimum number of samples needed to split a leaf node B) maximum tree height C) maximum iterations, and each configuration was run for three repetitions. The response value being examined is model training time (in seconds), the measured values are given in Table 3.2. The measured training times for each repetition of the various configurations is given in the y column. The $\hat{y}$ column contains the mean of the y values. The error values are the difference between $\hat{y}$ and the corresponding $y_{i,j}$ values.

Using the sign table, the variation for each factor is computed as

$$
\begin{aligned}
q_0 &= 1/2^3 \cdot (\hat{y}_0 + \hat{y}_1 + \hat{y}_2 + \hat{y}_3 + \hat{y}_4 + \hat{y}_5 + \hat{y}_6 + \hat{y}_7) \\
&= 1/8 \cdot (2.22 + 1.94 + 1.95 + 1.95 + 2.08 + 2.02 + 2.08 + 2.00) = 2.03 \\
q_1 &= 1/2^3 \cdot (-\hat{y}_0 + \hat{y}_1 - \hat{y}_2 + \hat{y}_3 - \hat{y}_4 + \hat{y}_5 - \hat{y}_6 + \hat{y}_7) \\
&= 1/8 \cdot (-2.22 + 1.94 - 1.95 + 1.95 - 2.08 + 2.02 - 2.08 + 2.00) = -0.05 \\
q_2 &= 1/2^3 \cdot (-\hat{y}_0 - \hat{y}_1 + \hat{y}_2 + \hat{y}_3 - \hat{y}_4 - \hat{y}_5 + \hat{y}_6 + \hat{y}_7) \\
&= 1/8 \cdot (-2.22 - 1.94 + 1.95 + 1.95 - 2.08 - 2.02 + 2.08 + 2.00) = -0.04 \\
q_3 &= 1/2^3 \cdot (-\hat{y}_0 - \hat{y}_1 - \hat{y}_2 - \hat{y}_3 + \hat{y}_4 + \hat{y}_5 + \hat{y}_6 + \hat{y}_7) \\
&= 1/8 \cdot (-2.22 - 1.94 - 1.95 - 1.95 + 2.08 + 2.02 + 2.08 + 2.00) = 0.02 \\
q_4 &= 1/2^3 \cdot (\hat{y}_0 - \hat{y}_1 - \hat{y}_2 + \hat{y}_3 + \hat{y}_4 - \hat{y}_5 - \hat{y}_6 + \hat{y}_7) \\
&= 1/8 \cdot (2.22 - 1.94 - 1.95 + 1.95 + 2.08 - 2.02 - 2.08 + 2.00) = 0.03 \\
q_5 &= 1/2^3 \cdot (\hat{y}_0 - \hat{y}_1 + \hat{y}_2 - \hat{y}_3 - \hat{y}_4 + \hat{y}_5 - \hat{y}_6 + \hat{y}_7) \\
&= 1/8 \cdot (2.22 - 1.94 + 1.95 - 1.95 - 2.08 + 2.02 - 2.08 + 2.00) = 0.02 \\
q_6 &= 1/2^3 \cdot (\hat{y}_0 + \hat{y}_1 - \hat{y}_2 - \hat{y}_3 - \hat{y}_4 - \hat{y}_5 + \hat{y}_6 + \hat{y}_7) \\
&= 1/8 \cdot (2.22 + 1.94 - 1.95 - 1.95 - 2.08 - 2.02 + 2.08 + 2.00) = 0.03 \\
q_7 &= 1/2^3 \cdot (-\hat{y}_0 + \hat{y}_1 + \hat{y}_2 - \hat{y}_3 + \hat{y}_4 - \hat{y}_5 - \hat{y}_6 + \hat{y}_7) \\
&= 1/8 \cdot (-2.22 + 1.94 + 1.95 - 1.95 + 2.08 - 2.02 - 2.08 + 2.00) = -0.04
\end{aligned}
\tag{3.12}
$$

.

The q_j values are then used to compute each of the sum of squares values

$$
\begin{aligned}
SS_{q_0} &= 2^k \cdot r \cdot + q_0^2 = 2^3 \cdot 3 \cdot 2.03^2 = 98.833 \\
SS_{q_1} &= 2^k \cdot r \cdot + q_1^2 = 2^3 \cdot 3 \cdot -0.05^2 = 0.063 \\
SS_{q_2} &= 2^k \cdot r \cdot + q_2^2 = 2^3 \cdot 3 \cdot -0.04^2 = 0.030 \\
SS_{q_3} &= 2^k \cdot r \cdot + q_3^2 = 2^3 \cdot 3 \cdot 0.02^2 = 0.006 \\
SS_{q_4} &= 2^k \cdot r \cdot + q_4^2 = 2^3 \cdot 3 \cdot 0.03^2 = 0.026 \\
SS_{q_5} &= 2^k \cdot r \cdot + q_5^2 = 2^3 \cdot 3 \cdot 0.02^2 = 0.008 \\
SS_{q_6} &= 2^k \cdot r \cdot + q_6^2 = 2^3 \cdot 3 \cdot 0.03^2 = 0.020 \\
SS_{q_7} &= 2^k \cdot r \cdot + q_7^2 = 2^3 \cdot 3 \cdot -0.04^2 = 0.035
\end{aligned}
\tag{3.13}
$$

.

The sum of squared error is computed as

$$\begin{aligned} SSE &= \sum_{i=1}^{2^k}\sum_{j=1}^{r} e_{i,j}^2 \\ &= +(-0.04)^2 + 0.11^2 + (-0.07)^2 \\ &\quad + 0.09^2 + (-0.06)^2 + (-0.03)^2 \\ &\quad + (-0.16)^2 + 0.13^2 + 0.03^2 \\ &\quad + (-0.10)^2 + 0.13^2 + (-0.03)^2 \\ &\quad + (-0.12)^2 + 0.09^2 + 0.03^2 \\ &\quad + (-0.06)^2 + (-0.02)^2 + 0.08^2 \\ &\quad + (-0.13)^2 + 0.03^2 + 0.09^2 \\ &\quad + 0.06^2 + (-0.11)^2 + 0.04^2 = 0.18 \end{aligned} \tag{3.14}$$

.

Finally to compute the percent of variation due to each set of factors, SST must first be computed, as

$$SST = \sum_{j=1}^{2^k} SS_{q_j} + SSE = 0.06 + 0.03 + 0.01 + 0.03 + 0.01 + 0.02 + 0.03 + 0.18 = 0.37 \tag{3.15}$$

, which can then be used to compute each of the individual percentages as

$$\begin{aligned} percent_{q_0} &= 100\% \cdot SS_{q_0}/SST = 100\% \cdot 98.83/0.37 = 26721.26\% \\ percent_{q_1} &= 100\% \cdot SS_{q_1}/SST = 100\% \cdot 0.06/0.37 = 16.96\% \\ percent_{q_2} &= 100\% \cdot SS_{q_2}/SST = 100\% \cdot 0.03/0.37 = 8.14\% \\ percent_{q_3} &= 100\% \cdot SS_{q_3}/SST = 100\% \cdot 0.01/0.37 = 1.49\% \\ percent_{q_4} &= 100\% \cdot SS_{q_4}/SST = 100\% \cdot 0.03/0.37 = 6.94\% \\ percent_{q_5} &= 100\% \cdot SS_{q_5}/SST = 100\% \cdot 0.01/0.37 = 2.18\% \\ percent_{q_6} &= 100\% \cdot SS_{q_6}/SST = 100\% \cdot 0.02/0.37 = 5.42\% \\ percent_{q_7} &= 100\% \cdot SS_{q_7}/SST = 100\% \cdot 0.03/0.37 = 9.34\% \\ percent_{error} &= 100\% \cdot SSE/SST = 100\% \cdot 0.18/0.37 = 49.53\% \end{aligned} \tag{3.16}$$

.

Thus the percent of variation due to the primary factors A, B, and C are 16.96%, 8.14%, and 1.49% respectively. The percent due to the combination of two factors AB, AC, and BC are 6.94%, 2.18%, and 5.42% respectively. Leaving 9.34% to the combination of all factors and 49.53% due to error.

Continuing this example, to determine which factors are statistically significant, the standard deviation of the factors is needed. The standard deviation of the error is computed as

$$s_e = \sqrt{\frac{SSE}{2^k \cdot (r-1)}} = \sqrt{\frac{0.18}{2^3 \cdot (3-1)}} = 0.11 \tag{3.17}$$

. The standard deviation for each factor is computed is then

$$s_{q_j} = \frac{s_e}{\sqrt{2^k r}} = \frac{0.11}{2^3 \cdot 3} = 0.004458 \tag{3.18}$$

Factor	Effect	Percent of Variation	Confidence Interval
I	2.029	26721.255	(1.983,2.076)
A	-0.051	16.955	(-0.097,-0.005)
B	-0.035	8.138	(-0.082,0.011)^I
C	0.015	1.494	(-0.031,0.061)^I
AB	0.033	6.944	(-0.014,0.079)^I
AC	0.018	2.181	(-0.028,0.065)^I
BC	0.029	5.424	(-0.017,0.075)^I
ABC	-0.038	9.336	(-0.084,0.008)^I
Error	0.183	49.528	(0.137,0.229)

Table 3.3: Example confidence intervals, ^{I}Factor is insignificant (95% confidence).

.

Using this standard deviation, and the degrees of freedom, which would be $2^k(r-1) = 16$ in this example, the required t-value can be looked up in any statistics text book. For 95% confidence, the p value will be 0.975, and n is the degrees of freedom, 16, this gives a t-value of 2.120. The confidence intervals for each factor are

$$\begin{aligned}
CI_{q_0} &= q_0 \pm s_{q_0} \cdot t = 2.029 \pm 0.022 \cdot 2.120 = (1.983, 2.076) \\
CI_{q_1} &= q_1 \pm s_{q_1} \cdot t = -0.051 \pm 0.022 \cdot 2.120 = (-0.097, -0.005) \\
CI_{q_2} &= q_2 \pm s_{q_2} \cdot t = -0.035 \pm 0.022 \cdot 2.120 = (-0.082, 0.011) \\
CI_{q_3} &= q_3 \pm s_{q_3} \cdot t = 0.015 \pm 0.022 \cdot 2.120 = (-0.031, 0.061) \\
CI_{q_4} &= q_4 \pm s_{q_4} \cdot t = 0.033 \pm 0.022 \cdot 2.120 = (-0.014, 0.079) \\
CI_{q_5} &= q_5 \pm s_{q_5} \cdot t = 0.018 \pm 0.022 \cdot 2.120 = (-0.028, 0.065) \\
CI_{q_6} &= q_6 \pm s_{q_6} \cdot t = 0.029 \pm 0.022 \cdot 2.120 = (-0.017, 0.075) \\
CI_{q_7} &= q_7 \pm s_{q_7} \cdot t = -0.038 \pm 0.022 \cdot 2.120 = (-0.084, 0.008) \\
CI_{error} &= SSE \pm s_e \cdot t = 0.183 \pm 0.107 \cdot 2.120 = (0.137, 0.229)
\end{aligned} \tag{3.19}$$

.

3.2 Hyper-parameter $2^k r$ Design

The number of factors being examined determines the value of k. For logistic regression, nine factors were chosen as shown in Table 3.4. With $k = 9$, 512 ($2^9 = 512$) combinations of parameters are required. For the tree induction algorithms, four factors were chosen for simple tree induction as shown in Table 3.5 and five were used to bagging and boosting as shown in Table 3.6 and Table 3.7 respectively. With $k = 4$ and $k = 5$ only 16 and 32 combinations are required. Each of these factors were considered for each node splitting method (information gain, variance minimization, and MSEE). With three different splitting methods, each requiring their own $2^k r$ runs, the total jumps to 48 and 96. Additionally a total of 10 trials (i.e. 10 different ways of splitting the data into training, validation and test sets) were used in the $2^k r$ study. The number of cores used was not considered as a factor since it is reasonable to assume that it would have a significant effect on the run time of some datasets even if the two datasets used are too small to demonstrate such effects.

To account for random variations, a total of three repetitions were run for each configuration. For $2^k r$ this means that $r = 3$.

Over the ten trials, each combination of factors was tested for significance. Combinations of factors that failed to be significant could obviously be dropped form further study.

Additionally, among the factors that were significant the magnitude of their significance within an algorithm could be estimated by looking at their significance across all trials and datasets. Since the percent of variation due to each combination of factors varies a lot between trials, the median variation for each combination was taken. The median error value was also taken. Next, for each combination of factors, the variation due to that combination was distributed evenly across all factors involved in it and added to the total variation for those factors. Finally, these summed variations for the individual factors were averaged across multiple datasets. The error values across the datasets were also averaged.

3.3 Choosing High and Low Levels for $2^k r$

Once the factors to be investigated were chosen, high and low levels for each factor needed to be determined. Since some hyper-parameters limit certain dimensions (e.g. height of a tree), it is important to understand what those dimensions would be if the limits were removed. It would be pointless to set the maximum tree height levels to 50 and 100 if the largest any trees get have a height of 30. Rather than pick values arbitrarily, a series of models were trained using extreme values for some hyper-parameters to determine reasonable bounds for other parameters. Specifically hyper-parameters that affect the size of the model (e.g., maximum stump height or minimum samples needed to split a node) were given extreme values. For example, setting the maximum height of a stump to two in boosting should cause the number of stumps produced to be a reasonable upper bound. Using this rationale, extreme sets of hyper-parameters were used to find reasonable upper bounds for each hyper-parameter being examined. These extreme hyper-parameters where then used on four datasets and the resulting models were then measured to determine reasonable limits. In some cases the upper bound for some datasets were orders of magnitude larger than the rest. Those were considered outliers and ignored for the purposes of determining high and low values. For the low values, 50% of the high value was chosen.

3.3.1 Logistic Regression

For determining the high and low values for logistic regression, the maximum number of iterations was set to unlimited. The use of bins, for probabilities and attribute levels, was disabled. The L_2 regularization factor was set to values of 0 (disabled), 1, and 10. The running of full regression was set to never, once at the end, and every iteration. All combinations of the full regression and L_2 regularization were done, for a total of nine runs for each dataset.

3.3.2 Tree Induction

For tree induction, the maximum number of iterations was set to unlimited. Likewise, no lower limits were placed on the number of samples needed before splitting a leaf node. The maximum tree height was set to either unlimited or just two. The two tree height limits were used for all three node splitting algorithms available.

3.3.3 Bagging

For bagging, the same values as tree induction were used. The number of stumps used was only one.

3.3.4 Boosting

For boosting, the same values as tree induction were again used, but the maximum number of stumps was set to unlimited. The probability used to determine correctly vs incorrectly classified samples was 0.5.

3.3.5 High and Low Levels

The resulting models, with either unconstrained or highly constrained hyper-parameters, were examined to determine common model sizes for Mushrooms and Pima. Other hyper-parameters that do not simply limit the size of the model or number of iterations allowed were chosen somewhat arbitrarily, but hopefully in useful range. In most cases, the low level was chosen to be 50% the high level. The resulting high and low levels used for the $2^k r$ study are summarized in Table 3.4 through Table 3.7. Due to the different characteristics of the datasets, some levels have multiple values. In those cases, the first number was used for Pima and the second number was used for Mushrooms.

ID	Low	High	Factor
A	1	10	L_2 regularization factor
B	off	on	L_2 regularization enabled
C	20	40	feature level bins
D	20	40	probability bins
E	off	on	binning enabled/disabled
F	on	off	full regression once vs per iteration
G	on	off	full regression enabled/disabled
H	3 / 30	7 / 60	max iterations (Pima / Mushrooms)
I	50%	100%	percent of training set used

Table 3.4: Factors for logistic regression

ID	Low	High	Factor
A	2	10	Minimum samples for splitting
B	12 / 14	24 / 28	Maximum tree height
C	15 / 50	30 / 100	Maximum iterations
D	50%	100%	Percent of training set used

Table 3.5: Factors for tree induction

ID	Low	High	Factor
A	4 / 16	8 / 32	number of stumps
B	2	10	minimum samples for splitting
C	12 / 14	24 / 28	maximum tree height
D	15 / 50	30 / 100	maximum iterations (Pima / Mushrooms)
E	50%	100%	percent of training set used

Table 3.6: Factors for tree induction with bagging

ID	Low	High	Factor
A	4 / 16	8 / 32	maximum number of stumps
B	2	10	minimum samples for splitting
C	12 / 14	24 / 28	maximum tree height
D	15 / 50	30 / 100	maximum iterations (Pima / Mushrooms)
E	50%	100%	percent of training set used

Table 3.7: Factors for tree induction with boosting

3.4 Runtime Environment

The hardware used for these experiments was a 24 node cluster with four 1.86Ghz Intel Xeon 5120 cores and 4GB of RAM per node. All Java code was run in a 32-bit version of Java 1.6.0 update 24. The nodes were connected via an Infiniband network, and a 1000baseT management network. Resources were allocated as 16 nodes used for computation, using only one core per node, and one additional node that was used both for launching jobs and running the overlay network manager. All communication between nodes was run over the 1000baseT network, as IP over infiniband was not enabled on the cluster. While this is a small fraction of the total computation power of the cluster, the datasets used are not large enough to benefit from more cores.

3.5 $2^k r$ Results

Using the high and low levels a series of models were trained using a 60% / 20% / 20% split of training / validation / test samples from each dataset. The Mushrooms and Pima datasets were used, both in their original form and converted to attributes that are all numeric in the range zero to one, for the logistic regression algorithm. For each of the models trained the total training time was measured in milliseconds with a microsecond precision timer. Each model was also scored using the test samples that were not used in training. The scoring consisted of five non-parametric metrics that that give a score between zero and one, the arithmetic mean of those five metrics was then used as the final score.

3.5.1 Statistical Significance Results

Due to the large nature of the $2^k r$ tables resulting from the data collected, they are are in the supplementary appendix. One of the smaller tables has been duplicated as Table 3.8. This table shows the percent of variation due to each factor or combination of factors as well as the variation due to error for each of the 10 trials. Factors for which the variation is statistically significant ($p = 0.95$), are show in italics. The count column gives the number of trials for which the factor is significant. The average and median columns gives the arithmetic mean and median of the variation over the ten trials.

Looking at the individual factors (A, B, C, D, and E), it is clear that A, B, and E are significant in all ten trials, while C is significant in nine. On the other hand factor D is only significant in four of them which might make is a a good candidate to remove from consideration in later portions of the study. However when looking at interactions with other factors such as C and E the number of trials where it was significant goes up to six and seven respectively.

Therefore, looking simply at if the factors are significant is not a reasonable way to eliminate factors. While this is just one measure of one algorithm being run on one dataset, the rest of the results indicate that statistical significance alone does not justify eliminating any of the hyper-parameters considered. Perhaps the selection of hyper-parameters could be done by examining the total variation caused by each one relative the error in the measurements.

Factor	Percent Variation in Trial										Count	Average	Median
	1	2	3	4	5	6	7	8	9	10			
A	*13.091*	*1.848*	*6.611*	*15.989*	*15.531*	*9.455*	*27.562*	*36.613*	*16.773*	*25.440*	10	16.89	15.76
B	*4.353*	*10.448*	*10.255*	*6.530*	*9.503*	*12.844*	*16.784*	*15.835*	*8.774*	*5.863*	10	10.12	9.88
C	*4.483*	*2.675*	0.04	*3.772*	*2.613*	*2.371*	*2.439*	*1.441*	*3.395*	*2.738*	9	2.60	2.64
D	*0.754*	0.28	0.04	0.10	*0.399*	0.25	0.09	0.24	*0.608*	*1.386*	4	0.41	0.27
E	*37.889*	*61.715*	*56.539*	*42.144*	*47.258*	*47.367*	*24.978*	*28.669*	*31.577*	*26.749*	10	40.49	40.02
AB	*2.213*	0.21	0.14	0.30	*0.649*	0.38	0.33	0.00	0.26	*0.722*	3	0.52	0.31
AC	*2.554*	0.00	0.16	*1.233*	*1.270*	*3.270*	*4.428*	*0.379*	*5.728*	*3.575*	8	2.26	1.91
AD	*1.715*	0.04	0.25	0.12	0.28	0.03	0.03	0.00	0.09	*3.081*	2	0.56	0.10
AE	0.06	*1.902*	*4.869*	*7.547*	0.01	*2.801*	*1.529*	*1.454*	*0.642*	*1.051*	8	2.19	1.49
BC	*0.316*	*0.656*	0.44	0.02	0.05	*0.690*	0.08	*0.418*	0.00	0.01	4	0.27	0.20
BD	0.02	0.17	0.26	0.09	0.05	*1.299*	0.03	0.06	0.10	0.11	1	0.22	0.10
BE	*0.317*	*1.706*	0.24	0.33	*0.424*	*1.913*	*0.459*	*1.626*	*1.916*	*0.339*	8	0.93	0.44
CD	*2.177*	0.01	0.44	*1.326*	*3.141*	0.23	*0.900*	0.02	*0.894*	*0.539*	6	0.97	0.72
CE	*4.928*	*4.752*	0.11	*3.184*	*3.850*	*4.071*	*3.629*	*1.559*	*4.710*	*5.463*	9	3.63	3.96
DE	*3.318*	*0.810*	0.43	*2.259*	*0.826*	0.01	*0.931*	0.26	*1.684*	*0.552*	7	1.11	0.82
ABC	0.06	0.02	0.11	0.01	0.29	0.08	0.11	*0.347*	0.00	0.12	1	0.11	0.09
ABD	0.17	*0.848*	0.25	0.26	*0.519*	0.02	0.33	*0.393*	0.01	0.00	3	0.28	0.25
ABE	*0.598*	0.42	0.64	0.04	0.00	*0.889*	*0.478*	*0.351*	0.00	0.07	4	0.35	0.39
ACD	*2.408*	0.01	0.00	*1.363*	*1.062*	0.34	*1.203*	0.12	*1.463*	*0.997*	6	0.90	1.03
ACE	*9.017*	*2.266*	*2.453*	*1.478*	*1.925*	*0.722*	*2.896*	*3.015*	*9.382*	*8.361*	10	4.15	2.67
ADE	*2.088*	0.09	0.35	*1.897*	*0.790*	0.30	*0.614*	0.04	*1.419*	*1.062*	6	0.87	0.70
BCD	0.10	0.07	0.52	0.02	0.03	0.03	0.01	*0.338*	0.02	0.18	1	0.13	0.05
BCE	0.09	0.12	0.00	0.14	0.18	*1.233*	0.04	0.06	0.08	0.02	1	0.20	0.09
BDE	0.16	0.00	0.05	0.14	0.13	0.28	0.00	0.21	0.08	0.13	0	0.12	0.13
CDE	*1.602*	0.16	0.30	*1.348*	*1.569*	0.02	0.06	0.00	*1.034*	*2.663*	5	0.88	0.66
ABCD	0.15	0.12	0.36	0.11	0.07	*1.400*	0.32	0.22	0.11	*0.510*	2	0.34	0.19
ABCE	0.15	0.00	0.45	0.04	0.01	0.16	0.17	*0.877*	0.05	0.04	1	0.20	0.10
ABDE	0.22	0.22	0.06	0.02	0.17	0.02	0.03	0.00	0.06	0.19	0	0.10	0.06
ACDE	*1.025*	0.21	0.45	0.13	*1.785*	0.01	*1.803*	0.14	*0.546*	*2.229*	5	0.83	0.50
BCDE	0.06	0.25	0.09	*0.993*	0.21	0.31	0.32	0.01	0.45	0.26	1	0.30	0.26
ABCDE	0.01	0.00	0.57	*0.634*	0.06	0.15	0.13	0.03	0.48	0.30	1	0.24	0.14
Error	*3.907*	*7.963*	*12.541*	*6.430*	*5.355*	*7.053*	*7.283*	*5.253*	*7.650*	*5.248*	10	6.87	6.74

Table 3.8: Time for bagged (MSEE) on PIMA (binorm). Statistically significant factors in *italics* ($p = 0.95$).

	Additional Factors					
Factor	0	1	2	3	4	Total
A	*15.76%*	1.91%	1.71%	0.21%	0.03%	*19.62%*
B	*9.88%*	0.53%	0.33%	0.15%	0.03%	*10.92%*
C	2.64%	3.39%	1.53%	0.26%	0.03%	*7.86%*
D	0.27%	0.87%	0.94%	0.25%	0.03%	2.36%
E	*40.02%*	3.36%	1.55%	0.23%	0.03%	*45.18%*
Error						6.74%

Table 3.9: Summary of variation due to factors for Time for bagged (MSEE) on Pima (binorm). Values greater than the median error (6.742%) are show in *italics*.

	Additional Factors					
Factor	0	1	2	3	4	Total
A	*18.18%*	*7.10%*	*0.73%*	*0.03%*	*0.01%*	*26.05%*
B	*13.36%*	*1.30%*	*0.72%*	*0.03%*	*0.01%*	*15.41%*
C	*0.18%*	*0.20%*	*0.11%*	*0.03%*	*0.01%*	*0.52%*
D	*0.01%*	*0.03%*	*0.04%*	*0.02%*	*0.01%*	*0.11%*
E	*15.20%*	*6.97%*	*0.73%*	*0.03%*	*0.01%*	*22.93%*
Error						0.00%

Table 3.10: Summary of variation due to factors for Combo Score for bagged (MSEE) on Pima (binorm). Values greater than the median error (0.000%) are show in *italics*.

3.5.2 Summing Interations

Since the goal of this exercise was to find individual factors that could be eliminated to reduce the complexity of later portions of this research, it is necessary to identify which factors affect the outcome either directly or though interactions with other factors. For each factor or combination of factors, the median percent variation over all 10 trials was chosen. This will not always result in a sum of 100%, but median is less affected by outliers. The median also tended to be lower than the mean, making it a more conservative estimate of variation.

The sum of these median variations over all interactions was computed, to get a better idea of how factors affect the overall variation. When summing interactions, the value was scaled by the inverse of the number of factors involved, thereby spreading the variation evenly across all factors involved. An example of such summing is given in Table 3.9. The error reported is the median error across the ten trials as reported in Table 3.8. A table like this can be made for each each metric for each algorithm run on each dataset. Thus further summarization is needed. Taking the average of the total variations across datasets should give a better idea of how the factors might affect other datasets overall.

Here it is clear that even across all interactions with other factors, D is still not as significant as error. When we look at the same factors for their effects on model quality, as shown in Table 3.10. Again factor D does not contribute much variation. Although higher than error, the error for model scores is always nearly, if not actually, zero.

It appears setting a modest threshold for percent variation may be a viable means to eliminate hyper-parameters from further study. However this is just one dataset and only one of the splitting methods.

3.5.3 Averages of Summed Interactions

Additional tables similar to Table 3.9 can also be made by averaging the results over multiple datasets for each algorithm. In Table 3.11, the average variation in model scores across all datasets for bagging with MSEE as the split measure is given for each factors, as well as the averages of the median error over those datasets. Likewise, Table 3.12 gives the average variation of training times.

Factor	Average Variation
A	24.34%
B	15.56%
C	0.28%
D	0.04%
E	31.69%
Error	0.00%

Table 3.11: Averages of variation in Combo Scores for bagged with MSEE

Factor	Average Variation
A	14.70%
B	11.16%
C	4.83%
D	2.17%
E	52.09%
Error	7.30%

Table 3.12: Averages of variation in Times for bagged with MSEE

This example of how the factors can be combined to get an overall sense of each factor's effect on performance is only for one of the splitting methods. When the different splitting methods are combined as in Table 3.13 through Table 3.18, it becomes clear that different splitting methods are affected by the hyper-parameters very differently.

Looking at the variation over all splitting methods in Table 3.14, when using the variance minimization split method, factor D does contribute more to the variation. This amount of variation is more than C contributed with MSEE, which had been shown to produce a statistically significant amounts in nearly all trials.

Taking the results of Table 3.13 through Table 3.20, none of the factors consistently provide a low percentage of variation relative the error.

3.6 Conclusion

Given these results, all hyper-parameters were considered worthy of inclusion in the remainder of this study. A full factorial approach to exploring the hyper-parameter space was unreasonable and a different method was needed. In Chapter 4, an approach for sampling the hyper-parameter space in a systematic fashion taking into account the relative importance of hyper-parameters is discussed.

	Average Variation		
Factor	Info Gain	Variance	MSEE
A	29.65%	25.39%	24.34%
B	20.32%	23.86%	15.56%
C	0.01%	0.00%	0.28%
D	0.00%	0.00%	0.04%
E	34.76%	32.74%	31.69%
Error	0.00%	0.00%	0.00%

Table 3.13: Super summary for bagging model score

	Average Variation		
Factor	Info Gain	Variance	MSEE
A	15.40%	21.54%	14.70%
B	17.40%	25.93%	11.16%
C	1.77%	19.27%	4.83%
D	1.76%	10.69%	2.17%
E	46.56%	19.65%	52.09%
Error	8.90%	1.58%	7.30%

Table 3.14: Super summary for bagging training time

	Average Variation		
Factor	Info Gain	Variance	MSEE
A	3.72%	5.00%	3.20%
B	0.54%	0.15%	0.03%
C	0.99%	1.00%	4.33%
D	14.96%	8.76%	5.77%
E	14.14%	18.95%	23.12%
Error	0.00%	0.00%	0.01%

Table 3.15: Super summary for boosting model score

	Average Variation		
Factor	Info Gain	Variance	MSEE
A	18.92%	24.67%	17.89%
B	1.98%	7.49%	0.04%
C	0.31%	7.51%	1.16%
D	27.36%	31.08%	38.75%
E	44.23%	24.82%	35.34%
Error	3.08%	0.71%	1.05%

Table 3.16: Super summary for boosting training time

	Average Variation		
Factor	Info Gain	Variance	MSEE
A	0.00%	0.00%	0.00%
B	0.00%	0.00%	4.60%
C	15.63%	13.29%	23.29%
D	28.21%	33.04%	25.70%
Error	0.00%	0.00%	0.00%

Table 3.17: Super summary for tree induction model score

	Average Variation		
Factor	Info Gain	Variance	MSEE
A	4.53%	16.35%	0.17%
B	1.79%	13.47%	0.23%
C	49.42%	53.15%	90.15%
D	31.55%	10.09%	6.90%
Error	8.94%	5.05%	2.18%

Table 3.18: Super summary for tree induction training time

Factor	Average Variation
A	5.85%
B	8.33%
C	0.84%
D	1.49%
E	5.09%
F	16.32%
G	19.39%
H	3.03%
I	7.33%
Error	0.42%

Table 3.19: Super summary for logistic regression model score

Factor	Average Variation
A	4.33%
B	6.23%
C	0.57%
D	0.75%
E	8.22%
F	8.15%
G	42.83%
H	3.81%
I	5.96%
Error	1.35%

Table 3.20: Super summary for logistic regression training time

Chapter 4

Phase 2: Hyper-parameter Optimization

Due to the large number of hyper-parameters to be examined, it is impractical to do a full factorial study. Instead a hill-climbing [99] strategy was employed to try to find optimal hyper-parameters for each dataset while also randomly sampling other regions of the hyper-parameter space.

In a typical hill climbing search, there is a function, *fitness*, that maps each point in the search space to a unique comparable value, often a real number. The goal of the search is to maximize the value of that function. To accomplish this, a second function, *expand*, that maps any point in search space to a set of adjacent points in search space is used. Using these two functions, given an arbitrary point in the search space, the expand function provides a set of nearby points. When passed to the fitness function each of these points can then be tested to see which one has the highest value. The point with the highest values from the fitness function is then chosen to be the next point to be passed to the expand function. This process continues until none of the points produced by the expand function have a higher score. This process of moving to adjacent points with higher scores is said to climb the hill represented by the fitness function until it finds a local maximum. However, since hill climbing can only find local maximums, in order to find a global maximum many initial start points may be needed. Thus a common approach is to use random restarts, where a random point in the search space is chosen whenever a local maximum is reached. The start point for the first hill climb is also chosen at random from the search space. Together this approach is called hill climbing with random restarts.

For the purposes of the hill climbing approach the point in search space is an n-tuple, where each element is a specific level for its given hyper-parameter. Some hyper-parameters are searchable while others are not. Combinations of non-searchable hyper-parameters are expanded externally and are treated as distinct searches. The expand function is only allowed to modify exactly one hyper-parameter at a time. Each point is assigned its score based on the composite score of the model produced using the hyper-parameters it represents. That composite score is the arithmetic mean of seven non-parametric in the range zero to one where zero is perfectly incorrect, and one is perfect correct.

The search starts by picking five random points in search space such that each searchable hyper-parameter is assigned a uniformly random value between 0 and 100. Each point is converted to a set of hyper-parameters. Each of the five random configurations is used to train models on five different splits of each dataset. The highest scoring one is chosen as the starting point for hill climbing and the remaining four are added to a pool of future candidates. From the start point, neighboring states are examined by first expanding the hyper-parameter that is responsible for the most variation according to the $2^k r$ analysis, and proceeding through each hyper-parameter until no better neighboring states can be found. Within each hyper-parameter the value change is initially quite large and is reduced exponentially. The initial change is 16 units, 16 units, followed by 8, then

4, 2, and finally 1. These large steps allow for faster climbing of hills when it's moving in the correct direction, as well as overcoming shoulders (flat regions between regions of improvement). Whenever a neighboring state with a higher score is found, that state becomes the new best state and all new neighbors are chosen relative it. When no neighbors get better average composite scores, the hill-climb is terminated and another set of five random states are chosen. The new hill-climb start state is then chosen from the five new states as well as the pool of previously rejected start points.

This process of picking random start points then hill-climbing continued until a pre-determined amount of time had elapsed. For most of the datasets a total time of one week was used. For three of the larger datasets (CoverType, Intrusion, and Thrombin), a total time of four weeks was used. The runtime system for this portion of the project, which is described more thoroughly in Section 4.1, used a queuing system where jobs were submitted then allowed to run when there were enough cores available. Due to issues in the runtime environment, it was known that jobs would randomly terminate. To deal with this random termination, when each job would start, it would check to see if it had more models to train, and if it did would submit a another copy of itself that would only be allowed to run when the current job ended. To account for gaps in useful execution time caused by random termination and having to wait in a queue, any gap of more than twelve hours was not counted towards the overall time limit.

Some hyper-parameters were either held constant or had multiple fixed values, and are not reported in the tables. For all algorithms, the data split for all searches was 60% training, 20% validation, and 20% test, and searches were performed using both 50% and 100% of the training samples. For boosting the threshold for positive versus negative classification for the purposes of reweighting samples is always 0.5. For tree induction pruning was always enabled. For logistic regression, separate searches were performed for each of the full regression modes (never, after every iteration, and once at the end). For using bins in logistic regression, the numbers of bins were searchable, but searches were also performed with binning completely disabled.

The hyper-parameters that were varied during the search for the tree based methods were maximum height of any tree/stump, the maximum number of iterations, and the minimum number of samples to split a leaf. For bagging, the number of stumps was also varied. For boosting, the maximum number of stumps as well ad the maximum number of iterations to train an individual stump were varied. For tree induction the point where it switched from splitting a single node to all of the leaves simultaneously was varied. For both boosting and single tree induction the interval between writing output values in the mapper was also varied.

For logistic regression, the maximum number of iterations and the L_2 regularization factor were varied. For searched where bins were used, the number of bins was also varied.

4.1 Runtime Environment

The hardware used for these experiments was a 92 node cluster with 16 2.60 Ghz Intel Xeon E5-2670 cores and 64GB of RAM per node. All Java code was run in a 64-bit version of Java 1.7.0 update 75. The nodes were connected via an Infiniband network, and a 1000baseT management network. The system also utilized the Open Grid Scheduler (OGS) [89] for allocating resources to compute jobs. The scheduler in OGS assigned jobs to a collection of slots, which in this case map to individual cores. Due to the nature of the scheduler, a single job could be scattered across the system with only one core on each node, or it could have entire nodes all to itself. There was no way of predicting how a job would be split across node ahead of time, and difficult to reconstruct after the fact.

Additionally, `ssh` [86] access to compute nodes was not allowed. To get around this limitation, the process of starting the MapReduce runtime system as well as copying data to the compute nodes was handled by an program called `mpi_wrapper` that used the MPI API for communication and synchronization operations as well as a set of shell scripts to perform various tasks.

Each job submitted to OGS consisted of 33 slots (i.e. cores). Of those, 32 slots were for actual computation, and the remaining core was used to handle the overlay network and the launching of MapReduce jobs. All MapReduce communication between nodes were run over the 1000baseT. The `mpi_wrapper`'s MPI calls used the Infiniband network, but some additional TCP/IP communication, due to MPI's busy waiting, was passed over the 1000baseT network.

4.2 Primary Model Training

The purpose of this hill climbing approach was to identify hyper-parameters that produce high quality models, and ideally with minimal training time for the quality. The incomplete models captured after each iteration also served as a means to skip large portions of the training process in the next phase.

The maximum model scores for each dataset are given in Table 4.1 through Table 4.3. The metrics reported are AUC, accuracy, precision, recall, specificity, root mean squared error, root mean squared accuracy, and a combination score which is the arithmetic mean of the seven measures. When computing the mean, $1 - \text{RMSE}$ is used since 1.0 is perfect and 0.0 is very bad.

4.2.1 Best Model Scores Found

In Table 4.1 though Table 4.4, the highest model scores found for each dataset are given. Each of the individual measure (e.g., AUC, accuracy) scores were the averaged over five trials. For the combo score, the average was computed by summing the individual scores over all trials, then dividing by the number of trials and the number of scores. For the purposes of picking the best score, only the mean was considered. The standard deviation and confidence intervals were not computed here, as this table is presented primarily to give an idea of both the range of score values different metrics provide for the same model, as well as give an idea of the relative difficulty of learning models for each dataset. For instance, Mushrooms typically gets a combined score around 0.9 across all algorithms, indicating that it is a more easily learned dataset. The Pima dataset on the other hand, had combined scores in the 0.6 to 0.7 range (ignoring bagging which failed to produce a useful model), which indicates that it is a harder dataset to learn.

The Example and Example2, being synthetic datasets generated by models similiar to the ones being learned offer an opportunity to see if datasets generated by a particular model type are easier to learn by that model type. Looking at the combination scores in Table 4.2 through Table 4.4, it does appear to be the case that higher scores were achieved for Example using the tree based methods, where logistic regression scored better on Example2.

4.2.2 Problematic Limitations

Of particular note is the precision column in Table 4.1. The -1 values are an error code that indicate that computing the actual value would have involved a division by zero. In this case, the cause was a bad interaction between the number of stumps in the model and the minimum number of samples

needed to compute a split. Due to the sizes of some datasets, the number of samples available to the root node of each stump was insufficient to perform any splits. As a result, which can be seen in the TN and FN versus TP and FP columns, all of the samples were classified as negative by bagged models for most datasets. This problem was not detected during the hyper-parameter search as only AUC, accuracy and the combination score were examined during spot checks of the progress. In hind-sight, a more extreme negative value that would be guaranteed to make the combo score would have been preferred.

This demonstrates an important and often overlooked aspect of scoring machine learning models. Had precision, recall and the confusion matrix values been omitted from these tables, it would be nearly impossible to detect that such a major error had occurred.

In addition to the stumps and samples interaction for bagging, problems can occur if the prior probability (i.e. the ratio of positive samples to the total samples) is sufficiently low. In such a case, even when nodes chose good splits, all of the leaf nodes may still give probabilities that are less than 0.5, and thus precision would be undefined as there would be no samples that get classified as positive. Likewise, a similar problem can occur when the prior probability is too high and the resulting model only produces probabilities greater than 0.5. A model that produces only positive predictions would be harder to detect based on scores, as none of the metrics used here would be undefined in such a case. This would suggest that the metrics used may be incomplete.

A further problem illustrated by the ADS, ADULT and EXAMPLE2 datasets in Table 4.1 is the conversion from categorical and integer values to real values when creating the binorm versions of the datasets. This discrepancy in model quality is most likely due to differences in how split point searches were done for *real* attributes as opposed to *integer* attributes. In an integer attribute, each of the unique values is tried as a split point. For a real attribute, this is assumed to be impractical, so instead twenty split points at 5-percentile intervals are considered. This alternate strategy will skip potentially useful split points. In hindsight, the number of split points to consider for a real attribute should have been a hyper-parameter to be optimized.

Dataset	TP	TN	FP	FN	AUC	Acc.	Prec.	Recall	Spec.	RMSE	RMSA	Combo
Ads	13	575	0	77	0.777	0.884	1.000	0.145	1.000	0.281	0.862	0.770
Ads (binorm)	0	575	0	90	0.500	0.864	-1.000	0.000	1.000	0.356	0.895	0.415
Adult	825	4587	356	731	0.808	0.833	0.699	0.530	0.928	0.344	0.789	0.749
Adult (binorm)	0	4517	0	1513	0.550	0.749	-1.000	0.000	1.000	0.500	0.863	0.380
Covertype	14487	50131	6491	27824	0.498	0.653	0.819	0.343	0.885	0.453	0.594	0.620
Covertype (binorm)	18042	49987	6535	24197	0.511	0.689	0.313	0.428	0.884	0.452	0.599	0.567
Credit	0	75	0	52	0.709	0.590	-1.000	0.000	1.000	0.576	0.708	0.347
Credit (binorm)	0	75	0	52	0.665	0.589	-1.000	0.000	1.000	0.572	0.695	0.340
Example	123	0	76	0	0.500	0.619	0.619	1.000	0.000	0.489	0.556	0.544
Example (binorm)	119	0	79	0	0.500	0.599	0.599	1.000	0.000	0.492	0.548	0.536
Example2	181	0	17	0	0.516	0.912	0.912	1.000	0.000	0.271	0.865	0.705
Example2 (binorm)	0	20	0	178	0.500	0.102	-1.000	0.000	1.000	0.886	0.304	0.146
IntCensor	610	462	489	458	0.542	0.531	0.568	0.568	0.493	0.499	0.503	0.529
IntCensor (binorm)	261	722	238	798	0.499	0.487	0.605	0.251	0.757	0.507	0.503	0.514
IntShopping	0	961	0	472	0.743	0.671	-1.000	0.000	1.000	0.515	0.753	0.379
IntShopping (binorm)	0	957	0	476	0.512	0.668	-1.000	0.000	1.000	0.574	0.813	0.346
Intrusion	192888	784002	1291	1613	0.800	0.997	0.993	0.992	0.998	0.049	0.992	0.960
Intrusion (binorm)	192536	775376	9914	1968	0.582	0.988	0.951	0.990	0.987	0.190	0.906	0.888
Mushrooms	681	755	21	163	0.686	0.886	0.970	0.806	0.973	0.262	0.867	0.847
Mushrooms (binorm)	0	787	0	834	0.550	0.485	-1.000	0.000	1.000	0.710	0.690	0.288
Pima	0	97	0	55	0.648	0.637	-1.000	0.000	1.000	0.564	0.761	0.355
Pima (binorm)	0	100	0	53	0.656	0.653	-1.000	0.000	1.000	0.543	0.758	0.361
Thrombin	0	376	0	7	0.758	0.981	-1.000	0.000	1.000	0.136	0.968	0.510
Thrombin (binorm)	0	376	0	7	0.635	0.981	-1.000	0.000	1.000	0.137	0.978	0.494

Table 4.1: Top model scores for bagged models.

Dataset	TP	TN	FP	FN	AUC	Acc.	Prec.	Recall	Spec.	RMSE	RMSA	Combo
Ads	55	536	38	34	0.769	0.889	0.590	0.614	0.933	0.322	0.790	0.752
Ads (binorm)	79	469	106	11	0.615	0.823	0.454	0.878	0.815	0.405	0.841	0.717
Adult	568	4719	224	988	0.659	0.813	0.718	0.365	0.955	0.376	0.799	0.705
Adult (binorm)	0	4517	0	1513	0.500	0.749	-1.000	0.000	1.000	0.391	0.785	0.378
Covertype	1486	56014	608	40824	0.576	0.581	0.743	0.035	0.989	0.492	0.523	0.565
Covertype (binorm)	556	56518	147	41481	0.587	0.578	0.791	0.013	0.997	0.491	0.522	0.571
Credit	46	67	8	5	0.923	0.890	0.846	0.894	0.889	0.317	0.792	0.845
Credit (binorm)	44	67	8	8	0.907	0.872	0.843	0.845	0.891	0.343	0.725	0.820
Example	114	66	10	9	0.915	0.905	0.920	0.928	0.869	0.302	0.795	0.861
Example (binorm)	111	65	14	7	0.889	0.886	0.883	0.934	0.818	0.323	0.795	0.840
Example2	168	10	7	13	0.526	0.895	0.957	0.926	0.544	0.261	0.892	0.783
Example2 (binorm)	178	0	20	0	0.500	0.898	0.898	1.000	0.000	0.313	0.927	0.701
IntCensor	828	277	673	240	0.546	0.547	0.552	0.775	0.292	0.501	0.522	0.533
IntCensor (binorm)	717	393	567	342	0.570	0.550	0.562	0.674	0.410	0.497	0.518	0.541
IntShopping	121	912	49	350	0.741	0.722	0.714	0.259	0.949	0.436	0.711	0.666
IntShopping (binorm)	201	841	116	275	0.762	0.727	0.639	0.423	0.879	0.424	0.678	0.669
Intrusion	193639	783954	1338	862	0.759	0.998	0.993	0.996	0.998	0.114	0.957	0.941
Intrusion (binorm)	191886	775343	9947	2619	0.500	0.987	0.951	0.987	0.987	0.111	0.981	0.897
Mushrooms	845	776	0	0	1.000	1.000	1.000	1.000	1.000	0.013	0.998	0.998
Mushrooms (binorm)	834	787	0	0	1.000	1.000	1.000	1.000	1.000	0.057	0.979	0.989
Pima	33	82	15	22	0.794	0.756	0.688	0.601	0.844	0.411	0.689	0.709
Pima (binorm)	34	84	15	19	0.806	0.775	0.693	0.649	0.847	0.408	0.678	0.720
Thrombin	1	338	38	6	0.556	0.882	-0.772	0.159	0.897	0.260	0.930	0.485
Thrombin (binorm)	0	376	0	7	0.533	0.981	-1.000	0.000	1.000	0.138	0.989	0.481

Table 4.2: Top model scores for boosted models.

Dataset	TP	TN	FP	FN	AUC	Acc.	Prec.	Recall	Spec.	RMSE	RMSA	Combo
Ads	6	555	20	84	0.550	0.843	0.460	0.064	0.965	0.373	0.799	0.615
Ads (binorm)	0	575	0	89	0.650	0.865	0.200	0.007	1.000	0.342	0.820	0.600
Adult	0	4939	4	1556	0.457	0.760	0.095	0.000	0.999	0.388	0.740	0.523
Adult (binorm)	480	3915	601	1032	0.500	0.729	-0.422	0.316	0.867	0.421	0.801	0.481
Covertype	31249	38491	18131	11061	0.486	0.705	0.638	0.738	0.680	0.470	0.637	0.631
Covertype (binorm)	29723	43345	13177	12516	0.498	0.740	0.694	0.704	0.767	0.461	0.709	0.664
Credit	46	65	10	5	0.550	0.876	0.826	0.890	0.870	0.315	0.865	0.795
Credit (binorm)	49	62	13	3	0.575	0.868	0.785	0.939	0.819	0.348	0.871	0.787
Example	110	67	8	12	0.603	0.894	0.928	0.899	0.889	0.295	0.874	0.827
Example (binorm)	100	67	12	18	0.566	0.846	0.896	0.844	0.851	0.343	0.831	0.785
Example2	181	1	16	0	0.508	0.917	0.918	0.998	0.100	0.271	0.878	0.721
Example2 (binorm)	175	2	18	3	0.500	0.891	0.907	0.980	0.116	0.307	0.923	0.716
IntCensor	929	180	771	139	0.562	0.549	0.546	0.869	0.189	0.503	0.515	0.533
IntCensor (binorm)	979	90	871	80	0.515	0.529	0.530	0.926	0.095	0.591	0.620	0.518
IntShopping	29	941	20	442	0.500	0.677	0.587	0.064	0.979	0.481	0.609	0.562
IntShopping (binorm)	148	835	122	327	0.504	0.686	0.548	0.313	0.871	0.473	0.705	0.593
Intrusion	194495	775331	9961	6	0.669	0.990	0.951	1.000	0.987	0.098	0.985	0.926
Intrusion (binorm)	191525	775827	9463	2979	0.525	0.987	0.953	0.985	0.988	0.107	0.982	0.902
Mushrooms	837	772	4	7	0.803	0.993	0.995	0.991	0.995	0.085	0.988	0.954
Mushrooms (binorm)	807	787	0	27	0.892	0.983	1.000	0.968	1.000	0.126	0.974	0.956
Pima	27	84	12	27	0.563	0.735	0.730	0.509	0.870	0.425	0.726	0.673
Pima (binorm)	26	89	10	26	0.526	0.759	0.726	0.499	0.894	0.434	0.754	0.675
Thrombin	0	376	0	6	0.550	0.982	-0.267	0.095	0.999	0.132	0.976	0.600
Thrombin (binorm)	0	376	0	6	0.600	0.982	-0.200	0.073	1.000	0.132	0.976	0.614

Table 4.3: Top model scores for tree models.

Dataset	TP	TN	FP	FN	AUC	Acc.	Prec.	Recall	Spec.	RMSE	RMSA	Combo
Ads (binorm)	7	575	0	82	0.906	0.875	0.928	0.085	0.999	0.330	0.827	0.756
Adult (binorm)	160	4507	9	1352	0.870	0.774	0.947	0.106	0.998	0.412	0.702	0.712
Covertype (binorm)	131	56520	2	42108	0.872	0.574	0.488	0.003	1.000	0.495	0.531	0.568
Credit (binorm)	48	58	14	3	0.889	0.855	0.770	0.930	0.801	0.334	0.828	0.820
Example (binorm)	115	17	60	3	0.743	0.674	0.655	0.969	0.224	0.464	0.559	0.623
Example2 (binorm)	172	7	13	6	0.683	0.902	0.929	0.965	0.316	0.293	0.885	0.770
IntCensor (binorm)	418	571	389	642	0.588	0.490	0.668	0.404	0.598	0.500	0.501	0.536
IntShopping (binorm)	3	956	1	473	0.526	0.669	0.044	0.006	0.998	0.471	0.582	0.479
Intrusion (binorm)	1190	785290	0	193314	0.964	0.803	1.000	0.006	1.000	0.398	0.747	0.732
Mushrooms (binorm)	834	759	28	0	0.998	0.983	0.968	1.000	0.964	0.200	0.841	0.936
Pima (binorm)	7	97	3	45	0.726	0.682	0.405	0.139	0.973	0.473	0.541	0.570
Thrombin (binorm)	0	375	1	6	0.599	0.979	0.433	0.095	0.996	0.144	0.936	0.699

Table 4.4: Top model scores for sfo models.

4.2.3 Best Hyper-Parameters Found

The hyper-parameters that achieved the highest scores, as described in Section 4.2.1, for each dataset are shown in Table 4.5 though Table 4.8.

Looking at the hyper-parameters that did well, one thing that is readily apparent is that the hyper-parameters vary greatly between datasets. In particular no one splitting method was preferred across all datasets, although information gain and variance minimization were more common than MSEE.

4.2.4 Temporal Learning Curves

The scores reported in Section 4.2.1 only give the final model scores. During training, each iteration produces a model that can be scored individually. The speed of the algorithms in terms of time per iteration time can also vary considerably. As a result, it is possible that one algorithm may produce reasonably good models very quickly, but then, given enough time, another algorithm may take over as the best.

To show this aspect of the learning process a type of learning curve is used. In this case the x-axis is the run time of the algorithm, and the y-axis is the model quality. Due to the time based nature of these curves, I shall refer to them as temporal learning curves (TLCs).

In Figure 4.1 though Figure 4.24, the TLCs for each algorithm are shown on each dataset. The curve chosen for each dataset is one that scored very highly, but may not have used the same hyper-parameters given in Section 4.2.3. For all datasets, except both versions of Adult and Example2, 32 compute cores were used. For Adult, Example2, and their "binorm" variants, only a single compute core was used as these datasets needed to be rerun later to correct an error (i.e., the target class had been specified incorrectly), but the original cluster was no longer available.

For the actual curves, the average of the most recent value from each trial is used. The error bars are the 95% confidence interval for the five trials. For readability, points that are too close together (>100 points total or <100ms apart) were filtered out before graphing. This allows for visual inspection to see if the difference in model scores are statistically significant. In Chapter 6 these confidence intervals will be used in determining which algorithm is deemed best for a given situation.

As can be seen in the TLCs, in some cases such as Adult (binorm) (Figure 4.4), sfo (i.e., logistic regression) does very well. However in others, such as Intrusion (binorm) (Figure 4.18), bagging and boosting both outperform sfo for model quality, with bagging being faster than boosting. In Mushrooms (binorm) (Figure 4.20), tree induction takes an early lead, but sfo eventually matches it, and both are eventually surpassed by boosting. These results are exactly what is expected based on the no free lunch theorem [8].

Another thing that can be seen in TLCs is over-fitting. In the case of tree induction, for both Ads and Ads (binorm) (Figure 4.1 and Figure 4.2), towards the end the error bars grow considerably. This is most likely due to some of the trials over-fitting more than others, leading to greater uncertainty in the model quality. Another symptom of over-fitting can be seen in a TLC when the average model quality drops. This drop can be seen in Thrombin (binorm) (Figure 4.24) around 7,000 seconds.

Dataset	Node Split Method	Perc. Used	Max. Iters	Max. Height	Samples per Leaf	Number of Stumps
Ads	info gain	100	76	110	51	30
Ads (binorm)	info gain	100	88	75	118	147
Adult	msee	100	34	21	42	74
Adult (binorm)	info gain	100	38	14	16	66
Covertype	variance	100	78	12	51	30
Covertype (binorm)	variance	50	78	12	51	30
Credit	variance	100	110	177	1	109
Credit (binorm)	info gain	100	113	94	1	130
Example	variance	50	78	36	133	14
Example (binorm)	msee	100	78	12	83	18
Example2	msee	100	102	28	51	6
Example2 (binorm)	info gain	100	20	102	91	14
IntCensor	msee	50	82	12	83	22
IntCensor (binorm)	msee	50	78	20	67	30
IntShopping	info gain	100	46	30	32	82
IntShopping (binorm)	info gain	100	104	35	110	127
Intrusion	info gain	50	28	78	67	46
Intrusion (binorm)	info gain	50	12	78	51	30
Mushrooms	info gain	50	12	80	59	14
Mushrooms (binorm)	info gain	100	21	34	58	98
Pima	info gain	100	113	94	1	115
Pima (binorm)	variance	100	94	97	1	97
Thrombin	info gain	50	113	110	1	97
Thrombin (binorm)	info gain	50	97	94	1	97

Table 4.5: Hyper-parameters for top scoring bagged models.

Dataset	Node Split Method	Perc. Used	Max. Iters	Max. Height	Samples per Leaf	Number of Stumps	Output Interval
Ads	variance	50	56	30	51	35	12
Ads (binorm)	info gain	100	70	53	1	97	75
Adult	info gain	100	78	54	16	66	14
Adult (binorm)	info gain	100	74	34	21	42	97
Covertype	info gain	50	78	38	16	66	14
Covertype (binorm)	variance	50	74	21	50	42	97
Credit	variance	100	202	21	34	74	97
Credit (binorm)	variance	50	210	21	34	42	105
Example	info gain	100	68	67	30	71	12
Example (binorm)	info gain	100	40	51	46	35	12
Example2	info gain	50	24	51	30	35	12
Example2 (binorm)	info gain	100	24	51	30	35	12
IntCensor	info gain	100	90	34	21	42	113
IntCensor (binorm)	info gain	100	90	34	21	42	97
IntShopping	msee	100	138	21	34	42	113
IntShopping (binorm)	info gain	100	86	85	1	81	75
Intrusion	info gain	100	74	34	21	42	97
Intrusion (binorm)	msee	100	8	8	32	10	256
Mushrooms	info gain	100	106	34	37	42	105
Mushrooms (binorm)	info gain	100	102	53	1	97	91
Pima	variance	100	88	30	51	35	12
Pima (binorm)	variance	100	90	22	34	42	113
Thrombin	info gain	50	17	95	110	113	94
Thrombin (binorm)	info gain	100	1	95	94	97	94

Table 4.6: Hyper-parameters for top scoring boosted models.

Dataset	Node Split Method	Perc. Used	Max. Iters	Max. Height	Samples per Leaf	In Mem. Limit	Output Interval
Ads	variance	50	145	111	110	94	40
Ads (binorm)	variance	50	46	28	115	94	66
Adult	msee	50	97	94	95	94	40
Adult (binorm)	msee	100	38	14	70	81	1
Covertype	variance	50	75	49	15	55	58
Covertype (binorm)	msee	50	129	94	111	94	40
Credit	info gain	50	86	70	30	81	1
Credit (binorm)	info gain	100	90	53	58	50	97
Example	info gain	100	99	97	19	55	90
Example (binorm)	info gain	100	91	49	31	56	58
Example2	info gain	50	38	70	14	81	1
Example2 (binorm)	msee	100	38	67	16	78	68
IntCensor	variance	100	138	53	58	50	129
IntCensor (binorm)	info gain	50	38	70	22	81	1
IntShopping	variance	100	91	49	63	79	58
IntShopping (binorm)	variance	50	8	8	32	100	256
Intrusion	msee	100	113	94	95	94	40
Intrusion (binorm)	info gain	50	38	70	14	81	1
Mushrooms	variance	50	90	37	42	34	97
Mushrooms (binorm)	msee	100	70	51	28	78	82
Pima	msee	100	90	42	21	50	97
Pima (binorm)	variance	100	90	21	42	34	97
Thrombin	variance	100	30	12	51	78	66
Thrombin (binorm)	info gain	100	38	70	14	81	1

Table 4.7: Hyper-parameters for top scoring tree models.

Dataset	Perc. Used	Max. Iters	Full Reg. Mode	Prob. Bins	Level Bins	L_2 Factor
Ads (binorm)	100	34	0	Disabled	Disabled	21
Adult (binorm)	100	5	2	Disabled	Disabled	28.4
Covertype (binorm)	50	34	0	Disabled	Disabled	21
Credit (binorm)	50	8	2	20	20	0
Example (binorm)	100	34	2	Disabled	Disabled	21
Example2 (binorm)	100	8	2	20	20	0
IntCensor (binorm)	100	8	0	Disabled	Disabled	0
IntShopping (binorm)	100	94	0	40	35	95
Intrusion (binorm)	100	51	0	12	78	30
Mushrooms (binorm)	100	79	1	Disabled	Disabled	21
Pima (binorm)	50	8	1	20	20	0
Thrombin (binorm)	100	8	0	20	20	0

Table 4.8: Hyper-parameters for top scoring sfo models.

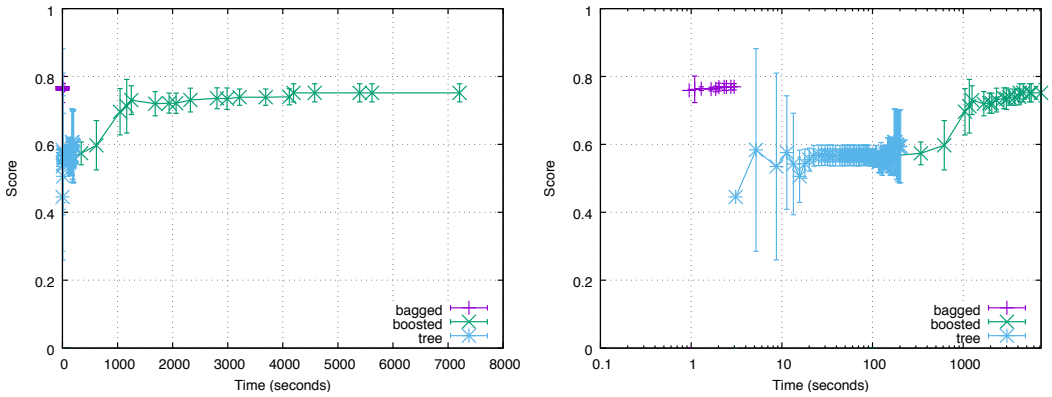

Figure 4.1: Learning curves for ADS on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

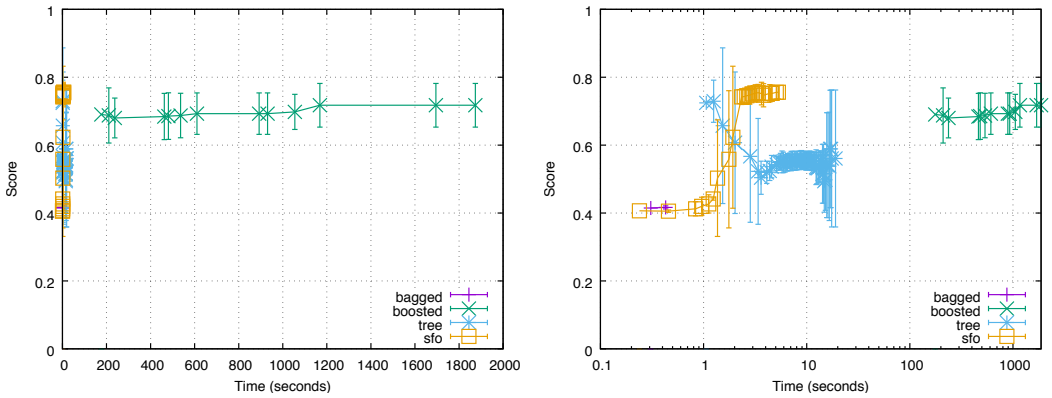

Figure 4.2: Learning curves for ADS (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

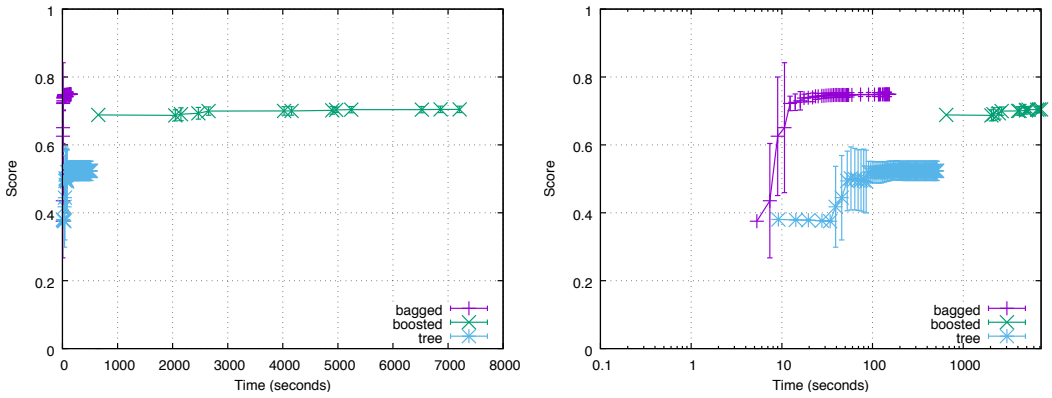

Figure 4.3: Learning curves for ADULT on 1 core with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

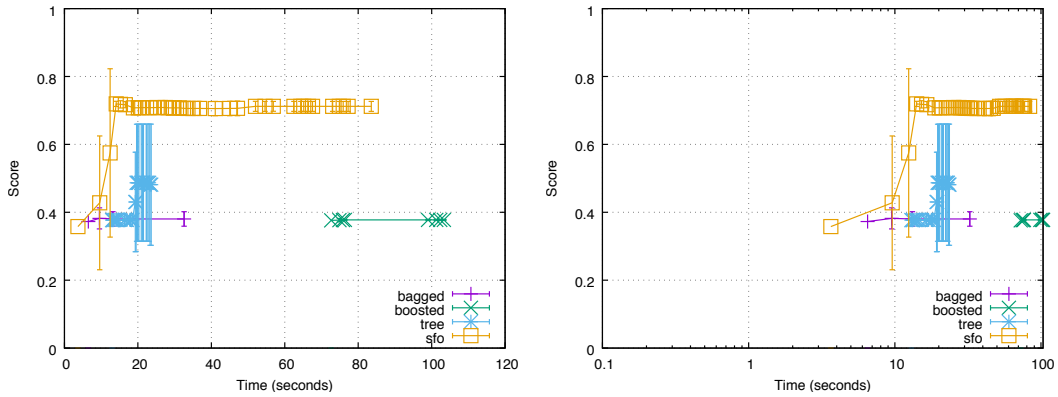

Figure 4.4: Learning curves for ADULT (binorm) on 1 core with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

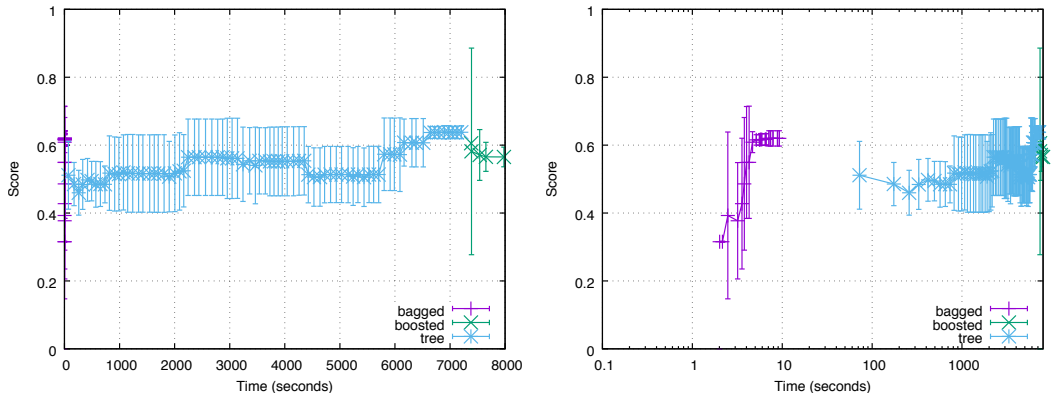

Figure 4.5: Learning curves for COVERTYPE on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

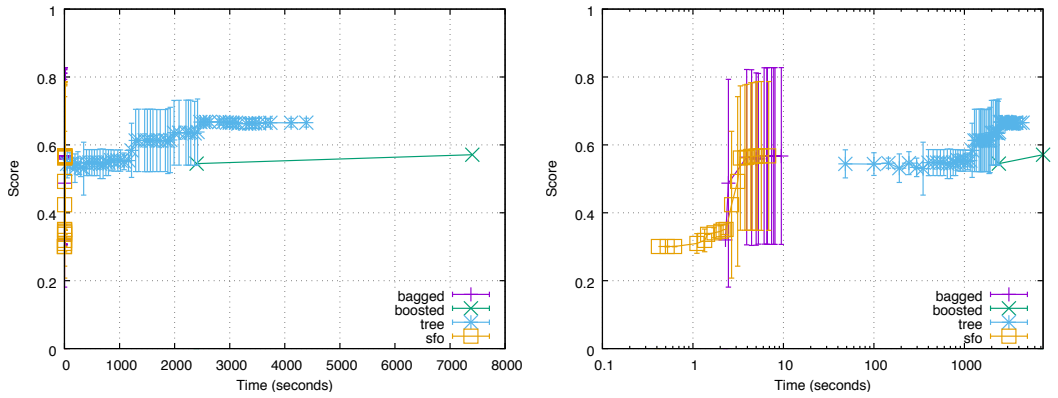

Figure 4.6: Learning curves for COVERTYPE (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

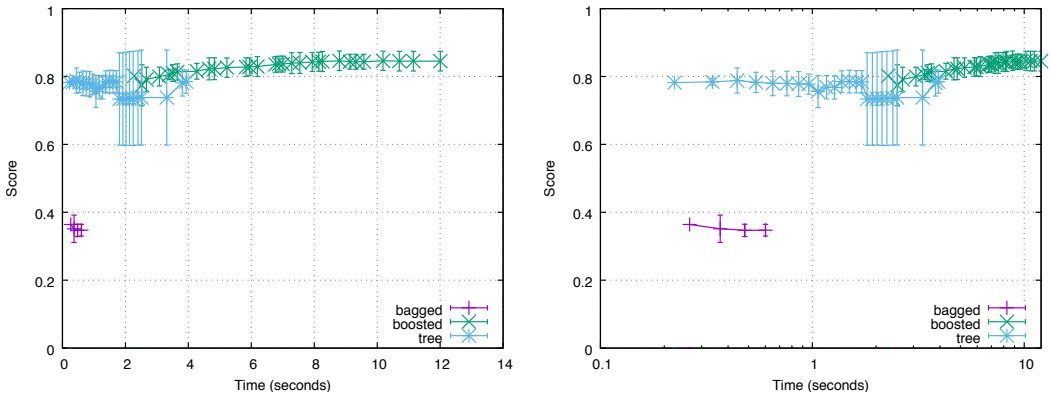

Figure 4.7: Learning curves for CREDIT on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

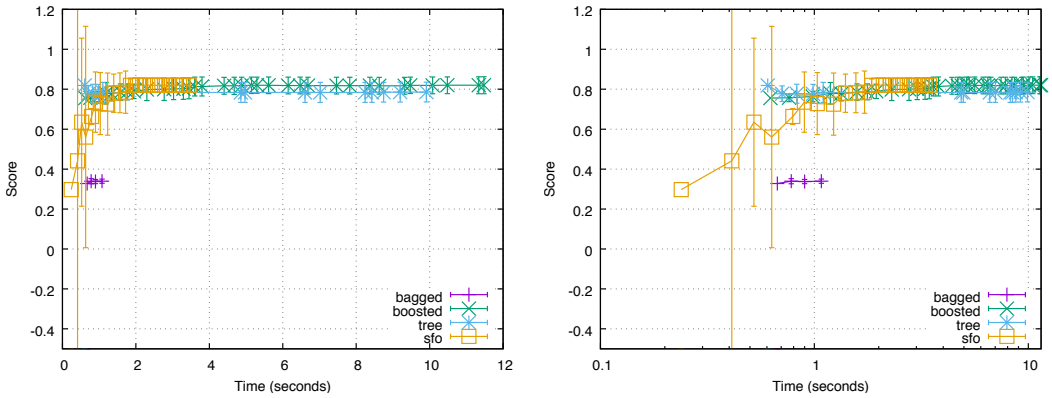

Figure 4.8: Learning curves for CREDIT (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

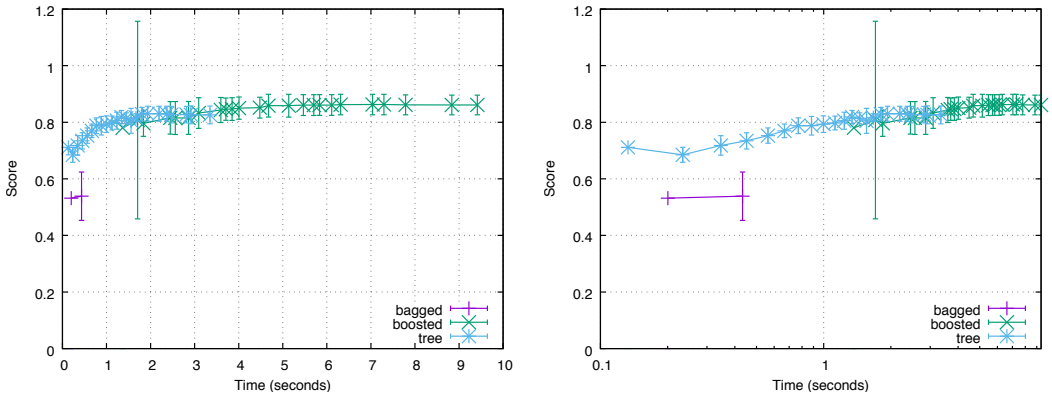

Figure 4.9: Learning curves for EXAMPLE on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

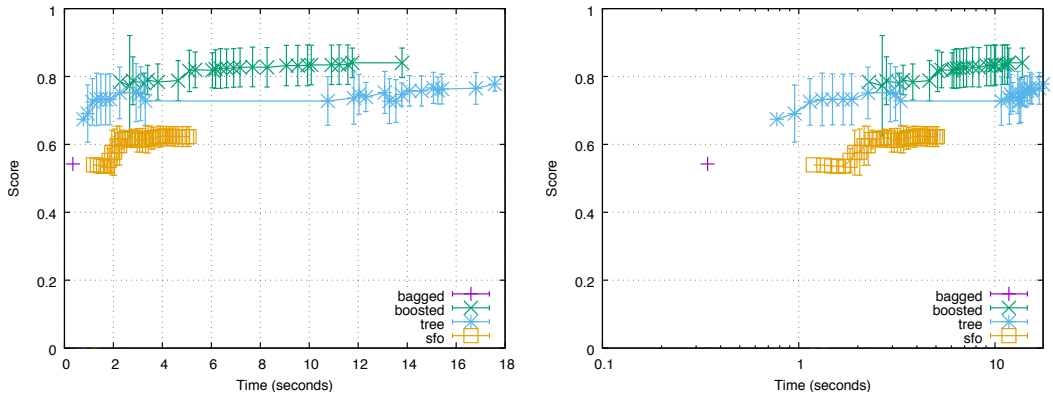

Figure 4.10: Learning curves for EXAMPLE (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

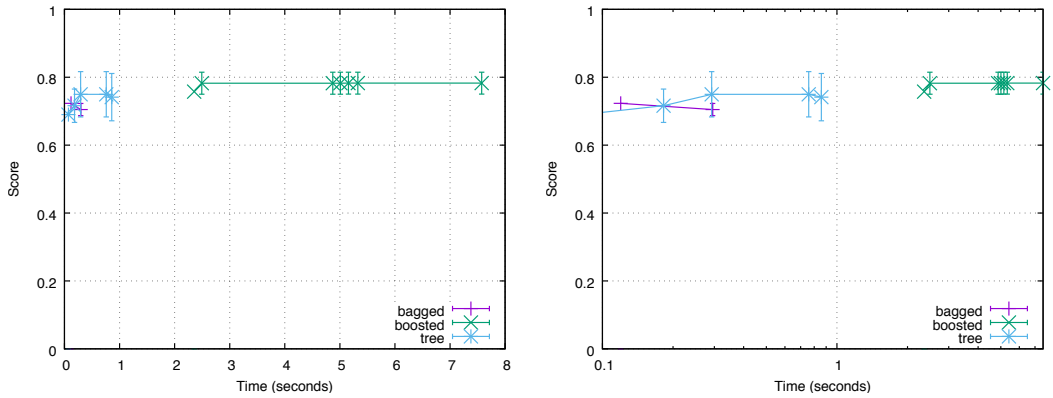

Figure 4.11: Learning curves for EXAMPLE2 on 1 core with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

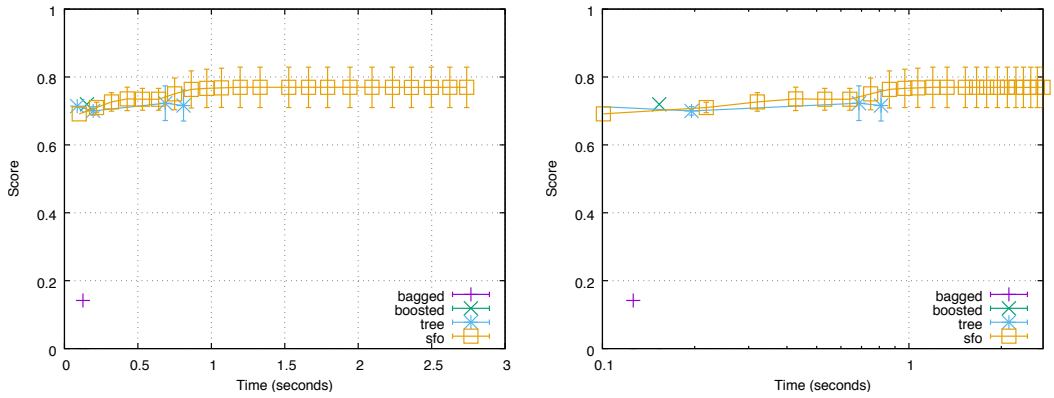

Figure 4.12: Learning curves for EXAMPLE2 (binorm) on 1 core with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

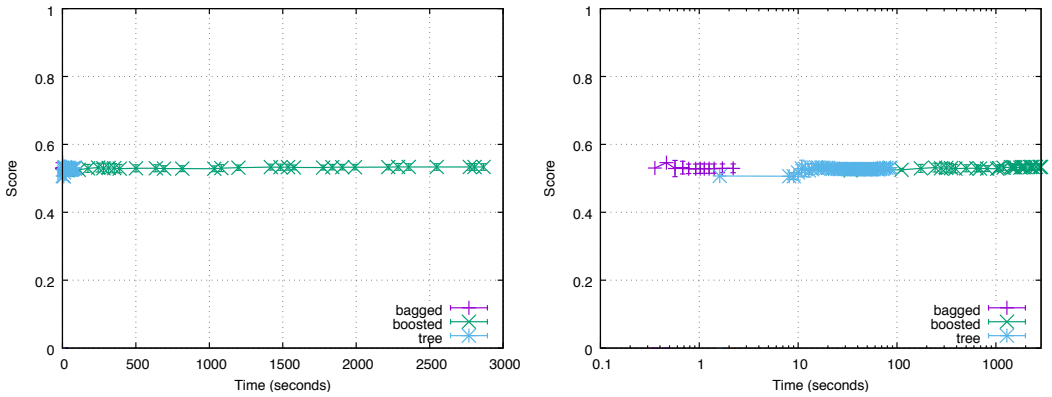

Figure 4.13: Learning curves for IntCensor on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

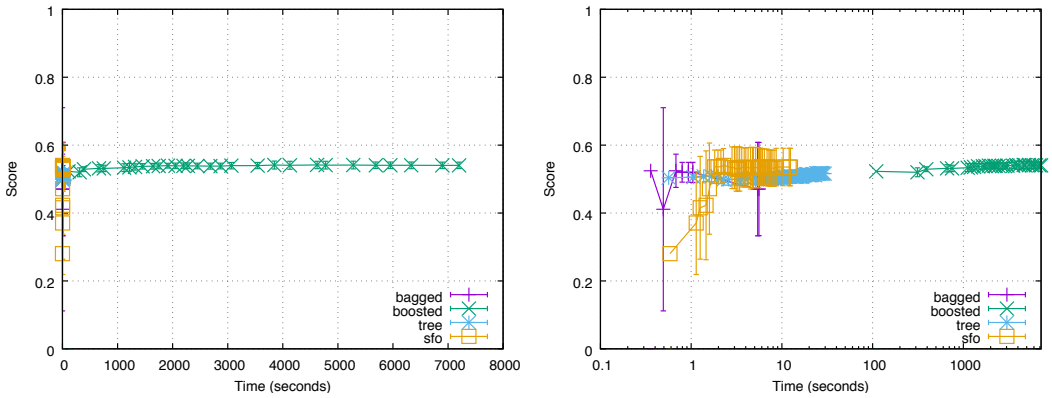

Figure 4.14: Learning curves for IntCensor (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

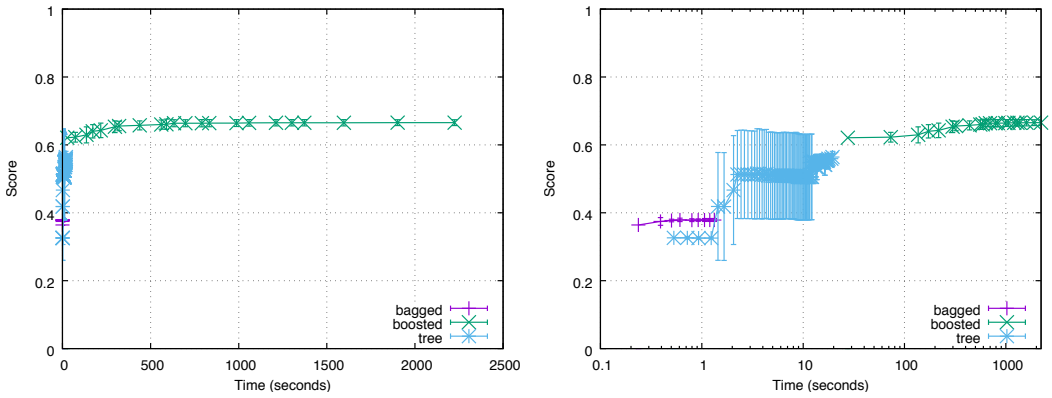

Figure 4.15: Learning curves for IntShopping on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

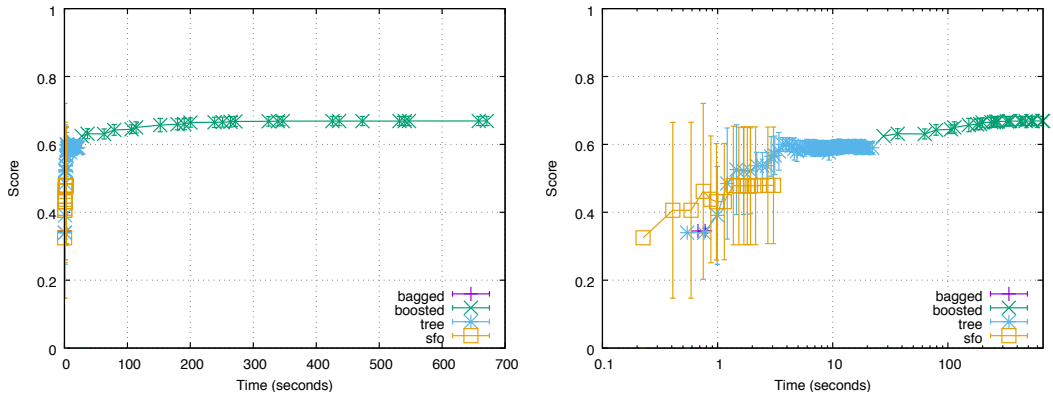

Figure 4.16: Learning curves for IntShopping (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

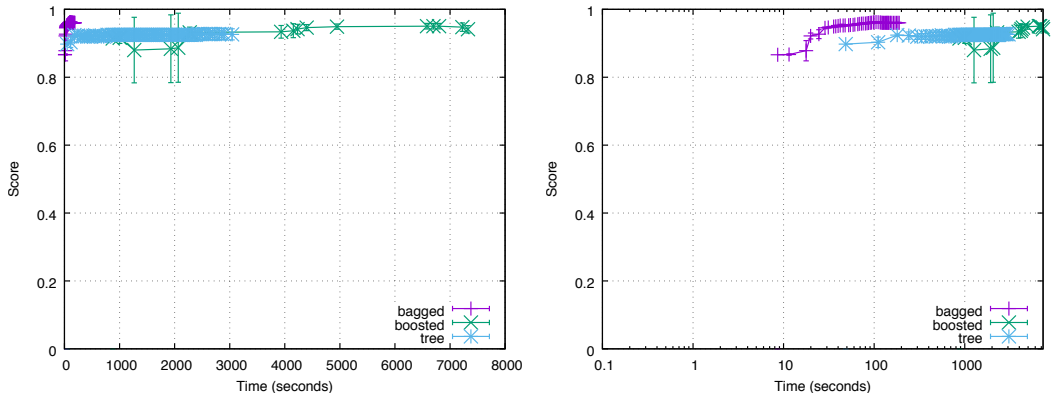

Figure 4.17: Learning curves for Intrusion on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

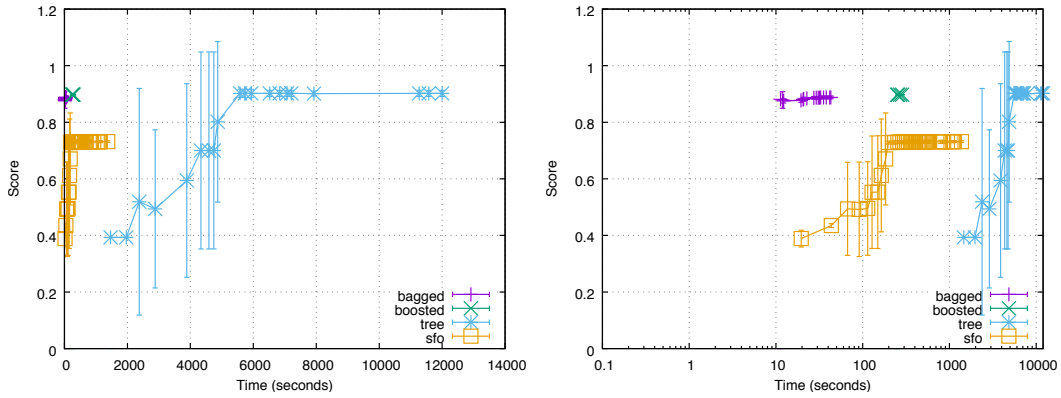

Figure 4.18: Learning curves for Intrusion (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

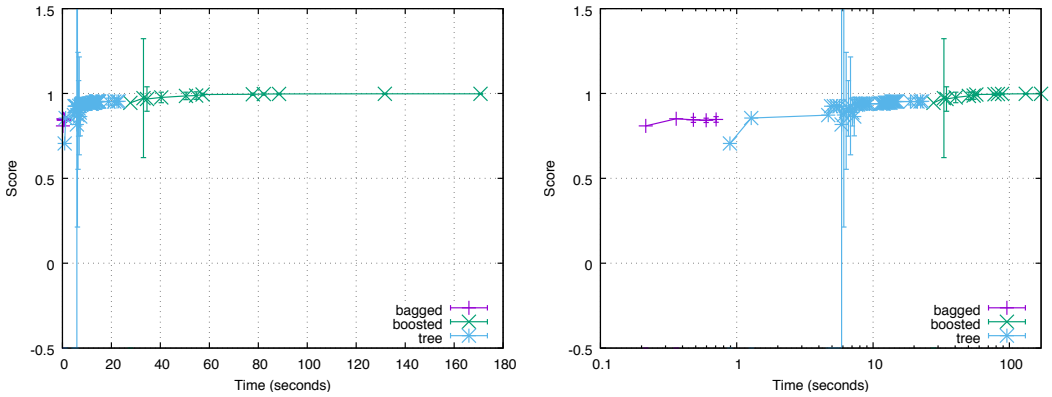

Figure 4.19: Learning curves for MUSHROOMS on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

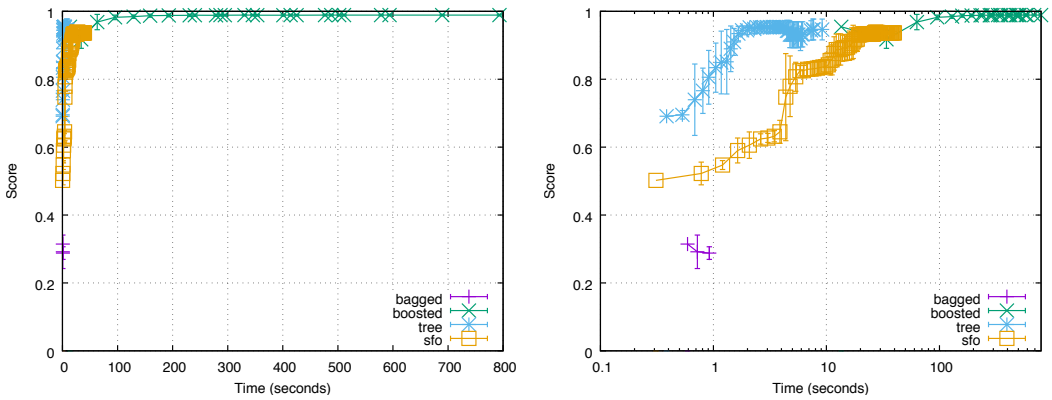

Figure 4.20: Learning curves for MUSHROOMS (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

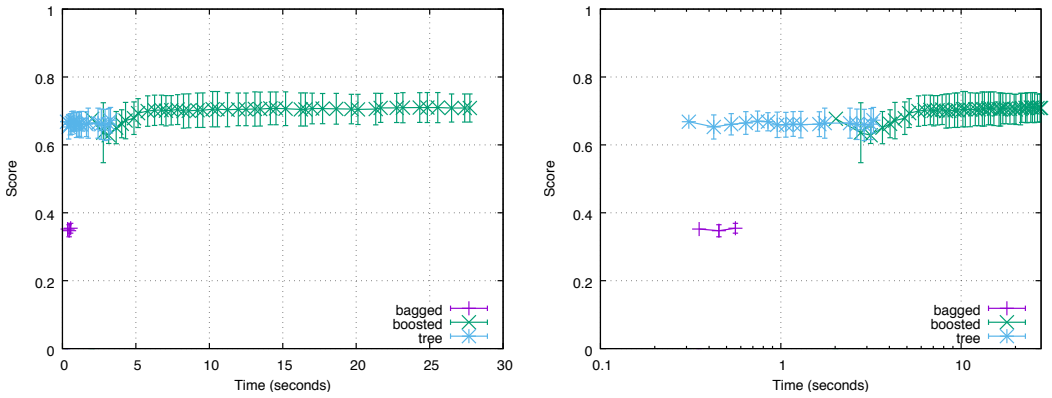

Figure 4.21: Learning curves for PIMA on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

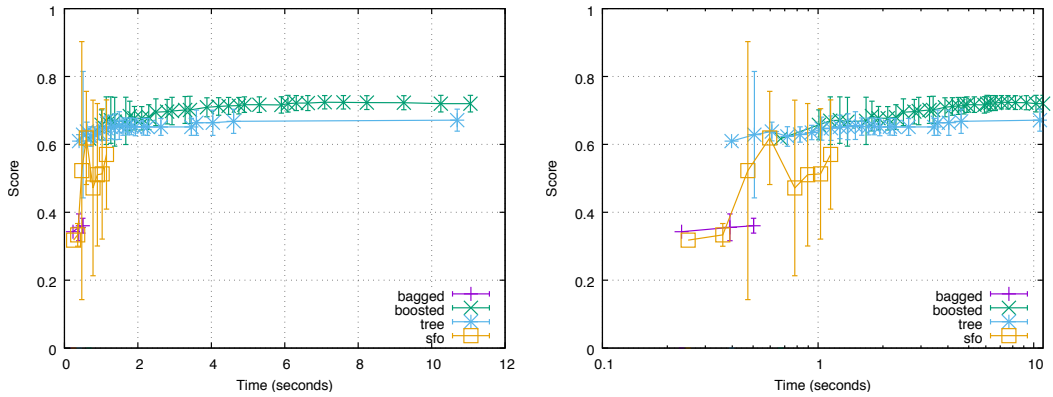

Figure 4.22: Learning curves for PIMA (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

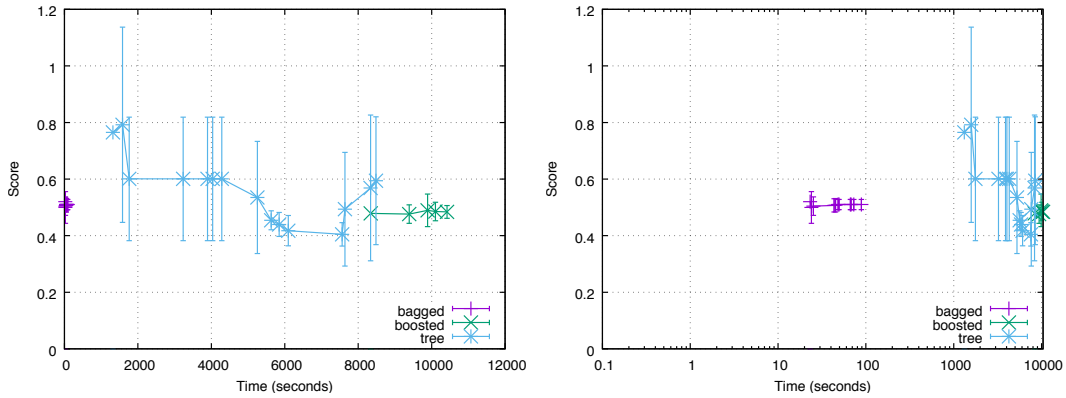

Figure 4.23: Learning curves for THROMBIN on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

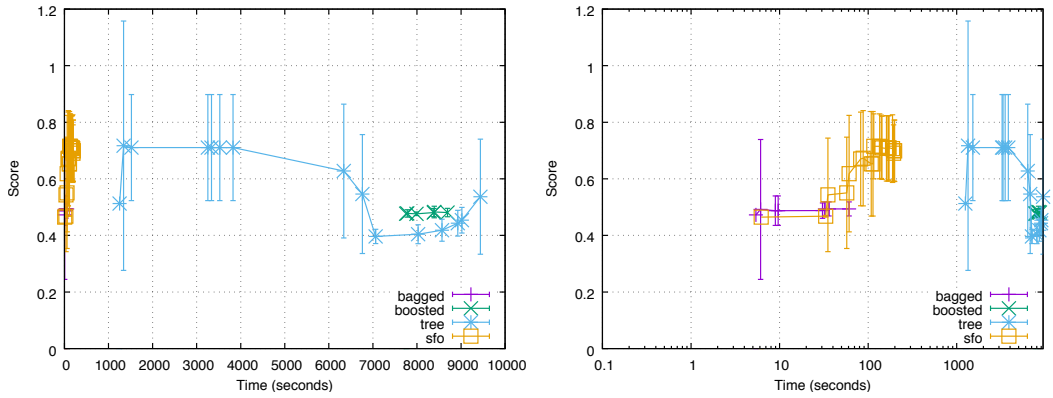

Figure 4.24: Learning curves for THROMBIN (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

4.3 Conclusion

The temporal learning curves produced demonstrate that for different datasets not only do the different algorithms produce vastly different model qualities, but over he course of training which algorithm produces the best models can change. This indicates that when training time is important, algorithm selection becomes an even more critical aspect of producing the best possible model. Additionally the wide ranges in hyper-parameters found while searching for optimal models indicate that fine tuning of hyper-parameters is also vital for getting the best performance, in terms of both speed and quality, out of each algorithms.

Chapter 5

Phase 3: Determining Parallel Performance by Relearning from Partial Models

When analyzing parallel algorithms, it is necessary rerun the exact same computation multiple times on different numbers of compute cores to measure the time it takes. Since the final product of the computation is always the same, this can be a very Sisyphean [100] task. Where Sisyphus was cursed by Zeus to push a boulder to the top of a hill, just to have the boulder roll back down the hill again, rerunning the same learning task over and over again also always ends with the same result. In this section, a method of collecting the required data is given that would be equivalent to allowing Sisyphus to start the boulder at any point on the hill he wishes.

In order to build a model to predict which algorithm would work best on a given number of compute cores in a fixed amount of time, data on the speed of each algorithm on different numbers of compute cores is needed. In a perfect world that would simply mean running the algorithms on a fixed number of cores, then scaling the run times linearly based on the number of cores. Unfortunately, due to serial portions of the algorithm, parallel overhead, and potentially unbalanced workloads, the change in run time is rarely linear with respect to the cores. Thus, the only way to truly determine the actual run time is to run the algorithm over and over again on different numbers of cores.

To rerun all of the experimental configurations on up to the 32 compute cores used when searching for optimal hyper-parameters would take an unreasonable amount of CPU time. Instead only powers of two were considered, with an upper limit of 16 compute cores.

All of the algorithms being studied are deterministic anytime algorithms, therefore it should be possible to use intermediate results obtained on a particular number of cores and rerun them on another number of cores and obtain similar, if not identical, results. Exploiting the anytime nature of these algorithms allows sampling of performance of individual model training iterations on varying numbers of cores without always starting from an empty model, and repeating the entire training process. However, due to data reordering and round-off issues, parallel algorithms that are mathematically deterministic may end up being slightly non-deterministic when implemented [101, 102].

In preparation for this re-sampling, the first time an algorithm is run on a dataset the models are recorded after each iteration. Then when run on a different number of cores, these models can be used as starting points, allowing the process to jump to any point in the training process.

The goal here is not only to determine which algorithm is best when run to completion, but also which one would have produced the best model if a training time limit were imposed. If the goal were simply to get overall speedup, using a weighted average of the iteration times at each recomputed point would be sufficient. However this study is interested in exploiting the anytime nature of the

algorithms being studied. For this reason, it is necessary to estimate the total run-time needed to get to each iteration of a run on a different number of cores.

To accomplish this, for single compute core runs, the iterations that were sampled were the first, a middle, and the final iteration. Since the number of iterations needed for each learning task varies, and can be quite large, if there was a sufficiently large number of iterations, every 50 iterations was also sampled. Using the iteration times for these sampled points, the intermediate iterations were interpolated by scaling the 32 core iteration times.

For numbers of cores other than one, repeating the process of rerunning the first, middle and last iterations for other numbers of cores would have been preferred, but the computational resources for that were not available. Instead, more extreme CPU time reduction was employed. When re-sampling on multiple cores, the iterations were sampled using the same pattern as with one core, but only every 20^{th} iteration was actually run. Since these resampled iterations are a subset of the resampled single core iterations, the ratio of run times can be used to scale the interpolated single core data to the number of cores being used. The ratio used for scaling was computed by taking the ratio at each re-sampled point, then averaging those ratios.

Once each iteration has an estimated run time, an estimate of how long it would have taken to complete a particular iteration assuming the learning had started from an initially empty model can be computed by summing the estimated iteration times for all of the previous iterations in that learning task.

5.1 Validation of Technique

There are many problems that can occur when comparing algorithms across different numbers of compute cores [102, 103]. Even on the same number of cores subtle reordering of computations can cause variations in the result due to the breakdown of the associative law. This is especially true of numerical methods due to loss of precision in floating point computations. Even the computationally simple task of summing a sequence of numbers can lead to different results depending on the order in which the additions are performed [101]. In Section 3.5, these effects can be seen in the fact that there is non-zero error in the model quality.

Since it is possible models might differ slightly when recomputed, it is necessary to verify that the models being produced are close enough to what would have been produced if the full algorithm had been run.

Some machine learning models can be represented as a vector (e.g. logistic regression), and in these cases direct comparisons of the actual models might be possible using the L_2 norm of the difference in the model vectors. However, for tree based models such simple comparisons are not so easy, as the difference between to graphs is not an easy to define concept.

In order to compare arbitrary models it would be better to focus not only on the model itself, but also its outputs. By taking two comparable models, the original dataset can be passed though both models and for each sample the mean absolute difference in their respective probability predictions can be computed as

$$\frac{1}{T \cdot N} \sum_{t=1}^{T} \sum_{i=1}^{N} |\hat{p}_{ti} - p_{ti}| \tag{5.1}$$

, where N is the number of samples, T is the number of unique splits of the data used, $\hat{p}_{ti}$ is the predicted value for sample x_i from one model trained during trial t, and p_i is the prediction from

the other model trained during trial t. This is not to be confused with mean absolute deviation, as described in Equation 6.4, which uses the absolute difference between samples and their mean. This instead is computing the difference between predictions of the same value from two different models. By aggregating these differences, an average and confidence interval can also be computed. If the confidence interval contains zero, then the difference between the models is statistically insignificant.

For each number of cores (1, 2, 4, 8, and 16), models were trained both by starting from an empty model, and by resampling each iteration from the models trained on 32 cores. Each of these models were then applied to the entire original dataset and the difference measures between the two types of models was computed. This was repeated for five datasets, using the highest scoring hyper-parameters for each dataset over five different splits of the dataset.

5.2 Runtime System

This portion of the project had originally been planned to use the same system as described in Section 4.1, but that system became unavailable. As a result the 24 node system described in Section 3.4 was used. In order to achieve a higher degree of parallelism, a rudimentary job scheduler (Plan N as described in Section 2.3.4) was implemented to allow dividing the cluster into equally sized chunks of cores. This scheduler however only allowed jobs of equal sizes to be run simultaneously.

5.3 Model Retraining on Varying Numbers of Cores

This phase of the project was intended to collect data that would allow estimating the total runtime of each algorithm on different numbers of cores. As such, no really meaningful conclusions were expected from the data.

5.3.1 Model Differences

As expected there was some variation between the predictions of the models. The differences shown in Table 5.1 through Table 5.4 show that while there is some variation in the predictions of the models produced, the variation is mostly small. In some cases the training of certain models failed leading to "missing" values. The "nan"s in Table 5.2 are most likely due to invalid models that produce "NaN" (not a number) predictions for some samples. A previous, far less rigorous, evaluation of this technique that looked only at the root mean squared error in the predictions and did not include confidence intervals had been used to justify using this technique. Unfortunately, these re-evaluated results indicate that the predictions made by the models when re-run on a different number of nodes are commonly statistically significant.

Timing information collected while measuring the difference in model prediction can also be compared. For the iteration time comparison, there are many more statistically significant differences. The most likely cause of the high variability in iterations times is excessive use of paging space, which is discussed further in Section 5.4.1.

Based on these results, rerunning only a portion of iterations based on previously trained models can lead to models that are significantly different than if they had been trained starting from an empty model. However, based on a preliminary evaluation it had been concluded that rerunning select iterations starting from partially trained models was a reasonable method for collecting the

Dataset	Cores					Number Significant
	1	2	4	8	16	
Credit	0.426	0.428	0.421	0.415	0.429	0
Credit (binorm)	0.00516	0.00812	0.0116	0.0299	0.066	0
IntCensor	0.224	0.225	missing	missing	*0.222*	1
IntCensor (binorm)	*0.00304*	*0.0265*	*0.0868*	*0.205*	*0.258*	5
IntShopping	*0.274*	*0.274*	*0.274*	*0.274*	*0.274*	5
IntShopping (binorm)	missing	missing	missing	missing	missing	0
Mushrooms	*0.275*	*0.275*	*0.275*	*0.275*	*0.275*	5
Mushrooms (binorm)	0	0	0	0	0	0
Pima	*0.00313*	*0.00442*	*0.0107*	*0.0245*	*0.043*	5
Pima (binorm)	*0.00439*	*0.0054*	*0.00767*	*0.024*	*0.047*	5

Table 5.1: Average differences in model predictions for bagged. Statistically significant differences from zero are show in *italics* (95% confidence)

Dataset	Cores					Number Significant
	1	2	4	8	16	
Credit	0.0837	0.0953	0.12	0.112	0.116	0
Credit (binorm)	0.0241	0.0649	nan	nan	nan	0
IntCensor	0.0412	0.136	missing	missing	*0.12*	1
IntCensor (binorm)	0	0	*0.178*	*0.103*	*0.0956*	3
IntShopping	*0.0626*	*0.142*	*0.136*	*0.136*	*0.12*	5
IntShopping (binorm)	0	*0.000788*	*0.0574*	*0.0869*	*0.0612*	4
Mushrooms	*0.00344*	*0.00223*	*0.00751*	*0.0215*	*0.00833*	5
Mushrooms (binorm)	0	*0.0997*	*0.0709*	*0.0657*	*0.0845*	4
Pima	0	*0.0499*	*0.198*	*0.169*	*0.217*	4
Pima (binorm)	0	*0.0879*	*0.241*	*0.184*	*0.162*	4

Table 5.2: Average differences in model predictions for boosted. Statistically significant differences from zero are show in *italics* (95% confidence)

Dataset	Cores					Number Significant
	1	2	4	8	16	
Credit	0.203	0.13	0.0839	0.0781	0.094	0
Credit (binorm)	0.118	0.0795	0.0754	0.0906	0.0929	0
IntCensor	0.0241	0.0168	missing	missing	*0.0245*	1
IntCensor (binorm)	missing	missing	missing	missing	missing	0
IntShopping	*0.059*	*0.0597*	*0.0611*	*0.06*	*0.0513*	5
IntShopping (binorm)	missing	missing	missing	missing	missing	0
Mushrooms	*0.444*	*0.359*	*0.152*	*0.0947*	*0.107*	5
Mushrooms (binorm)	*0.175*	*0.188*	*0.183*	*0.126*	*0.139*	5
Pima	*0.117*	*0.135*	*0.147*	*0.133*	*0.128*	5
Pima (binorm)	*0.16*	*0.145*	*0.14*	*0.172*	*0.174*	5

Table 5.3: Average differences in model predictions for tree. Statistically significant differences from zero are show in *italics* (95% confidence)

Dataset	Cores					Number Significant
	1	2	4	8	16	
Credit (binorm)	missing	missing	missing	missing	missing	0
IntCensor (binorm)	0	0	0	0	0	0
IntShopping (binorm)	missing	missing	missing	missing	missing	0
Mushrooms (binorm)	*2.98e-09*	*2.94e-09*	*6.6e-18*	*2.94e-09*	*2.94e-09*	5
Pima (binorm)	*0.000678*	*0.000678*	*0.000678*	*0.000678*	*0.000678*	5

Table 5.4: Average differences in model predictions for sfo. Statistically significant differences from zero are show in *italics* (95% confidence)

Dataset	Cores					Number Significant
	1	2	4	8	16	
Credit	0.292	0.366	0.628	0.521	0.512	0
Credit (binorm)	0.154	0.855	0.893	1.03	0.861	0
IntCensor	0.829	1.45	missing	missing	*1.41*	1
IntCensor (binorm)	*0.61*	*0.99*	*0.993*	*1.35*	*1.92*	5
IntShopping	0.918	*0.496*	*0.561*	*0.686*	*0.724*	4
IntShopping (binorm)	missing	missing	missing	missing	missing	0
Mushrooms	0.445	*0.589*	*0.718*	*0.844*	*1.09*	4
Mushrooms (binorm)	1.3	*0.275*	0.342	*0.629*	*1.21*	3
Pima	0.303	*0.461*	*0.623*	*0.487*	*0.398*	4
Pima (binorm)	0.214	*0.428*	*0.607*	*0.856*	*0.617*	4

Table 5.5: Average differences in iteration times (in seconds) for bagged. Statistically significant differences from zero are show in *italics* (95% confidence)

Dataset	Cores					Number Significant
	1	2	4	8	16	
Credit	2.06	0.96	1.81	1.77	1.98	0
Credit (binorm)	0.0731	1.3	1.02	1.65	2.44	0
IntCensor	888	827	missing	missing	*268*	1
IntCensor (binorm)	1.22	7.19	1.05e+04	*7.07e+03*	265	1
IntShopping	*3.86e+03*	*662*	*1.35e+03*	*276*	*192*	5
IntShopping (binorm)	0.322	1.6	1.28e+03	9.97	*1.08*	1
Mushrooms	325	27.6	14	5.13	3.88	0
Mushrooms (binorm)	0.554	0.405	3.1	3.61	4.94	0
Pima	0.129	*0.117*	*0.409*	0.378	*0.544*	3
Pima (binorm)	0.104	*0.149*	0.319	1.13	1.32	1

Table 5.6: Average differences in iteration times (in seconds) for boosted. Statistically significant differences from zero are show in *italics* (95% confidence)

Dataset	Cores					Number Significant
	1	2	4	8	16	
Credit	0.309	0.339	0.399	0.399	0.607	0
Credit (binorm)	0.255	0.282	0.335	0.496	0.797	0
IntCensor	16.8	2.83	missing	missing	*0.607*	1
IntCensor (binorm)	missing	missing	missing	missing	missing	0
IntShopping	*24.9*	*5.35*	*3.43*	*257*	*61.5*	5
IntShopping (binorm)	missing	missing	missing	missing	missing	0
Mushrooms	*1.38*	*0.404*	*0.304*	*0.439*	*0.53*	5
Mushrooms (binorm)	*0.467*	*0.509*	*0.424*	*0.484*	*0.755*	5
Pima	*0.266*	*0.269*	*0.348*	*0.447*	*0.657*	5
Pima (binorm)	*0.232*	*0.272*	*0.396*	*0.497*	*0.613*	5

Table 5.7: Average differences in iteration times (in seconds) for tree. Statistically significant differences from zero are show in *italics* (95% confidence)

Dataset	Cores					Number Significant
	1	2	4	8	16	
Credit (binorm)	missing	missing	missing	missing	missing	0
IntCensor (binorm)	*0.33*	*2.26*	*2.05*	7.81	*2.21*	4
IntShopping (binorm)	missing	missing	missing	missing	missing	0
Mushrooms (binorm)	*0.261*	*0.349*	*0.975*	*3.17*	*0.639*	5
Pima (binorm)	*0.24*	*0.433*	*0.464*	*0.701*	*0.663*	5

Table 5.8: Average differences in iteration times (in seconds) for sfo. Statistically significant differences from zero are show in *italics* (95% confidence)

required data in a the amount of time available. Also, given time and processor time constraints using this technique was necessary.

5.4 Speedup

To better understand how algorithms perform on differing numbers of cores, speedup curves are used. Speedup ideally is the ratio of the run time for the best serial algorithm to the parallel run time as given in Equation 5.2.

$$\text{speedup} = \frac{\text{time}_{\text{serial}}}{\text{time}_{\text{parallel}}} \tag{5.2}$$

However, fast serial implementations of these algorithms were not available, so the parallel algorithm run on a single core was used. To get a speedup curve, speedup is computed for many different numbers of computational units (cores).

5.4.1 Speedup Results

The speedup results are show in Figure 5.1 through Figure 5.24 and are arrange alphabetically by dataset name. For the binorm datasets, all four algorithms evaluated are shown, logistic regression (sfo) is not shown for the non-binorm datasets, as logistic regression cannot be run directly on those datasets. Each graph includes an "ideal" line which shows what speedup would be if it exactly followed the increase in number of cores used (i.e., *speedup* = *cores*). For each number of cores, the 95% confidence interval is given based on the average over all comparable iterations (i.e., iterations with the same hyper-parameters, data split, and iteration number).

Overall the speedup data shows expectable patterns. For smaller datasets, as the number of cores increased, the run time increased as in Figure 5.21. This is common when the parallel overhead (e.g., communication and synchronization) costs exceed the computation time. For some of the larger datasets, the speedup does increase as the number of cores is increased, but often comes back down as the parallel overhead starts to take over as shown in Figure 5.7 and Figure 5.19.

There are some anomalies in this data that are worth noting. For some datasets, Ads, Adult, Adult (binorm), IntCensor, and IntShopping, some of the speedup curves go way outside the possible range. In the case of Adult, the lower bound for the 95% confidence interval is around 17,000 for 16 cores, where the maximum possible speedup should be only 16. If these weird readings were due simply to random variations in the runtime system, it would be expected that the error bars would include part of the valid range as in Figure 5.2 and Figure 5.6. This is not the case, so the problem most likely was not just random fluctuations. Since speedup is the ratio of two different run times, the problem with these measurements could be in one or both of the run times. In all of these anomalous cases, the speedup is way too high, which means either the single core run time is too high, or the multiple core run is way too low. Of these options, it is more plausible that single core runs were consistently slower than expected, than it is that multi-core runs were consistently impossibly fast. Possible explanations for this slowness include memory usage, which could cause slowdowns due to excessive paging or garbage collection. In order to have the best chance of learning tasks finishing without running out of memory and minimize the need for aggressive garbage collection, the Java Virtual Machine (JVM) was configured to have the maximum amount of heap space, 3712MB for the 32-bit JVM used, and the nodes were configured with plenty of swap space to allow for such large processes. This means that for memory hungry learning tasks, most of the heap space would

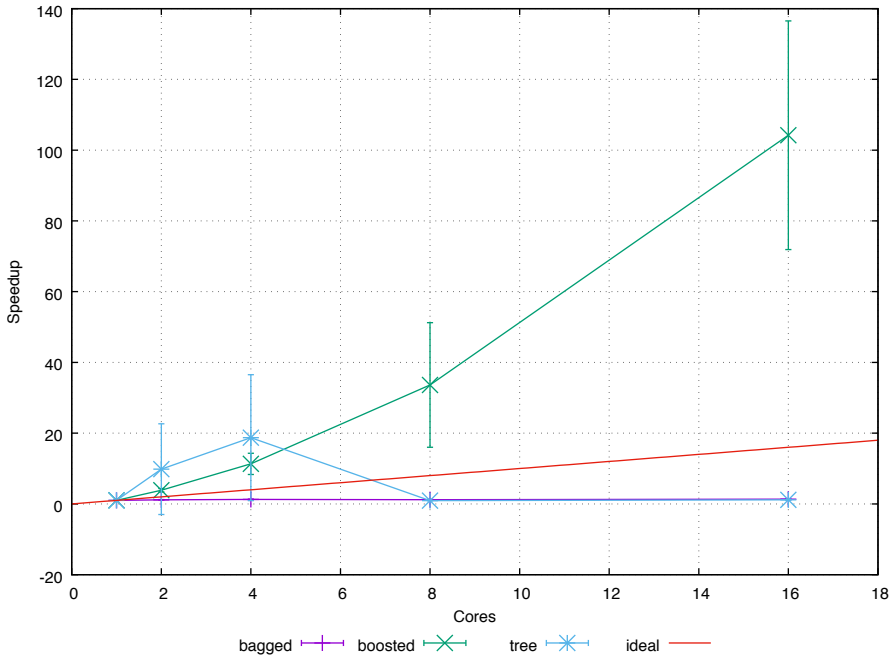

Figure 5.1: Speedup curves for ADS. Error bars are 95% confidence intervals.

end up paged out. It is worth reiterating that the cluster used here only has 1GB of RAM per core. The cluster that was used in Chapter 4, which was originally going to be used here as well, had 4GB of RAM per core.

Another type of anomaly that occurred can be seen in INTRUSION, INTRUSION (binorm), THROMBIN, and THROMBIN (binorm). In some cases, the anomaly is that certain algorithms are missing speedup data (e.g Figure 5.17 is missing bagging and boosting). In other cases not all data points for some algorithms are available, as in Figure 5.23. In all of these cases, the problem is that a single compute node was incapable of running the required learning tasks. So, while there may be data for the multiple core runs, there is no single core run with the same hyper-parameters to compare it to. Counter intuitively, the one data point for THROMBIN is for a single core. This is possibly because the single core iterations were sampled more frequently, so it was more likely that there would be a configuration that could finish on a single core.

The practical effect of these speedup anomalies is that the estimated time to reach a particular iteration may be incorrect. This could adversely affect the predictions of the final algorithm selection model when a time limit is applied. It should not affect the results when no time limit (i.e., an arbitrarily high limit) is specified.

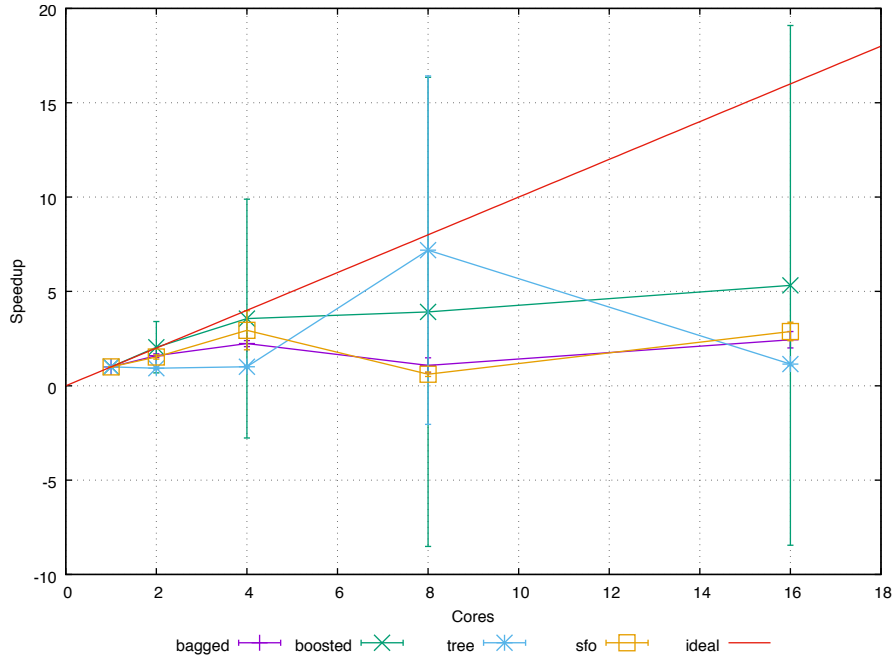

Figure 5.2: Speedup curves for ADS (binorm). Error bars are 95% confidence intervals.

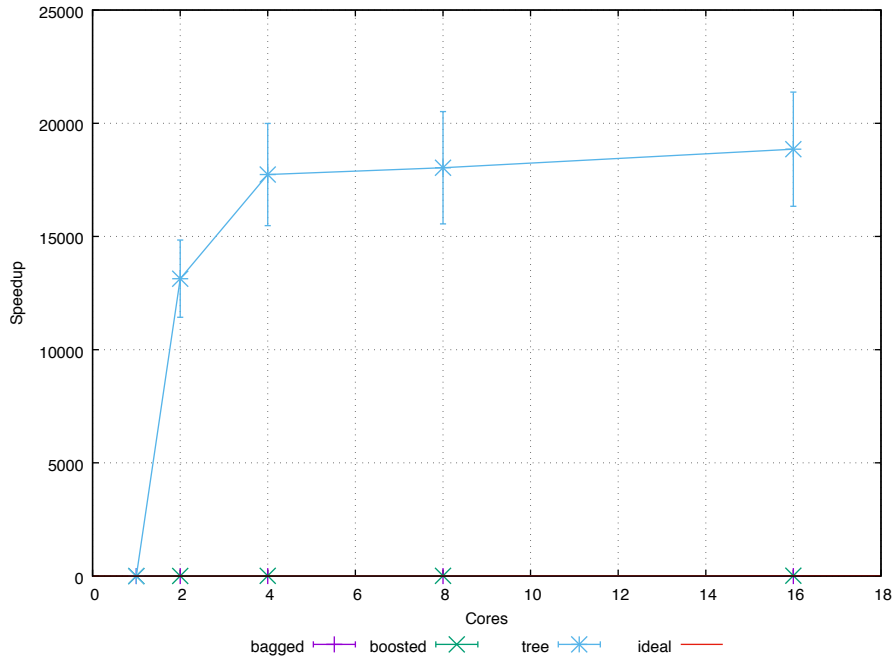

Figure 5.3: Speedup curves for ADULT. Error bars are 95% confidence intervals.

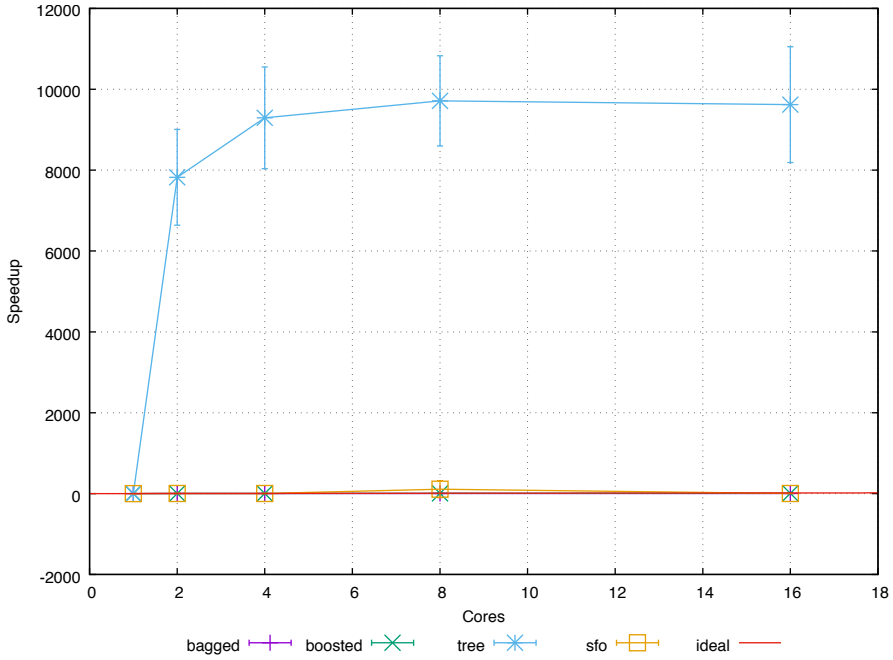

Figure 5.4: Speedup curves for ADULT (binorm). Error bars are 95% confidence intervals.

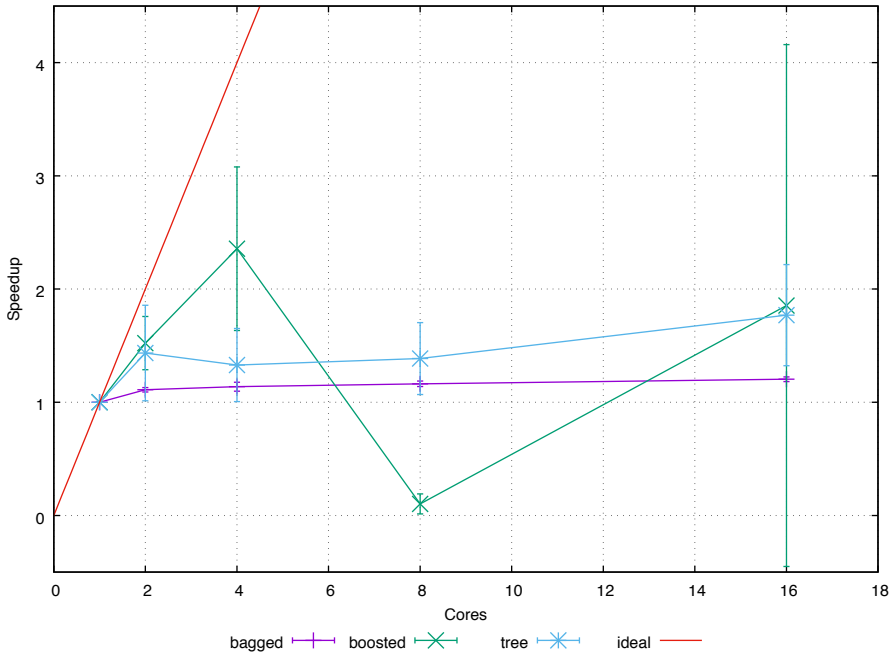

Figure 5.5: Speedup curves for COVERTYPE. Error bars are 95% confidence intervals.

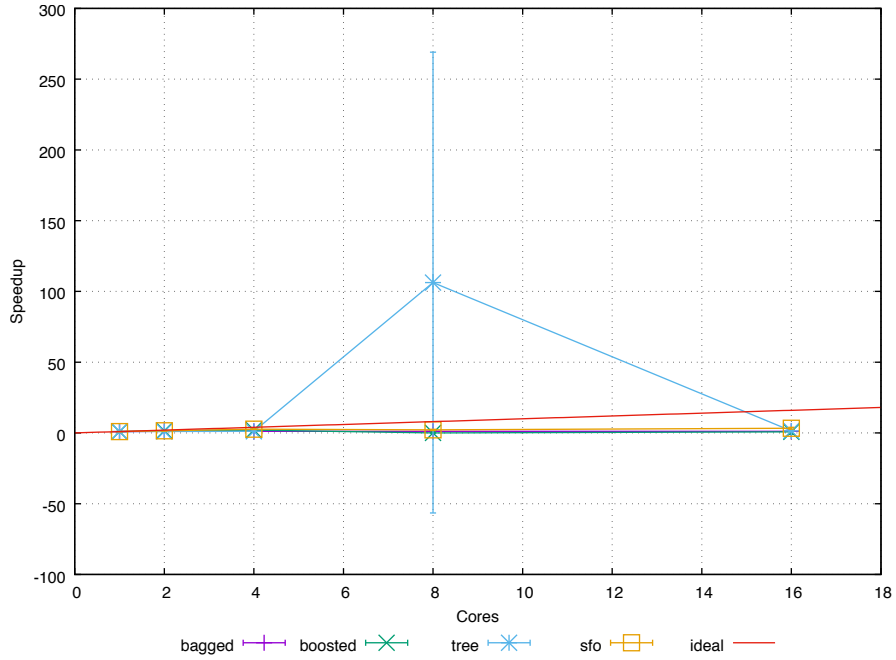

Figure 5.6: Speedup curves for COVERTYPE (binorm). Error bars are 95% confidence intervals.

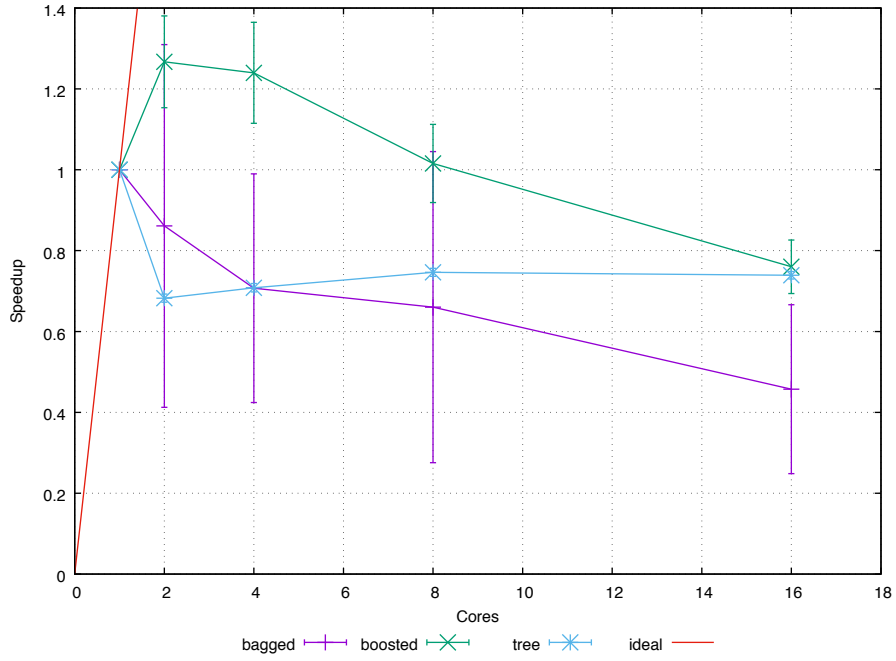

Figure 5.7: Speedup curves for CREDIT. Error bars are 95% confidence intervals.

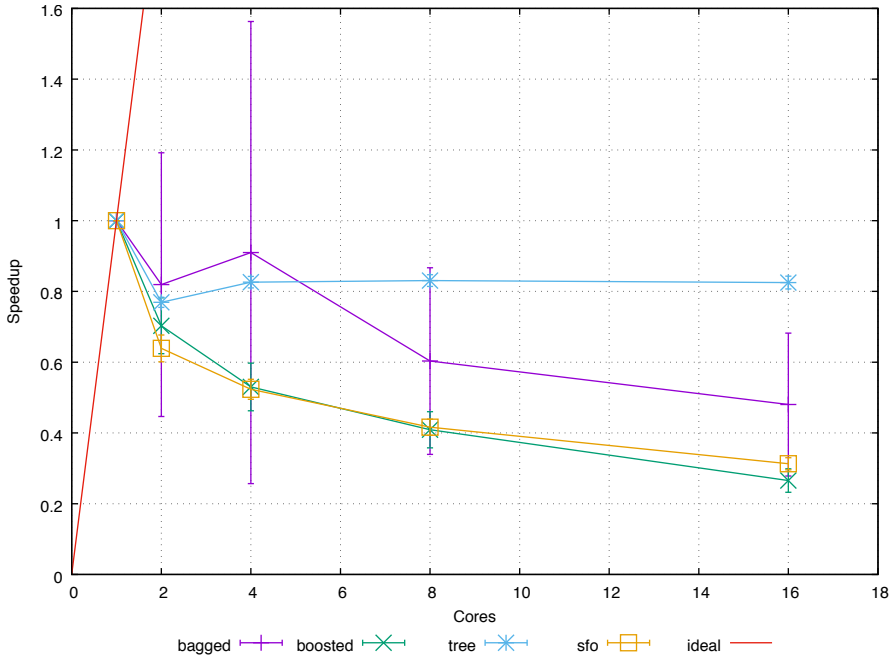

Figure 5.8: Speedup curves for CREDIT (binorm). Error bars are 95% confidence intervals.

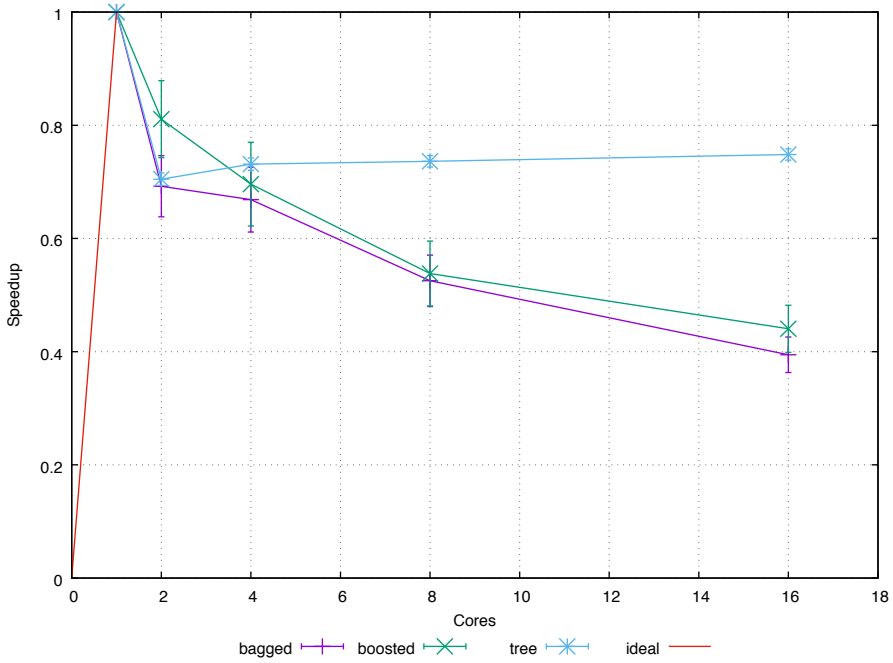

Figure 5.9: Speedup curves for EXAMPLE. Error bars are 95% confidence intervals.

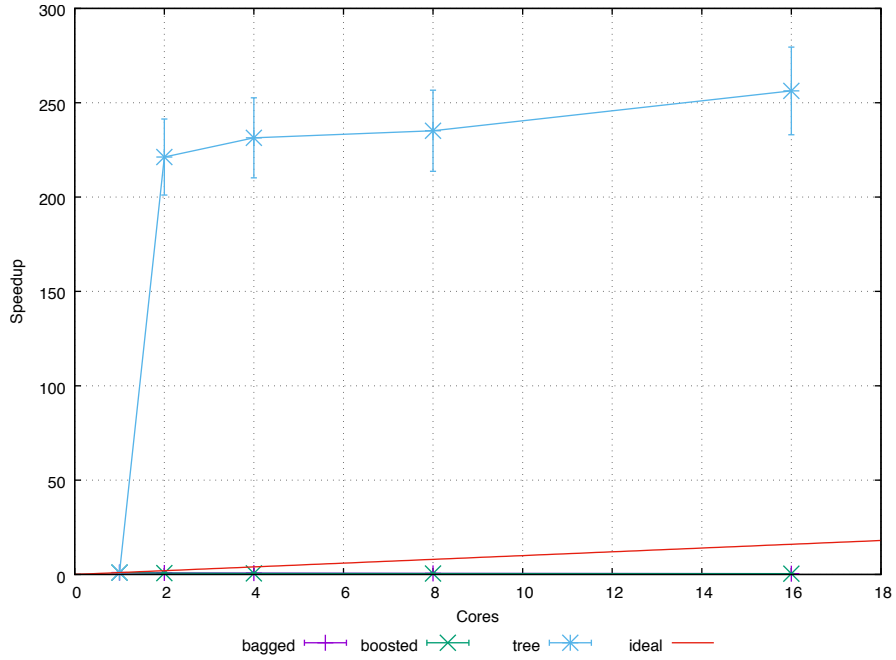

Figure 5.10: Speedup curves for EXAMPLE2. Error bars are 95% confidence intervals.

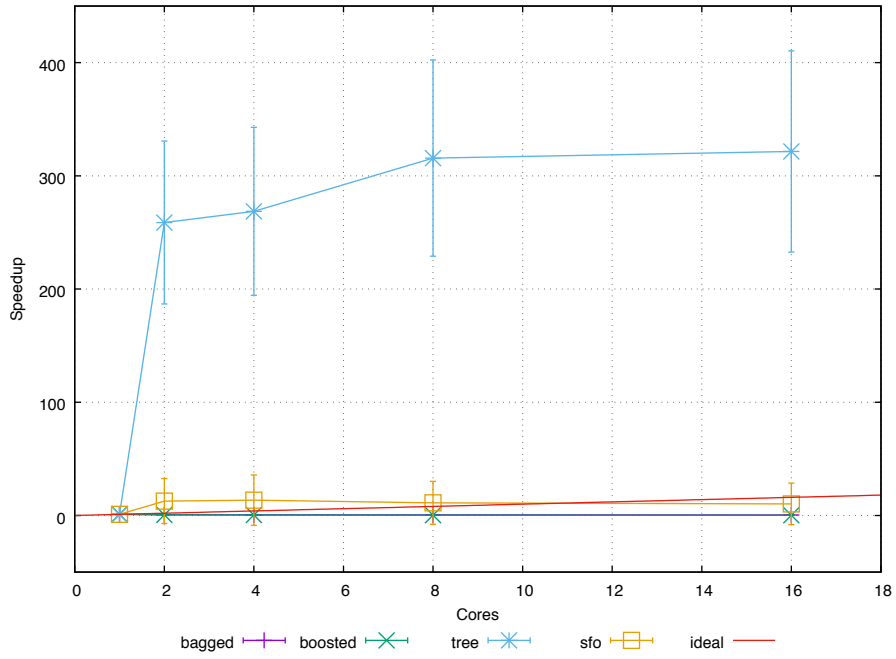

Figure 5.11: Speedup curves for EXAMPLE2 (binorm). Error bars are 95% confidence intervals.

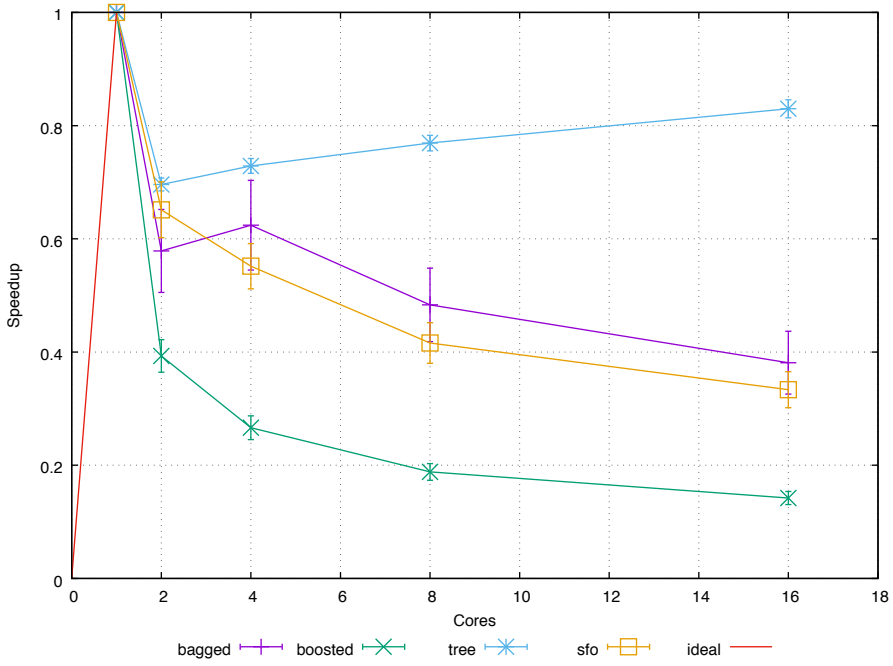

Figure 5.12: Speedup curves for EXAMPLE (binorm). Error bars are 95% confidence intervals.

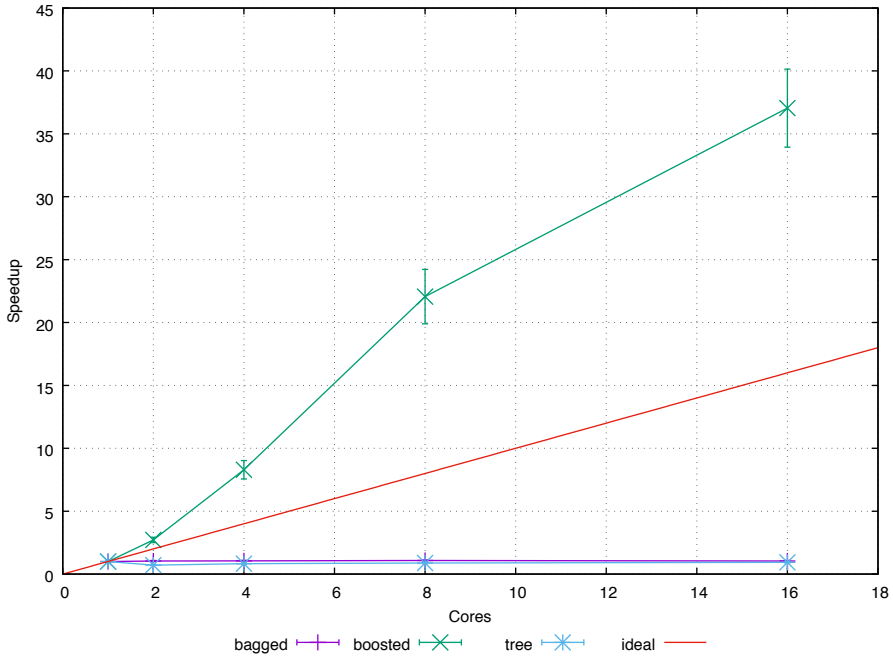

Figure 5.13: Speedup curves for INTCENSOR. Error bars are 95% confidence intervals.

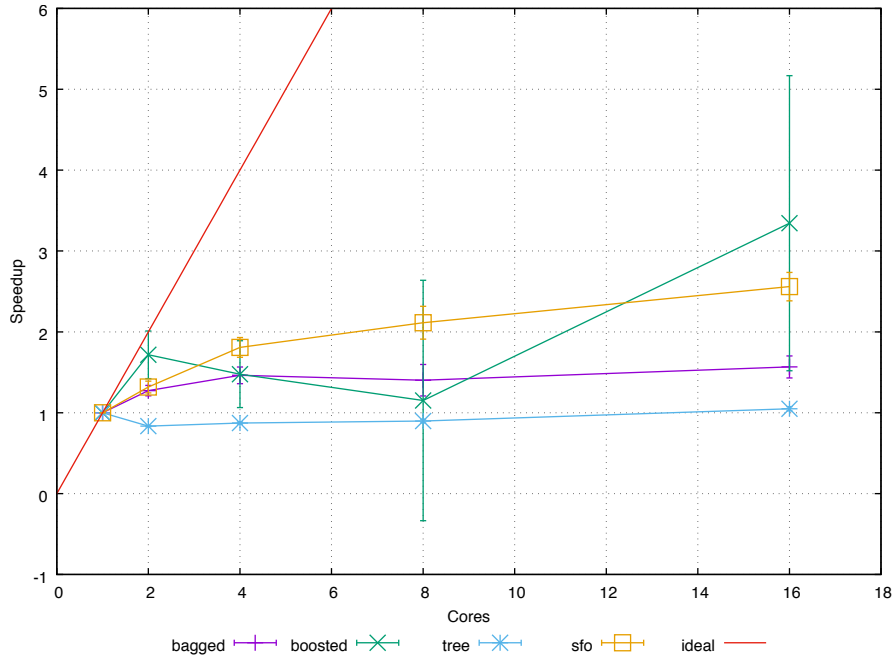

Figure 5.14: Speedup curves for IntCensor (binorm). Error bars are 95% confidence intervals.

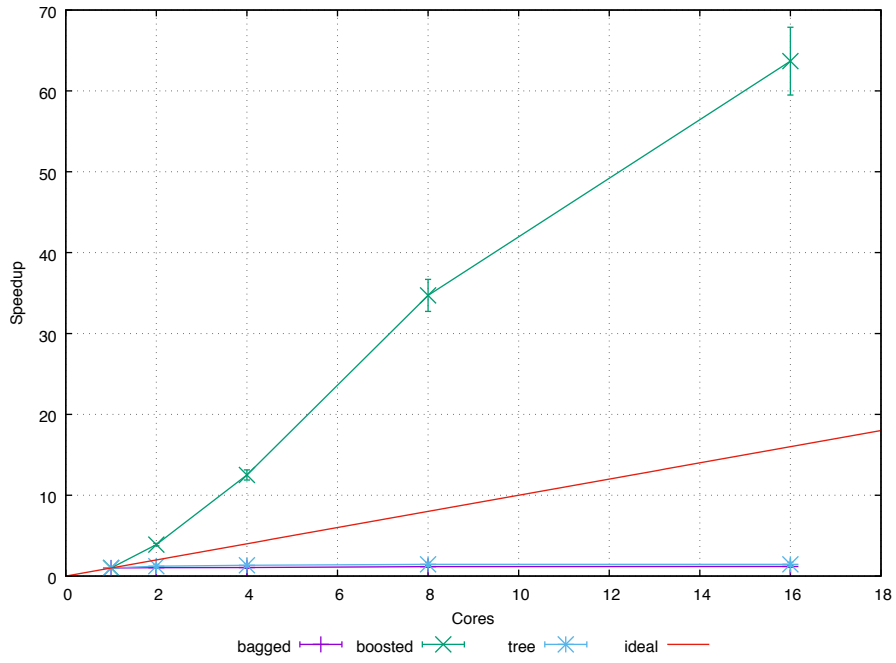

Figure 5.15: Speedup curves for IntShopping. Error bars are 95% confidence intervals.

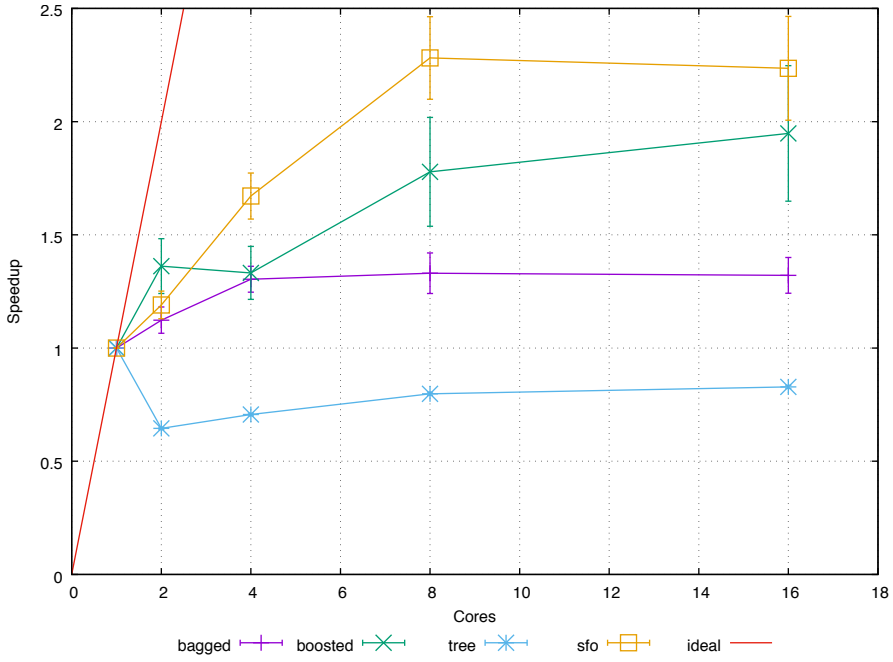

Figure 5.16: Speedup curves for IntShopping (binorm). Error bars are 95% confidence intervals.

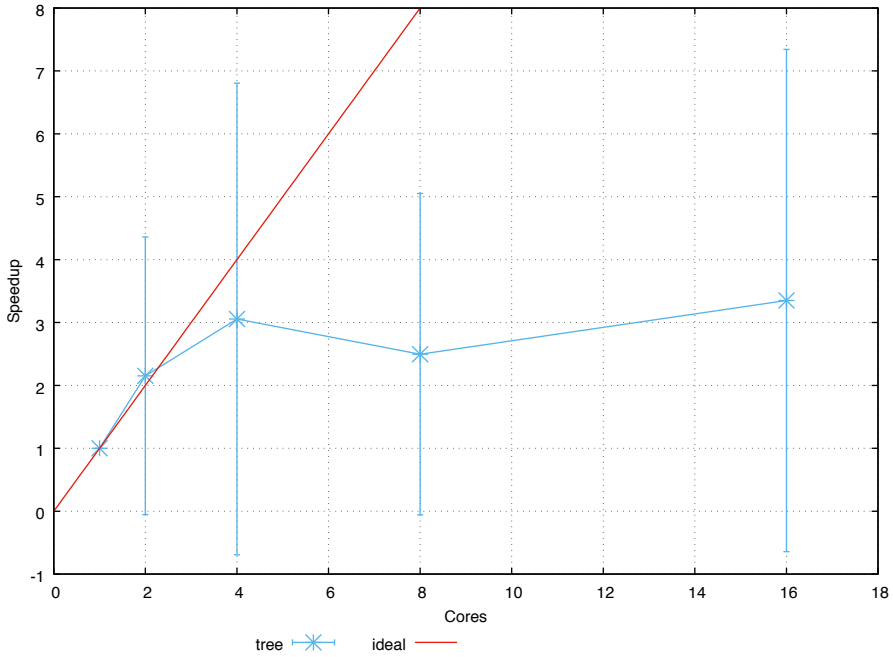

Figure 5.17: Speedup curves for Intrusion. Error bars are 95% confidence intervals.

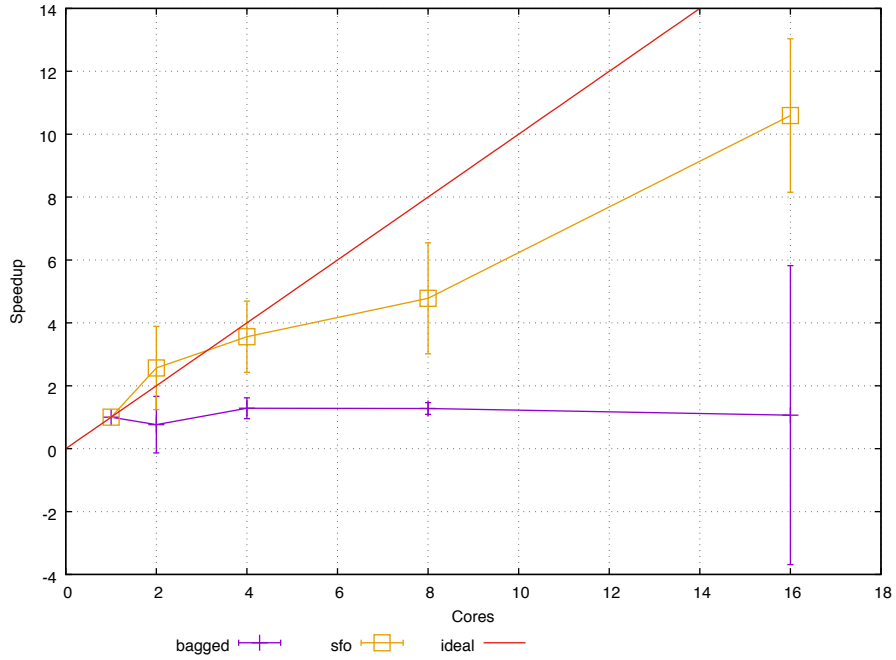

Figure 5.18: Speedup curves for INTRUSION (binorm). Error bars are 95% confidence intervals.

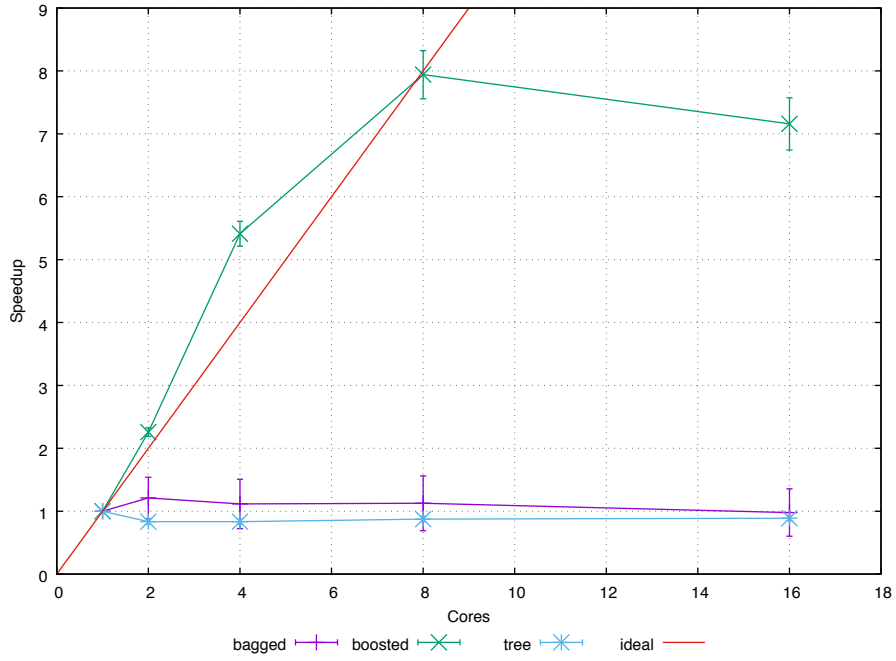

Figure 5.19: Speedup curves for MUSHROOMS. Error bars are 95% confidence intervals.

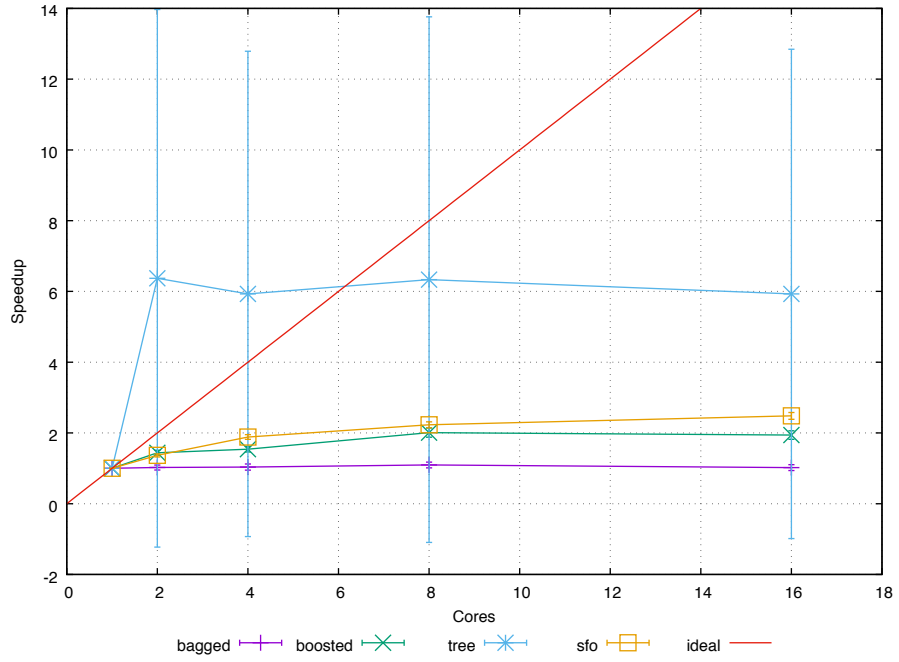

Figure 5.20: Speedup curves for MUSHROOMS (binorm). Error bars are 95% confidence intervals.

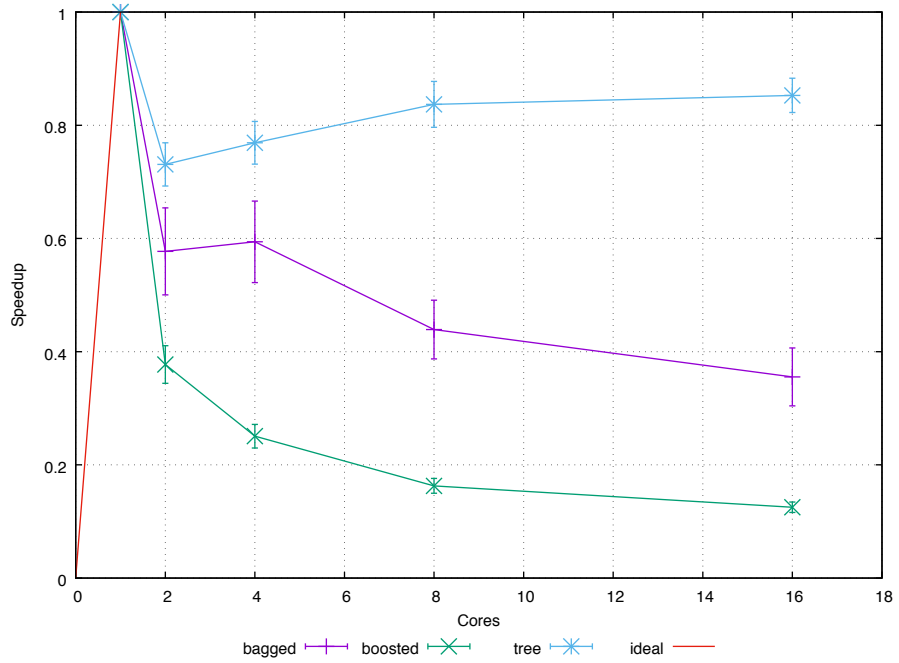

Figure 5.21: Speedup curves for PIMA. Error bars are 95% confidence intervals.

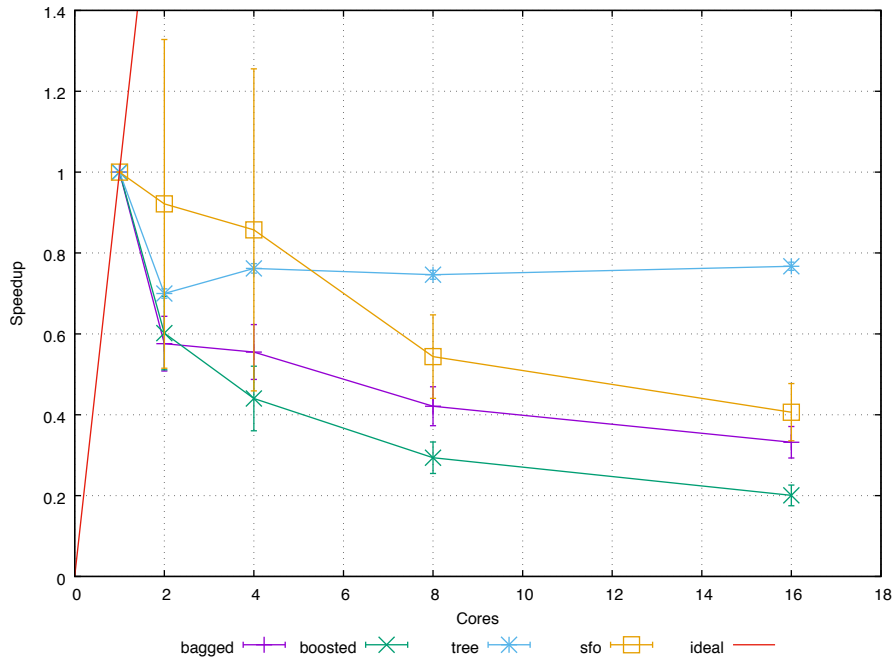

Figure 5.22: Speedup curves for PIMA (binorm). Error bars are 95% confidence intervals.

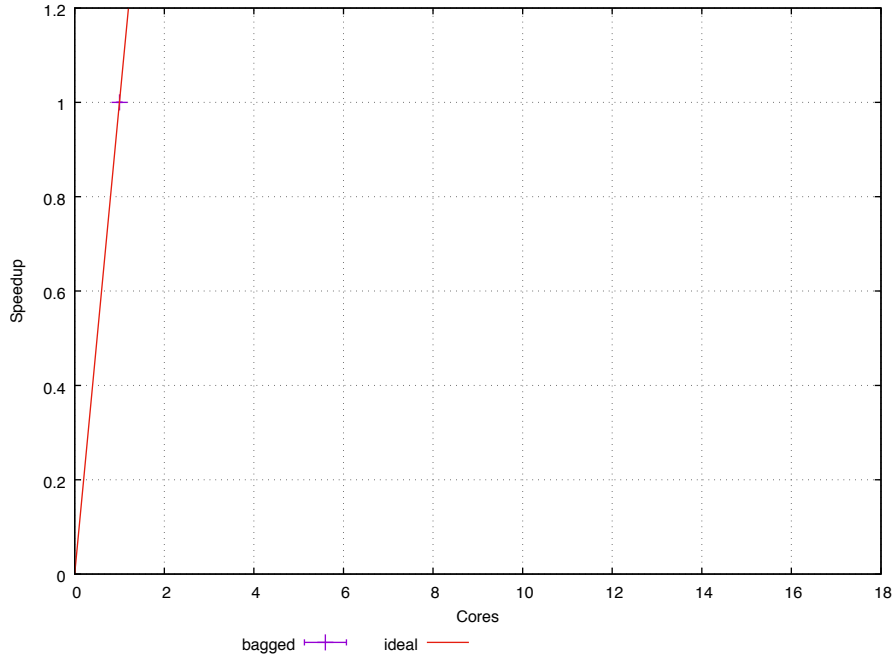

Figure 5.23: Speedup curves for THROMBIN. Error bars are 95% confidence intervals.

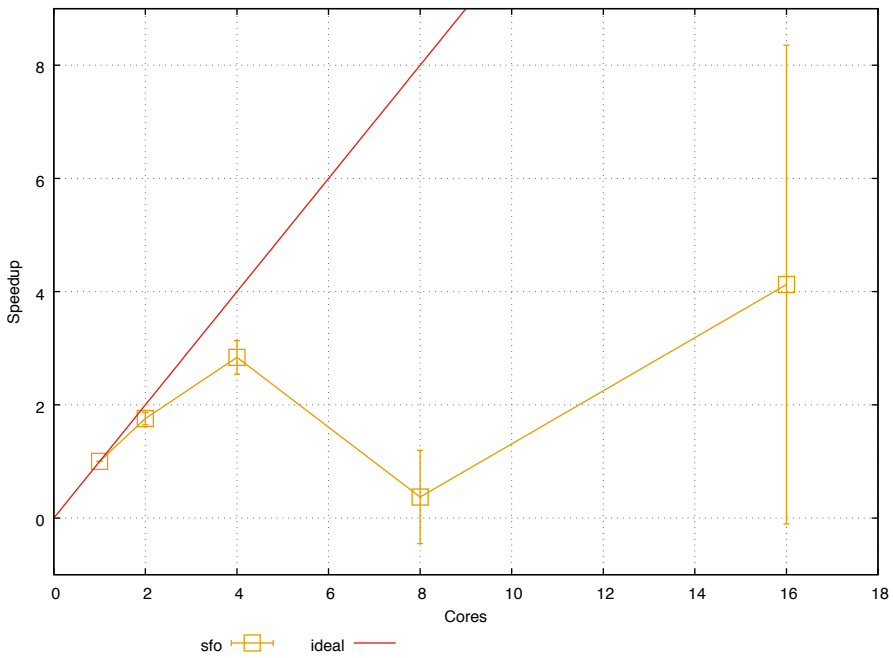

Figure 5.24: Speedup curves for THROMBIN (binorm). Error bars are 95% confidence intervals.

5.5 CONCLUSION

By comparing the model outputs from models trained from initially empty models to models produced by repeating single iterations based on models recorded during previous training runs on different numbers of cores, it was concluded that the differences in the resulting models were not significant. This result indicated that parallel performance data can be estimated by resampling individual iterations of each learning algorithms rather than having to start each algorithm from an empty model, thereby greatly reducing the time needed to collect the data required to build an algorithm selection model.

In terms of speedup, the results were a mix of mediocre results for smaller datasets and impossibly good results for many of the larger datasets. These impossibly good results are most likely due to extremely poor performance on a single core due to unfortunate interactions between the Java Virtual Machine (JVM) and the host virtual memory system.

Chapter 6

Phase 4: Training Algorithm Selection Models

Based on the data collected, models were trained to predict which algorithm would perform best on previously unseen datasets, given the characteristics of each dataset, a number of compute cores, and a time limit. The work of Perlich *et. al.* [7] indicates that small datasets with low separability tend to prefer logistic regression, where as large and more easily separable datasets tend to favor tree induction. Mitchie *et. al.* [46], King *et al.* [10], Kalousis [65], and Ali and Smith [47], all examined different measures of datasets and building models for algorithm selection based on those measures. However all of these studies were done using serial algorithms and did not take run-time into consideration. This chapter describes the dataset measures used to describe the datasets in this study, then uses those measurements to construct a dataset which models were then trained on to predict which algorithm will perform best on a dataset based on its characteristics. The dataset prediction model were then tested against datasets that had not been used in the creation of the models to see how well the models perform.

6.1 Dataset Properties

To build a model that can make predictions about datasets, an additional dataset that describes which algorithms perform best on known problems is needed. The classes in the additional dataset are the various algorithms tested, in this case, bagging, boosting, tree induction, and logistic regression. The attributes of the additional dataset are made up primarily of the characteristics of the datasets. Therefore the distinguishing characteristics of the datasets must be measured. These measurements, along with the number of compute cores and time limits, make up the attributes for the meta-learning problem of predicting which algorithm should work best. The measures used are described in Section 6.1.2 through 6.1.4.

The dataset measures used in this study were mostly adapted from Kalousis [65], but many of them have been used in additional studies as well [10, 47].

A full listing of the measures for each dataset is given in Appendix B.

6.1.1 Applying Univariate Measures to Multivariate Datasets

Many of the measures used to describe datasets are normally applied only to single set of numbers. Since datasets have multiple attributes, this presents a problem. When applying a single-variate measure, the samples are first divided into groups of the same class. Within each of those class based groups, the single-variate measure is applied to each attribute individually. This produces a single set of numbers (one for each class and attribute pair), the minimum, maximum, arithmetic

mean, median, and standard deviation of that set are all reported as shown in Table 6.2. For some statistics that are combined this way, a ten bin histogram is also produced, as well as a count of non-computable values as shown in Table 6.3.

6.1.2 Simple Measures

The simple measures describe the overall size, shape and general attribute and class makeup (i.e., the aspects of the dataset that can be counted). Many of these simple measures are used to define the more complex statistics and information theory based measures. The total number of samples, also known as instances or examples, in the dataset, is referred to as N. For datasets that are provided as a training set and a separate testing set, N is the number of samples in the training set. The total number of attributes, also known as attributes or variables that define individual samples, is referred to as J. The number of class labels in the dataset is referred to as C. While this work focuses exclusively on binary probability estimation, all of the metrics that are sensitive to the number of classes will be reported both for the original number of classes and just the two classes considered. The number of samples belonging to the positive (or majority) class is referred to as num_{pos} and the negative (or minority) class is referred to as num_{neg}. Likewise the percentage of samples belonging to positive and negative classes are referred to as $perc_{pos}$ and $perc_{neg}$, respectively. Similarly, percentages of attributes of different types are given. The number and percentage of: binary attributes are num_{bin} and $perc_{bin}$, categorical (nominal) attributes are num_{cat} and $perc_{cat}$, discrete ordered attributes are num_{ord} and $perc_{ord}$, continuous attributes are num_{cont} and $perc_{cont}$. The number and percentage of missing values across all samples and attributes are num_{miss} and $perc_{miss}$. In this study samples with missing values were either removed or the missing values were treated as an additional level. The ratio of attributes to samples (J/N) is the *dimensionality* of the dataset.

In addition to counting the samples and attributes, the unique values within each categorical attribute can be counted. The minimum, maximum, mean median and standard deviation of the unique value counts are $unique_{min}$, $unique_{max}$, $unique_{mean}$, $unique_{median}$, $unique_{stddev}$, respectively. Since binary attributes can be thought of as categorical with only two values, some of the unique value counts are repeated to include the binary attributes. These unique value counts with binary attributes includes are $unique2_{mean}$, $unique2_{median}$, and $unique2_{stddev}$.

These simple measures do not provide much information about the separability of the classes (i.e. how learnable the dataset is), but some of them are the main factors affecting the run times of a singe iteration of each algorithm. Some algorithms will be faster with large numbers of samples (N) where others will benefit more from large numbers of features (J).

6.1.3 Statistical Measures

This section provides descriptions of the various statistical measures used to describe datasets. Additionally, all non-numeric attributes are ignored for statistical measures.

Arithmetic Mean

The arithmetic mean ($\bar{x}$) of a set of numbers is the sum of the numbers divided by size of the set of numbers. That is to say, that if you have a set of numbers $x_i \in \{x_1, \ldots, x_N\}$, the arithmetic mean

is

$$arithMean = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{6.1}$$

.

Geometric Mean

The geometric mean of a set of numbers is the nth root of the product of n numbers. That is to say, if you have a set of numbers $x_i \in \{x_1, \ldots, x_N\}$, the geometric mean is

$$geoMean = \left\{\prod_{i=1}^{N} |x_i|\right\}^{\frac{1}{N}} \tag{6.2}$$

.

Harmonic Mean

The harmonic mean of a set of numbers is the reciprocal of the arithmetic mean of the reciprocals of each element in the set. That is to say, that if you have a set of numbers $x_i \in \{x_1, \ldots, x_N\}$, the harmonic mean is

$$harmMean = \left\{\frac{1}{N}\sum_{i=1}^{N} \frac{1}{x_i}\right\}^{-1} \tag{6.3}$$

.

Trim Mean

The *trimMean* of a set of numbers is the arithmetic mean of a trimmed version of the set. More specifically, an equal number of values are removed from both the high and low ends of the dataset. For this study the top 10% and the bottom 10% were removed. To get a trim mean for the entire dataset, the arithmetic mean of trim means for each attribute for each class are reported.

Mean Absolute Deviation

The mean absolute deviation (*mad*) of a set is the arithmetic mean of the distances from the overall mean of the set,

$$mad = \frac{1}{N}\sum_{i=1}^{N} |x_i - \bar{x}| \tag{6.4}$$

, where $\bar{x}$ is the arithmetic mean, given in Equation 6.1.

Variance

The variance of a set is similar to the *mad*, but uses the quadratic difference from the mean. That is to say, the average of the square of the distances between each value and the overall mean.

$$variance = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2 \tag{6.5}$$

, where $\bar{x}$ is the arithmetic mean, given in Equation 6.1.

Standard Deviation

The standard deviation of a set is the square root of the variance,

$$stdDev = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \tag{6.6}$$

, where $\bar{x}$ is the arithmetic mean, given in Equation 6.1.

90^{th} Percentile

The 90^{th} percentile ($perc_{90}$) is the smallest element of a set where 90% of the elements in the set are smaller than it. This is equivalent to sorting the set and finding the element where the largest 10% of values are on one side and the smallest 90% values are on the other side.

Inter-Quartile Range

The inter-quartile range (*IQ-range*) is the difference between the first and third quartile value. The first quartile, is similar to the 90^{th} percentile, but is the value for which one quarter are smaller than it, and three quarters are larger than it. Likewise, the third quartile is the value for which three quarters are smaller than it and one quarter is larger than it. The difference between these two numbers is the inter-quartile range.

Median

The *median* of a set of numbers is the value for which 50% of the values in the set are smaller than it. The median is also the 50^{th} percentile.

Canonical Correlation

In canonical correlation, the samples are considered to be vectors in J-dimensional space. These vectors are then projected onto $J-1$ mutually orthogonal vectors. The first vector that the samples are projected onto is chosen with the following constraints: 1) the centroids of each class (i.e. the arithmetic mean of all samples belonging to that class) have the distance between them maximized, and 2) the distance between samples within each class are minimized when projected onto the vector. A second vector can then be chosen according the the same constraints that is orthogonal to the first

vector. A total of $J-1$ vectors numbered 1 to $J-1$ can be chosen this way, where each subsequent vector is mutually orthogonal to all previous vectors.

This measure was implemented using the `canoncorr` function in MATLAB. The first matrix consisted of a row per sample, with the columns being the numeric attributes of the dataset. The second matrix also has a row per sample, but there was a column per class where all of the values in a row were set to zero, except the position corresponding to the sample's class, which was set to one. The values reported for $cancorr_1$ through $cancorr_4$ are elements one through four of the r vector returned by the `canoncorr` function.

For datasets where r contains fewer than four elements, NaN is reported for the non-existent elements.

Fraction of Separability (due to Canonical Correlation)

The fraction of separability ($frac_k$) [46, 10, 65] gives the proportion of variation due to the first k linear discriminants. It is computed by taking the ratio of the sum of the k largest eigenvalues to the sum of all of the eigenvalues corresponding the the eigenvectors produced in canonical correlation.

This process involves inverting a matrix. However, if there is an attribute for which all of the values are the same for any class, the matrix that needs to be inverted will contain columns and rows that are all zeros. This makes it impossible to invert the matrix. To get around this limitation, any such rows and columns are removed from the matrix prior to inversion. The effect of this modification is that it ignores any attributes that lack sufficient variation to contribute to learning.

Inverting matrices is also a very time consuming operation, so for datasets with large (>10,000) numbers of numeric attributes, this measure was not computed and is reported as NaN.

It may also be the case that the eigenvalues found might be complex. To ensure that all of the dataset characteristics are real values, the absolute value of the eigenvalues was used.

As with canonical correlation, for any dataset that does not have sufficient linear discriminants, NaN is reported for the remainder of the values.

Difference in Fraction of Separability

Since $frac_k$ gives the total variation given by the first k linear discriminants, it does not directly convey any information about how much variation is added by each individual discriminant. To provide information regarding the individual discriminant, $dfrac_k$ gives the amount of variation added by each discriminant (i.e., the difference between $frac_k$ values) and is computed as

$$dfrac_k = frac_k - frac_{k-1} \tag{6.7}$$

.

Mean Absolute Correlation

The mean absolute correlation [10, 65] is the arithmetic mean over all classes of the absolute values of the correlation coefficients of all pairs of attributes within each class. That is to say, that if you

take all pairs of numeric attributes i and j and perform linear regression between them and measure the correlation as ρ_{ij}, mean absolute correlation $\bar{\rho}$ is

$$|\bar{\rho}| = \frac{2}{J^2 - J} \sum_{i<j} |\rho_{ij}| \tag{6.8}$$

, where J is the number of attributes.

Skew

The skew of a dataset is similar to variance and standard deviation, but uses the third power of the differences between each value and the mean. In terms of moments of central tendency, skew is the third moment. This is often referred to as measuring how much the set deviates from a normal distribution or how asymmetric the distribution is [104]. Skew is computed as

$$skew = \frac{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^3}{\sigma^3} \tag{6.9}$$

, where $\bar{x}$ is the arithmetic mean of the set as shown in Equation 6.1 and σ is the standard deviation of the set as shown in Equation 6.6.

Since skew can be negative, the absolute value of skew for each attribute was used. The absolute values were then averaged the same way as all other univariate statistics.

Kurtosis

The kurtosis of a dataset is an extension of skew, and uses the fourth power of the differences from the mean. In terms of moments of central tendency, kurtosis is related to the fourth moment. This is often referred to as measuring the peakedness of the distribution, however Westfall [105] gives a more detailed treatment of what it really means. Kurtosis instead can be thought of as the heaviness or lightness of the tails of the distribution [104]. Kurtosis is computed as

$$kurtosis = \frac{\frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})^4}{\sigma^4} \tag{6.10}$$

where $\bar{x}$ is the arithmetic mean of the set, given in Equation 6.1, and σ is the standard deviation of the set, given in Equation 6.6.

Normal Cumulative Distribution Test

The normal cumulative distribution test ($normP$) first attempts to fit the values of a numeric attribute to a normal distribution. The resulting distribution is then tested against the original values. A p-value for rejecting the hypothesis that the data comes from the fit distribution is then computed.

Chi-square Test

The chi-square test χ^2 determines how closely a set of numbers fits a normal distribution. The value computed for each class and attribute pair is the confidence level at which the hypothesis that the values are normally distributed can be rejected.

Standard Deviation Ratio

The *sdRatio* [10, 65] provides a measure of similarity between the covariance matrices of the attributes across all of the classes. An implementation in MATLAB of *sdRatio* based on the descriptions provided by Kalousis [65] and King *et al.* is provided in the electronic appendix. Because computing *sdRatio* involves finding a determinant of a potentially very large matrix, this measure is not reported for datasets that have over 10,000 numeric attributes.

Z-score

The z-score (*zScore*) of a value is the number of standard deviations above or below the mean that a value is and is computed as

$$zScore = \frac{x_i - \bar{x}}{\sigma} \tag{6.11}$$

, where $\bar{x}$ is the arithmetic mean and σ is the standard deviation of a numeric attribute's values. For each attribute the maximum z-score is used when computing the averages over all classes and attributes.

Concentration Coefficient

The concentration coefficient [65] gives a measure of how well nominal attributes predict the value of other nominal attributes. Because the class attribute is also a nominal attribute, this measure is given both with (*concCoeffC*) and without (*concCoeff*) the class being included.

ANOVA p-Values

The ANOVA p-values ($pVal$) measure performs ANOVA on all pairings of numeric to nominal attributes for each class. The p-values for each pairing is then averaged to estimate the strength of the relationships between the nominal and numeric attibutes. As with concentration coefficient, because the class attribute is also a nominal attribute, this measure is given both with ($pValC$) and without ($pVal$) the class being included.

6.1.4 Information Theoretic Measures

Another category of measure is based on information theory. These measures focus on the entropy of the dataset.

$$Entropy = -\sum_{i=1}^{N} p_i \lg p_i \tag{6.12}$$

, where p_i is the probability of event p_i occurring. Note that since p_i is between zero and one, $\lg p_i$ is always non-positive, so the final result is always non-negative.

Class Entropy

The class entropy of a dataset is the total entropy over all classes.

$$H(C) = -\sum_{i=1}^{K} P(C = C_i) \lg P(C = C_i) \tag{6.13}$$

, where K is the number of classes and $P(C = C_i)$ is the probability that a sample belongs to class C_i.

Mean Attribute Entropy

The mean attribute entropy is the arithmetic mean of the entropies of each nominal attribute.

$$\bar{H}(X) = -\frac{1}{J} \sum_{j=1}^{J} \sum_{i=1}^{K} P(X = X_{ij}) \lg P(X = X_{ij}) \tag{6.14}$$

, where X_{ij} is the i^{th} sample of the j^{th} class and K is the number of classes.

Mean Joint Entropy for Attributes and Class

The joint entropy for attributes and classes ($H(X_j, C)$) is similar to class entropy and attribute entropy, but uses the joint probability of the class and attribute j having a combination of values $P(X = X_{ij}$ and $C = C_i)$. That is

$$H(X_j, C) = -\sum_{i=1}^{K} p_{ij} \lg p_{ij} \tag{6.15}$$

where p_{ij} is the joint probability $P(X = X_{ij}$ and $C = C_i)$. The mean joint entropy is then the arithmetic mean of the joint entropies for all attributes,

$$\bar{H}(X, C) = -\frac{1}{J} \sum_{j=1}^{J} H(X_j, C) \tag{6.16}$$

.

Mean Mutual Information for Attributes and Class

The mean mutual information [46, 65] ($\bar{M}(X, C)$) is

$$M(X, C) = \sum_{i=1}^{K} \sum_{j=1}^{K} p_{ij} \lg \left(\frac{p_{ij}}{\pi_i q_j} \right) \tag{6.17}$$

, where p_{ij} is the joint probability $P(X = X_{ij}$ and $C = C_i)$, π_i is the marginal probability $P(C = C_i)$, and q_j is the marginal probability $P(X = X_j)$. The mean mutual information ($\bar{M}(X, C)$) is the arithmetic mean of mutual information over all nominal attributes.

Equivalent Number of Attributes

Each attribute hopefully provides some information related to the class of each sample. The equivilant number of variables [46] (env) is computed by taking the ratio of the total entropy of the class, and dividing it by the average information of the attributes,

$$env = \frac{H(C)}{\bar{M}(X, C)} \tag{6.18}$$

, which gives an estimate of the number of attributes (variables) that should be needed to classify the samples.

Noise to Signal Ratio

The noise to signal ratio [46] (nsr) gives a measure of how much useless information (noise) there is in the dataset, and it computed as

$$nsr = \frac{\bar{H}X - \bar{M}(X, C)}{\bar{M}(X, C)} \tag{6.19}$$

.

6.1.5 Dataset Measurements

The characteristics, as described in Section 6.1.2 through 6.1.4, for the IntCensor dataset are given in the Table 6.1 through Table 6.3. A full listing of all dataset measures is given in Appendix B.

Characteristic	Value
$cancorr_1$	0.090
$cancorr_2$	0.060
$cancorr_3$	0.021
$cancorr_4$	NaN
$H(C)$	1.726
$dfrac_2$	0.295
$dfrac_3$	0.037
$dfrac_4$	0.000
$dimensionality$	0.007
env	116.698
$frac_1$	0.668
$frac_2$	0.963
$frac_3$	1
$frac_4$	1
nsr	70.969
J	74
N	10108
$perc_{nin}$	66.2%
$perc_{cont}$	6.8%
$perc_{ord}$	27.0%
$perc_{miss}$	0.0%
$perc_{neg}$	47.6%
$perc_{pos}$	52.4%
$sdRatio$	1.000
num_{bin}	49
C	4
num_{cont}	5
num_{ord}	20
num_{miss}	0
num_{pos}	4816
num_{neg}	5292

Table 6.1: Characteristics of the IntCensor dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.030	0.321	0.159	0.152	0.084
skew	0.056	2.049	0.472	0.875	0.707
arithMean	-0.159	35.169	3.194	8.470	13.301
χ^2	0	0.000	0.000	0.000	0.000
concCoeff	0	1	0.001	0.023	0.121
concCoeffC	0	0	0	0	0
$H(X)$	0.044	5.715	0.701	1.065	1.087
$H(X, C)$	1.043	6.706	1.693	2.048	1.083
geoMean	0	29.868	0	5.877	12.065
harmMean	-305.540	111.486	0	-10.475	76.257
IQ-range	0.250	22	2.625	5.638	7.809
kurtosis	2.610	7.185	3.389	3.936	1.375
mad	0.354	13.036	1.533	3.498	4.385
median	0	34	3	7.950	12.668
$\bar{M}(X, C)$	0.000	0.998	0.000	0.015	0.116
normP	1	1	1	1	0
pValC	0	1	0.000	0.090	0.223
pVal	0.000	0.004	0.000	0.001	0.002
$perc_{90}$	0.500	56	5	14.050	20.499
stdDev	0.652	16.462	1.936	4.411	5.431
trimMean	-0.039	34.982	3.034	8.305	13.165
unique	3	129	8	20.600	36.535
unique2			2	7.391	21.100
variance	0.425	271.005	3.757	47.474	90.697
zScore	1.536	3.401	2.395	2.407	0.554

Table 6.2: Summary characteristics of the IntCensor dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	35.00	27.50	35.00	2.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skew	20.00	20.00	20.00	0.00	0.00	5.00	5.00	10.00	5.00	15.00	0.00
concCoeff	96.86	1.35	0.20	0.04	0.08	0.02	0.02	0.00	0.00	1.43	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	41.89	33.78	2.70	12.16	4.05	2.70	0.00	0.00	0.00	2.70	0.00
$H(X, C)$	41.89	35.14	1.35	12.16	4.05	2.70	0.00	0.00	0.00	2.70	0.00
kurtosis	30.00	30.00	10.00	0.00	10.00	0.00	5.00	5.00	5.00	5.00	0.00
$\bar{M}(X, C)$	98.65	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.35	0.00
pValC	83.19	3.77	2.03	0.87	2.03	1.45	1.45	1.16	1.74	2.32	0.00
pVal	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table 6.3: Histogram based characteristics of the IntCensor dataset.

6.2 Algorithm Selection Dataset

Building a model to select algorithms that will perform well on a dataset requires first building an additional dataset that can be used to train the model. The classes for this dataset are the algorithms. Each sample is labeled with the algorithm that produced the best model quality under certain constraints. The attributes of this dataset are the properties of the datasets as described in Section 6.1, along with the machine configurations and a time limit.

6.2.1 Algorithm Selection Attributes

The attributes for the algorithm selection dataset are primarily made up of the measured characteristics described in Section 6.1. Any characteristic that is not computable, or results in Not a Number (NaN) or Infinity (Inf), are treated as missing values. In addition to the dataset characteristics, two additional attributes are added. The first is the number of cores. The second is a time limit (in milliseconds) that the algorithm would be allowed to run.

The levels for the time limit attribute of the dataset are constructed by sampling the range of times at an interval of ∼10% the range being sampled. The relationship between sampling interval and the value being sampled is given by Equation 6.20.

$$\text{interval} = 10^{\lfloor \log \text{value} \rfloor - 1} \tag{6.20}$$

For consistent sampling within each region, the start point is always a multiple of the interval for the lower end of the range of times being sampled. Additionally, to represent the final model generated if no time limit were given, a time limit of 10^9ms (∼11.5 days) is also included.

6.2.2 Algorithm Selection Classes

The classes for the algorithm selection dataset are, not surprisingly, the individual algorithms. In order to label each sample, a means of determining the "best" algorithm is needed. To pick the algorithm to label a sample with, temporal learning curves (TLCs) as described in Section 4.2.4 are used. Since each sample represents a potential stopping time, for each time limit considered, all of the TLCs are compared based on the confidence intervals for their combination scores. For example, in Figure 6.1, up to about 10 seconds, tree induction would be the best algorithm. After 10 seconds, sfo's and boosting's confidence intervals overlap the tree confidence intervals. Around 100 or 200 seconds boosting model scores improve to a point where the confidence intervals no longer overlap those of any of the other algorithms, and boosting becomes the best algorithm.

For each time limit and number of cores considered, there are potentially many overlapping confidence intervals as is the case between 10 seconds and 100 seconds for Figure 6.1. There are several possible ways to handle the overlaps. The first is to create a sample for each overlapping algorithm, and would be to create a label for each combination of algorithms, and finally a single one could be chosen based on some reasonable criteria. In this case it was decided to use the algorithm that had the highest lower bound. For each sampled time, all TLCs over all hyper-parameters were considered when picking the best scores. This means that from one sample to the next, even if the same algorithm is selected, it may have been from a different set of hyper-parameters.

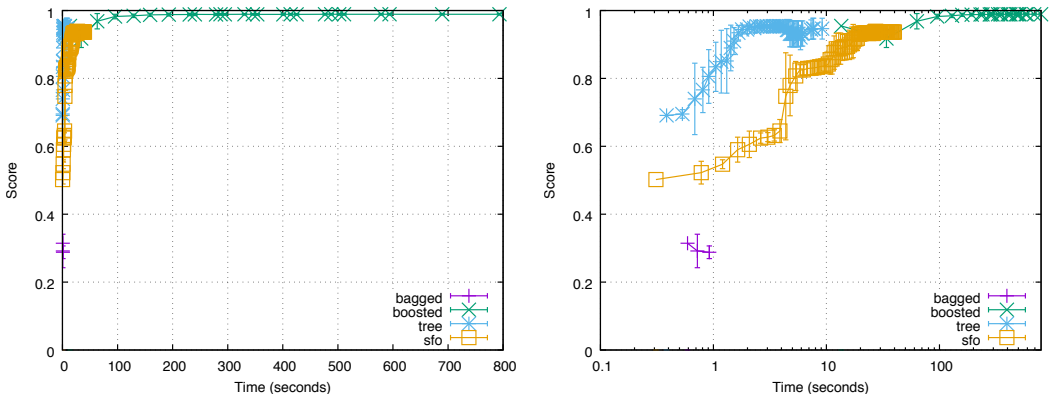

Figure 6.1: Learning curves for MUSHROOMS (binorm) on 32 cores with linear (left) and logrithmic (right) time axes. Error bars are 95% confidence intervals.

6.2.3 Conversion to Binary Classification

Since the algorithms being classified perform probability estimation for binary classification, the classes in the original datasets can be simplified to a positive class and a negative class. This simplification can affect the dataset characteristics used to build the model. To address this issue, two datasets were created. One with the characteristics of the datasets including all of the original classes. The other dataset uses characteristics as measured with only two classes.

6.2.4 Building a Test Dataset

To test the algorithm prediction models several additional datasets were chosen. Each of these datasets were then put through the same hill climbing process as described in Chapter 4 using a single compute core. To give the hill-climbing a head start, and ensure that some set of hyper-parameters were tried on all numbers of cores, the set of random restarts was seeded with some of the hyper-parameters that had scored highly on the datasets in Chapter 4. To avoid using hyper-parameters that are overly similar to others, all hyper-parameters were rounded to the nearest multiple of five. After the rounding, duplicates were removed. Initially the top five hyper-parameters from each algorithm were going to be used, but only the very highest scoring hyper-parameters ended up being used, due to an error when creating the initial learning task files.

After the hill climbing was completed on a single core, it was again repeated on different numbers of cores to collect run-time data. The numbers of cores used were again 2, 4, 8, and 16 as in Chapter 5. However, unlike in Chapter 5, all training on multiple cores was done from initially empty models. The hill climbing on multiple cores was also seeded with the top five sets of hyper-parameters for each algorithm.

Using the data from the hill climbing, a dataset was built just as the training set for algorithm selection had been constructed.

6.3 Runtime System

For measuring of dataset characteristics, Matlab R2016b (9.1.0.441655) [106] running on a 2.3GHz Intel Core i7 with 16GB of RAM was used. For training the models for the meta-learning problem, Weka 3.8.1 [39] running on the same Core i7 system was used.

The training of models for the test dataset for the meta-learning problem were trained using the the same configuration as in Section 5.2.

6.4 Algorithm Selection Results

6.4.1 Additional Model Quality Metrics

The model quality metrics used by Weka are different than the ones used in Chapter 3 through Chapter 5. As in the previous chapters, Accuracy, Error, Precision, Recall, and AUC are all reported. However, F-measure (described as F-score in Equation 1.39), and two additional metrics, MCC and area under the precision-recall curve, are also reported.

Matthews Correlation Coefficient

The Matthews Correlation Coefficient (MCC) [107, 108] is intended to give a quality measure that is useful when the class sizes differ greatly. Unlike AUC which also handles class size imbalances well, MCC is computed strictly from the confusion matrix,

$$MCC = \frac{TP \cdot TN - FN \cdot FP}{\sqrt{(TP + FN) \cdot (TP + FP) \cdot (TN + FP) \cdot (TN + FN)}} \tag{6.21}$$

, and does not require finding the area under a curve. The values given by MCC range from -1 to 1, where -1 is very bad, and 1 is very good.

Area Under the Precision-Recall Curve

The area under the precision-recall curve [109] (AUPRC) is similar to AUC, however instead of using specificity and sensitivity, precision and recall are used to construct the curve.

6.4.2 Model Training Results

A series of 18 different models were trained on the algorithm selection datasets (regular and binary class only) using Weka and default settings for most learners. For the clustering algorithms, an additional flags to label each cluster and to use 4 clusters were given. The scores reported for those models after training are shown in Table 6.4 and Table 6.5. The last column of the tables (Avg.) is an average of all of the measures. For measures that do not have the range 0 to 1, or for which lower is better, the values were mapped to the range 0 to 1, with one being the better end of the scale. Thus, error was subtracted from zero before being averaged, and MCC was divided by two and had 0.5 added to it.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.475	0.525	0.226	0.475	0.306	0.000	0.754	0.605	0.483
Decision Stump	0.620	0.79	0.603	0.620	0.574	0.408	0.727	0.533	0.646
EM	0.591	0.776	0.581	0.591	0.524	0.367	0.768	0.588	0.638
Farthest First	0.491	0.661	0.468	0.491	0.433	0.172	0.679	0.502	0.539
Filtered Clusterer	0.512	0.617	0.468	0.512	0.426	0.171	0.689	0.509	0.540
Hierarchical Clusterer	0.495	0.551	0.401	0.495	0.364	0.099	0.529	0.373	0.470
Hoeffding Tree	0.626	0.845	0.660	0.626	0.610	0.472	0.763	0.586	0.681
IBk	0.820	0.917	0.821	0.820	0.817	0.733	0.956	0.900	0.865
J48	0.936	0.97	0.936	0.936	0.936	0.906	0.978	0.947	0.949
LMT	0.833	0.927	0.842	0.833	0.835	0.752	0.914	0.845	0.863
LWL	0.625	0.825	0.629	0.625	0.591	0.448	0.829	0.676	0.691
Logistic	0.706	0.832	0.681	0.706	0.689	0.535	0.892	0.789	0.758
Make Density Based Clusterer	0.512	0.617	0.468	0.512	0.426	0.171	0.689	0.509	0.540
REPTree	0.921	0.963	0.921	0.921	0.921	0.885	0.981	0.952	0.940
Random Forest	0.947	0.975	0.947	0.947	0.947	0.922	0.994	0.986	0.963
Random Tree	0.937	0.971	0.938	0.937	0.938	0.908	0.956	0.901	0.942
SimpleKMeans	0.571	0.733	0.543	0.571	0.512	0.314	0.766	0.596	0.619
Simple Logistic	0.696	0.83	0.672	0.696	0.679	0.521	0.887	0.783	0.750

Table 6.4: Model scores for training set.

While some of the models scored very poorly, some of the algorithms scored very high. In particular, the tree based methods J48, REPTree, Random Forest, and Random Tree all managed to get average scores over 0.9. This would suggest that one should expect the models produced to do well in predicting which algorithm will perform well based on their characteristics.

On the other hand, the clustering methods tended to do very poorly, and commonly had average scores below 0.5. One possible explanation for the bad clustering is that for all cases, only four clusters were used. Had differing numbers of clusters been tried, it is possible clustering may have produced better predictions.

Surprisingly, the scores in Table 6.13 are nearly identical to those in Table 6.4. This would indicate that the models are not very sensitive to the dataset characteristics that are affected by the number of classes.

To look more closely at the results of these models, the confusion matrices can be examined. The confusion matrices for highest scoring (J48) and lowest scoring (EM) models are shown in Table 6.6 and Table C.3. A complete listing of confusion matrices is given in Appendix C.

In the high scoring model (Table 6.6), the largest numbers are aligned along the diagonal, as expected. One interesting thing to note is that along the diagonal, class 'a', bagging, has the smallest number of samples falling into that bin. This is most likely due to the issues regarding bagging on small datasets and with the binorm versions of the datasets as discussed in Section 4.2.2.

In the low scoring model (Table 6.7), the model apparently always guesses 'boosted'. If a model had to only give one answer every time, 'boosted' is certainly a better option than 'bagging', since more samples have that label. This does not however make for a very useful model.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.475	0.525	0.226	0.475	0.306	0.000	0.744	0.609	0.483
Decision Stump	0.620	0.79	0.603	0.620	0.574	0.408	0.727	0.533	0.646
EM	0.610	0.774	0.578	0.610	0.551	0.385	0.781	0.604	0.650
Farthest First	0.505	0.678	0.475	0.505	0.444	0.198	0.675	0.488	0.546
Filtered Clusterer	0.485	0.555	0.464	0.485	0.355	0.087	0.670	0.495	0.507
Hierarchical Clusterer	0.495	0.551	0.401	0.495	0.364	0.099	0.529	0.373	0.470
Hoeffding Tree	0.631	0.864	0.693	0.631	0.624	0.500	0.773	0.612	0.697
IBk	0.815	0.915	0.816	0.815	0.811	0.726	0.954	0.896	0.861
J48	0.936	0.97	0.935	0.936	0.935	0.905	0.978	0.947	0.949
LMT	0.827	0.909	0.824	0.827	0.824	0.735	0.938	0.869	0.861
LWL	0.629	0.836	0.639	0.629	0.601	0.462	0.830	0.672	0.696
Logistic	0.706	0.832	0.681	0.706	0.689	0.535	0.892	0.789	0.758
Make Density Based Clusterer	0.485	0.555	0.464	0.485	0.355	0.087	0.669	0.494	0.506
REPTree	0.921	0.963	0.921	0.921	0.921	0.885	0.981	0.952	0.940
Random Forest	0.948	0.975	0.948	0.948	0.947	0.922	0.994	0.986	0.963
Random Tree	0.940	0.971	0.940	0.940	0.940	0.912	0.958	0.906	0.944
SimpleKMeans	0.545	0.75	0.513	0.545	0.515	0.293	0.745	0.572	0.604
Simple Logistic	0.697	0.829	0.678	0.697	0.681	0.523	0.889	0.785	0.752

Table 6.5: Model scores for training set with binary classes.

a	b	c	d	
294	10	38	24	a = bagged
12	3785	39	109	b = boosted
26	27	1915	41	c = sfo
17	128	61	1775	d = tree

Table 6.6: Confusion matrix for the J48 model applied to the training dataset.

a	b	c	d	
0	366	0	0	a = bagged
0	3945	0	0	b = boosted
0	2009	0	0	c = sfo
0	1981	0	0	d = tree

Table 6.7: Confusion matrix for the EM model applied to the training dataset.

6.5 Testing Dataset Results

To test the actual usefulness of the model produced, a set of 8 datasets where run through a similar hyper-parameter tuning and parallel run time process as described in Chapters 4 and 5. In this case the hyper-parameter finding was done using a single compute core, and the parallel runs were all started from empty models.

Dataset	TP	TN	FP	FN	AUC	Acc.	Prec.	Recall	Spec.	RMSE	RMSA	Combo
Abalone	0	702	0	133	0.550	0.840	-1.000	0.000	1.000	0.365	0.769	0.399
Abalone (binorm)	0	702	0	133	0.500	0.840	-1.000	0.000	1.000	0.399	0.916	0.408
Botswana	0	594	0	64	0.550	0.903	-1.000	0.000	1.000	0.309	0.947	0.442
Botswana (binorm)	0	594	0	64	0.550	0.903	-1.000	0.000	1.000	0.309	0.947	0.442
German	0	138	0	59	0.698	0.699	-0.200	0.011	0.994	0.433	0.633	0.486
German (binorm)	0	139	0	59	0.550	0.700	-1.000	0.000	1.000	0.544	0.832	0.363
Heart	14	21	2	12	0.796	0.706	0.881	0.528	0.900	0.423	0.651	0.720
Heart (binorm)	0	24	0	27	0.571	0.475	-1.000	0.000	1.000	0.717	0.681	0.287
Indian pines	0	1572	0	476	0.500	0.768	-1.000	0.000	1.000	0.473	0.867	0.380
Indian pines (binorm)	0	1572	0	476	0.500	0.768	-1.000	0.000	1.000	0.480	0.874	0.380
KSC	184	843	10	0	0.700	0.990	0.949	0.999	0.988	0.096	0.985	0.931
KSC (binorm)	184	840	13	0	0.700	0.987	0.934	0.999	0.985	0.109	0.982	0.925
PaviaU	0	4826	0	3721	0.500	0.565	-1.000	0.000	1.000	0.474	0.692	0.326
PaviaU (binorm)	0	4826	0	3721	0.500	0.565	-1.000	0.000	1.000	0.481	0.684	0.324
SalinasA	0	752	0	310	0.600	0.708	-1.000	0.000	1.000	0.523	0.841	0.375
SalinasA (binorm)	0	752	0	310	0.600	0.708	-1.000	0.000	1.000	0.523	0.841	0.375

Table 6.8: Top model scores for bagged models on test datasets.

Dataset	TP	TN	FP	FN	AUC	Acc.	Prec.	Recall	Spec.	RMSE	RMSA	Combo
Abalone	0	704	0	132	0.500	0.841	-1.000	0.000	1.000	0.391	0.898	0.407
Abalone (binorm)	0	702	0	133	0.500	0.840	-1.000	0.000	1.000	0.392	0.898	0.407
Botswana	0	594	0	64	0.600	0.903	-1.000	0.000	1.000	0.311	0.945	0.448
Botswana (binorm)	0	594	0	64	0.600	0.903	-1.000	0.000	1.000	0.311	0.950	0.449
German	21	125	14	37	0.717	0.739	0.620	0.369	0.900	0.426	0.701	0.660
German (binorm)	0	139	0	59	0.500	0.700	-1.000	0.000	1.000	0.540	0.827	0.355
Heart	24	17	5	2	0.782	0.840	0.817	0.909	0.763	0.368	0.762	0.787
Heart (binorm)	17	18	5	10	0.500	0.695	0.752	0.628	0.768	0.488	0.730	0.655
Indian pines	0	1572	0	476	0.500	0.768	-1.000	0.000	1.000	0.481	0.875	0.380
Indian pines (binorm)	0	1572	0	476	0.500	0.768	-1.000	0.000	1.000	0.481	0.875	0.380
KSC	77	846	6	107	0.650	0.889	-0.231	0.400	0.992	0.321	0.919	0.614
KSC (binorm)	57	849	4	125	0.708	0.875	-0.359	0.333	0.995	0.340	0.915	0.590
PaviaU	0	4826	0	3721	0.500	0.565	-1.000	0.000	1.000	0.656	0.747	0.308
PaviaU (binorm)	0	4826	0	3721	0.500	0.565	-1.000	0.000	1.000	0.656	0.747	0.308
SalinasA	0	755	0	307	0.583	0.710	-1.000	0.000	1.000	0.527	0.842	0.373
SalinasA (binorm)	0	752	0	310	0.550	0.708	-1.000	0.000	1.000	0.471	0.843	0.376

Table 6.9: Top model scores for boosted models on test datasets.

Dataset	TP	TN	FP	FN	AUC	Acc.	Prec.	Recall	Spec.	RMSE	RMSA	Combo
Abalone	0	702	0	133	0.640	0.840	-1.000	0.000	1.000	0.381	0.833	0.419
Abalone (binorm)	0	701	0	133	0.500	0.839	-0.800	0.000	0.999	0.363	0.780	0.422
Botswana	9	585	9	54	0.600	0.904	-0.696	0.166	0.985	0.233	0.914	0.520
Botswana (binorm)	9	585	9	54	0.600	0.904	-0.697	0.166	0.984	0.233	0.914	0.520
German	10	124	15	49	0.547	0.675	0.528	0.180	0.897	0.466	0.643	0.572
German (binorm)	3	137	2	56	0.667	0.704	-0.686	0.052	0.982	0.455	0.627	0.413
Heart	22	19	4	4	0.698	0.821	0.836	0.823	0.819	0.374	0.785	0.772
Heart (binorm)	19	17	6	7	0.533	0.726	0.747	0.703	0.728	0.487	0.759	0.673
Indian pines	351	1243	329	124	0.500	0.778	0.516	0.739	0.790	0.365	0.782	0.677
Indian pines (binorm)	351	1243	329	124	0.500	0.778	0.516	0.738	0.791	0.365	0.782	0.677
KSC	184	842	10	0	0.767	0.989	0.947	0.997	0.988	0.098	0.984	0.939
KSC (binorm)	184	842	10	0	0.767	0.989	0.947	0.995	0.988	0.100	0.983	0.938
PaviaU	3583	3431	1394	137	0.600	0.821	0.720	0.963	0.711	0.365	0.777	0.747
PaviaU (binorm)	3584	3426	1400	137	0.600	0.820	0.719	0.963	0.710	0.365	0.776	0.746
SalinasA	309	748	3	0	0.800	0.996	0.989	0.998	0.995	0.056	0.992	0.959
SalinasA (binorm)	309	748	4	1	0.800	0.995	0.987	0.997	0.995	0.060	0.991	0.958

Table 6.10: Top model scores for tree models on test datasets.

Dataset	TP	TN	FP	FN	AUC	Acc.	Prec.	Recall	Spec.	RMSE	RMSA	Combo
Abalone (binorm)	0	702	0	133	0.656	0.840	-1.000	0.000	1.000	0.365	0.771	0.415
Botswana (binorm)	0	594	0	64	0.917	0.903	-1.000	0.000	1.000	0.296	0.872	0.485
German (binorm)	1	139	0	58	0.556	0.703	-0.667	0.019	0.995	0.457	0.629	0.397
Heart (binorm)	19	14	9	7	0.807	0.669	0.729	0.735	0.634	0.449	0.631	0.679
Indian pines (binorm)	0	1572	0	476	0.869	0.768	-1.000	0.000	1.000	0.396	0.706	0.421
KSC (binorm)	0	853	0	185	1.000	0.822	-1.000	0.000	1.000	0.331	0.883	0.482
PaviaU (binorm)	559	4728	97	3161	0.875	0.619	0.714	0.151	0.980	0.469	0.539	0.630
SalinasA (binorm)	0	752	0	310	0.997	0.708	-1.000	0.000	1.000	0.445	0.655	0.416

Table 6.11: Top model scores for sfo models on test datasets.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.135	0.865	0.018	0.135	0.032	0.000	0.460	0.289	0.304
Decision Stump	0.252	0.871	0.075	0.252	0.115	0.088	0.589	0.358	0.382
EM	0.096	0.855	0.026	0.096	0.033	-0.054	0.497	0.324	0.300
Farthest First	0.396	0.756	0.294	0.396	0.319	0.133	0.569	0.353	0.456
Filtered Clusterer	0.192	0.857	0.051	0.192	0.080	0.039	0.493	0.315	0.337
Hierarchical Clusterer	0.135	0.865	0.018	0.135	0.032	0.000	0.500	0.317	0.313
Hoeffding Tree	0.264	0.732	0.275	0.264	0.224	-0.024	0.479	0.314	0.380
IBk	0.281	0.717	0.264	0.281	0.261	-0.003	0.473	0.351	0.391
J48	0.331	0.743	0.343	0.331	0.296	0.065	0.538	0.345	0.432
KStar	0.242	0.758	0.059	0.242	0.094	0.000	0.500	0.317	0.339
LMT	0.261	0.785	0.319	0.261	0.246	0.037	0.560	0.356	0.413
LWL	0.250	0.861	0.102	0.250	0.131	0.103	0.565	0.373	0.385
Logistic	0.159	0.841	0.025	0.159	0.044	0.000	0.500	0.317	0.318
Make Density Based Clusterer	0.252	0.87	0.074	0.252	0.114	0.088	0.583	0.355	0.381
REPTree	0.246	0.724	0.286	0.246	0.215	-0.044	0.506	0.345	0.381
Random Forest	0.241	0.719	0.227	0.241	0.205	-0.048	0.579	0.383	0.384
Random Tree	0.242	0.814	0.377	0.242	0.221	0.060	0.528	0.337	0.411
SimpleKMeans	0.158	0.867	0.074	0.158	0.070	0.033	0.481	0.311	0.329
Simple Logistic	0.495	0.624	0.364	0.495	0.395	0.155	0.579	0.384	0.489

Table 6.12: Model scores for test set.

6.6 Model Testing Results

Based on the scores of the models in Section 6.4.2, it would be expected that the models might do well in predicting algorithms for additional datasets. In Table 6.12 and Table 6.13 are the scores for the 16 models when applied to the data collected using datasets not considered in the creation of the training data.

Here the results are far less encouraging. In all cases the average model score is less than 0.5 with most around 0.3. However, there are some glimmers of hope in this data. Specifically, looking at the AUC column, the tree based methods that had the highest scores during training managed to get above 0.5, which indicates they are at least better than random guessing. Thus it is possible, that had many more datasets been used in building the training data, the models may have performed better. It is also likely that some of the errors are due to situations where multiple algorithms may do well, but in building the training dataset a single algorithm had to be selected to label each sample, thus the prediction cannot indicate that the tie occurred.

Looking at the confusion matrices for the testing dataset, many of the models committed the same blunder as EM on the training dataset. For instance, Decision Stump, as shown in Table 6.14, only ever outputs classes 'b' or 'c', especially when 'd' would have been the correct value. When it comes to deciding between tree based models of logistic regression, things look slightly more promising. Only seven sfo samples were incorrectly assigned the boosting label.

In the case of J48, which had performed very well on the training data, it also seemed to get stuck

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.135	0.865	0.018	0.135	0.032	0.000	0.494	0.317	0.312
Decision Stump	0.252	0.871	0.075	0.252	0.115	0.088	0.589	0.358	0.382
EM	0.139	0.857	0.041	0.139	0.061	-0.003	0.480	0.306	0.315
Farthest First	0.178	0.853	0.044	0.178	0.069	0.026	0.494	0.317	0.331
Filtered Clusterer	0.135	0.865	0.018	0.135	0.032	0.000	0.431	0.296	0.301
Hierarchical Clusterer	0.135	0.865	0.018	0.135	0.032	0.000	0.500	0.317	0.313
Hoeffding Tree	0.133	0.79	0.086	0.133	0.060	-0.122	0.415	0.281	0.292
IBk	0.272	0.783	0.250	0.272	0.211	0.027	0.496	0.324	0.390
J48	0.324	0.759	0.357	0.324	0.291	0.064	0.535	0.342	0.433
LMT	0.269	0.72	0.322	0.269	0.269	-0.011	0.524	0.324	0.399
LWL	0.250	0.861	0.102	0.250	0.131	0.103	0.575	0.373	0.387
Logistic	0.275	0.712	0.306	0.275	0.289	0.003	0.471	0.332	0.395
Make Density Based Clusterer	0.135	0.865	0.018	0.135	0.032	0.000	0.522	0.325	0.317
REPTree	0.235	0.808	0.292	0.235	0.189	0.017	0.499	0.358	0.391
Random Forest	0.240	0.736	0.257	0.240	0.216	-0.021	0.578	0.363	0.390
Random Tree	0.316	0.733	0.372	0.316	0.314	0.050	0.520	0.327	0.428
SimpleKMeans	0.152	0.865	0.053	0.152	0.061	0.018	0.447	0.297	0.317
Simple Logistic	0.178	0.796	0.220	0.178	0.145	-0.052	0.512	0.324	0.353

Table 6.13: Model scores for test set with binary classes.

a	b	c	d	
0	212	125	0	a = bagged
0	136	52	0	b = boosted
0	7	214	0	c = sfo
0	319	326	0	d = tree

Table 6.14: Confusion matrix for the Decision Stump model applied to the test dataset.

outputting 'b' when other classes would have been more appropriate. It did however do a very good job avoiding 'a' (bagging) which is known to have some implementation issues. J48 also did quite well identifying when class 'd' (tree) was the correct choice with 310 correct, however somewhat to the detriment of class 'a' (bagging) with 185 mistakes.

a	b	c	d	
1	113	38	185	a = bagged
4	121	5	58	b = boosted
0	144	28	49	c = sfo
0	259	76	310	d = tree

Table 6.15: Confusion matrix for the J48 model applied to the test dataset.

6.6.1 Principal Component Analysis

An examination of the separability of the datasets based on the attributes of the dataset was also done. Specifically, principle component analysis [98] (PCA) was performed. Each of the attributes in the algorithm selection dataset was first normalized to have a zero mean and variance of one. To map the training and testing datasets into the same 2-dimensional space, the samples from both datasets where combined into one dataset before computing the coefficients matrix. The resulting coefficients matrix was then applied to both the training and testing datasets individually. Figure 6.2 shows the first two principle components of the attributes, with classes in the training and testing datasets labeled individually. The tight clustering of most of the points into a single big group indicates that the attributes provide insufficent detail about the datasets to easily separate them.

PCA was also performed on the dataset constructed from the characteristics measured after converting each original dataset to having binary classes. Figure 6.3 shows the results of PCA on the binary class based dataset. Again, most of the points fall into once big cluster.

These PCA results suggest that the dataset characteristics are insufficient for the learning task of selecting among these four algorithms.

6.7 Model Per Algorithm

Since it doesn't appear to be the case that building a model that can select an algorithm among all four tested. Is it instead possible to make a model for each algorithm that attempts to determine of that algorithm will work at least as well as the best algorithm tested.

To test this, a datasets for each algorithm was created similar to the one described in Section 6.2. However, the labels will be restricted to a just the algorithm for that dataset, or "other". The label is set to the algorithm's name when it is the best, or its confidence interval overlaps the best confidence interval. If there is another algorithm for which the confidence interval's lower bound is above the upper bound of the algorithm being checked, the label is "other". For each algorithm testing datasets were similarly constructed. The scores for the training datasets are given in Table 6.16 through Table 6.19. The scores for the test datasets are given in Table 6.20 though Table 6.23.

These results look far more promising. The original models trained on all of the datasets had average scores that were in the 0.4 range with the highest individual metric being less than 0.6. However, the

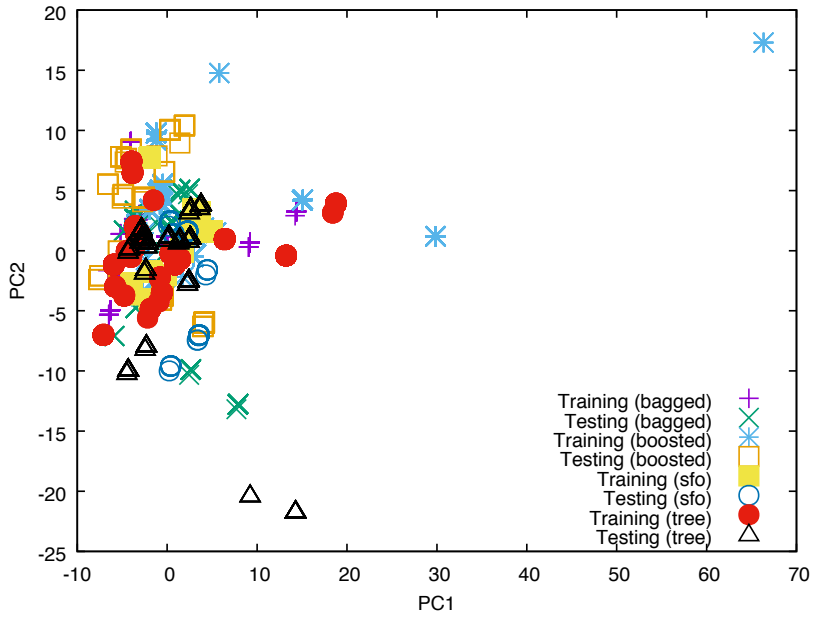

Figure 6.2: First two principle components for algorithm selection dataset. First and second principle components account for 35.1% and 32.0% of variation respectively.

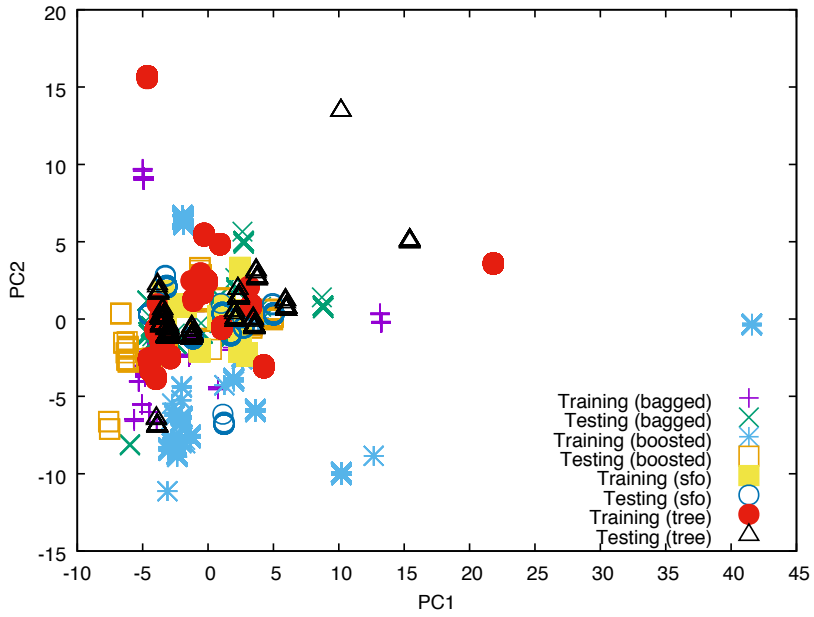

Figure 6.3: First two principle components for algorithm selection dataset(binary classes). First and second principle components account for 37.4% and 26.9% of variation respectively.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.881	0.119	0.776	0.881	0.825	0.000	0.716	0.869	0.696
Decision Stump	0.881	0.119	0.776	0.881	0.825	0.000	0.691	0.845	0.690
EM	0.881	0.119	0.776	0.881	0.825	0.000	0.700	0.852	0.692
Farthest First	0.881	0.119	0.776	0.881	0.825	0.000	0.589	0.818	0.674
Filtered Clusterer	0.881	0.119	0.776	0.881	0.825	0.000	0.582	0.820	0.673
Hierarchical Clusterer	0.884	0.296	0.857	0.884	0.860	0.281	0.584	0.818	0.728
Hoeffding Tree	0.880	0.121	0.810	0.880	0.825	0.015	0.578	0.817	0.677
IBk	0.915	0.58	0.908	0.915	0.910	0.558	0.924	0.952	0.860
J48	0.977	0.901	0.977	0.977	0.977	0.889	0.974	0.982	0.964
LMT	0.979	0.917	0.979	0.979	0.979	0.900	0.983	0.989	0.969
LWL	0.881	0.119	0.776	0.881	0.825	0.000	0.815	0.896	0.712
Logistic	0.883	0.291	0.856	0.883	0.859	0.273	0.854	0.907	0.771
Make Density Based Clusterer	0.881	0.119	0.776	0.881	0.825	0.000	0.582	0.820	0.673
REPTree	0.969	0.861	0.968	0.969	0.968	0.848	0.972	0.982	0.952
Random Forest	0.980	0.899	0.980	0.980	0.980	0.905	0.993	0.996	0.970
Random Tree	0.978	0.906	0.978	0.978	0.978	0.894	0.943	0.967	0.959
SimpleKMeans	0.881	0.119	0.776	0.881	0.825	0.000	0.681	0.850	0.689
Simple Logistic	0.882	0.251	0.850	0.882	0.852	0.233	0.806	0.895	0.754

Table 6.16: Model scores for training set (bagged).

models trained to predict individual algorithm's performance against the others did much better. The best results were for predicting when logistic regression will do well, most models had average scores in the 0.6 to 0.8 range. The highest AUC reported was 0.886.

In the case of bagging most of the averaged scores are in the 0.5 to 0.6 range. For boosted stumps and regular trees, the results were comparable to the models built on all four algorithms.

When the process of building a model for each algorithm is repeated using the dataset characteristics based on binary classes, the results are neither better not worse than the results using all classes. These results are shown in Table 6.28 through Table 6.31.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.558	0.442	0.312	0.558	0.400	0.000	0.733	0.726	0.529
Decision Stump	0.706	0.745	0.756	0.706	0.701	0.466	0.740	0.722	0.726
EM	0.699	0.688	0.698	0.699	0.698	0.388	0.750	0.726	0.707
Farthest First	0.594	0.521	0.595	0.594	0.548	0.147	0.589	0.594	0.576
Filtered Clusterer	0.628	0.639	0.640	0.628	0.629	0.266	0.673	0.658	0.641
Hierarchical Clusterer	0.575	0.47	0.618	0.575	0.460	0.106	0.520	0.526	0.537
Hoeffding Tree	0.606	0.507	0.694	0.606	0.513	0.217	0.578	0.584	0.587
IBk	0.865	0.864	0.865	0.865	0.865	0.727	0.951	0.950	0.886
J48	0.968	0.967	0.968	0.968	0.968	0.934	0.980	0.970	0.970
LMT	0.972	0.972	0.972	0.972	0.972	0.943	0.990	0.988	0.976
LWL	0.732	0.689	0.747	0.732	0.721	0.461	0.884	0.891	0.766
Logistic	0.737	0.727	0.736	0.737	0.736	0.465	0.832	0.839	0.760
Make Density Based Clusterer	0.628	0.639	0.640	0.628	0.629	0.266	0.673	0.658	0.641
REPTree	0.958	0.957	0.958	0.958	0.958	0.915	0.983	0.977	0.963
Random Forest	0.972	0.971	0.972	0.972	0.972	0.942	0.996	0.995	0.978
Random Tree	0.968	0.967	0.968	0.968	0.968	0.935	0.969	0.956	0.966
SimpleKMeans	0.666	0.664	0.669	0.666	0.667	0.328	0.745	0.733	0.684
Simple Logistic	0.730	0.728	0.732	0.730	0.730	0.455	0.822	0.828	0.753

Table 6.17: Model scores for training set (boosted).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.578	0.467	0.733	0.578	0.444	0.150	0.735	0.745	0.607
Decision Stump	0.678	0.745	0.814	0.678	0.656	0.495	0.703	0.697	0.715
EM	0.672	0.66	0.671	0.672	0.672	0.333	0.739	0.715	0.683
Farthest First	0.607	0.582	0.602	0.607	0.603	0.193	0.638	0.632	0.608
Filtered Clusterer	0.619	0.619	0.624	0.619	0.620	0.237	0.662	0.650	0.629
Hierarchical Clusterer	0.598	0.493	0.724	0.598	0.488	0.210	0.542	0.552	0.575
Hoeffding Tree	0.656	0.629	0.653	0.656	0.651	0.293	0.735	0.721	0.668
IBk	0.860	0.86	0.861	0.860	0.860	0.718	0.946	0.944	0.881
J48	0.971	0.971	0.971	0.971	0.971	0.942	0.986	0.981	0.974
LMT	0.973	0.973	0.973	0.973	0.973	0.945	0.991	0.988	0.977
LWL	0.678	0.745	0.814	0.678	0.656	0.495	0.822	0.822	0.745
Logistic	0.754	0.761	0.761	0.754	0.755	0.511	0.854	0.854	0.781
Make Density Based Clusterer	0.621	0.622	0.627	0.621	0.622	0.242	0.663	0.650	0.631
REPTree	0.963	0.962	0.963	0.963	0.963	0.925	0.989	0.987	0.969
Random Forest	0.974	0.972	0.974	0.974	0.974	0.948	0.997	0.997	0.980
Random Tree	0.970	0.968	0.970	0.970	0.970	0.939	0.971	0.959	0.968
SimpleKMeans	0.672	0.686	0.686	0.672	0.673	0.357	0.743	0.725	0.692
Simple Logistic	0.753	0.761	0.761	0.753	0.753	0.511	0.846	0.845	0.778

Table 6.18: Model scores for training set (tree).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.661	0.339	0.436	0.661	0.526	0.000	0.887	0.884	0.612
Decision Stump	0.881	0.939	0.912	0.881	0.884	0.779	0.905	0.880	0.896
EM	0.817	0.826	0.832	0.817	0.820	0.619	0.872	0.852	0.831
Farthest First	0.678	0.405	0.666	0.678	0.594	0.163	0.709	0.723	0.629
Filtered Clusterer	0.669	0.395	0.637	0.669	0.584	0.123	0.672	0.693	0.610
Hierarchical Clusterer	0.661	0.339	0.436	0.661	0.526	0.000	0.535	0.588	0.531
Hoeffding Tree	0.661	0.339	0.436	0.661	0.526	0.000	0.811	0.797	0.591
IBk	0.947	0.951	0.949	0.947	0.947	0.885	0.986	0.985	0.957
J48	0.976	0.974	0.976	0.976	0.976	0.946	0.990	0.986	0.978
LMT	0.977	0.975	0.977	0.977	0.977	0.950	0.994	0.993	0.981
LWL	0.881	0.939	0.912	0.881	0.884	0.779	0.946	0.936	0.909
Logistic	0.895	0.922	0.908	0.895	0.897	0.786	0.960	0.956	0.916
Make Density Based Clusterer	0.669	0.395	0.637	0.669	0.584	0.123	0.672	0.693	0.610
REPTree	0.971	0.968	0.971	0.971	0.971	0.936	0.992	0.990	0.975
Random Forest	0.980	0.974	0.980	0.980	0.980	0.955	0.997	0.996	0.983
Random Tree	0.977	0.97	0.976	0.977	0.976	0.948	0.973	0.965	0.973
SimpleKMeans	0.764	0.677	0.759	0.764	0.760	0.459	0.841	0.832	0.766
Simple Logistic	0.895	0.922	0.908	0.895	0.897	0.786	0.955	0.948	0.914

Table 6.19: Model scores for training set (sfo).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.654	0.346	0.428	0.654	0.517	0.000	0.437	0.515	0.506
Decision Stump	0.654	0.346	0.428	0.654	0.517	0.000	0.543	0.570	0.526
EM	0.654	0.346	0.428	0.654	0.517	0.000	0.482	0.540	0.515
Farthest First	0.654	0.346	0.428	0.654	0.517	0.000	0.536	0.566	0.525
Filtered Clusterer	0.654	0.346	0.428	0.654	0.517	0.000	0.523	0.558	0.522
Hierarchical Clusterer	0.654	0.346	0.428	0.654	0.517	0.000	0.500	0.548	0.518
Hoeffding Tree	0.654	0.346	0.428	0.654	0.517	0.000	0.537	0.566	0.525
IBk	0.646	0.38	0.582	0.646	0.556	0.050	0.538	0.573	0.556
J48	0.554	0.412	0.531	0.554	0.540	-0.036	0.358	0.498	0.491
KStar	0.346	0.654	0.120	0.346	0.178	0.000	0.500	0.548	0.399
LMT	0.530	0.539	0.579	0.530	0.541	0.065	0.552	0.585	0.549
LWL	0.654	0.346	0.428	0.654	0.517	0.000	0.539	0.579	0.527
Logistic	0.629	0.547	0.628	0.629	0.628	0.177	0.599	0.604	0.607
Make Density Based Clusterer	0.654	0.346	0.428	0.654	0.517	0.000	0.535	0.564	0.525
REPTree	0.656	0.351	0.706	0.656	0.524	0.057	0.477	0.547	0.556
Random Forest	0.654	0.346	0.428	0.654	0.517	0.000	0.686	0.692	0.560
Random Tree	0.671	0.383	0.731	0.671	0.561	0.169	0.527	0.566	0.587
SimpleKMeans	0.654	0.346	0.428	0.654	0.517	0.000	0.495	0.548	0.518
Simple Logistic	0.550	0.346	0.484	0.550	0.506	-0.128	0.504	0.556	0.491

Table 6.20: Model scores for test set (bagged).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.306	0.694	0.094	0.306	0.144	0.000	0.505	0.572	0.390
Decision Stump	0.720	0.394	0.721	0.720	0.645	0.224	0.557	0.610	0.622
EM	0.291	0.573	0.383	0.291	0.207	-0.200	0.432	0.555	0.392
Farthest First	0.610	0.662	0.692	0.610	0.625	0.251	0.640	0.659	0.640
Filtered Clusterer	0.571	0.437	0.578	0.571	0.574	0.008	0.504	0.577	0.539
Hierarchical Clusterer	0.306	0.694	0.094	0.306	0.144	0.000	0.500	0.575	0.390
Hoeffding Tree	0.661	0.436	0.626	0.661	0.635	0.115	0.578	0.614	0.596
IBk	0.429	0.483	0.534	0.429	0.447	-0.082	0.482	0.560	0.478
J48	0.566	0.519	0.610	0.566	0.581	0.080	0.547	0.606	0.567
KStar	0.306	0.694	0.094	0.306	0.144	0.000	0.500	0.575	0.390
LMT	0.372	0.577	0.537	0.372	0.348	-0.057	0.448	0.545	0.459
LWL	0.282	0.518	0.371	0.282	0.223	-0.254	0.351	0.493	0.362
Logistic	0.306	0.694	0.094	0.306	0.144	0.000	0.509	0.575	0.391
Make Density Based Clusterer	0.591	0.577	0.644	0.591	0.606	0.155	0.584	0.616	0.598
REPTree	0.515	0.517	0.589	0.515	0.534	0.030	0.586	0.638	0.551
Random Forest	0.530	0.545	0.607	0.530	0.548	0.069	0.564	0.622	0.560
Random Tree	0.451	0.426	0.523	0.451	0.472	-0.114	0.434	0.551	0.469
SimpleKMeans	0.346	0.673	0.611	0.346	0.251	0.032	0.478	0.568	0.474
Simple Logistic	0.492	0.524	0.582	0.492	0.511	0.015	0.575	0.629	0.539

Table 6.21: Model scores for test set (boosted).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.369	0.631	0.136	0.369	0.199	0.000	0.446	0.514	0.396
Decision Stump	0.631	0.369	0.398	0.631	0.488	0.000	0.500	0.534	0.506
EM	0.606	0.402	0.542	0.606	0.533	0.012	0.504	0.536	0.529
Farthest First	0.575	0.441	0.543	0.575	0.551	0.019	0.507	0.541	0.530
Filtered Clusterer	0.495	0.396	0.482	0.495	0.488	-0.111	0.446	0.512	0.470
Hierarchical Clusterer	0.369	0.631	0.136	0.369	0.199	0.000	0.500	0.534	0.405
Hoeffding Tree	0.592	0.349	0.408	0.592	0.473	-0.140	0.515	0.544	0.488
IBk	0.524	0.326	0.419	0.524	0.455	-0.206	0.495	0.546	0.461
J48	0.564	0.438	0.535	0.564	0.543	0.002	0.537	0.560	0.530
KStar	0.369	0.631	0.136	0.369	0.199	0.000	0.500	0.534	0.405
LMT	0.436	0.581	0.547	0.436	0.403	0.020	0.545	0.575	0.504
LWL	0.631	0.369	0.398	0.631	0.488	0.000	0.452	0.498	0.496
Logistic	0.369	0.631	0.136	0.369	0.199	0.000	0.457	0.516	0.397
Make Density Based Clusterer	0.507	0.479	0.528	0.507	0.514	-0.013	0.493	0.531	0.507
REPTree	0.510	0.407	0.494	0.510	0.501	-0.086	0.431	0.506	0.477
Random Forest	0.558	0.344	0.438	0.558	0.473	-0.155	0.321	0.439	0.444
Random Tree	0.539	0.428	0.518	0.539	0.526	-0.034	0.483	0.527	0.505
SimpleKMeans	0.520	0.416	0.503	0.520	0.510	-0.066	0.464	0.518	0.490
Simple Logistic	0.590	0.382	0.508	0.590	0.513	-0.043	0.391	0.463	0.489

Table 6.22: Model scores for test set (tree).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.771	0.229	0.595	0.771	0.672	0.000	0.714	0.787	0.630
Decision Stump	0.678	0.905	0.866	0.678	0.702	0.492	0.791	0.793	0.770
EM	0.712	0.288	0.647	0.712	0.672	0.000	0.500	0.647	0.585
Farthest First	0.771	0.229	0.595	0.771	0.672	0.000	0.556	0.670	0.596
Filtered Clusterer	0.554	0.868	0.849	0.554	0.574	0.378	0.711	0.747	0.693
Hierarchical Clusterer	0.771	0.229	0.595	0.771	0.672	0.000	0.500	0.647	0.586
Hoeffding Tree	0.771	0.229	0.595	0.771	0.672	0.000	0.555	0.678	0.596
IBk	0.638	0.831	0.823	0.638	0.665	0.395	0.688	0.776	0.720
J48	0.728	0.902	0.867	0.728	0.749	0.528	0.858	0.849	0.806
KStar	0.771	0.229	0.595	0.771	0.672	0.000	0.500	0.647	0.586
LMT	0.721	0.915	0.873	0.721	0.743	0.534	0.816	0.825	0.798
LWL	0.678	0.905	0.866	0.678	0.702	0.492	0.871	0.881	0.791
Logistic	0.229	0.771	0.052	0.229	0.085	0.000	0.448	0.617	0.366
Make Density Based Clusterer	0.678	0.905	0.866	0.678	0.702	0.492	0.791	0.793	0.770
REPTree	0.678	0.876	0.850	0.678	0.703	0.466	0.775	0.796	0.761
Random Forest	0.773	0.877	0.862	0.773	0.790	0.553	0.886	0.882	0.827
Random Tree	0.735	0.769	0.812	0.735	0.754	0.433	0.752	0.769	0.755
SimpleKMeans	0.801	0.408	0.782	0.801	0.762	0.321	0.716	0.757	0.711
Simple Logistic	0.717	0.854	0.843	0.717	0.739	0.480	0.833	0.843	0.786

Table 6.23: Model scores for test set (sfo).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.881	0.119	0.776	0.881	0.825	0.000	0.639	0.848	0.684
Decision Stump	0.881	0.119	0.776	0.881	0.825	0.000	0.691	0.845	0.690
EM	0.881	0.119	0.776	0.881	0.825	0.000	0.694	0.851	0.691
Farthest First	0.881	0.119	0.776	0.881	0.825	0.000	0.624	0.830	0.679
Filtered Clusterer	0.881	0.119	0.776	0.881	0.825	0.000	0.580	0.818	0.672
Hierarchical Clusterer	0.884	0.296	0.857	0.884	0.860	0.281	0.584	0.818	0.728
Hoeffding Tree	0.879	0.129	0.814	0.879	0.827	0.041	0.577	0.820	0.681
IBk	0.912	0.562	0.905	0.912	0.907	0.539	0.920	0.949	0.855
J48	0.977	0.902	0.976	0.977	0.977	0.888	0.976	0.984	0.964
LMT	0.979	0.908	0.979	0.979	0.979	0.899	0.976	0.986	0.967
LWL	0.881	0.119	0.776	0.881	0.825	0.000	0.827	0.899	0.714
Logistic	0.883	0.291	0.856	0.883	0.859	0.273	0.854	0.907	0.771
Make Density Based Clusterer	0.881	0.119	0.776	0.881	0.825	0.000	0.577	0.817	0.672
REPTree	0.970	0.874	0.970	0.970	0.970	0.857	0.973	0.983	0.955
Random Forest	0.980	0.895	0.979	0.980	0.979	0.900	0.993	0.996	0.969
Random Tree	0.977	0.899	0.977	0.977	0.977	0.891	0.939	0.966	0.957
SimpleKMeans	0.881	0.119	0.776	0.881	0.825	0.000	0.658	0.844	0.686
Simple Logistic	0.882	0.252	0.850	0.882	0.853	0.233	0.807	0.894	0.755

Table 6.24: Model scores for training (bagged) set with binary classes.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.558	0.442	0.312	0.558	0.400	0.000	0.737	0.733	0.530
Decision Stump	0.706	0.745	0.756	0.706	0.701	0.466	0.740	0.722	0.726
EM	0.698	0.704	0.705	0.698	0.699	0.399	0.753	0.728	0.711
Farthest First	0.656	0.713	0.744	0.656	0.641	0.408	0.711	0.703	0.691
Filtered Clusterer	0.628	0.639	0.640	0.628	0.629	0.266	0.674	0.658	0.641
Hierarchical Clusterer	0.575	0.47	0.618	0.575	0.460	0.106	0.520	0.526	0.537
Hoeffding Tree	0.569	0.46	0.614	0.569	0.440	0.085	0.717	0.685	0.575
IBk	0.861	0.861	0.862	0.861	0.861	0.719	0.948	0.947	0.883
J48	0.968	0.968	0.968	0.968	0.968	0.936	0.981	0.974	0.970
LMT	0.972	0.972	0.972	0.972	0.972	0.944	0.989	0.987	0.976
LWL	0.734	0.696	0.744	0.734	0.725	0.462	0.882	0.890	0.767
Logistic	0.737	0.727	0.736	0.737	0.736	0.465	0.832	0.839	0.760
Make Density Based Clusterer	0.628	0.639	0.640	0.628	0.629	0.266	0.674	0.658	0.641
REPTree	0.960	0.959	0.960	0.960	0.960	0.919	0.983	0.977	0.965
Random Forest	0.971	0.971	0.971	0.971	0.971	0.942	0.996	0.995	0.977
Random Tree	0.966	0.966	0.966	0.966	0.966	0.932	0.967	0.953	0.965
SimpleKMeans	0.678	0.696	0.696	0.678	0.678	0.373	0.749	0.732	0.699
Simple Logistic	0.733	0.736	0.737	0.733	0.734	0.466	0.822	0.827	0.757

Table 6.25: Model scores for training (boosted) set with binary classes.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.563	0.447	0.755	0.563	0.411	0.076	0.670	0.682	0.579
Decision Stump	0.678	0.745	0.814	0.678	0.656	0.495	0.703	0.697	0.715
EM	0.660	0.643	0.658	0.660	0.658	0.305	0.732	0.714	0.672
Farthest First	0.608	0.584	0.604	0.608	0.604	0.196	0.641	0.630	0.610
Filtered Clusterer	0.610	0.611	0.616	0.610	0.612	0.220	0.649	0.637	0.619
Hierarchical Clusterer	0.598	0.493	0.724	0.598	0.488	0.210	0.542	0.552	0.575
Hoeffding Tree	0.703	0.706	0.708	0.703	0.704	0.406	0.756	0.734	0.715
IBk	0.851	0.852	0.853	0.851	0.852	0.701	0.942	0.941	0.874
J48	0.973	0.974	0.973	0.973	0.973	0.946	0.986	0.981	0.976
LMT	0.974	0.973	0.974	0.974	0.974	0.947	0.992	0.990	0.978
LWL	0.679	0.746	0.813	0.679	0.657	0.495	0.822	0.820	0.745
Logistic	0.754	0.761	0.761	0.754	0.755	0.511	0.854	0.854	0.781
Make Density Based Clusterer	0.612	0.614	0.618	0.612	0.613	0.224	0.651	0.638	0.621
REPTree	0.964	0.964	0.964	0.964	0.964	0.927	0.989	0.987	0.970
Random Forest	0.975	0.973	0.975	0.975	0.975	0.948	0.997	0.997	0.980
Random Tree	0.971	0.97	0.971	0.971	0.971	0.942	0.973	0.961	0.970
SimpleKMeans	0.644	0.66	0.660	0.644	0.645	0.304	0.720	0.708	0.667
Simple Logistic	0.753	0.767	0.766	0.753	0.754	0.517	0.848	0.846	0.781

Table 6.26: Model scores for training (tree) set with binary classes.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.661	0.339	0.436	0.661	0.526	0.000	0.886	0.884	0.612
Decision Stump	0.881	0.939	0.912	0.881	0.884	0.779	0.905	0.880	0.896
EM	0.795	0.8	0.812	0.795	0.799	0.573	0.873	0.854	0.814
Farthest First	0.680	0.548	0.664	0.680	0.667	0.246	0.753	0.754	0.671
Filtered Clusterer	0.682	0.447	0.659	0.682	0.626	0.192	0.712	0.723	0.641
Hierarchical Clusterer	0.661	0.339	0.436	0.661	0.526	0.000	0.535	0.588	0.531
Hoeffding Tree	0.661	0.339	0.436	0.661	0.526	0.000	0.811	0.797	0.591
IBk	0.945	0.95	0.947	0.945	0.946	0.881	0.986	0.984	0.955
J48	0.976	0.974	0.976	0.976	0.976	0.947	0.990	0.986	0.978
LMT	0.978	0.975	0.978	0.978	0.978	0.950	0.994	0.992	0.981
LWL	0.881	0.939	0.912	0.881	0.884	0.779	0.950	0.940	0.910
Logistic	0.895	0.922	0.908	0.895	0.897	0.786	0.960	0.956	0.916
Make Density Based Clusterer	0.682	0.447	0.659	0.682	0.626	0.192	0.711	0.723	0.641
REPTree	0.973	0.969	0.973	0.973	0.973	0.940	0.992	0.990	0.977
Random Forest	0.979	0.973	0.979	0.979	0.979	0.954	0.997	0.997	0.983
Random Tree	0.975	0.968	0.975	0.975	0.975	0.944	0.972	0.963	0.972
SimpleKMeans	0.774	0.722	0.774	0.774	0.774	0.495	0.862	0.850	0.785
Simple Logistic	0.894	0.918	0.906	0.894	0.895	0.782	0.956	0.947	0.913

Table 6.27: Model scores for training (sfo) set with binary classes.

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.654	0.346	0.428	0.654	0.517	0.000	0.467	0.539	0.513
Decision Stump	0.654	0.346	0.428	0.654	0.517	0.000	0.543	0.570	0.526
EM	0.654	0.346	0.428	0.654	0.517	0.000	0.457	0.529	0.511
Farthest First	0.654	0.346	0.428	0.654	0.517	0.000	0.489	0.543	0.516
Filtered Clusterer	0.654	0.346	0.428	0.654	0.517	0.000	0.511	0.553	0.520
Hierarchical Clusterer	0.654	0.346	0.428	0.654	0.517	0.000	0.500	0.548	0.518
Hoeffding Tree	0.654	0.346	0.428	0.654	0.517	0.000	0.500	0.548	0.518
IBk	0.649	0.387	0.594	0.649	0.563	0.068	0.498	0.553	0.553
J48	0.666	0.461	0.637	0.666	0.621	0.171	0.483	0.545	0.583
LMT	0.630	0.333	0.422	0.630	0.506	-0.115	0.514	0.567	0.506
LWL	0.654	0.346	0.428	0.654	0.517	0.000	0.479	0.528	0.513
Logistic	0.564	0.539	0.593	0.564	0.573	0.098	0.536	0.573	0.561
Make Density Based Clusterer	0.654	0.346	0.428	0.654	0.517	0.000	0.599	0.601	0.537
REPTree	0.630	0.462	0.599	0.630	0.603	0.107	0.572	0.598	0.581
Random Forest	0.654	0.346	0.428	0.654	0.517	0.000	0.647	0.675	0.553
Random Tree	0.523	0.393	0.508	0.523	0.514	-0.087	0.454	0.528	0.487
SimpleKMeans	0.654	0.346	0.428	0.654	0.517	0.000	0.487	0.540	0.516
Simple Logistic	0.654	0.346	0.428	0.654	0.517	0.000	0.540	0.566	0.526

Table 6.28: Model scores for test set with binary classes (bagged).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.306	0.694	0.094	0.306	0.144	0.000	0.521	0.585	0.394
Decision Stump	0.720	0.394	0.721	0.720	0.645	0.224	0.557	0.610	0.622
EM	0.383	0.527	0.522	0.383	0.381	-0.091	0.441	0.550	0.455
Farthest First	0.564	0.373	0.547	0.564	0.555	-0.066	0.468	0.563	0.513
Filtered Clusterer	0.306	0.694	0.094	0.306	0.144	0.000	0.553	0.600	0.400
Hierarchical Clusterer	0.306	0.694	0.094	0.306	0.144	0.000	0.500	0.575	0.390
Hoeffding Tree	0.661	0.584	0.676	0.661	0.667	0.237	0.618	0.646	0.641
IBk	0.661	0.513	0.651	0.661	0.655	0.179	0.509	0.570	0.601
J48	0.661	0.58	0.675	0.661	0.667	0.234	0.648	0.666	0.647
LMT	0.546	0.619	0.650	0.546	0.562	0.153	0.564	0.614	0.585
LWL	0.605	0.456	0.601	0.605	0.603	0.062	0.491	0.564	0.557
Logistic	0.490	0.48	0.562	0.490	0.510	-0.028	0.443	0.531	0.499
Make Density Based Clusterer	0.306	0.694	0.094	0.306	0.144	0.000	0.664	0.665	0.422
REPTree	0.653	0.515	0.648	0.653	0.651	0.172	0.644	0.671	0.628
Random Forest	0.516	0.632	0.648	0.516	0.527	0.140	0.630	0.690	0.591
Random Tree	0.520	0.6	0.631	0.520	0.536	0.112	0.569	0.608	0.568
SimpleKMeans	0.368	0.676	0.638	0.368	0.293	0.065	0.489	0.574	0.492
Simple Logistic	0.299	0.603	0.406	0.299	0.203	-0.159	0.433	0.555	0.402

Table 6.29: Model scores for test set with binary classes (boosted).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.369	0.631	0.136	0.369	0.199	0.000	0.470	0.530	0.401
Decision Stump	0.631	0.369	0.398	0.631	0.488	0.000	0.500	0.534	0.506
EM	0.559	0.388	0.500	0.559	0.512	-0.067	0.494	0.532	0.501
Farthest First	0.521	0.393	0.490	0.521	0.501	-0.094	0.457	0.516	0.481
Filtered Clusterer	0.462	0.395	0.468	0.462	0.465	-0.141	0.428	0.506	0.452
Hierarchical Clusterer	0.369	0.631	0.136	0.369	0.199	0.000	0.500	0.534	0.405
Hoeffding Tree	0.583	0.343	0.402	0.583	0.468	-0.160	0.486	0.562	0.481
IBk	0.510	0.363	0.464	0.510	0.479	-0.145	0.433	0.504	0.461
J48	0.451	0.568	0.547	0.451	0.433	0.020	0.521	0.554	0.504
LMT	0.446	0.399	0.463	0.446	0.453	-0.151	0.430	0.492	0.444
LWL	0.631	0.369	0.398	0.631	0.488	0.000	0.462	0.505	0.498
Logistic	0.516	0.431	0.510	0.516	0.513	-0.053	0.423	0.482	0.483
Make Density Based Clusterer	0.485	0.425	0.493	0.485	0.488	-0.089	0.455	0.515	0.475
REPTree	0.460	0.397	0.469	0.460	0.464	-0.141	0.430	0.514	0.453
Random Forest	0.555	0.34	0.432	0.555	0.470	-0.165	0.346	0.443	0.445
Random Tree	0.541	0.413	0.510	0.541	0.520	-0.050	0.489	0.531	0.502
SimpleKMeans	0.446	0.399	0.463	0.446	0.453	-0.151	0.424	0.501	0.445
Simple Logistic	0.575	0.384	0.502	0.575	0.513	-0.057	0.487	0.526	0.504

Table 6.30: Model scores for test set with binary classes (tree).

Model Type	Acc.	Error	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC	Avg.
Canopy	0.771	0.229	0.595	0.771	0.672	0.000	0.733	0.797	0.633
Decision Stump	0.678	0.905	0.866	0.678	0.702	0.492	0.791	0.793	0.770
EM	0.617	0.402	0.653	0.617	0.633	0.017	0.533	0.658	0.578
Farthest First	0.771	0.229	0.595	0.771	0.672	0.000	0.673	0.727	0.617
Filtered Clusterer	0.771	0.229	0.595	0.771	0.672	0.000	0.358	0.608	0.563
Hierarchical Clusterer	0.771	0.229	0.595	0.771	0.672	0.000	0.500	0.647	0.586
Hoeffding Tree	0.771	0.229	0.595	0.771	0.672	0.000	0.555	0.678	0.596
IBk	0.472	0.813	0.810	0.472	0.482	0.272	0.725	0.790	0.650
J48	0.727	0.855	0.845	0.727	0.749	0.490	0.867	0.868	0.798
LMT	0.740	0.845	0.843	0.740	0.760	0.495	0.843	0.852	0.796
LWL	0.678	0.905	0.866	0.678	0.702	0.492	0.804	0.842	0.778
Logistic	0.602	0.179	0.559	0.602	0.580	-0.245	0.451	0.626	0.497
Make Density Based Clusterer	0.771	0.229	0.595	0.771	0.672	0.000	0.615	0.695	0.606
REPTree	0.703	0.817	0.824	0.703	0.727	0.438	0.767	0.802	0.758
Random Forest	0.785	0.901	0.874	0.785	0.802	0.584	0.895	0.884	0.840
Random Tree	0.756	0.638	0.778	0.756	0.764	0.366	0.697	0.740	0.727
SimpleKMeans	0.740	0.341	0.689	0.740	0.704	0.108	0.712	0.752	0.654
Simple Logistic	0.624	0.889	0.858	0.624	0.648	0.440	0.746	0.809	0.740

Table 6.31: Model scores for test set with binary classes (sfo).

6.7.1 Principal Component Analysis

Principle component analysis was also performed on the per algorithm datasets. The first two components of for each of these datasets is given in Figure 6.4 through Figure 6.7. As with the unified dataset with all of the algorithms, most of the datasets do not show any clear clustering of algorithms. The one dataset that did give somewhat promising scores was sfo (Figure 6.7). Here there does seem to be a bit more grouping among the individual symbols. However, when the sfo and non-sfo (other) symbols are considered as groups it seems that many of the non-sfo (i.e. 'other') points lie outside the main cluster.

As with the unified dataset, PCA was also performed on versions of the dataset where the characteristics are based on a binary class version of the original datasets. These PCA results, shown in Figure 6.8 through Figure 6.11, show essentially the same pattern. In most, there is one large grouping of all points. In the logistic regression (sfo) specific graph, the points that lay outside the main cluster tend to be 'other', which could indicate that the higher model scores for logistic regression are mostly due to it being easier to spot areas where logistic regression will perform poorly.

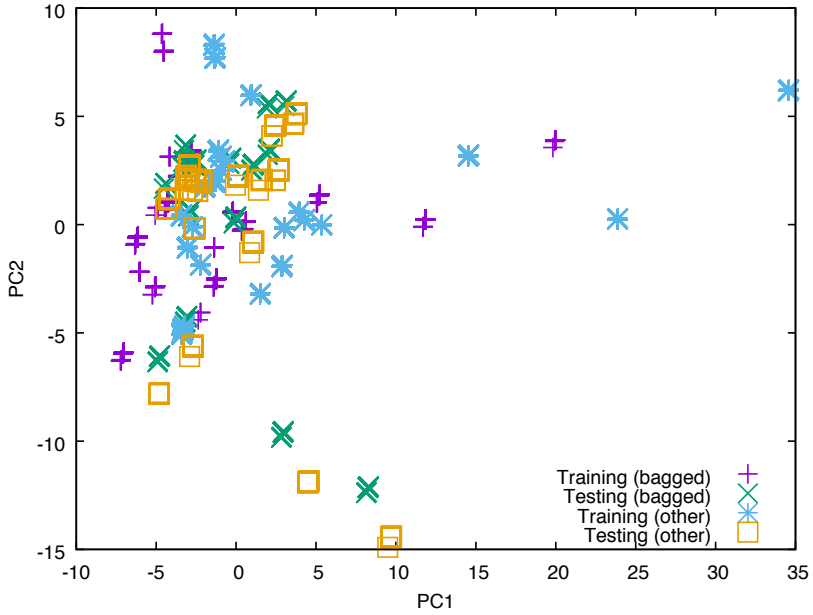

Figure 6.4: First two principle components for bagged specific algorithm selection dataset. First and second principle components account for 35.1% and 32.0% of variation respectively.

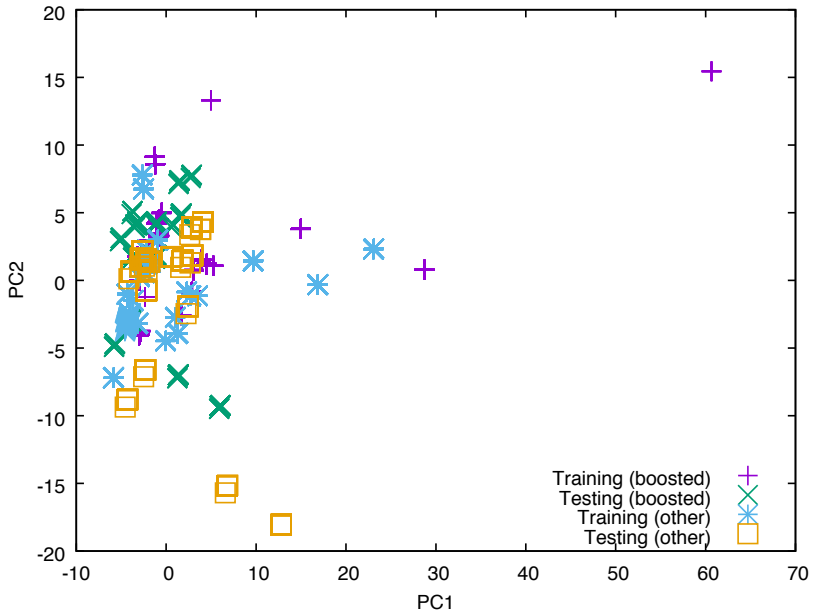

Figure 6.5: First two principle components for boosted specific algorithm selection dataset. First and second principle components account for 35.1% and 32.0% of variation respectively.

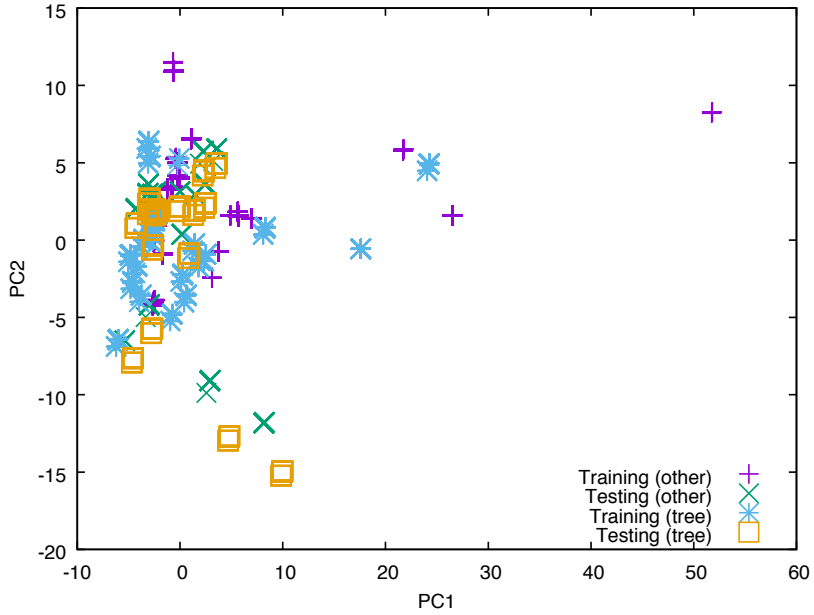

Figure 6.6: First two principle components for tree specific algorithm selection dataset. First and second principle components account for 35.1% and 32.0% of variation respectively.

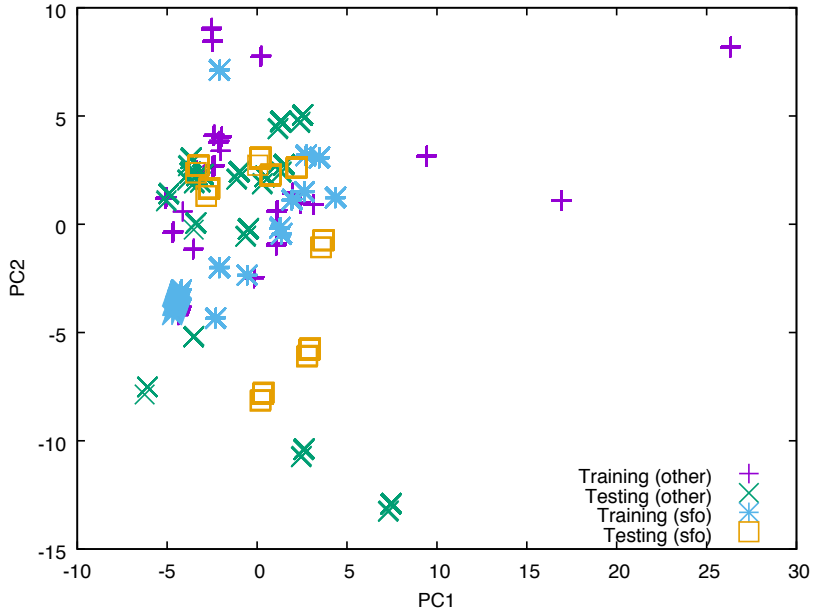

Figure 6.7: First two principle components for sfo specific algorithm selection dataset. First and second principle components account for 35.1% and 32.0% of variation respectively.

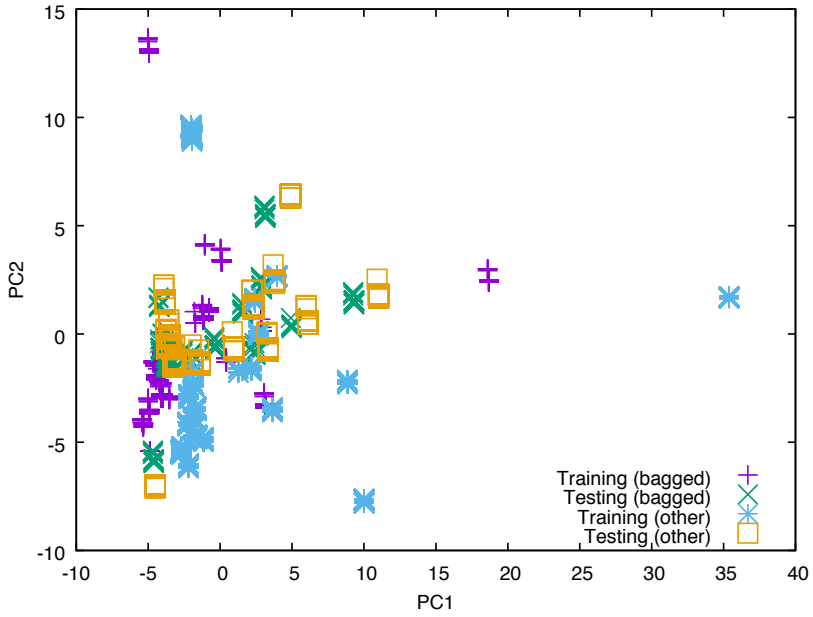

Figure 6.8: First two principle components for bagged specific algorithm selection dataset(binary classes). First and second principle components account for 37.4% and 26.9% of variation respectively.

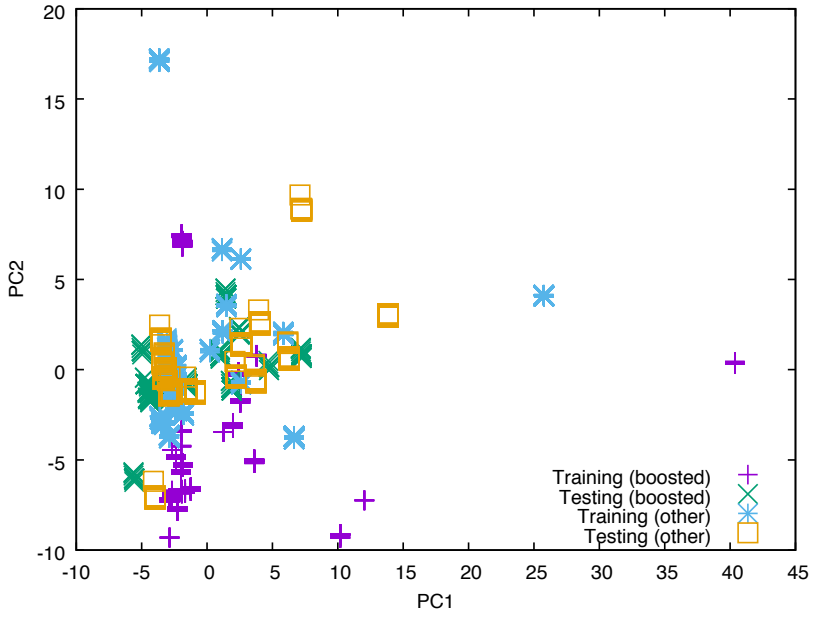

Figure 6.9: First two principle components for boosted specific algorithm selection dataset(binary classes). First and second principle components account for 37.4% and 26.9% of variation respectively.

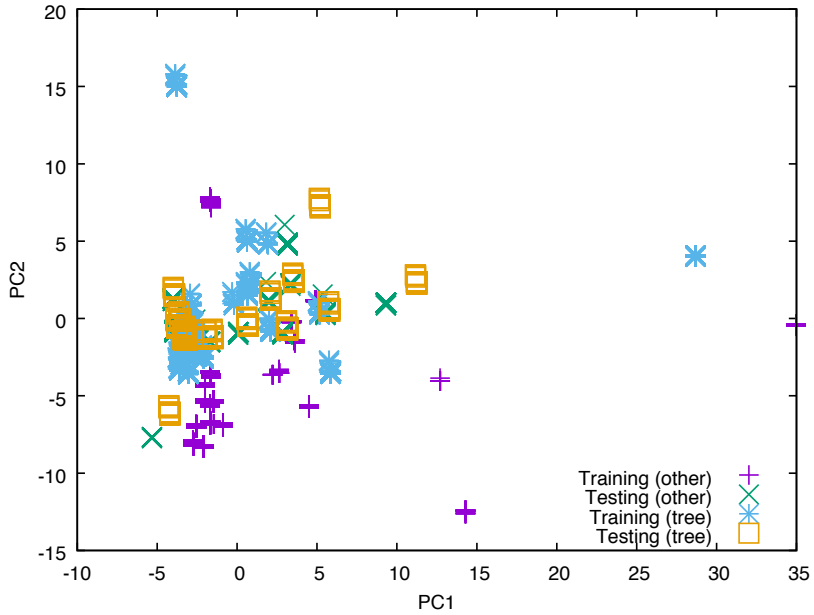

Figure 6.10: First two principle components for tree specific algorithm selection dataset(binary classes). First and second principle components account for 37.4% and 26.9% of variation respectively.

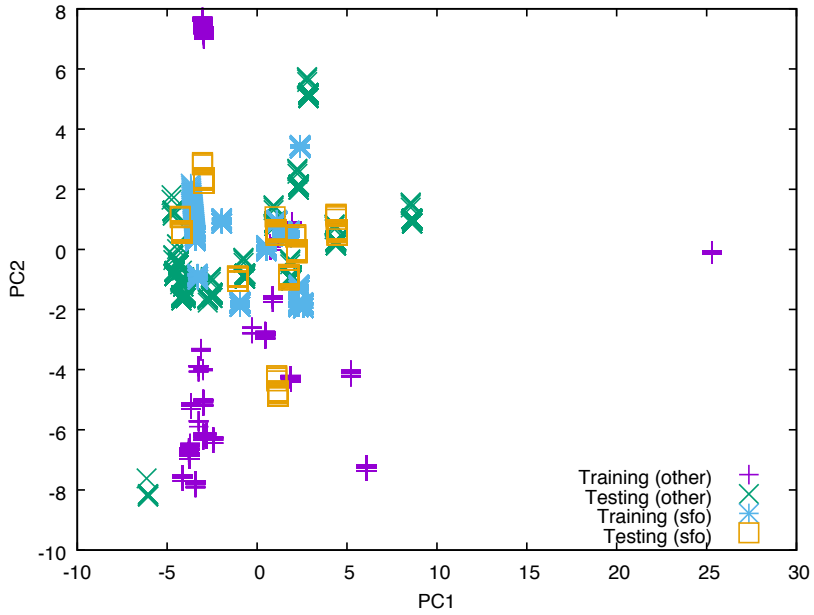

Figure 6.11: First two principle components for sfo specific algorithm selection dataset(binary classes). First and second principle components account for 37.4% and 26.9% of variation respectively.

6.8 Cross Validation

The model scores in Section 6.6 and Section 6.7 are low enough to suggest that building a model to select algorithms is not viable. To check to see if these scores were low due to an unlucky assignment of datasets to the training and testing sets, 5x5-fold cross validation was performed. This consisted of splitting the datasets randomly into five equally sized groups. Each of the five groups then served as a testing set for models trained on the samples in the other four sets. This was repeated five times with different random assignments to the groups. The various model metrics were then averaged over all 25 models produced, and the 95% confidence intervals are reported in Table 6.32. For the purposes of grouping, the samples for a binorm dataset are always assigned to the same group as the non-binorm version. The 5x5-fold cross validation was repeated for the datasets based on the binary class versions of the original datasets. The results for those models are given in Table 6.33. Between the binary class based datasets and regular versions of datasets there is no statistically significant difference in the scores. For all of the model types the confidence intervals for AUC do not include 0.5, which indicates that they are doing better than random guessing. The best performing model type, Random Forest, has a lower end of its confidence interval around 0.75, which is far better than expected based on the initial training/testing split. These results are consistent what Nural *et al.* [77] found regarding building models to predict algorithms for regression problems.

Cross validation was also performed using the samples that were labeled based on individual algorithms The results of that cross validation, given in Table 6.34 through Table 6.41. With the original splitting of the datasets, which showed only some of the sfo specific AUC results were better than random guessing. The cross validation AUC results show that bagged, boosted, and sfo algorithm specific models achieved scores that are above 0.6 with 95% confidence. For the binary class versions of the datasets, the lower end of the confidence intervals were even higher for bagged, boosted and sfo, but still crossed 0.5 for tree.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.253,0.350)	(0.078,0.142)	(0.253,0.350)	(0.117,0.198)	(0.000,0.000)	(0.571,0.641)	(0.415,0.479)
Decision Stump	(0.339,0.422)	(0.232,0.324)	(0.339,0.422)	(0.243,0.326)	(0.115,0.199)	(0.599,0.657)	(0.394,0.471)
EM	(0.411,0.517)	(0.375,0.491)	(0.411,0.517)	(0.348,0.463)	(0.181,0.300)	(0.632,0.684)	(0.421,0.493)
Farthest First	(0.386,0.489)	(0.276,0.380)	(0.386,0.489)	(0.284,0.390)	(0.085,0.180)	(0.550,0.598)	(0.370,0.422)
Filtered Clusterer	(0.315,0.419)	(0.169,0.280)	(0.315,0.419)	(0.201,0.313)	(0.033,0.129)	(0.526,0.587)	(0.342,0.415)
Hoeffding Tree	(0.331,0.424)	(0.339,0.469)	(0.331,0.424)	(0.260,0.368)	(0.125,0.237)	(0.589,0.658)	(0.404,0.487)
J48	(0.346,0.416)	(0.445,0.518)	(0.346,0.416)	(0.338,0.412)	(0.131,0.220)	(0.567,0.630)	(0.426,0.477)
Logistic	(0.201,0.334)	(0.188,0.351)	(0.201,0.334)	(0.166,0.311)	(-0.010,0.145)	(0.502,0.597)	(0.367,0.466)
Make Density Based Clusterer	(0.323,0.428)	(0.189,0.305)	(0.323,0.428)	(0.212,0.326)	(0.052,0.147)	(0.537,0.598)	(0.351,0.421)
REPTree	(0.428,0.534)	(0.479,0.584)	(0.428,0.534)	(0.426,0.535)	(0.216,0.356)	(0.640,0.719)	(0.480,0.577)
Random Forest	(0.489,0.565)	(0.556,0.611)	(0.489,0.565)	(0.487,0.562)	(0.301,0.394)	(0.758,0.801)	(0.607,0.671)
Random Tree	(0.426,0.519)	(0.496,0.568)	(0.426,0.519)	(0.434,0.524)	(0.222,0.334)	(0.616,0.672)	(0.433,0.493)
SimpleKMeans	(0.338,0.433)	(0.246,0.360)	(0.338,0.433)	(0.253,0.357)	(0.067,0.183)	(0.557,0.619)	(0.384,0.449)
Simple Logistic	(0.406,0.515)	(0.443,0.545)	(0.406,0.515)	(0.377,0.488)	(0.161,0.298)	(0.655,0.738)	(0.470,0.573)

Table 6.32: Cross validation scores for unified models.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.253,0.350)	(0.078,0.142)	(0.253,0.350)	(0.117,0.198)	(0.000,0.000)	(0.562,0.635)	(0.405,0.487)
Decision Stump	(0.330,0.414)	(0.220,0.316)	(0.330,0.414)	(0.231,0.317)	(0.106,0.193)	(0.592,0.653)	(0.389,0.469)
EM	(0.391,0.493)	(0.312,0.433)	(0.391,0.493)	(0.314,0.432)	(0.124,0.239)	(0.584,0.649)	(0.390,0.467)
Farthest First	(0.314,0.410)	(0.208,0.300)	(0.314,0.410)	(0.209,0.301)	(0.027,0.102)	(0.522,0.576)	(0.347,0.406)
Filtered Clusterer	(0.268,0.378)	(0.135,0.253)	(0.268,0.378)	(0.163,0.281)	(-0.007,0.098)	(0.531,0.599)	(0.349,0.431)
Hoeffding Tree	(0.315,0.408)	(0.330,0.454)	(0.315,0.408)	(0.254,0.349)	(0.122,0.217)	(0.610,0.677)	(0.432,0.506)
J48	(0.345,0.409)	(0.431,0.502)	(0.345,0.409)	(0.335,0.402)	(0.123,0.206)	(0.556,0.609)	(0.416,0.461)
Logistic	(0.174,0.309)	(0.167,0.324)	(0.174,0.309)	(0.141,0.286)	(-0.022,0.133)	(0.485,0.576)	(0.351,0.450)
Make Density Based Clusterer	(0.272,0.381)	(0.134,0.247)	(0.272,0.381)	(0.158,0.273)	(0.016,0.107)	(0.524,0.576)	(0.348,0.411)
REPTree	(0.441,0.541)	(0.491,0.589)	(0.441,0.541)	(0.440,0.540)	(0.234,0.363)	(0.656,0.726)	(0.494,0.584)
Random Forest	(0.505,0.578)	(0.572,0.631)	(0.505,0.578)	(0.503,0.573)	(0.328,0.417)	(0.766,0.809)	(0.623,0.683)
Random Tree	(0.403,0.496)	(0.479,0.564)	(0.403,0.496)	(0.396,0.494)	(0.202,0.317)	(0.611,0.667)	(0.422,0.487)
SimpleKMeans	(0.329,0.422)	(0.249,0.364)	(0.329,0.422)	(0.246,0.346)	(0.080,0.183)	(0.544,0.611)	(0.378,0.445)
Simple Logistic	(0.449,0.544)	(0.480,0.583)	(0.449,0.544)	(0.429,0.528)	(0.211,0.340)	(0.668,0.736)	(0.493,0.584)

Table 6.33: Cross validation scores for unified models with binary classes.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.584,0.648)	(0.748,0.808)
Decision Stump	(0.781,0.833)	(0.642,0.722)	(0.781,0.833)	(0.705,0.770)	(-0.026,0.003)	(0.475,0.521)	(0.688,0.752)
EM	(0.787,0.843)	(0.643,0.723)	(0.787,0.843)	(0.707,0.776)	(-0.022,0.006)	(0.513,0.598)	(0.710,0.775)
Farthest First	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.550,0.607)	(0.714,0.780)
Filtered Clusterer	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.489,0.555)	(0.697,0.763)
Hoeffding Tree	(0.725,0.799)	(0.681,0.765)	(0.725,0.799)	(0.687,0.758)	(0.003,0.100)	(0.450,0.538)	(0.688,0.758)
J48	(0.689,0.766)	(0.736,0.792)	(0.689,0.766)	(0.701,0.766)	(0.085,0.184)	(0.550,0.626)	(0.735,0.791)
Logistic	(0.525,0.657)	(0.696,0.778)	(0.525,0.657)	(0.539,0.677)	(0.008,0.115)	(0.533,0.624)	(0.702,0.781)
Make Density Based Clusterer	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.509,0.557)	(0.703,0.765)
REPTree	(0.665,0.752)	(0.736,0.789)	(0.665,0.752)	(0.678,0.747)	(0.062,0.165)	(0.556,0.626)	(0.741,0.796)
Random Forest	(0.781,0.825)	(0.754,0.810)	(0.781,0.825)	(0.742,0.799)	(0.121,0.213)	(0.622,0.685)	(0.773,0.826)
Random Tree	(0.689,0.756)	(0.720,0.783)	(0.689,0.756)	(0.696,0.758)	(0.057,0.152)	(0.534,0.588)	(0.711,0.774)
SimpleKMeans	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.484,0.569)	(0.706,0.768)
Simple Logistic	(0.757,0.814)	(0.686,0.769)	(0.757,0.814)	(0.710,0.777)	(0.042,0.116)	(0.538,0.629)	(0.724,0.791)

Table 6.34: Cross validation scores for bagged specific models.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.353,0.435)	(0.135,0.223)	(0.353,0.435)	(0.194,0.283)	(-0.007,0.027)	(0.527,0.625)	(0.596,0.661)
Decision Stump	(0.556,0.656)	(0.599,0.730)	(0.556,0.656)	(0.451,0.580)	(0.140,0.247)	(0.548,0.593)	(0.571,0.624)
EM	(0.556,0.639)	(0.551,0.664)	(0.556,0.639)	(0.511,0.618)	(0.095,0.251)	(0.579,0.662)	(0.604,0.664)
Farthest First	(0.459,0.561)	(0.473,0.631)	(0.459,0.561)	(0.378,0.511)	(0.087,0.205)	(0.524,0.597)	(0.571,0.620)
Filtered Clusterer	(0.506,0.592)	(0.416,0.551)	(0.506,0.592)	(0.425,0.542)	(-0.023,0.106)	(0.500,0.567)	(0.544,0.600)
Hoeffding Tree	(0.539,0.633)	(0.508,0.630)	(0.539,0.633)	(0.472,0.583)	(0.053,0.183)	(0.518,0.608)	(0.575,0.644)
J48	(0.518,0.628)	(0.588,0.675)	(0.518,0.628)	(0.476,0.609)	(0.109,0.265)	(0.530,0.644)	(0.616,0.672)
Logistic	(0.533,0.622)	(0.576,0.663)	(0.533,0.622)	(0.509,0.612)	(0.072,0.229)	(0.566,0.655)	(0.601,0.677)
Make Density Based Clusterer	(0.513,0.602)	(0.433,0.572)	(0.513,0.602)	(0.432,0.552)	(-0.014,0.130)	(0.510,0.581)	(0.553,0.608)
REPTree	(0.589,0.683)	(0.639,0.718)	(0.589,0.683)	(0.570,0.681)	(0.215,0.356)	(0.626,0.719)	(0.677,0.726)
Random Forest	(0.618,0.693)	(0.682,0.734)	(0.618,0.693)	(0.603,0.689)	(0.255,0.378)	(0.746,0.807)	(0.779,0.821)
Random Tree	(0.603,0.685)	(0.645,0.713)	(0.603,0.685)	(0.597,0.683)	(0.206,0.343)	(0.605,0.678)	(0.615,0.675)
SimpleKMeans	(0.483,0.561)	(0.414,0.553)	(0.483,0.561)	(0.397,0.512)	(-0.032,0.099)	(0.530,0.596)	(0.569,0.625)
Simple Logistic	(0.574,0.653)	(0.570,0.671)	(0.574,0.653)	(0.535,0.628)	(0.076,0.245)	(0.621,0.724)	(0.667,0.751)

Table 6.35: Cross validation scores for boosted specific models.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.359,0.429)	(0.136,0.194)	(0.359,0.429)	(0.196,0.265)	(0.000,0.000)	(0.457,0.534)	(0.530,0.576)
Decision Stump	(0.527,0.612)	(0.488,0.639)	(0.527,0.612)	(0.418,0.507)	(0.068,0.176)	(0.520,0.567)	(0.549,0.594)
EM	(0.502,0.598)	(0.420,0.562)	(0.502,0.598)	(0.429,0.553)	(0.052,0.158)	(0.547,0.612)	(0.574,0.622)
Farthest First	(0.355,0.431)	(0.317,0.454)	(0.355,0.431)	(0.274,0.380)	(-0.127,-0.003)	(0.463,0.517)	(0.526,0.568)
Filtered Clusterer	(0.422,0.515)	(0.309,0.449)	(0.422,0.515)	(0.322,0.441)	(-0.049,0.027)	(0.462,0.516)	(0.526,0.560)
Hoeffding Tree	(0.423,0.518)	(0.488,0.573)	(0.423,0.518)	(0.406,0.510)	(-0.100,0.051)	(0.427,0.526)	(0.530,0.585)
J48	(0.561,0.675)	(0.588,0.695)	(0.561,0.675)	(0.562,0.677)	(0.119,0.328)	(0.556,0.676)	(0.600,0.681)
Logistic	(0.517,0.604)	(0.518,0.611)	(0.517,0.604)	(0.504,0.597)	(-0.020,0.143)	(0.480,0.596)	(0.531,0.623)
Make Density Based Clusterer	(0.424,0.520)	(0.339,0.488)	(0.424,0.520)	(0.323,0.444)	(-0.033,0.057)	(0.484,0.538)	(0.536,0.572)
REPTree	(0.542,0.645)	(0.581,0.669)	(0.542,0.645)	(0.539,0.643)	(0.098,0.270)	(0.565,0.667)	(0.607,0.685)
Random Forest	(0.486,0.602)	(0.541,0.633)	(0.486,0.602)	(0.484,0.600)	(0.007,0.204)	(0.501,0.624)	(0.596,0.670)
Random Tree	(0.461,0.561)	(0.512,0.599)	(0.461,0.561)	(0.463,0.562)	(-0.047,0.127)	(0.468,0.576)	(0.544,0.600)
SimpleKMeans	(0.455,0.541)	(0.428,0.537)	(0.455,0.541)	(0.377,0.477)	(-0.029,0.049)	(0.488,0.557)	(0.544,0.589)
Simple Logistic	(0.542,0.640)	(0.548,0.643)	(0.542,0.640)	(0.521,0.618)	(0.037,0.205)	(0.533,0.652)	(0.586,0.684)

Table 6.36: Cross validation scores for tree specific models.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.638,0.725)	(0.423,0.536)	(0.638,0.725)	(0.504,0.614)	(0.000,0.000)	(0.569,0.661)	(0.645,0.719)
Decision Stump	(0.570,0.667)	(0.437,0.560)	(0.570,0.667)	(0.476,0.576)	(-0.038,0.045)	(0.507,0.580)	(0.603,0.654)
EM	(0.600,0.697)	(0.533,0.662)	(0.600,0.697)	(0.535,0.644)	(0.039,0.148)	(0.488,0.567)	(0.591,0.657)
Farthest First	(0.674,0.741)	(0.626,0.690)	(0.674,0.741)	(0.626,0.692)	(0.114,0.208)	(0.547,0.610)	(0.616,0.674)
Filtered Clusterer	(0.620,0.718)	(0.448,0.569)	(0.620,0.718)	(0.497,0.616)	(-0.001,0.048)	(0.496,0.572)	(0.588,0.655)
Hoeffding Tree	(0.537,0.643)	(0.588,0.674)	(0.537,0.643)	(0.498,0.611)	(0.042,0.157)	(0.497,0.581)	(0.601,0.662)
J48	(0.611,0.673)	(0.653,0.709)	(0.611,0.673)	(0.609,0.671)	(0.116,0.234)	(0.600,0.667)	(0.673,0.719)
Logistic	(0.555,0.645)	(0.527,0.637)	(0.555,0.645)	(0.501,0.607)	(0.014,0.095)	(0.484,0.569)	(0.601,0.664)
Make Density Based Clusterer	(0.641,0.727)	(0.449,0.571)	(0.641,0.727)	(0.517,0.627)	(0.002,0.052)	(0.491,0.562)	(0.579,0.648)
REPTree	(0.598,0.663)	(0.639,0.699)	(0.598,0.663)	(0.600,0.660)	(0.076,0.214)	(0.573,0.658)	(0.671,0.724)
Random Forest	(0.681,0.726)	(0.694,0.740)	(0.681,0.726)	(0.664,0.714)	(0.196,0.311)	(0.696,0.766)	(0.750,0.801)
Random Tree	(0.608,0.674)	(0.645,0.707)	(0.608,0.674)	(0.611,0.675)	(0.095,0.241)	(0.544,0.617)	(0.621,0.676)
SimpleKMeans	(0.614,0.713)	(0.465,0.601)	(0.614,0.713)	(0.508,0.631)	(0.020,0.111)	(0.457,0.530)	(0.578,0.640)
Simple Logistic	(0.586,0.673)	(0.544,0.659)	(0.586,0.673)	(0.529,0.629)	(0.041,0.154)	(0.565,0.656)	(0.647,0.719)

Table 6.37: Cross validation scores for sfo specific models.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.495,0.562)	(0.706,0.778)
Decision Stump	(0.786,0.840)	(0.643,0.723)	(0.786,0.840)	(0.706,0.774)	(-0.023,0.005)	(0.484,0.537)	(0.691,0.756)
EM	(0.792,0.843)	(0.643,0.723)	(0.792,0.843)	(0.709,0.777)	(-0.017,0.004)	(0.520,0.593)	(0.715,0.775)
Farthest First	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.523,0.578)	(0.705,0.770)
Filtered Clusterer	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.438,0.505)	(0.683,0.750)
Hoeffding Tree	(0.633,0.763)	(0.674,0.749)	(0.633,0.763)	(0.613,0.730)	(-0.021,0.051)	(0.481,0.570)	(0.698,0.765)
J48	(0.719,0.778)	(0.736,0.793)	(0.719,0.778)	(0.714,0.770)	(0.067,0.164)	(0.526,0.598)	(0.718,0.782)
Logistic	(0.530,0.663)	(0.697,0.774)	(0.530,0.663)	(0.545,0.678)	(0.026,0.108)	(0.518,0.600)	(0.703,0.778)
Make Density Based Clusterer	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.505,0.563)	(0.698,0.764)
REPTree	(0.637,0.730)	(0.744,0.788)	(0.637,0.730)	(0.662,0.733)	(0.071,0.179)	(0.515,0.610)	(0.736,0.790)
Random Forest	(0.785,0.826)	(0.763,0.814)	(0.785,0.826)	(0.747,0.802)	(0.139,0.231)	(0.618,0.678)	(0.773,0.824)
Random Tree	(0.641,0.728)	(0.707,0.770)	(0.641,0.728)	(0.658,0.734)	(0.008,0.112)	(0.499,0.568)	(0.708,0.767)
SimpleKMeans	(0.800,0.848)	(0.644,0.724)	(0.800,0.848)	(0.712,0.780)	(0.000,0.000)	(0.467,0.550)	(0.698,0.763)
Simple Logistic	(0.707,0.804)	(0.697,0.776)	(0.707,0.804)	(0.686,0.776)	(0.048,0.123)	(0.559,0.648)	(0.723,0.795)

Table 6.38: Cross validation scores for bagged specific models with binary classes.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.352,0.429)	(0.133,0.196)	(0.352,0.429)	(0.191,0.266)	(0.000,0.000)	(0.530,0.611)	(0.590,0.647)
Decision Stump	(0.556,0.656)	(0.599,0.730)	(0.556,0.656)	(0.451,0.580)	(0.140,0.247)	(0.548,0.593)	(0.571,0.624)
EM	(0.556,0.643)	(0.463,0.602)	(0.556,0.643)	(0.482,0.594)	(0.068,0.199)	(0.534,0.624)	(0.572,0.640)
Farthest First	(0.462,0.551)	(0.468,0.620)	(0.462,0.551)	(0.376,0.493)	(0.069,0.181)	(0.530,0.586)	(0.563,0.609)
Filtered Clusterer	(0.465,0.560)	(0.372,0.519)	(0.465,0.560)	(0.381,0.508)	(-0.054,0.082)	(0.491,0.573)	(0.544,0.607)
Hoeffding Tree	(0.530,0.619)	(0.579,0.668)	(0.530,0.619)	(0.485,0.589)	(0.081,0.225)	(0.542,0.624)	(0.586,0.649)
J48	(0.531,0.627)	(0.602,0.677)	(0.531,0.627)	(0.497,0.614)	(0.125,0.260)	(0.499,0.610)	(0.590,0.644)
Logistic	(0.501,0.586)	(0.546,0.638)	(0.501,0.586)	(0.473,0.575)	(0.008,0.177)	(0.516,0.619)	(0.569,0.654)
Make Density Based Clusterer	(0.470,0.572)	(0.363,0.523)	(0.470,0.572)	(0.366,0.508)	(-0.018,0.125)	(0.487,0.561)	(0.543,0.598)
REPTree	(0.585,0.671)	(0.633,0.703)	(0.585,0.671)	(0.572,0.668)	(0.179,0.322)	(0.623,0.716)	(0.665,0.730)
Random Forest	(0.634,0.693)	(0.682,0.732)	(0.634,0.693)	(0.621,0.687)	(0.258,0.369)	(0.750,0.806)	(0.782,0.822)
Random Tree	(0.576,0.652)	(0.614,0.698)	(0.576,0.652)	(0.562,0.649)	(0.156,0.300)	(0.583,0.662)	(0.594,0.657)
SimpleKMeans	(0.482,0.568)	(0.437,0.590)	(0.482,0.568)	(0.407,0.534)	(0.027,0.164)	(0.520,0.599)	(0.572,0.624)
Simple Logistic	(0.599,0.668)	(0.590,0.674)	(0.599,0.668)	(0.543,0.633)	(0.093,0.243)	(0.581,0.679)	(0.639,0.715)

Table 6.39: Cross validation scores for boosted specific models with binary classes.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.359,0.429)	(0.136,0.194)	(0.359,0.429)	(0.196,0.265)	(0.000,0.000)	(0.456,0.508)	(0.525,0.560)
Decision Stump	(0.527,0.612)	(0.488,0.639)	(0.527,0.612)	(0.418,0.507)	(0.068,0.176)	(0.520,0.567)	(0.549,0.594)
EM	(0.474,0.574)	(0.355,0.508)	(0.474,0.574)	(0.382,0.516)	(0.019,0.126)	(0.538,0.607)	(0.565,0.623)
Farthest First	(0.357,0.421)	(0.258,0.386)	(0.357,0.421)	(0.251,0.345)	(-0.106,-0.013)	(0.451,0.502)	(0.522,0.558)
Filtered Clusterer	(0.441,0.534)	(0.357,0.497)	(0.441,0.534)	(0.357,0.473)	(-0.023,0.053)	(0.473,0.530)	(0.536,0.565)
Hoeffding Tree	(0.466,0.547)	(0.506,0.600)	(0.466,0.547)	(0.424,0.516)	(-0.029,0.101)	(0.472,0.544)	(0.538,0.590)
J48	(0.525,0.635)	(0.571,0.670)	(0.525,0.635)	(0.521,0.631)	(0.073,0.261)	(0.536,0.640)	(0.583,0.652)
Logistic	(0.500,0.580)	(0.502,0.600)	(0.500,0.580)	(0.478,0.564)	(-0.043,0.104)	(0.479,0.582)	(0.528,0.614)
Make Density Based Clusterer	(0.418,0.520)	(0.315,0.459)	(0.418,0.520)	(0.315,0.438)	(-0.053,0.052)	(0.469,0.540)	(0.531,0.575)
REPTree	(0.471,0.592)	(0.513,0.622)	(0.471,0.592)	(0.466,0.588)	(-0.033,0.175)	(0.487,0.614)	(0.566,0.648)
Random Forest	(0.492,0.603)	(0.547,0.633)	(0.492,0.603)	(0.488,0.601)	(0.018,0.204)	(0.491,0.621)	(0.595,0.669)
Random Tree	(0.456,0.557)	(0.513,0.602)	(0.456,0.557)	(0.449,0.555)	(-0.035,0.136)	(0.487,0.569)	(0.550,0.600)
SimpleKMeans	(0.447,0.537)	(0.387,0.523)	(0.447,0.537)	(0.366,0.476)	(-0.027,0.057)	(0.469,0.533)	(0.538,0.575)
Simple Logistic	(0.596,0.677)	(0.621,0.697)	(0.596,0.677)	(0.571,0.656)	(0.158,0.300)	(0.569,0.680)	(0.608,0.701)

Table 6.40: Cross validation scores for tree specific models with binary classes.

Model Type	Acc.	Prec.	Recall	F-Meas.	MCC	AUC	AUPRC
Canopy	(0.634,0.725)	(0.418,0.537)	(0.634,0.725)	(0.499,0.614)	(0.000,0.000)	(0.587,0.663)	(0.649,0.722)
Decision Stump	(0.586,0.671)	(0.430,0.555)	(0.586,0.671)	(0.482,0.581)	(-0.014,0.033)	(0.522,0.578)	(0.601,0.652)
EM	(0.592,0.700)	(0.511,0.654)	(0.592,0.700)	(0.517,0.646)	(0.086,0.172)	(0.564,0.622)	(0.629,0.679)
Farthest First	(0.669,0.742)	(0.573,0.668)	(0.669,0.742)	(0.596,0.680)	(0.079,0.178)	(0.533,0.596)	(0.612,0.669)
Filtered Clusterer	(0.616,0.714)	(0.434,0.558)	(0.616,0.714)	(0.495,0.613)	(-0.009,0.051)	(0.502,0.561)	(0.586,0.646)
Hoeffding Tree	(0.548,0.669)	(0.574,0.682)	(0.548,0.669)	(0.511,0.633)	(0.011,0.170)	(0.513,0.615)	(0.608,0.684)
J48	(0.684,0.750)	(0.704,0.764)	(0.684,0.750)	(0.674,0.743)	(0.240,0.380)	(0.637,0.714)	(0.693,0.755)
Logistic	(0.577,0.678)	(0.567,0.704)	(0.577,0.678)	(0.532,0.656)	(0.108,0.239)	(0.564,0.657)	(0.633,0.712)
Make Density Based Clusterer	(0.617,0.712)	(0.427,0.545)	(0.617,0.712)	(0.499,0.611)	(-0.030,0.022)	(0.446,0.519)	(0.568,0.628)
REPTree	(0.657,0.719)	(0.684,0.735)	(0.657,0.719)	(0.649,0.709)	(0.177,0.302)	(0.638,0.720)	(0.710,0.765)
Random Forest	(0.714,0.753)	(0.729,0.762)	(0.714,0.753)	(0.697,0.739)	(0.261,0.362)	(0.731,0.792)	(0.776,0.820)
Random Tree	(0.625,0.696)	(0.662,0.722)	(0.625,0.696)	(0.630,0.696)	(0.132,0.280)	(0.562,0.637)	(0.634,0.685)
SimpleKMeans	(0.597,0.701)	(0.443,0.570)	(0.597,0.701)	(0.491,0.612)	(-0.022,0.056)	(0.461,0.528)	(0.579,0.635)
Simple Logistic	(0.608,0.683)	(0.593,0.672)	(0.608,0.683)	(0.568,0.648)	(0.037,0.141)	(0.560,0.650)	(0.646,0.711)

Table 6.41: Cross validation scores for sfo specific models with binary classes.

6.9 Conclusion

Using the data collected while searching for optimal hyper-parameters and rerunning select iterations of the training algorithms on different numbers of processors, a couple of datasets for selecting which algorithm should perform the best on a given dataset, number of cores, within a time limit, were constructed. A collection of models were then trained using these datasets. While the datasets initially scored quite well when trying to make predictions about the datasets used to construct it, when predictions about datasets that had not been used were made and compared to actual results, the models scored very poorly. One likely cause of the bad results is that the training datasets were too dissimilar to the test datasets. If the training set contained many more datasets with characteristics more closely resembling those of the test datasets, it is possible the predictions would have been more correct. However, PCA analysis of the datasets indicates that the characteristics used to describe the datasets do not do a sufficient job to allow proper classification. Further cross validation indicates that the initial assignment of datasets to training and testing lead to atypical results. Overall it appears that building models for prediction when bagging, boosting, or logistic regression may perform well is possible. It is unclear as to why tree induction is hard to predict.

Chapter 7

Discussion

7.1 Summary of Problem

When a researcher is faced with the task of building a model using machine learning, there is a wide variety of algorithms to choose from, making the problem of selecting an algorithm that will work well for a particular dataset a daunting one. When tackling a problem large enough to warrant using a cluster to solve it, the problem becomes even harder, as different algorithms may exhibit different parallel performance characteristics that are dataset dependant. For extremely large datasets, it will likely soon be the case that running algorithms to completion will no longer be possible, meaning that not only will it be important to pick an algorithm that can produce a good model from the data, but to pick one that will produce a good model in a reasonable amount of time.

While one could try all available algorithms on the task at hand and compare the results to pick the best algorithm, that may be a very time consuming process. Adding to the difficulty is that many algorithms have hyper-parameters that may need to be fine tuned before giving good results. It may be tempting to try each of the algorithms on a smaller subsample of the dataset, but as Perlich *et al.* [7] demonstrated, a subsamping of the dataset may favor a different algorithm than the one that will work on the complete dataset. It would be far better if it were possible to first examine the properties of the dataset and use a model to predict which algorithm would perform best. Additionally, it would be great if a model could also recommend reasonable hyper-parameters to try based on the properties of the dataset.

7.2 Summary of Results

Attempting to build a model to select among all of the algorithms available would be way beyond the scope of this project. That would first involve implementing each of the algorithms, then collecting data on both the model quality and run times over a large number of datasets. Instead a smaller number of MapReduce [49] machine learning algorithms found in the literature were used. The algorithms chosen were Logistic Regression [20] and several forms of Tree Induction [21], including bagging [32] and boosting [33]. The overall task of building a model for algorithm selection was divided into four phases.

In the first phase (Chapter 3), only two smaller datasets were used two perform $2^k r$ analysis on the hyper-parameters for each algorithm. The goal of this phase was to find hyper-parameters that had little effect on either the run time or the quality of models produced. Unfortunately, it turns out all of the available hyper-parameters were deemed useful enough for inclusion in the rest of the study. The $2^k r$ results did provide insight into the relative importance of each hyper-parameter though.

In the second phase (Chapter 4), a search for optimal hyper-parameters for each dataset using each algorithm was performed. The search was conducted using a hill climbing algorithm with random

restarts, using model quality, with run time as a tie breaker, as a fitness function. While the first phase failed to rule out any hyper-parameters, it did determine which one would affect the fitness function more. Using this information, the hill climbing performed its search one hyper-parameter at a time, starting from the one with the largest impact and working its way to the one with the least impact before cycling back to the largest again. During this phase, models were recorded at the end of each iteration of each learning task for use in the next phase.

The third phase (Chapter 5), looked at how changing the number of compute cores affected the run time of each algorithm. Rerunning each algorithm on different numbers of cores can be a very time consuming task and may seem like a waste of resources if the solution to a problem has already been found on one number of cores. Changing the number of compute cores, ideally, should only change the run time for an otherwise identical training task, but still produce the same overall result. Using this notion, a data collection plan was devised where only a portion of the training iterations would be rerun on each number of cores, each starting from one of the models recorded during training in the second phase. In the real world it is not the case that changing the number of cores only affects the run time, in reality, the model produced on a different number of cores may also be different. An experiment was performed to determine if the variation due to changing the number of cores significantly affected the predictions made by the resulting models. In some cases, statistically significant differences were measured in the predictions produced by models trained exclusively on a certain number of cores and the models produced by rerunning iterations from a different number of cores, but overall the differences were determined to be small enough to use this technique. Utilizing these results, data for building an algorithm selection model for parallel machine learning algorithms was performed much more quickly.

The fourth and final phase (Chapter 6), took the run time and model quality results from phases two and three, and combined it with measurements taken of each dataset to create another machine learning dataset. This new dataset had as attributes, the characteristics of each dataset, a number of cores, and a run time limit. The classes in this new dataset are the individual algorithms. Each sample was then assigned its class based on which algorithm was estimated to provide the highest model score on that number of cores within that time limit. A collection of models were then trained on this new dataset using Weka [39]. The model scores reported by Weka upon training these models indicated that the models were very good. To verify the models produced, datasets that had not been used in phases two and three were run through the same hill climbing process for hyper-parameter optimization, on various numbers of cores. The best algorithms for each of these datasets were then compared to predictions made by the models, but the results were abysmal. Even the best models only performed slightly better than random. However when 5x5-fold cross validation was performed to assign datasets randomly among the training and testing datasets, the average scores were much higher for most of the learning problems examined. This indicates that the original splitting between training and testing was problematic.

Overall the results of training models to select which algorithm will perform best, or at least as good as any other algorithm are somewhat mixed. When cross validation was performed to randomly choose which datasets the models are trained on and which are tested against, the results for three of the four algorithms as well as all four in a single model look promising. However, for one algorithm, tree induction, no models were able to predict when it might do well. Also, the original sets of datasets used for training produced models that performed very poorly on the original testing datasets. This indicates that if these models were applied to an arbitrary dataset not included in this study, the predictions may not as trustworthy as the cross validation results may imply.

7.3 Future Work

Given the poor performance predicting the best algorithms for the test datasets, using additional datasets in the construction of the training dataset would be helpful. In particular devising synthetic datasets to fill in gaps in the dataset characteristic space could prove useful.

If additional datasets were to be utilized, optimal hyper-parameters for them will also need to be found. While the hill climbing technique used was effective, there are other search algorithms (e.g., Nelder-Mead [110]) that may be worth exploring, as it may converge on optimal hyper-parameters more quickly. Also, having an additional hold-out set that is not used in either training or testing, that can be used to score the models afterwards for labeling samples may lead to more reliable results.

Assuming adding more datasets does provide models good enough to reliably predict the best algorithm for a dataset, there is still the problem of picking hyper-parameters that are needed to produce a good model using that algorithm.

For this study, only the hyper-parameters for the learning algorithms were examined, however there are many other tunable settings involved in this study, in particular, the settings used for the Java Virtual Machine (JVM) were problematic when running on smaller numbers of cores. Additional work on estimating the optimal parameters for the JVM, as well as algorithm selection and hyper-parameter estimation, could be very useful.

Bibliography

[1] A. L. Guth, "Evolution of the Southern Kenya Rift from Miocene to present with a focus on the Magadi area," Master's thesis, 2007.

[2] ——, "Spatial and temporal evolution of the volcanics and sediments of the Kenya Rift," Ph.D. dissertation, 2013.

[3] L. E. Brown, "NOVEL METHODS FOR VARIABLE SELECTION IN NON-FAITHFUL DOMAINS, UNDERSTANDING SUPPORT VECTOR MACHINES, LEARNING REGIONS OF BAYESIAN NETWORKS, AND PREDICTION UNDER MANIPULATION," Ph.D. dissertation, 2009.

[4] D. Dheeru and E. Karra Taniskidou, "UCI Machine Learning Repository," Tech. Rep., 2017.

[5] (1997) GVU WWW User Survey. [Online]. Available: http://www.cc.gatech.edu/gvu/user_surveys/

[6] L. Sweeney, "Information explosion," *Confidentiality, disclosure, and data access: theory and practical applications for statistical agencies*, pp. 43–74, 2001.

[7] C. Perlich, F. Provost, and J. Simonoff, "Tree induction vs. logistic regression: a learning-curve analysis," *The Journal of Machine Learning Research*, vol. 4, pp. 211–255, 2003.

[8] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Computation*, 1996.

[9] ——, "The Supervised Learning No-Free-Lunch Theorems," in *Soft Computing and Industry*. London: Springer London, 2002, pp. 25–42.

[10] R. King, C. Feng, and A. Sutherland, "Statlog: comparison of classification algorithms on large real-world problems," *Applied Artificial Intelligence an International Journal*, vol. 9, no. 3, pp. 289–333, 1995.

[11] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih, "A Comparison of Prediction Accuracy, Complexity, and Training Time of Thirty-three Old and New Classification Algorithms," *Machine learning*, vol. 40, pp. 203–228, 2000.

[12] P. Kumar, Nitin, V. K. Sehgal, and D. S. Chauhan, "A Benchmark to Select Data Mining Based Classification Algorithms For Business Intelligence And Decision Support Systems," *International Journal of Data Mining & Knowledge Management Process*, vol. 2, no. 5, pp. 25–42, Sep. 2012.

[13] R. Caruana and A. Niculescu-Mizil, "An empirical comparison of supervised learning algorithms," *Proceedings of the 23rd international conference on Machine learning*, pp. 161–168, 2006.

[14] J. Huang and C. X. Ling, "Using AUC and Accuracyin Evaluating Learning Algorithms," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 3, pp. 299–310, Jan. 2005.

[15] J. M. Lobo, A. Jiménez-Valverde, and R. Real, "AUC: a misleading measure of the performance of predictive distribution models," *Global Ecology and Biogeography*, vol. 17, no. 2, pp. 145–151, Mar. 2008.

[16] M. Sokolova, N. Japkowicz, Szpakowicz, and Stan, "Beyond Accuracy, F-score, and ROC: A Family of Discriminant Measures for Performance Evaluation," *AAAI Workshop on Evaluation Methods for Machine Learning*, pp. 24–29, Jun. 2006.

[17] J. Bradford and J. Fortes, "Characterization and parallelization of decision-tree induction," *Journal of Parallel and Distributed Computing*, vol. 61, no. 3, pp. 322–349, 2001.

[18] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun, "Map-reduce for machine learning on multicore," *Advances in Neural Information Processing Systems*, vol. 19, p. 281, 2007.

[19] V. Galtier, O. Pietquin, and S. Vialle, "AdaBoost Parallelization on PC Clusters with Virtual Shared Memory for Fast Feature Selection," *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*, pp. 165–168, 2008.

[20] S. Singh, J. Kubica, S. Larsen, and D. Sorokina, "Parallel Large Scale Feature Selection for Logistic Regression," *Proceedings of the Ninth SIAM International Conference on Data Mining*, 2009.

[21] B. Panda, J. Herbach, S. Basu, and R. Bayardo, "Planet: Massively Parallel Learning of Tree Ensembles with MapReduce," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1426–1437, 2009.

[22] P. Haller and H. Miller, "Parallelizing Machine Learning–Functionally," *The Second Annual Scala Workshop*, Jun. 2011.

[23] E. Y. Chang, H. Bai, and K. Zhu, "Parallel algorithms for mining large-scale rich-media data," *Proceedings of the 17th ACM international conference on Multimedia*, pp. 917–918, 2009.

[24] D. Bailey, "Twelve ways to fool the masses when giving performance results on parallel computers," *Supercomputing Review*, vol. 4, no. 8, pp. 54–55, 1991.

[25] G. Marcus, "Deep Learning: A Critical Appraisal," *arXiv.org*, Jan. 2018.

[26] F. Beaufort, "Beaufort Wind Scale," Tech. Rep., 1805.

[27] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. McLachlan, A. Ng, B. Liu, and P. Yu, "Top 10 algorithms in data mining," *Knowledge and Information Systems*, vol. 14, no. 1, pp. 1–37, 2008.

[28] F. Provost and P. Domingos, "Tree Induction for Probability-Based Ranking," *Machine learning*, vol. 52, no. 3, pp. 199–215, 2003.

[29] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[30] C. Ferri, P. Flach, and J. Hernández-Orallo, "Improving the AUC of probabilistic estimation trees," *Machine Learning: ECML 2003*, pp. 121–132, 2003.

[31] J. Mingers, "An empirical comparison of selection measures for decision-tree induction," *Machine learning*, vol. 3, no. 4, pp. 319–342, 1989.

[32] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[33] J. R. Quinlan, "Boosting first-order learning," *Algorithmic Learning Theory*, pp. 143–155, 1996.

[34] R. E. Schapire and Y. SINGER, "Improved Boosting Algorithms Using Confidence-rated Predictions," 1999.

[35] R. Wolke and H. Schwetlick, "Iteratively reweighted least squares: Algorithms, convergence analysis, and numerical comparisons," *SIAM journal on scientific and statistical computing*, vol. 9, no. 5, pp. 907–921, 1988.

[36] N. Japkowicz, "Why question machine learning evaluation methods," *AAAI Workshop on Evaluation Methods for Machine Learning*, pp. 6–11, 2006.

[37] F. Provost and T. Fawcett, "Analysis and visualization of classifier performance: Comparison under imprecise class and cost distributions," *Proceedings of the third international conference on knowledge discovery and data mining*, pp. 43–48, 1997.

[38] T. Fawcett, "An introduction to ROC analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, Jun. 2006.

[39] Weka. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/

[40] G. W. Brier, "Verification of Forecasts Expressed in Terms of Probability," *Monthly Weather Review*, vol. 78, no. 1, pp. 1–3, 1950.

[41] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Applied statistics*, pp. 191–201, 1992.

[42] J. Mingers, "An empirical comparison of pruning methods for decision tree induction," *Machine learning*, vol. 4, no. 2, pp. 227–243, 1989.

[43] L. Breslow and D. Aha, "Simplifying decision trees: A survey," *The Knowledge Engineering Review*, vol. 12, no. 01, pp. 1–40, 1997.

[44] J. R. Quinlan, "Simplifying decision trees," *International Journal of Human-Computer Studies*, vol. 51, no. 2, pp. 497–510, Aug. 1999.

[45] K. Grabczewski, "Meta-Learning in Decision Tree Induction," *Studies in Computational Intelligence*, vol. 498, 2014.

[46] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine learning, neural and statistical classification," 1994.

[47] S. Ali and K. A. Smith, "On learning algorithm selection for classification," *Applied Soft Computing*, vol. 6, no. 2, pp. 119–138, Jan. 2006.

[48] P. Brazdil, C. G. Giraud-Carrier, C. Soares, and R. Vilalta, "Metalearning - Applications to Data Mining." *Cognitive Technologies*, 2009.

[49] S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Sixth Symposium on Operating System Design and Implementation*, 2004.

[50] The Apache Software Foundation, "Hadoop," http://hadoop.apache.org/, accessed: 14 Apr 2018.

[51] M. Schwarzkopf, D. G. Murray, and S. Hand, "The seven deadly sins of cloud computing research," *HotCloud, June*, 2012.

[52] Mahout. [Online]. Available: http://mahout.apache.org/

[53] HaLoop. [Online]. Available: http://code.google.com/p/haloop/

[54] Twister. [Online]. Available: http://www.iterativemapreduce.org/

[55] J. Ekanayake, H. Li, B. Zhang, T. Gunarathne, S.-H. Bae, J. Qiu, and G. Fox, "Twister: a runtime for iterative MapReduce," in *HPDC '10: Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing.* New York, New York, USA: ACM Request Permissions, Jun. 2010, p. 810.

[56] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, and C. Guestrin, "GraphLab: A Distributed Framework for Machine Learning in the Cloud," *arXiv.org*, Jul. 2011.

[57] J. R. Quinlan, "Bagging, boosting, and C4. 5," *Proceedings of the National Conference on Artificial Intelligence*, pp. 725–730, 1996.

[58] F. Provost and T. Fawcett, "Robust classification for imprecise environments," *Machine learning*, vol. 42, no. 3, pp. 203–231, 2001.

[59] F. Esposito, D. Malerba, G. Semeraro, and J. Kay, "A comparative analysis of methods for pruning decision trees," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 19, no. 5, pp. 476–491, 1997.

[60] P. Domingos and M. Pazzani, "On the optimality of the simple Bayesian classifier under zero-one loss," *Machine learning*, vol. 29, no. 2-3, pp. 103–130, 1997.

[61] R. Caruana and A. Niculescu-Mizil, "Data mining in metric space: an empirical analysis of supervised learning performance criteria," *the 2004 ACM SIGKDD international conference*, pp. 69–78, 2004.

[62] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, and E. Sackinger, "Comparison of learning algorithms for handwritten digit recognition," *International conference on artificial neural networks*, vol. 60, 1995.

[63] G. F. Cooper, C. F. Aliferis, R. Ambrosino, J. Aronis, B. G. Buchanan, R. Caruana, M. J. Fine, C. Glymour, G. Gordon, B. H. Hanusa, J. E. Janosky, C. Meek, T. Mitchell, T. Richardson, and P. Spirtes, "An evaluation of machine-learning methods for predicting pneumonia mortality," *Artificial Intelligence in Medicine*, vol. 9, no. 2, pp. 107–138, Feb. 1997.

[64] MetaL. [Online]. Available: http://www.ofai.at/research/impml/metal/mlee-metal/

[65] A. Kalousis, "Algorithm Selection via Meta Learning ," Ph.D. dissertation, Jan. 2002.

[66] F. Provost and V. Kolluri, "A survey of methods for scaling up inductive algorithms," *Data mining and knowledge discovery*, vol. 3, no. 2, pp. 131–169, 1999.

[67] J. R. A. i. computers and 1976, "The algorithm selection problem," *Elsevier*.

[68] S. M. Abdulrahman, A. Adamu, Y. I. I. J. of, and 2017, "An Overview of the Algorithm Selection Problem," *International Journal of Computer*, vol. 26, pp. 71–98, 2017.

[69] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, *Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms*, ser. combined selection and hyperparameter optimization of classification algorithms. New York, New York, USA: ACM, Aug. 2013.

[70] M. Pelikan, D. E. Goldberg, and E. Cantú-Paz, *BOA: the Bayesian optimization algorithm*. Morgan Kaufmann Publishers Inc., Jul. 1999.

[71] M. R. Smith, L. Mitchell, C. Giraud-Carrier, and T. Martinez, "Recommending Learning Algorithms and Their Associated Hyperparameters," *arXiv.org*, Jul. 2014.

[72] M. Wistuba, N. Schilling, and L. Schmidt-Thieme, "Hyperparameter Search Space Pruning – A New Component for Sequential Model-Based Hyperparameter Optimization," in *Robust Harris-Laplace Detector by Scale Multiplication*. Cham: Springer International Publishing, Aug. 2015, pp. 104–119.

[73] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient Global Optimization of Expensive Black-Box Functions," *Journal of Global Optimization*, vol. 13, no. 4, pp. 455–492, 1998.

[74] S. M. Abdulrahman, P. Brazdil, J. N. van Rijn, and J. Vanschoren, "Algorithm Selection via Meta-learning and Sample-based Active Testing." *MetaSel@PKDD/ECML*, 2015.

[75] S. M. Abdulrahman, P. Brazdil, J. N. Rijn, and J. Vanschoren, "Speeding up algorithm selection using average ranking and active testing by introducing runtime," *Machine learning*, pp. 1–30, Nov. 2017.

[76] M. Cachada, S. M. Abdulrahman, and P. Brazdil, "Combining Feature and Algorithm Hyperparameter Selection using some Metalearning Methods." *AutoML@PKDD/ECML*, 2017.

[77] M. V. Nural, H. Peng, and J. A. Miller, "Using meta-learning for model type selection in predictive big data analytics." *BigData*, 2017.

[78] S. Sanders and C. Giraud-Carrier, "Informing the Use of Hyperparameter Optimization Through Metalearning," in *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, Oct. 2017, pp. 1051–1056.

[79] "MetaSel 2014," pp. 1–66, Aug. 2014.

[80] K. Eggensperger, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Efficient Benchmarking of Hyperparameter Optimizers via Surrogates." *AAAI*, 2015.

[81] A. F. M. Sousa, R. B. C. Prudêncio, T. B. Ludermir, and C. Soares, "Active learning and data manipulation techniques for generating training examples in meta-learning," *Neurocomputing*, vol. 194, no. C, pp. 45–55, Jun. 2016.

[82] T. T. Joy, S. Rana, S. K. G. 0001, and S. Venkatesh, "Hyperparameter tuning for big data using Bayesian optimisation." *ICPR*, pp. 2574–2579, 2016.

[83] D. Horn and B. Bischl, "Multi-objective parameter configuration of machine learning algorithms using model-based optimization." *SSCI*, pp. 1–8, 2016.

[84] J. N. van Rijn and F. Hutter, "Hyperparameter Importance Across Datasets," Oct. 2017.

[85] P. Brazdil and C. Giraud-Carrier, "Metalearning and Algorithm Selection: progress, state of the art and introduction to the 2018 Special Issue," *Machine learning*, vol. 107, no. 1, pp. 1–14, Dec. 2017.

[86] D. J. Barrett and R. E. Silverman, *SSH, the secure shell: the definitive guide.* Beijing: O'Reilly, 2001.

[87] L. Clarke, I. Glendinning, and R. Hempel, "The MPI Message Passing Interface Standard," in *Programming Environments for Massively Parallel Distributed Systems.* Basel: Birkhäuser, Basel, 1994, pp. 213–218.

[88] ActiveMQ. [Online]. Available: http://activemq.apache.org/activemq-542-release.html

[89] Open Grid Scheduler. [Online]. Available: http://gridscheduler.sourceforge.net

[90] S. Lee, "Strange Tales," vol. 1, no. 135, Aug. 1965.

[91] R. K. Pearson, "The problem of disguised missing data," *ACM SIGKDD Explorations Newsletter*, vol. 8, no. 1, pp. 83–92, 2006.

[92] J. Snow. (2018, Jan.) Algorithms are making American inequality worse. [Online]. Available: https://www.technologyreview.com/s/610026/algorithms-are-making-american-inequality-worse/

[93] Y. Wang and M. Kosinski, "Deep neural networks are more accurate than humans at detecting sexual orientation from facial images." *Journal of personality and social psychology*, vol. 114, no. 2, pp. 246–257, Feb. 2018.

[94] J. Dressel and H. Farid, "The accuracy, fairness, and limits of predicting recidivism," *Science Advances*, vol. 4, no. 1, p. eaao5580, Jan. 2018.

[95] S. Desmarais and J. Singh, "Risk assessment instruments validated and implemented in correctional settings in the United States," 2013.

[96] (2017, Oct.) Asking the Right Questions About AI. [Online]. Available: https://medium.com/@yonatanzunger/asking-the-right-questions-about-ai-7ed2d9820c48

[97] J. Radin, ""Digital Natives": How Medical and Indigenous Histories Matter for Big Data," *Osiris*, vol. 32, no. 1, pp. 43–64, Oct. 2017.

[98] R. K. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*, 1st ed. Wiley, Apr. 1991.

[99] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Pearson Education, Feb. 2003.

[100] Homer, The Odyssey.

[101] N. J. Higham, "The Accuracy of Floating Point Summation," *SIAM Journal on Scientific Computing*, vol. 14, no. 4, pp. 783–799, 1993.

[102] S. Collange, D. Defour, S. Graillat, and R. Iakymchuk, "Numerical reproducibility for the parallel reduction on multi- and many-core architectures," *Parallel Computing*, vol. 49, no. C, pp. 83–97, Nov. 2015.

[103] N. Whitehead and A. F.-F. N. A.-C.-F.-P. pdf, "Precision and Performance: Floating Point and IEEE 754 Compliance for NVIDIA GPUs, nVidia technical white paper, 2011 https://developer. nvidia. com ...."

[104] M. Natrella, "NIST/SEMATECH e-Handbook of Statistical Methods," *(July 2010)*, 2010.

[105] P. H. Westfall, "Kurtosis as Peakedness, 1905–2014. R.I.P." *The American Statistician*, vol. 68, no. 3, pp. 191–195, Aug. 2014.

[106] MATLAB, *version 9.1.0 (R2016b)*. Natick, Massachusetts: The MathWorks Inc., 2016.

[107] B. W. Matthews, "Comparison of the predicted and observed secondary structure of T4 phage lysozyme," *Biochimica et Biophysica Acta (BBA) - Protein Structure*, vol. 405, no. 2, pp. 442–451, Oct. 1975.

[108] J. Yan and L. Kurgan, "Consensus-Based Prediction of RNA and DNA Binding Residues from Protein Sequences," in *Pattern Recognition and Machine Intelligence*. Cham: Springer, Cham, Jun. 2015, pp. 501–511.

[109] K. Boyd, K. H. Eng, and C. D. Page, "Area under the Precision-Recall Curve: Point Estimates and Confidence Intervals," in *Robust Harris-Laplace Detector by Scale Multiplication*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 451–466.

[110] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *Comput. J.*, vol. 7, no. 4, pp. 308–313, 1965.

Appendix A

Mathematical Notation

A.1 Variable Names

Below is an alphabetical listing of variables used in this dissertation along with their meaning.

Symbol	Type	Meaning
$\vec{\beta}$	vector	Logistic Regression model vector
C	integer	The number of individual classes
c_i	integer or string	is an individual class label
J	integer	Number of attributes in a dataset
L	set of integers or strings	The set of class labels
N	integer	Total number of samples in a dataset
N_i	integer	Number of samples that reach node v_i
N_{ic}	integer	Number of samples that reach node v_i, belonging to class c
p_i	real	Probability that sample x_i belongs to the target class
v_i	tree node	Unique node in a tree induction model
X	set of vectors	A set of machine samples, also referred to as a dataset.
$\vec{x_i}$	vector	logistic regression sample vector
$x_{i,j}$	scalar or string	Element j in a sample vector $\vec{x_i}$
y_i	integer or string	Label for sample $x_i \in X$
$y_{i,j}$	integer or string	is the label assigned to sample j that reached node i

Table A.1: List of variables

Appendix B

Dataset Characteristics

B.1 Meta-model Training Datasets

Characteristic	Value
$cancorr_1$	0.584
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.584
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	-1
dimensionality	0.475
env	107.862
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	0
nsr	11.073
J	1558
N	3279
$perc_{nin}$	99.8%
$perc_{cont}$	0.2%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	86.0%
$perc_{pos}$	14.0%
sdRatio	1.000
num_{bin}	1555
C	2
num_{cont}	3
num_{ord}	0
num_{miss}	0
num_{pos}	2820
num_{neg}	459

Table B.1: Characteristics of the Ads dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.038	0.944	0.355	0.406	0.362
skew	0.110	7.471	2.165	2.685	2.770
arithMean	2.221	272.216	50.484	77.723	100.511
χ^2	0.000	0.000	0.000	0.000	0.000
concCoeff	0	1	0.000	0.007	0.072
concCoeffC	0	1	0	0.000	0.001
$H(X)$	0.014	4.948	0.042	0.065	0.167
$H(X, C)$	0.584	5.260	0.623	0.644	0.159
geoMean	1.728	109.001	19.743	30.935	40.032
harmMean	-6.967	6.614	-3.612	-1.905	5.401
IQ-range	3.839	371	41	98.440	141.565
kurtosis	1.328	73.030	11.223	23.083	28.116
mad	2.729	180.983	32.782	53.519	67.200
median	1.088	234	42.500	68.292	87.101
$M(X, C)$	0.000	0.584	0.001	0.005	0.020
normP	1	1	1	1	0
pValC	0	0.999	0.086	0.243	0.298
pVal	0.000	0.000	0.000	0.000	0.000
$perc_{90}$	6.110	468	105	148.652	172.649
stdDev	3.714	192.154	50.113	66.199	70.522
trimMean	1.345	280.926	43.702	74.397	104.806
unique	0	0	0	0	0
unique2			2	2	0
variance	13.792	36922.960	2551.395	8526.775	14307.220
zScore	1.768	10.552	7.705	6.961	3.637

Table B.2: Summary characteristics of the Ads dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	33.33	16.67	0.00	0.00	0.00	16.67	16.67	0.00	0.00	16.67	0.00
skew	33.33	0.00	33.33	0.00	0.00	16.67	0.00	0.00	0.00	16.67	0.00
concCoeff	99.00	0.14	0.09	0.09	0.07	0.05	0.04	0.07	0.08	0.36	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	99.23	0.64	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.06	0.00
$H(X, C)$	99.29	0.58	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.06	0.00
kurtosis	33.33	33.33	0.00	0.00	0.00	16.67	0.00	0.00	0.00	16.67	0.00
$M(X, C)$	98.52	1.09	0.26	0.00	0.06	0.00	0.00	0.00	0.00	0.06	0.00
pValC	51.13	9.41	6.80	6.11	5.19	4.37	4.52	3.67	5.08	3.73	0.00
pVal	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.3: Histogram based characteristics of the Ads dataset.

Characteristic	Value
$cancorr_1$	0.941
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.584
$dfrac_2$	NaN
$dfrac_3$	NaN
$dfrac_4$	NaN
$dimensionality$	0.475
env	107.862
$frac_1$	NaN
$frac_2$	NaN
$frac_3$	NaN
$frac_4$	NaN
nsr	11.073
J	1558
N	3279
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	86.0%
$perc_{pos}$	14.0%
$sdRatio$	1
num_{bin}	0
C	2
num_{cont}	1558
num_{ord}	0
num_{miss}	0
num_{pos}	2820
num_{neg}	459

Table B.4: Characteristics of the Ads (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	1	0.006	0.021	0.088
skew	0.110	53.104	12.770	13.184	7.469
arithMean	0	0.767	0.004	0.011	0.033
χ^2	0	0.000	0.000	0.000	0.000
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0.014	4.948	0.042	0.065	0.167
$H(X, C)$	0.584	5.260	0.623	0.644	0.159
geoMean	0	0	0	0	0
harmMean	0	0	0	0	0
IQ-range	0	1	0	0.003	0.055
kurtosis	1.121	2823	164.176	230.944	343.003
mad	0	0.484	0.008	0.018	0.042
median	0	1	0	0.001	0.026
$M(X, C)$	0.000	0.584	0.001	0.005	0.020
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0.000	0.982	0.095	0.155	0.212
$perc_{90}$	0	1	0	0.014	0.114
stdDev	0	0.493	0.062	0.065	0.071
trimMean	0	0.834	0	0.002	0.028
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.243	0.004	0.009	0.021
zScore	0	53.085	7.224	8.606	8.700

Table B.5: Summary characteristics of the Ads (binorm) dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	44.60	0.46	0.16	0.09	0.06	0.06	0.05	0.04	0.05	0.22	54.20
skew	7.12	16.11	27.25	6.51	5.68	0.45	0.00	0.99	0.00	0.71	35.17
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	99.23	0.64	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.06	0.00
$H(X, C)$	99.29	0.58	0.00	0.00	0.00	0.00	0.00	0.06	0.00	0.06	0.00
kurtosis	55.26	7.12	0.29	0.45	0.99	0.00	0.00	0.00	0.00	0.71	35.17
$M(X, C)$	98.52	1.09	0.26	0.00	0.06	0.00	0.00	0.00	0.00	0.06	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	50.32	26.12	11.10	1.54	1.93	1.86	1.73	1.73	2.57	1.09	0.00

Table B.6: Histogram based characteristics of the Ads (binorm) dataset.

Characteristic	Value
$cancorr_1$	0.478
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.796
$dfrac_2$	1
$dfrac_3$	-1
$dfrac_4$	0
$dimensionality$	0.000
env	6.450
$frac_1$	0.000
$frac_2$	1
$frac_3$	0
$frac_4$	0
nsr	17.491
J	14
N	32561
$perc_{nin}$	7.1%
$perc_{cont}$	42.9%
$perc_{ord}$	50.0%
$perc_{miss}$	0.9%
$perc_{neg}$	75.9%
$perc_{pos}$	24.1%
$sdRatio$	1.237
num_{bin}	1
C	2
num_{cont}	6
num_{ord}	7
num_{miss}	4262
num_{pos}	24720
num_{neg}	7841

Table B.7: Characteristics of the ADULT dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	0.127	0.029	0.041	0.033
skew	0.214	18.341	1.077	3.229	5.184
arithMean	9.595	190340.900	49.308	31911.290	73467.170
χ^2	0	0.000	0	0.000	0.000
concCoeff	0	1	0.009	0.139	0.303
concCoeffC	0	1	0	0.015	0.112
$H(X)$	0.512	8.658	1.741	2.283	2.118
$H(X, C)$	0.796	9.432	2.448	2.956	2.170
geoMean	0	160212.800	22.810	26689.930	62292.740
harmMean	0	127415.200	21.094	21132.280	49316.710
IQ-range	0	121422	4	19445.040	45446.700
kurtosis	2.820	608.315	7.533	61.707	172.649
mad	1.724	78354.530	57.273	13386.920	29668.760
median	0	179465	23	29645.420	69198.690
$\bar{M}(X, C)$	0.008	0.796	0.077	0.123	0.200
normP	1	1	1	1	0
pValC	0	0.455	0.000	0.010	0.066
pVal	0	0.088	0	0.015	0.036
$perc_{90}$	0	329144	53.500	55579.920	127759.500
stdDev	2.385	106482.300	162.388	18793.040	40259.190
trimMean	0	181308.900	37.250	30131.260	70112.780
unique	5	41	8	13.857	12.668
unique2			7.500	12.375	12.455
variance	5.689	1.13385e+10	48382.860	1.83891e+09	4.24894e+09
zScore	1.840	42.737	5.370	9.473	11.121

Table B.8: Summary characteristics of the ADULT dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	93.33	6.67	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skew	66.67	8.33	0.00	16.67	0.00	0.00	0.00	0.00	0.00	8.33	0.00
concCoeff	80.25	3.70	2.47	0.00	2.47	1.23	0.00	0.00	0.00	9.88	0.00
concCoeffC	97.53	1.23	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.23	0.00
$H(X)$	42.86	14.29	21.43	14.29	0.00	0.00	0.00	0.00	0.00	7.14	0.00
$H(X, C)$	28.57	28.57	7.14	28.57	0.00	0.00	0.00	0.00	0.00	7.14	0.00
kurtosis	91.67	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.33	0.00
$M(X, C)$	50.00	42.86	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.14	0.00
pValC	97.92	0.00	0.00	0.00	2.08	0.00	0.00	0.00	0.00	0.00	0.00
pVal	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.9: Histogram based characteristics of the ADULT dataset.

Characteristic	Value
$cancorr_1$	0.608
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.810
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	0.000
$dimensionality$	0.003
env	42.998
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	1
nsr	18.500
J	104
N	30162
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	75.1%
$perc_{pos}$	24.9%
$sdRatio$	1
num_{bin}	0
C	2
num_{cont}	104
num_{ord}	0
num_{miss}	0
num_{pos}	22654
num_{neg}	7508

Table B.10: Characteristics of the ADULT (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	0.913	0.006	0.022	0.051
skew	0.295	150.513	10.610	17.867	20.566
arithMean	0	0.932	0.008	0.089	0.193
χ^2	0	0.000	0	0.000	0.000
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0	8.549	0.105	0.367	0.948
$H(X, C)$	0.810	9.337	0.904	1.158	0.939
geoMean	0	0.687	0	0.006	0.054
harmMean	0	0.659	0	0.005	0.051
IQ-range	0	1	0	0.048	0.205
kurtosis	1.138	22657	113.618	742.825	1971.652
mad	0	0.483	0.016	0.077	0.115
median	0	1	0	0.062	0.226
$M(X, C)$	0	0.810	0.001	0.019	0.083
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0	0.880	0.000	0.090	0.193
$perc_{90}$	0	1	0	0.185	0.378
stdDev	0	0.492	0.078	0.140	0.132
trimMean	0	1	0	0.071	0.208
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.242	0.006	0.037	0.058
zScore	0	150.506	10.478	17.787	20.396

Table B.11: Summary characteristics of the ADULT (binorm) dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	90.12	2.83	0.73	0.27	0.19	0.08	0.04	0.04	0.02	0.01	5.68
skew	56.25	20.19	12.02	3.37	3.37	1.44	0.00	0.00	0.00	0.48	2.88
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	92.31	4.81	0.00	0.96	0.96	0.00	0.00	0.00	0.00	0.96	0.00
$H(X, C)$	94.23	2.88	0.00	1.92	0.00	0.00	0.00	0.00	0.00	0.96	0.00
kurtosis	89.42	5.77	0.00	1.44	0.00	0.00	0.00	0.00	0.00	0.48	2.88
$M(X, C)$	94.23	4.81	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.96	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	78.85	3.85	3.85	3.85	0.96	1.92	2.88	1.92	0.96	0.00	0.96

Table B.12: Histogram based characteristics of the ADULT (binorm) dataset.

Characteristic	Value
$cancorr_1$	0.588
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.985
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	0.000
$dimensionality$	0.000
env	42.713
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	1
nsr	62.490
J	54
N	495141
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	57.2%
$perc_{pos}$	42.8%
$sdRatio$	1
num_{bin}	0
C	2
num_{cont}	54
num_{ord}	0
num_{miss}	0
num_{pos}	283301
num_{neg}	211840

Table B.13: Characteristics of the COVERTYPE dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	0.918	0.016	0.046	0.083
skew	0.004	153.641	5.344	16.718	27.074
arithMean	0	3128.645	0.011	161.206	591.430
χ^2	0	0	0	0	0
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0	11.816	0.126	1.464	3.031
$H(X, C)$	0.985	12.780	1.075	2.426	3.034
geoMean	0	3124.587	0	57.983	409.538
harmMean	0	3120.446	0	57.880	408.846
IQ-range	0	2270	0	86.056	364.849
kurtosis	1.000	23606.830	29.564	1006.080	3305.167
mad	0	1336.296	0.022	52.171	220.734
median	0	3146	0	149.454	556.008
$M(X, C)$	0	0.985	0.002	0.023	0.134
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0	0.029	0.000	0.001	0.004
$perc_{90}$	0	5098	0	258.431	953.712
stdDev	0	1618.719	0.105	65.453	275.851
trimMean	0	3136.110	0	155.189	572.669
unique	0	0	0	0	0
unique2			0	0	0
variance	0	2620251	0.011	79673.250	406274.400
zScore	0	153.647	5.239	15.057	25.476

Table B.14: Summary characteristics of the COVERTYPE dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	65.13	5.77	1.71	0.77	0.21	0.24	0.17	0.07	0.03	0.03	25.86
skew	59.26	13.89	4.63	2.78	1.85	0.93	0.00	0.93	0.93	0.93	13.89
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	83.33	0.00	0.00	0.00	1.85	5.56	3.70	1.85	0.00	3.70	0.00
$H(X, C)$	83.33	0.00	0.00	0.00	1.85	5.56	3.70	1.85	0.00	3.70	0.00
kurtosis	78.70	1.85	2.78	0.00	0.00	0.93	0.93	0.00	0.00	0.93	13.89
$M(X, C)$	98.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.85	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	90.74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.26

Table B.15: Histogram based characteristics of the COVERTYPE dataset.

Characteristic	Value
$cancorr_1$	0.588
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.985
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	0.000
$dimensionality$	0.000
env	42.713
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	1
nsr	62.490
J	54
N	495141
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	57.2%
$perc_{pos}$	42.8%
$sdRatio$	1
num_{bin}	0
C	2
num_{cont}	54
num_{ord}	0
num_{miss}	0
num_{pos}	283301
num_{neg}	211840

Table B.16: Characteristics of the COVERTYPE (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	0.918	0.016	0.046	0.083
skew	0.004	153.641	5.344	16.718	27.074
arithMean	0	0.887	0.011	0.122	0.213
χ^2	0	0	0	0	0
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0	11.816	0.126	1.464	3.031
$H(X, C)$	0.985	12.780	1.075	2.426	3.034
geoMean	0	0.877	0	0.016	0.106
harmMean	0	0.873	0	0.016	0.105
IQ-range	0	1	0	0.084	0.226
kurtosis	1.000	23606.830	29.564	1006.080	3305.167
mad	0	0.500	0.022	0.075	0.114
median	0	1	0	0.091	0.223
$M(X, C)$	0	0.985	0.002	0.023	0.134
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0	0.029	0.000	0.001	0.004
$perc_{90}$	0	1	0	0.216	0.373
stdDev	0	0.500	0.077	0.126	0.126
trimMean	0	0.893	0	0.105	0.217
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.250	0.006	0.032	0.056
zScore	0	153.647	5.239	15.057	25.476

Table B.17: Summary characteristics of the COVERTYPE (binorm) dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	65.13	5.77	1.71	0.77	0.21	0.24	0.17	0.07	0.03	0.03	25.86
skew	59.26	13.89	4.63	2.78	1.85	0.93	0.00	0.93	0.93	0.93	13.89
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	83.33	0.00	0.00	0.00	1.85	5.56	3.70	1.85	0.00	3.70	0.00
$H(X, C)$	83.33	0.00	0.00	0.00	1.85	5.56	3.70	1.85	0.00	3.70	0.00
kurtosis	78.70	1.85	2.78	0.00	0.00	0.93	0.93	0.00	0.00	0.93	13.89
$M(X, C)$	98.15	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.85	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	90.74	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.26

Table B.18: Histogram based characteristics of the COVERTYPE (binorm) dataset.

Characteristic	Value
$cancorr_1$	0.483
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.994
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	-1
$dimensionality$	0.023
env	6.215
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	0
nsr	10.174
J	15
N	653
$perc_{nin}$	26.7%
$perc_{cont}$	40.0%
$perc_{ord}$	33.3%
$perc_{miss}$	0.0%
$perc_{neg}$	54.7%
$perc_{pos}$	45.3%
$sdRatio$	1.209
num_{bin}	4
C	2
num_{cont}	6
num_{ord}	5
num_{miss}	0
num_{pos}	357
num_{neg}	296

Table B.19: Characteristics of the CREDIT dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	0.463	0.103	0.132	0.122
skew	0.856	9.169	2.763	3.308	2.484
arithMean	0.667	2009.726	17.767	219.923	568.935
χ^2	0.000	0.000	0.000	0.000	0.000
concCoeff	0	1	0.007	0.131	0.311
concCoeffC	0	0	0	0	0
$H(X)$	0.436	4.248	0.997	1.787	1.271
$H(X,C)$	0.994	5.173	1.988	2.620	1.338
geoMean	0	31.687	0	4.966	11.626
harmMean	0	29.768	0	4.688	10.971
IQ-range	0	1223	10.508	148.835	348.901
kurtosis	3.109	104.645	12.815	25.190	29.241
mad	1.020	2744.967	6.392	276.013	782.358
median	0	210.500	3.740	46.800	73.703
$\bar{M}(X,C)$	0.002	0.994	0.069	0.160	0.257
normP	1	1	1	1	0
pValC	0.000	0.883	0.014	0.146	0.253
pVal	0.000	0.029	0.000	0.005	0.012
$perc_{90}$	2	4150.200	28.502	454.618	1175.424
stdDev	1.958	7660.949	8.559	723.015	2192.424
trimMean	0.211	624.314	16.688	88.370	178.619
unique	3	14	4	6.600	4.827
unique2			3	4.556	4.187
variance	3.835	5.86901e+07	77.918	4928914	1.69308e+07
zScore	3.381	12.791	6.120	7.041	3.114

Table B.20: Summary characteristics of the CREDIT dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	46.67	30.00	13.33	6.67	3.33	0.00	0.00	0.00	0.00	0.00	0.00
skew	33.33	16.67	8.33	16.67	0.00	16.67	0.00	0.00	0.00	8.33	0.00
concCoeff	83.00	5.00	0.00	1.00	0.00	0.00	0.00	0.00	0.00	11.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	26.67	26.67	6.67	6.67	0.00	6.67	6.67	0.00	13.33	6.67	0.00
$H(X,C)$	13.33	26.67	13.33	6.67	6.67	6.67	6.67	0.00	13.33	6.67	0.00
kurtosis	50.00	8.33	0.00	33.33	0.00	0.00	0.00	0.00	0.00	8.33	0.00
$\bar{M}(X,C)$	60.00	20.00	6.67	0.00	6.67	0.00	0.00	0.00	0.00	6.67	0.00
pValC	70.37	5.56	5.56	5.56	1.85	0.00	3.70	1.85	5.56	0.00	0.00
pVal	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.21: Histogram based characteristics of the CREDIT dataset.

Characteristic	Value
$cancorr_1$	0.790
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.994
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	0.000
$dimensionality$	0.066
env	17.343
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	1
nsr	11.680
J	43
N	653
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	54.7%
$perc_{pos}$	45.3%
$sdRatio$	1
num_{bin}	0
C	2
num_{cont}	43
num_{ord}	0
num_{miss}	0
num_{pos}	357
num_{neg}	296

Table B.22: Characteristics of the CREDIT (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	1	0.047	0.080	0.130
skew	0.041	18.894	3.163	4.991	4.825
arithMean	0	0.946	0.077	0.183	0.257
χ^2	0.000	0.000	0.000	0.000	0.000
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0	4.247	0.412	0.727	0.944
$H(X, C)$	0.994	5.172	1.393	1.663	0.924
geoMean	0	0.199	0	0.002	0.021
harmMean	0	0.151	0	0.002	0.016
IQ-range	0	1	0	0.157	0.346
kurtosis	0.988	360	11.260	50.292	86.267
mad	0	0.500	0.109	0.155	0.149
median	0	1	0	0.174	0.369
$M(X, C)$	0	0.994	0.008	0.057	0.166
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0.000	0.994	0.020	0.221	0.324
$perc_{90}$	0	1	0	0.383	0.465
stdDev	0	0.501	0.216	0.229	0.148
trimMean	0	1	0	0.158	0.288
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.251	0.047	0.074	0.076
zScore	0	18.842	3.854	5.320	4.884

Table B.23: Summary characteristics of the CREDIT (binorm) dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	72.59	13.01	2.05	0.89	0.33	0.28	0.22	0.50	0.06	0.94	9.14
skew	27.91	25.58	13.95	8.14	3.49	2.33	5.81	2.33	0.00	5.81	4.65
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	55.81	20.93	11.63	2.33	0.00	0.00	4.65	0.00	2.33	2.33	0.00
$H(X, C)$	60.47	20.93	6.98	2.33	0.00	2.33	2.33	0.00	2.33	2.33	0.00
kurtosis	66.28	10.47	4.65	0.00	8.14	0.00	0.00	0.00	2.33	3.49	4.65
$M(X, C)$	88.37	4.65	2.33	0.00	2.33	0.00	0.00	0.00	0.00	2.33	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	55.81	11.63	4.65	2.33	2.33	2.33	6.98	0.00	6.98	4.65	2.33

Table B.24: Histogram based characteristics of the CREDIT (binorm) dataset.

Characteristic	Value
$cancorr_1$	0.460
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.956
$dfrac_2$	0.000
$dfrac_3$	-1
$dfrac_4$	0
$dimensionality$	0.004
env	3.059
$frac_1$	1
$frac_2$	1
$frac_3$	0
$frac_4$	0
nsr	7.922
J	4
N	1000
$perc_{nin}$	25.0%
$perc_{cont}$	50.0%
$perc_{ord}$	25.0%
$perc_{miss}$	0.0%
$perc_{neg}$	37.7%
$perc_{pos}$	62.3%
$sdRatio$	1.073
num_{bin}	1
C	2
num_{cont}	2
num_{ord}	1
num_{miss}	0
num_{pos}	377
num_{neg}	623

Table B.25: Characteristics of the EXAMPLE dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.150	0.162	0.156	0.156	0.009
skew	0.033	1.381	0.091	0.399	0.656
arithMean	1.669	28.606	15.957	15.548	14.585
χ^2	0.000	0.000	0.000	0.000	0.000
concCoeff	0	1	0	0.222	0.441
concCoeffC	0	0	0	0	0
$H(X)$	0.956	5.558	2.319	2.788	1.973
$H(X, C)$	0.956	6.421	3.175	3.432	2.259
geoMean	0	21.812	10.075	10.491	12.132
harmMean	-7.714	22.066	0	3.588	12.844
IQ-range	3	50.325	24.109	25.386	23.887
kurtosis	1.557	4.804	1.859	2.520	1.539
mad	1.565	25.355	12.000	12.730	12.376
median	1	34.040	15.235	16.378	16.366
$M(X, C)$	0.009	0.956	0.143	0.312	0.435
normP	1	1	1	1	0
pValC	0.068	0.704	0.513	0.449	0.291
pVal	0.000	0.603	0.301	0.301	0.426
$perc_{90}$	4	66.878	36.993	36.216	34.931
stdDev	1.961	28.291	14.440	14.783	14.243
trimMean	1.347	28.698	15.898	15.460	14.650
unique	4	4	4	4	0
unique2			3	3	1.414
variance	3.845	800.375	339.236	370.673	423.967
zScore	1.251	3.228	1.730	1.985	0.863

Table B.26: Summary characteristics of the EXAMPLE dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	0.00	100.00	0.00	0.00	0.00	0.0	0.00	0.00	0.0	0.00	0.00
skew	75.00	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.0	25.00	0.00
concCoeff	77.78	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.0	22.22	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.0	0.00	0.00
$H(X)$	25.00	0.00	25.00	25.00	0.00	0.0	0.00	0.00	0.0	25.00	0.00
$H(X, C)$	25.00	0.00	0.00	25.00	25.00	0.0	0.00	0.00	0.0	25.00	0.00
kurtosis	50.00	25.00	0.00	0.00	0.00	0.0	0.00	0.00	0.0	25.00	0.00
$M(X, C)$	50.00	25.00	0.00	0.00	0.00	0.0	0.00	0.00	0.0	25.00	0.00
pValC	25.00	0.00	0.00	25.00	0.00	0.0	25.00	25.00	0.0	0.00	0.00
pVal	50.00	0.00	0.00	0.00	0.00	0.0	50.00	0.00	0.0	0.00	0.00

Table B.27: Histogram based characteristics of the EXAMPLE dataset.

Characteristic	Value
$cancorr_1$	0.478
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.956
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	0.000
$dimensionality$	0.007
env	5.340
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	1
nsr	8.893
J	7
N	1000
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	37.7%
$perc_{pos}$	62.3%
$sdRatio$	1
num_{bin}	0
C	2
num_{cont}	7
num_{ord}	0
num_{miss}	0
num_{pos}	377
num_{neg}	623

Table B.28: Characteristics of the EXAMPLE (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.002	0.406	0.104	0.151	0.131
skew	0.033	1.557	1.116	0.811	0.568
arithMean	0.194	0.544	0.257	0.339	0.135
χ^2	0.000	0.000	0.000	0.000	0.000
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0.778	5.558	0.841	1.771	1.801
$H(X, C)$	0.956	6.421	1.770	2.548	1.859
geoMean	0	0.379	0	0.027	0.101
harmMean	0	0.139	0	0.010	0.037
IQ-range	0	1	0.938	0.658	0.422
kurtosis	1.025	4.804	2.244	2.251	0.987
mad	0.196	0.496	0.368	0.357	0.089
median	0	1	0	0.192	0.318
$\bar{M}(X, C)$	0.000	0.956	0.008	0.179	0.350
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0.000	1.000	0.034	0.263	0.391
$perc_{90}$	0.500	1	1	0.951	0.134
stdDev	0.245	0.499	0.429	0.403	0.079
trimMean	0.116	0.555	0.196	0.301	0.167
unique	0	0	0	0	0
unique2			0	0	0
variance	0.060	0.249	0.184	0.168	0.059
zScore	0.915	3.228	1.721	1.708	0.538

Table B.29: Summary characteristics of the EXAMPLE (binorm) dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	47.62	21.43	7.14	21.43	2.38	0.0	0.00	0.00	0.00	0.00	0.00
skew	28.57	7.14	0.00	0.00	7.14	0.0	7.14	21.43	21.43	7.14	0.00
concCoeff	100.0	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.0	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00
$H(X)$	71.43	0.00	0.00	14.29	0.00	0.0	0.00	0.00	0.00	14.29	0.00
$H(X, C)$	14.29	57.14	0.00	0.00	14.29	0.0	0.00	0.00	0.00	14.29	0.00
kurtosis	14.29	21.43	7.14	28.57	14.29	0.0	7.14	0.00	0.00	7.14	0.00
$\bar{M}(X, C)$	71.43	0.00	14.29	0.00	0.00	0.0	0.00	0.00	0.00	14.29	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.0	0.00	0.00	0.00	0.00	0.00
pVal	57.14	14.29	0.00	0.00	0.00	0.0	14.29	0.00	0.00	14.29	0.00

Table B.30: Histogram based characteristics of the EXAMPLE (binorm) dataset.

Characteristic	Value
$cancorr_1$	0.173
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.469
$dfrac_2$	1
$dfrac_3$	-1
$dfrac_4$	0
$dimensionality$	0.004
env	3.181
$frac_1$	0.000
$frac_2$	1
$frac_3$	0
$frac_4$	0
nsr	17.102
J	4
N	1000
$perc_{nin}$	25.0%
$perc_{cont}$	50.0%
$perc_{ord}$	25.0%
$perc_{miss}$	0.0%
$perc_{neg}$	10.0%
$perc_{pos}$	90.0%
$sdRatio$	1.006
num_{bin}	1
C	2
num_{cont}	2
num_{ord}	1
num_{miss}	0
num_{pos}	100
num_{neg}	900

Table B.31: Characteristics of the EXAMPLE2 dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.003	0.164	0.083	0.083	0.114
skew	0.147	0.739	0.526	0.484	0.264
arithMean	2.672	30.280	9.188	12.832	13.001
χ^2	0.000	0.013	0.000	0.003	0.006
concCoeff	0	1	0	0.223	0.441
concCoeffC	0	0	0	0	0
$H(X)$	0.469	5.547	2.329	2.669	2.127
$H(X, C)$	0.469	5.953	2.770	2.990	2.272
geoMean	0	23.226	6.402	9.008	11.238
harmMean	-4635.228	1100.438	0	-883.698	2554.253
IQ-range	4	46.550	21.173	23.224	21.467
kurtosis	1.804	2.399	2.061	2.081	0.306
mad	2.286	23.680	12.033	12.508	11.757
median	2	32.540	5.708	11.489	14.314
$\bar{M}(X, C)$	0.004	0.469	0.058	0.147	0.216
normP	1	1	1	1	0
pValC	0.047	0.906	0.416	0.446	0.416
pVal	0.000	0.095	0.047	0.047	0.067
$perc_{90}$	8	65.900	31.283	34.116	30.509
stdDev	2.698	27.243	14.197	14.584	13.672
trimMean	2.340	30.846	7.920	12.257	13.306
unique	4	4	4	4	0
unique2			3	3	1.414
variance	7.280	742.205	331.028	352.885	400.305
zScore	1.637	2.308	1.848	1.910	0.302

Table B.32: Summary characteristics of the EXAMPLE2 dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	50.00	50.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skew	25.00	0.00	0.00	0.00	25.00	0.00	0.00	0.00	25.00	25.00	0.00
concCoeff	77.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	22.22	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	25.00	0.00	0.00	25.00	25.00	0.00	0.00	0.00	0.00	25.00	0.00
$H(X, C)$	25.00	0.00	0.00	25.00	25.00	0.00	0.00	0.00	0.00	25.00	0.00
kurtosis	50.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	25.00	25.00	0.00
$\bar{M}(X, C)$	25.00	50.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	25.00	0.00
pValC	25.00	25.00	0.00	0.00	0.00	0.00	25.00	0.00	0.00	25.00	0.00
pVal	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.33: Histogram based characteristics of the EXAMPLE2 dataset.

Characteristic	Value
$cancorr_1$	0.381
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.469
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	0.000
dimensionality	0.007
env	5.354
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	1
nsr	18.438
J	7
N	1000
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	10.0%
$perc_{pos}$	90.0%
sdRatio	1
num_{bin}	0
C	2
num_{cont}	7
num_{ord}	0
num_{miss}	0
num_{pos}	100
num_{neg}	900

Table B.34: Characteristics of the EXAMPLE2 (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.001	0.548	0.115	0.178	0.162
skew	0.147	2.707	1.064	1.113	0.778
arithMean	0.100	0.630	0.266	0.318	0.164
χ^2	0.000	0.013	0.000	0.001	0.003
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0.469	5.548	0.814	1.703	1.844
$H(X, C)$	0.469	5.953	1.273	2.084	1.887
geoMean	0	0.257	0	0.018	0.069
harmMean	0	0.146	0	0.010	0.039
IQ-range	0	1	0.625	0.580	0.437
kurtosis	1.059	8.439	2.216	3.038	2.158
mad	0.180	0.492	0.317	0.325	0.096
median	0	1	0	0.248	0.365
$\bar{M}(X, C)$	0.004	0.469	0.011	0.088	0.170
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0.000	0.095	0.000	0.016	0.035
$perc_{90}$	0.500	1	1	0.942	0.141
stdDev	0.270	0.496	0.383	0.385	0.075
trimMean	0	0.663	0.208	0.273	0.205
unique	0	0	0	0	0
unique2			0	0	0
variance	0.073	0.246	0.147	0.153	0.057
zScore	0.763	2.985	1.706	1.855	0.608

Table B.35: Summary characteristics of the EXAMPLE2 (binorm) dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	45.24	19.05	4.76	19.05	9.52	2.38	0.00	0.00	0.00	0.00	0.00
skew	14.29	21.43	7.14	28.57	7.14	0.00	0.00	7.14	7.14	7.14	0.00
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	71.43	0.00	0.00	0.00	14.29	0.00	0.00	0.00	0.00	14.29	0.00
$H(X, C)$	14.29	57.14	0.00	0.00	14.29	0.00	0.00	0.00	0.00	14.29	0.00
kurtosis	14.29	57.14	7.14	0.00	0.00	7.14	0.00	7.14	0.00	7.14	0.00
$\bar{M}(X, C)$	57.14	28.57	0.00	0.00	0.00	0.00	0.00	0.00	0.00	14.29	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.36: Histogram based characteristics of the EXAMPLE2 (binorm) dataset.

Characteristic	Value
$cancorr_1$	0.090
$cancorr_2$	0.060
$cancorr_3$	0.021
$cancorr_4$	NaN
$H(C)$	1.726
$dfrac_2$	0.295
$dfrac_3$	0.037
$dfrac_4$	0.000
$dimensionality$	0.007
env	116.698
$frac_1$	0.668
$frac_2$	0.963
$frac_3$	1
$frac_4$	1
nsr	70.969
J	74
N	10108
$perc_{nin}$	66.2%
$perc_{cont}$	6.8%
$perc_{ord}$	27.0%
$perc_{miss}$	0.0%
$perc_{neg}$	47.6%
$perc_{pos}$	52.4%
$sdRatio$	1.000
num_{bin}	49
C	4
num_{cont}	5
num_{ord}	20
num_{miss}	0
num_{pos}	4816
num_{neg}	5292

Table B.37: Characteristics of the IntCensor dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.030	0.321	0.159	0.152	0.084
skew	0.056	2.049	0.472	0.875	0.707
arithMean	-0.159	35.169	3.194	8.470	13.301
χ^2	0	0.000	0.000	0.000	0.000
concCoeff	0	1	0.001	0.023	0.121
concCoeffC	0	0	0	0	0
$H(X)$	0.044	5.715	0.701	1.065	1.087
$H(X, C)$	1.043	6.706	1.693	2.048	1.083
geoMean	0	29.868	0	5.877	12.065
harmMean	-305.540	111.486	0	-10.475	76.257
IQ-range	0.250	22	2.625	5.638	7.809
kurtosis	2.610	7.185	3.389	3.936	1.375
mad	0.354	13.036	1.533	3.498	4.385
median	0	34	3	7.950	12.668
$\bar{M}(X, C)$	0.000	0.998	0.000	0.015	0.116
normP	1	1	1	1	0
pValC	0	1	0.000	0.090	0.223
pVal	0.000	0.004	0.000	0.001	0.002
$perc_{90}$	0.500	56	5	14.050	20.499
stdDev	0.652	16.462	1.936	4.411	5.431
trimMean	-0.039	34.982	3.034	8.305	13.165
unique	3	129	8	20.600	36.535
unique2			2	7.391	21.100
variance	0.425	271.005	3.757	47.474	90.697
zScore	1.536	3.401	2.395	2.407	0.554

Table B.38: Summary characteristics of the IntCensor dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	35.00	27.50	35.00	2.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skew	20.00	20.00	20.00	0.00	0.00	5.00	5.00	10.00	5.00	15.00	0.00
concCoeff	96.86	1.35	0.20	0.04	0.08	0.02	0.02	0.00	0.00	1.43	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	41.89	33.78	2.70	12.16	4.05	2.70	0.00	0.00	0.00	2.70	0.00
$H(X, C)$	41.89	35.14	1.35	12.16	4.05	2.70	0.00	0.00	0.00	2.70	0.00
kurtosis	30.00	30.00	10.00	0.00	10.00	0.00	5.00	5.00	5.00	5.00	0.00
$\bar{M}(X, C)$	98.65	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.35	0.00
pValC	83.19	3.77	2.03	0.87	2.03	1.45	1.45	1.16	1.74	2.32	0.00
pVal	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.39: Histogram based characteristics of the IntCensor dataset.

Characteristic	Value
$cancorr_1$	0.294
$cancorr_2$	0.261
$cancorr_3$	0.240
$cancorr_4$	NaN
$H(C)$	1.726
$dfrac_2$	0.008
$dfrac_3$	0.005
$dfrac_4$	0.004
$dimensionality$	0.046
env	722.759
$frac_1$	0.983
$frac_2$	0.991
$frac_3$	0.996
$frac_4$	1.000
nsr	92.348
J	466
N	10108
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	47.6%
$perc_{pos}$	52.4%
$sdRatio$	1
num_{bin}	0
C	4
num_{cont}	466
num_{ord}	0
num_{miss}	0
num_{pos}	4816
num_{neg}	5292

Table B.40: Characteristics of the IntCensor (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	1	0.010	0.021	0.040
skew	0.022	72.746	8.253	15.163	17.629
arithMean	0	0.941	0.006	0.066	0.148
χ^2	0	0.000	0	0.000	0.000
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0.001	5.715	0.051	0.223	0.435
$H(X, C)$	1.000	6.706	1.050	1.219	0.430
geoMean	0	0	0	0	0
harmMean	0	0	0	0	0
IQ-range	0	1	0	0.068	0.249
kurtosis	0.998	5295	69.164	541.921	1175.697
mad	0	0.500	0.011	0.077	0.129
median	0	1	0	0.030	0.163
$M(X, C)$	0.000	0.998	0.000	0.002	0.046
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0.000	0.992	0.294	0.355	0.324
$perc_{90}$	0	1	0	0.170	0.373
stdDev	0	0.500	0.075	0.131	0.144
trimMean	0	1	0	0.048	0.152
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.250	0.006	0.038	0.064
zScore	0	72.732	5.320	11.954	16.723

Table B.41: Summary characteristics of the IntCensor (binorm) dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	59.48	1.21	0.27	0.10	0.05	0.02	0.01	0.01	0.00	0.01	38.85
skew	36.59	16.09	6.60	5.79	3.92	2.31	2.52	0.59	0.00	3.59	22.00
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	86.05	12.66	0.00	0.64	0.21	0.21	0.00	0.00	0.00	0.21	0.00
$H(X, C)$	86.05	12.88	0.00	0.43	0.21	0.21	0.00	0.00	0.00	0.21	0.00
kurtosis	61.05	6.92	1.02	2.31	3.11	0.00	0.00	0.00	0.00	3.59	22.00
$M(X, C)$	99.79	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.21	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	33.05	10.52	12.02	5.58	5.36	4.08	4.94	6.44	13.95	4.08	0.00

Table B.42: Histogram based characteristics of the IntCensor (binorm) dataset.

Characteristic	Value
$cancorr_1$	NaN
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	2.239
$dfrac_2$	NaN
$dfrac_3$	NaN
$dfrac_4$	NaN
$dimensionality$	0.014
env	130.961
$frac_1$	NaN
$frac_2$	NaN
$frac_3$	NaN
$frac_4$	NaN
nsr	59.110
J	103
N	7219
$perc_{nin}$	79.6%
$perc_{cont}$	0.0%
$perc_{ord}$	20.4%
$perc_{miss}$	0.0%
$perc_{neg}$	66.9%
$perc_{pos}$	33.1%
$sdRatio$	NaN
num_{bin}	82
C	10
num_{cont}	0
num_{ord}	21
num_{miss}	0
num_{pos}	4832
num_{neg}	2387

Table B.43: Characteristics of the IntShopping dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	NaN	NaN	NaN	NaN	NaN
skew	NaN	NaN	NaN	NaN	NaN
arithMean	NaN	NaN	NaN	NaN	NaN
χ^2	NaN	NaN	NaN	NaN	NaN
concCoeff	0	1	0.002	0.018	0.100
concCoeffC	0	1	0	0.000	0.010
$H(X)$	0.002	2.894	0.877	1.028	0.745
$H(X, C)$	0.917	3.787	1.772	1.926	0.728
geoMean	NaN	NaN	NaN	NaN	NaN
harmMean	NaN	NaN	NaN	NaN	NaN
IQ-range	NaN	NaN	NaN	NaN	NaN
kurtosis	NaN	NaN	NaN	NaN	NaN
mad	NaN	NaN	NaN	NaN	NaN
median	NaN	NaN	NaN	NaN	NaN
$\bar{M}(X, C)$	0.000	0.916	0.003	0.017	0.091
normP	NaN	NaN	NaN	NaN	NaN
pValC	NaN	NaN	NaN	NaN	NaN
pVal	NaN	NaN	NaN	NaN	NaN
$perc_{90}$	NaN	NaN	NaN	NaN	NaN
stdDev	NaN	NaN	NaN	NaN	NaN
trimMean	NaN	NaN	NaN	NaN	NaN
unique	4	10	9	8.333	1.983
unique2			2	3.291	2.710
variance	NaN	NaN	NaN	NaN	NaN
zScore	NaN	NaN	NaN	NaN	NaN

Table B.44: Summary characteristics of the IntShopping dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skew	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeff	97.60	1.01	0.31	0.06	0.03	0.00	0.00	0.02	0.02	0.95	0.00
concCoeffC	99.99	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	0.00
$H(X)$	9.71	16.50	23.30	31.07	0.00	0.97	0.97	6.80	4.85	5.83	0.00
$H(X, C)$	9.71	16.50	24.27	30.10	0.97	0.97	0.97	5.83	4.85	5.83	0.00
kurtosis	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$\bar{M}(X, C)$	97.09	1.94	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.97	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.45: Histogram based characteristics of the IntShopping dataset.

Characteristic	Value
$cancorr_1$	1
$cancorr_2$	0.607
$cancorr_3$	0.390
$cancorr_4$	0.328
$H(C)$	2.239
$dfrac_2$	0.145
$dfrac_3$	0.074
$dfrac_4$	0.063
$dimensionality$	0.036
env	308.612
$frac_1$	0.559
$frac_2$	0.703
$frac_3$	0.778
$frac_4$	0.841
nsr	70.303
J	257
N	7219
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	66.9%
$perc_{pos}$	33.1%
$sdRatio$	NaN
num_{bin}	0
C	10
num_{cont}	257
num_{ord}	0
num_{miss}	0
num_{pos}	4832
num_{neg}	2387

Table B.46: Characteristics of the IntShopping (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0	1	0.044	0.066	0.072
skew	0	36.810	2.260	3.201	3.295
arithMean	0	1	0.100	0.184	0.223
χ^2	0	0.000	0.000	0.000	0.000
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0.002	2.239	0.502	0.517	0.329
$H(X, C)$	0.917	2.239	1.413	1.426	0.312
geoMean	0	1	0	0.011	0.102
harmMean	0	1	0	0.011	0.102
IQ-range	0	1	0	0.208	0.403
kurtosis	0.902	1356.997	6.113	22.253	61.013
mad	0	0.500	0.175	0.201	0.166
median	0	1	0	0.111	0.313
$M(X, C)$	0.000	0.916	0.001	0.007	0.058
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0	0.831	0.000	0.039	0.118
$perc_{90}$	0	1	0.500	0.501	0.496
stdDev	0	0.506	0.296	0.272	0.165
trimMean	0	1	0	0.146	0.246
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.256	0.088	0.101	0.084
zScore	0	36.810	2.184	2.926	3.186

Table B.47: Summary characteristics of the IntShopping (binorm) dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	69.72	13.12	3.19	0.90	0.31	0.10	0.04	0.02	0.01	0.01	12.6
skew	61.28	15.21	4.79	1.71	0.62	0.23	0.16	0.08	0.00	0.04	15.8
concCoeff	100.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
concCoeffC	100.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
$H(X)$	22.96	22.57	20.23	17.12	16.73	0.00	0.00	0.00	0.00	0.39	0.0
$H(X, C)$	15.18	12.84	13.62	9.73	12.84	12.06	11.67	11.67	0.00	0.39	0.0
kurtosis	81.95	1.13	0.54	0.31	0.08	0.08	0.00	0.00	0.00	0.04	15.8
$M(X, C)$	99.22	0.39	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.39	0.0
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.0
pVal	90.27	1.95	2.72	2.33	1.17	0.00	0.78	0.39	0.39	0.00	0.0

Table B.48: Histogram based characteristics of the IntShopping (binorm) dataset.

Characteristic	Value
$cancorr_1$	0.998
$cancorr_2$	0.998
$cancorr_3$	0.992
$cancorr_4$	0.819
$H(C)$	1.487
$dfrac_2$	0.192
$dfrac_3$	0.001
$dfrac_4$	0.000
$dimensionality$	0.000
env	11.348
$frac_1$	0.806
$frac_2$	0.998
$frac_3$	0.999
$frac_4$	1.000
nsr	5.364
J	41
N	4898431
$perc_{nin}$	9.8%
$perc_{cont}$	82.9%
$perc_{ord}$	7.3%
$perc_{miss}$	0.0%
$perc_{neg}$	80.1%
$perc_{pos}$	19.9%
$sdRatio$	1
num_{bin}	4
C	23
num_{cont}	34
num_{ord}	3
num_{miss}	0
num_{pos}	3925650
num_{neg}	972781

Table B.49: Characteristics of the INTRUSION dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0	1	0.138	0.277	0.311
skew	0	1035.383	2.816	27.398	112.114
arithMean	0	3922088	0.011	6700.727	141859
χ^2	0	0.516	0	0.007	0.061
concCoeff	0	1	0.000	0.202	0.359
concCoeffC	0	0	0	0	0
$H(X)$	0	4.649	0.477	0.834	1.071
$H(X,C)$	0.719	4.718	1.153	1.422	0.941
geoMean	0	915043.100	0	1273.709	32776.900
harmMean	0	53662.980	0	103.154	1949.763
IQ-range	0	1996785	0	3046.362	72519.840
kurtosis	0.200	1072020	11.464	14753.060	106825.700
mad	0	1844056	0.001	5255.835	77589.020
median	0	5150830	0	6718.202	184199.700
$\bar{M}(X,C)$	0	0.719	0.057	0.131	0.193
normP	1	1	1	1	0
pValC	0	0.999	0.000	0.190	0.359
pVal	0	0.000	0	0.000	0.000
$perc_{90}$	0	5153616	0	8192.629	187078.700
stdDev	0	2.03835e+07	0.014	49287.090	887815.100
trimMean	0	4258217	0	5702.269	152319.900
unique	3	70	11	28	36.592
unique2			2	13.143	25.288
variance	0	4.15489e+14	0.000	7.89637e+11	1.64136e+13
zScore	0	1035.382	0.304	16.971	93.392

Table B.50: Summary characteristics of the INTRUSION dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	13.74	3.84	2.64	2.27	1.78	1.37	1.25	0.95	0.90	2.46	68.79
skew	48.98	2.56	0.00	0.00	0.00	0.13	0.26	0.13	0.00	0.38	47.57
concCoeff	71.88	0.00	4.69	1.56	3.13	1.56	1.56	1.56	1.56	12.50	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	48.78	19.51	7.32	12.20	4.88	2.44	0.00	0.00	2.44	2.44	0.00
$H(X,C)$	48.78	21.95	9.76	4.88	4.88	2.44	2.44	0.00	2.44	2.44	0.00
kurtosis	50.38	0.00	0.00	0.00	0.26	0.26	0.00	0.00	0.00	0.38	48.72
$M(X,C)$	58.54	9.76	9.76	2.44	4.88	7.32	0.00	0.00	2.44	4.88	0.00
pValC	73.95	1.26	1.26	0.42	0.42	0.84	1.68	2.94	2.94	11.34	2.94
pVal	97.06	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.94

Table B.51: Histogram based characteristics of the INTRUSION dataset.

Characteristic	Value
$cancorr_1$	0.999
$cancorr_2$	0.998
$cancorr_3$	0.995
$cancorr_4$	0.907
$H(C)$	1.487
$dfrac_2$	0.431
$dfrac_3$	0.090
$dfrac_4$	0.006
$dimensionality$	0.000
env	31.477
$frac_1$	0.463
$frac_2$	0.893
$frac_3$	0.983
$frac_4$	0.989
nsr	5.433
J	122
N	4898431
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	80.1%
$perc_{pos}$	19.9%
$sdRatio$	1
num_{bin}	0
C	23
num_{cont}	122
num_{ord}	0
num_{miss}	0
num_{pos}	3925650
num_{neg}	972781

Table B.52: Characteristics of the INTRUSION (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0	1	0.002	0.087	0.207
skew	0	1035.383	7.284	39.765	109.856
arithMean	0	1	0	0.068	0.223
χ^2	0	0.516	0	0.005	0.053
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0	4.649	0.003	0.304	0.718
$H(X, C)$	0.719	4.718	0.722	0.976	0.628
geoMean	0	1	0	0.034	0.173
harmMean	0	1	0	0.031	0.167
IQ-range	0	1	0	0.028	0.147
kurtosis	0.200	1072020	82.576	14350.580	98336.380
mad	0	0.500	0	0.023	0.080
median	0	1	0	0.065	0.240
$M(X, C)$	0	0.719	0.000	0.047	0.121
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0	0.997	0	0.016	0.125
$perc_{90}$	0	1	0	0.095	0.282
stdDev	0	0.707	0	0.033	0.098
trimMean	0	1	0	0.067	0.229
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.500	0	0.011	0.041
zScore	0	1035.382	0	11.932	65.401

Table B.53: Summary characteristics of the INTRUSION (binorm) dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	11.07	0.68	0.35	0.30	0.22	0.19	0.15	0.11	0.11	0.31	86.51
skew	26.48	1.10	0.04	0.14	0.04	0.11	0.14	0.04	0.00	0.18	71.74
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	79.51	9.84	3.28	4.10	0.82	0.82	0.00	0.00	0.82	0.82	0.00
$H(X, C)$	80.33	11.48	2.46	0.82	1.64	0.82	0.82	0.00	0.82	0.82	0.00
kurtosis	27.37	0.18	0.00	0.07	0.14	0.07	0.00	0.00	0.00	0.18	71.99
$M(X, C)$	83.61	4.92	3.28	2.46	1.64	2.46	0.00	0.00	0.00	1.64	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	97.54	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.64	0.82

Table B.54: Histogram based characteristics of the INTRUSION (binorm) dataset.

Characteristic	Value
$cancorr_1$	NaN
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.999
$dfrac_2$	NaN
$dfrac_3$	NaN
$dfrac_4$	NaN
$dimensionality$	0.003
env	4.032
$frac_1$	NaN
$frac_2$	NaN
$frac_3$	NaN
$frac_4$	NaN
nsr	4.626
J	21
N	8124
$perc_{nin}$	19.0%
$perc_{cont}$	0.0%
$perc_{ord}$	81.0%
$perc_{miss}$	0.0%
$perc_{neg}$	48.2%
$perc_{pos}$	51.8%
$sdRatio$	1
num_{bin}	4
C	2
num_{cont}	0
num_{ord}	17
num_{miss}	0
num_{pos}	3916
num_{neg}	4208

Table B.55: Characteristics of the MUSHROOMS dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	NaN	NaN	NaN	NaN	NaN
skew	NaN	NaN	NaN	NaN	NaN
arithMean	NaN	NaN	NaN	NaN	NaN
χ^2	NaN	NaN	NaN	NaN	NaN
concCoeff	-0.000	1	0.064	0.154	0.233
concCoeffC	0	1	0	0.009	0.058
$H(X)$	0	3.030	1.399	1.394	0.850
$H(X, C)$	0.999	3.613	2.126	2.145	0.789
geoMean	NaN	NaN	NaN	NaN	NaN
harmMean	NaN	NaN	NaN	NaN	NaN
IQ-range	NaN	NaN	NaN	NaN	NaN
kurtosis	NaN	NaN	NaN	NaN	NaN
mad	NaN	NaN	NaN	NaN	NaN
median	NaN	NaN	NaN	NaN	NaN
$\bar{M}(X, C)$	0	0.999	0.202	0.248	0.272
normP	NaN	NaN	NaN	NaN	NaN
pValC	NaN	NaN	NaN	NaN	NaN
pVal	NaN	NaN	NaN	NaN	NaN
$perc_{90}$	NaN	NaN	NaN	NaN	NaN
stdDev	NaN	NaN	NaN	NaN	NaN
trimMean	NaN	NaN	NaN	NaN	NaN
unique	3	12	6	6.529	2.809
unique2			4	5.667	3.104
variance	NaN	NaN	NaN	NaN	NaN
zScore	NaN	NaN	NaN	NaN	NaN

Table B.56: Summary characteristics of the MUSHROOMS dataset.

Char.	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skew	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeff	56.82	17.98	7.02	5.17	1.86	1.24	0.83	0.41	0.00	4.75	3.93
concCoeffC	96.07	2.27	1.03	0.21	0.00	0.00	0.00	0.00	0.00	0.21	0.21
$H(X)$	14.29	4.76	9.52	14.29	9.52	9.52	14.29	14.29	4.76	4.76	0.00
$H(X, C)$	19.05	4.76	9.52	14.29	9.52	9.52	19.05	0.00	4.76	9.52	0.00
kurtosis	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$\bar{M}(X, C)$	33.33	14.29	28.57	4.76	9.52	0.00	0.00	0.00	0.00	9.52	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.57: Histogram based characteristics of the MUSHROOMS dataset.

Characteristic	Value
$cancorr_1$	0.998
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.999
$dfrac_2$	0.009
$dfrac_3$	0.006
$dfrac_4$	0.003
$dimensionality$	0.014
env	17.528
$frac_1$	0.962
$frac_2$	0.971
$frac_3$	0.977
$frac_4$	0.980
nsr	6.193
J	115
N	8124
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	48.2%
$perc_{pos}$	51.8%
$sdRatio$	1
num_{bin}	0
C	2
num_{cont}	115
num_{ord}	0
num_{miss}	0
num_{pos}	3916
num_{neg}	4208

Table B.58: Characteristics of the MUSHROOMS (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.000	1	0.076	0.132	0.155
skew	0.059	31.253	2.762	4.920	5.606
arithMean	0	0.998	0.046	0.161	0.240
χ^2	0	0.000	0	0.000	0
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0	1.000	0.308	0.410	0.351
$H(X, C)$	0.999	1.991	1.259	1.352	0.313
geoMean	0	0	0	0	0
harmMean	0	0	0	0	0
IQ-range	0	1	0	0.187	0.391
kurtosis	1.002	978.247	8.633	56.478	134.730
mad	0	0.500	0.066	0.155	0.175
median	0	1	0	0.096	0.295
$M(X, C)$	0	0.999	0.014	0.057	0.122
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0	0.144	0.000	0.003	0.017
$perc_{90}$	0	1	0	0.383	0.487
stdDev	0	0.500	0.182	0.213	0.180
trimMean	0	1	0	0.136	0.256
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.250	0.033	0.077	0.088
zScore	0	31.269	1.853	3.647	5.085

Table B.59: Summary characteristics of the MUSHROOMS (binorm) dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	33.04	11.37	4.86	2.75	1.63	0.91	0.50	0.23	0.16	0.31	44.23
skew	40.00	13.48	10.87	3.48	2.17	2.17	0.00	1.74	0.00	0.87	25.22
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	31.30	7.83	10.43	9.57	3.48	3.48	6.09	6.96	7.83	13.04	0.00
$H(X, C)$	32.17	8.70	12.17	6.96	7.83	7.83	5.22	6.96	6.09	6.09	0.00
kurtosis	64.35	4.78	2.61	0.43	1.74	0.00	0.00	0.00	0.00	0.87	25.22
$M(X, C)$	84.35	7.83	5.22	0.87	0.00	0.87	0.00	0.00	0.00	0.87	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	93.04	0.87	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.09

Table B.60: Histogram based characteristics of the MUSHROOMS (binorm) dataset.

Characteristic	Value
$cancorr_1$	0.551
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.933
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	0.000
$dimensionality$	0.010
env	4.579
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	1
nsr	16.077
J	8
N	768
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	65.1%
$perc_{pos}$	34.9%
$sdRatio$	1.074
num_{bin}	0
C	2
num_{cont}	8
num_{ord}	0
num_{miss}	0
num_{pos}	500
num_{neg}	268

Table B.61: Characteristics of the Pima dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.016	0.573	0.123	0.160	0.130
skew	0.001	2.499	0.890	1.067	0.837
arithMean	0.430	141.258	33.166	46.503	42.682
χ^2	0.000	0.034	0.000	0.003	0.009
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0.933	4.427	3.916	3.480	1.124
$H(X,C)$	0.933	5.149	4.772	4.209	1.390
geoMean	0	35.516	0	4.113	11.137
harmMean	0	34.052	0	3.930	10.665
IQ-range	0.333	167.500	16	31.922	44.383
kurtosis	1.962	12.458	5.511	5.916	2.928
mad	0.221	107.043	10.526	19.523	28.737
median	0	140	28.525	38.255	40.679
$M(X,C)$	0.044	0.933	0.101	0.204	0.299
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0.000	0.072	0.000	0.014	0.027
$perc_{90}$	0.803	273.100	45.940	77.871	79.054
stdDev	0.299	138.689	13.279	25.799	38.103
trimMean	0.384	141.664	32.602	43.517	40.746
unique	0	0	0	0	0
unique2			0	0	0
variance	0.089	19234.670	178.922	2026.664	5175.626
zScore	1.808	6.830	3.420	3.900	1.438

Table B.62: Summary characteristics of the Pima dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	46.43	25.00	14.29	5.36	7.14	1.79	0.00	0.00	0.00	0.00	0.00
skew	25.00	6.25	18.75	0.00	6.25	0.00	12.50	18.75	6.25	6.25	0.00
concCoeff	100.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	12.50	0.00	0.00	0.00	0.00	12.50	0.00	12.50	37.50	25.00	0.00
$H(X,C)$	12.50	0.00	0.00	0.00	0.00	0.00	12.50	0.00	12.50	62.50	0.00
kurtosis	25.00	6.25	18.75	6.25	0.00	25.00	12.50	0.00	0.00	6.25	0.00
$M(X,C)$	75.00	12.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	12.50	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	100.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.63: Histogram based characteristics of the Pima dataset.

Characteristic	Value
$cancorr_1$	0.551
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.933
$dfrac_2$	0.000
$dfrac_3$	0.000
$dfrac_4$	0.000
$dimensionality$	0.010
env	4.570
$frac_1$	1
$frac_2$	1
$frac_3$	1
$frac_4$	1
nsr	16.033
J	8
N	768
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	65.1%
$perc_{pos}$	34.9%
$sdRatio$	1
num_{bin}	0
C	2
num_{cont}	8
num_{ord}	0
num_{miss}	0
num_{pos}	500
num_{neg}	268

Table B.64: Characteristics of the Pima (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	0.016	0.573	0.123	0.160	0.130
skew	0.001	2.499	0.890	1.067	0.837
arithMean	0.081	0.710	0.246	0.329	0.199
χ^2	0.000	0.034	0.000	0.003	0.009
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0.933	4.427	3.916	3.478	1.123
$H(X, C)$	0.933	5.149	4.772	4.207	1.390
geoMean	0	0.148	0	0.009	0.037
harmMean	0	0.097	0	0.006	0.024
IQ-range	0.120	0.382	0.199	0.212	0.085
kurtosis	1.962	12.458	5.511	5.916	2.928
mad	0.078	0.185	0.123	0.121	0.030
median	0	0.704	0.243	0.305	0.224
$\bar{M}(X, C)$	0.044	0.933	0.102	0.204	0.299
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0.000	0.072	0.000	0.014	0.027
$perc_{90}$	0.219	0.923	0.494	0.527	0.188
stdDev	0.108	0.220	0.160	0.157	0.031
trimMean	0.058	0.712	0.236	0.319	0.213
unique	0	0	0	0	0
unique2			0	0	0
variance	0.012	0.048	0.026	0.026	0.010
zScore	1.808	6.830	3.420	3.900	1.438

Table B.65: Summary characteristics of the Pima (binorm) dataset.

Char.	Histogram (%)										NaN
$\overline{\lvert\rho\rvert}$	46.43	25.00	14.29	5.36	7.14	1.79	0.00	0.00	0.00	0.00	0.00
skew	25.00	6.25	18.75	0.00	6.25	0.00	12.50	18.75	6.25	6.25	0.00
concCoeff	100.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	12.50	0.00	0.00	0.00	0.00	12.50	0.00	12.50	37.50	25.00	0.00
$H(X, C)$	12.50	0.00	0.00	0.00	0.00	0.00	12.50	0.00	12.50	62.50	0.00
kurtosis	25.00	6.25	18.75	6.25	0.00	25.00	12.50	0.00	0.00	6.25	0.00
$\bar{M}(X, C)$	75.00	12.50	0.00	0.00	0.00	0.00	0.00	0.00	0.00	12.50	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	100.0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.66: Histogram based characteristics of the Pima (binorm) dataset.

Characteristic	Value
$cancorr_1$	NaN
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.153
$dfrac_2$	NaN
$dfrac_3$	NaN
$dfrac_4$	NaN
dimensionality	72.997
env	32.782
$frac_1$	NaN
$frac_2$	NaN
$frac_3$	NaN
$frac_4$	NaN
nsr	9.949
J	139351
N	1909
$perc_{nin}$	100.0%
$perc_{cont}$	0.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	97.8%
$perc_{pos}$	2.2%
sdRatio	1
num_{bin}	139351
C	2
num_{cont}	0
num_{ord}	0
num_{miss}	0
num_{pos}	1867
num_{neg}	42

Table B.67: Characteristics of the THROMBIN dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	NaN	NaN	NaN	NaN	NaN
skew	NaN	NaN	NaN	NaN	NaN
arithMean	NaN	NaN	NaN	NaN	NaN
χ^2	NaN	NaN	NaN	NaN	NaN
concCoeff	NaN	NaN	NaN	NaN	NaN
concCoeffC	NaN	NaN	NaN	NaN	NaN
$H(X)$	0.006	0.917	0.026	0.051	0.065
$H(X, C)$	0.153	1.070	0.175	0.199	0.064
geoMean	NaN	NaN	NaN	NaN	NaN
harmMean	NaN	NaN	NaN	NaN	NaN
IQ-range	NaN	NaN	NaN	NaN	NaN
kurtosis	NaN	NaN	NaN	NaN	NaN
mad	NaN	NaN	NaN	NaN	NaN
median	NaN	NaN	NaN	NaN	NaN
$\bar{M}(X, C)$	0.000	0.153	0.003	0.005	0.005
normP	NaN	NaN	NaN	NaN	NaN
pValC	NaN	NaN	NaN	NaN	NaN
pVal	NaN	NaN	NaN	NaN	NaN
$perc_{90}$	NaN	NaN	NaN	NaN	NaN
stdDev	NaN	NaN	NaN	NaN	NaN
trimMean	NaN	NaN	NaN	NaN	NaN
unique	0	0	0	0	0
unique2			2	2	0
variance	NaN	NaN	NaN	NaN	NaN
zScore	NaN	NaN	NaN	NaN	NaN

Table B.68: Summary characteristics of the THROMBIN dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skew	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeff	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	85.74	9.95	2.75	0.99	0.37	0.13	0.03	0.02	0.01	0.00	0.00
$H(X, C)$	85.54	10.08	2.81	0.99	0.38	0.14	0.03	0.02	0.01	0.00	0.00
kurtosis	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$\bar{M}(X, C)$	94.99	4.66	0.33	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Table B.69: Histogram based characteristics of the THROMBIN dataset.

Characteristic	Value
$cancorr_1$	0.975
$cancorr_2$	NaN
$cancorr_3$	NaN
$cancorr_4$	NaN
$H(C)$	0.153
$dfrac_2$	NaN
$dfrac_3$	NaN
$dfrac_4$	NaN
$dimensionality$	72.997
env	32.782
$frac_1$	NaN
$frac_2$	NaN
$frac_3$	NaN
$frac_4$	NaN
nsr	9.949
J	139351
N	1909
$perc_{nin}$	0.0%
$perc_{cont}$	100.0%
$perc_{ord}$	0.0%
$perc_{miss}$	0.0%
$perc_{neg}$	97.8%
$perc_{pos}$	2.2%
$sdRatio$	NaN
num_{bin}	0
C	2
num_{cont}	139351
num_{ord}	0
num_{miss}	0
num_{pos}	1867
num_{neg}	42

Table B.70: Characteristics of the THROMBIN (binorm) dataset.

Characteristic	Minimum	Maximum	Median	Mean	Std. Dev.
$\overline{\lvert\rho\rvert}$	NaN	NaN	NaN	NaN	NaN
skew	0	43.209	6.481	11.968	11.816
arithMean	0	0.786	0.024	0.035	0.055
χ^2	0	0.000	0.000	0.000	0.000
concCoeff	0	0	0	0	0
concCoeffC	0	0	0	0	0
$H(X)$	0.006	0.917	0.026	0.051	0.065
$H(X, C)$	0.153	1.070	0.175	0.199	0.064
geoMean	0	0	0	0	0
harmMean	0	0	0	0	0
IQ-range	0	1	0	0.012	0.110
kurtosis	0.897	1870	45	284.806	516.475
mad	0	0.500	0.046	0.062	0.086
median	0	1	0	0.000	0.017
$M(X, C)$	0.000	0.153	0.003	0.005	0.005
normP	1	1	1	1	0
pValC	NaN	NaN	NaN	NaN	NaN
pVal	0.000	0.999	0.000	0.059	0.184
$perc_{90}$	0	1	0	0.101	0.288
stdDev	0	0.506	0.154	0.137	0.114
trimMean	0	0.853	0	0.009	0.039
unique	0	0	0	0	0
unique2			0	0	0
variance	0	0.256	0.024	0.032	0.044
zScore	0	43.186	6.326	10.340	11.696

Table B.71: Summary characteristics of the THROMBIN (binorm) dataset.

Characteristic	Histogram (%)										NaNs
$\overline{\lvert\rho\rvert}$	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
skew	18.58	35.90	6.46	4.68	6.18	3.25	0.00	4.38	0.00	6.72	13.84
concCoeff	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
concCoeffC	100.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
$H(X)$	85.74	9.95	2.75	0.99	0.37	0.13	0.03	0.02	0.01	0.00	0.00
$H(X, C)$	85.54	10.08	2.81	0.99	0.38	0.14	0.03	0.02	0.01	0.00	0.00
kurtosis	61.88	7.41	2.51	3.25	4.38	0.00	0.00	0.00	0.00	6.72	13.84
$M(X, C)$	94.99	4.66	0.33	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pValC	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
pVal	88.94	2.09	1.67	1.02	0.84	0.88	0.83	1.18	1.96	0.59	0.00

Table B.72: Histogram based characteristics of the THROMBIN (binorm) dataset.

Algorithm Selection Confusion Matrices

C.1 Confusion Matrices for Training Datasets

a	b	c	d	
0	366	0	0	a = bagged
0	3945	0	0	b = boosted
0	2009	0	0	c = sfo
0	1981	0	0	d = tree

Table C.1: Confusion matrix for the Canopy model applied to the training dataset.

a	b	c	d	
0	253	103	10	a = bagged
0	2752	931	262	b = boosted
0	26	1983	0	c = sfo
0	981	587	413	d = tree

Table C.2: Confusion matrix for the Decision Stump model applied to the training dataset.

a	b	c	d	
0	205	117	44	a = bagged
0	2969	910	66	b = boosted
0	218	1791	0	c = sfo
0	902	932	147	d = tree

Table C.3: Confusion matrix for the EM model applied to the training dataset.

a	b	c	d	
0	189	87	90	a = bagged
0	3075	797	73	b = boosted
0	1183	826	0	c = sfo
0	1132	678	171	d = tree

Table C.4: Confusion matrix for the Farthest First model applied to the training dataset.

a	b	c	d	
0	366	0	0	a = bagged
0	3832	0	113	b = boosted
0	2009	0	0	c = sfo
0	1700	0	281	d = tree

Table C.5: Confusion matrix for the Hierarchical Clusterer model applied to the training dataset.

a	b	c	d	
76	140	103	47	a = bagged
88	2490	986	381	b = boosted
0	0	2009	0	c = sfo
164	484	708	625	d = tree

Table C.6: Confusion matrix for the Hoeffding Tree model applied to the training dataset.

a	b	c	d	
151	70	36	109	a = bagged
25	3337	284	299	b = boosted
24	83	1876	26	c = sfo
26	248	267	1440	d = tree

Table C.7: Confusion matrix for the IBk model applied to the training dataset.

a	b	c	d	
294	10	38	24	a = bagged
12	3785	39	109	b = boosted
26	27	1915	41	c = sfo
17	128	61	1775	d = tree

Table C.8: Confusion matrix for the J48 model applied to the training dataset.

a	b	c	d	
197	67	41	61	a = bagged
128	3162	59	596	b = boosted
27	27	1904	51	c = sfo
17	271	43	1650	d = tree

Table C.9: Confusion matrix for the LMT model applied to the training dataset.

a	b	c	d	
0	164	109	93	a = bagged
0	2694	987	264	b = boosted
0	11	1998	0	c = sfo
0	612	871	498	d = tree

Table C.10: Confusion matrix for the LWL model applied to the training dataset.

a	b	c	d	
1	157	103	105	a = bagged
0	3104	401	440	b = boosted
0	282	1580	147	c = sfo
5	619	181	1176	d = tree

Table C.11: Confusion matrix for the Logistic model applied to the training dataset.

a	b	c	d	
260	14	49	43	a = bagged
13	3753	44	135	b = boosted
23	34	1901	51	c = sfo
36	153	58	1734	d = tree

Table C.12: Confusion matrix for the REPTree model applied to the training dataset.

a	b	c	d	
299	22	26	19	a = bagged
15	3795	29	106	b = boosted
16	28	1928	37	c = sfo
12	95	31	1843	d = tree

Table C.13: Confusion matrix for the Random Forest model applied to the training dataset.

a	b	c	d	
294	24	24	24	a = bagged
17	3765	36	127	b = boosted
28	37	1901	43	c = sfo
20	102	37	1822	d = tree

Table C.14: Confusion matrix for the Random Tree model applied to the training dataset.

a	b	c	d	
0	220	112	34	a = bagged
0	3103	712	130	b = boosted
0	565	1435	9	c = sfo
0	1052	728	201	d = tree

Table C.15: Confusion matrix for the SimpleKMeans model applied to the training dataset.

a	b	c	d	
1	161	103	101	a = bagged
0	3039	447	459	b = boosted
0	250	1602	157	c = sfo
5	637	200	1139	d = tree

Table C.16: Confusion matrix for the Simple Logistic model applied to the training dataset.

C.2 Confusion Matrices for Test Datasets

a	b	c	d	
0	337	0	0	a = bagged
0	188	0	0	b = boosted
0	221	0	0	c = sfo
0	645	0	0	d = tree

Table C.17: Confusion matrix for the Canopy model applied to the test dataset.

a	b	c	d	
0	212	125	0	a = bagged
0	136	52	0	b = boosted
0	7	214	0	c = sfo
0	319	326	0	d = tree

Table C.18: Confusion matrix for the Decision Stump model applied to the test dataset.

a	b	c	d	
0	305	32	0	a = bagged
0	121	67	0	b = boosted
0	209	12	0	c = sfo
0	603	42	0	d = tree

Table C.19: Confusion matrix for the EM model applied to the test dataset.

a	b	c	d	
0	198	0	139	a = bagged
0	170	0	18	b = boosted
0	85	0	136	c = sfo
0	264	0	381	d = tree

Table C.20: Confusion matrix for the Farthest First model applied to the test dataset.

a	b	c	d	
0	337	0	0	a = bagged
0	188	0	0	b = boosted
0	221	0	0	c = sfo
0	645	0	0	d = tree

Table C.21: Confusion matrix for the Hierarchical Clusterer model applied to the test dataset.

a	b	c	d	
205	62	25	45	a = bagged
72	72	7	37	b = boosted
77	40	39	65	c = sfo
498	54	42	51	d = tree

Table C.22: Confusion matrix for the Hoeffding Tree model applied to the test dataset.

a	b	c	d	
0	156	22	159	a = bagged
0	83	8	97	b = boosted
0	71	78	72	c = sfo
0	294	121	230	d = tree

Table C.23: Confusion matrix for the IBk model applied to the test dataset.

a	b	c	d	
1	113	38	185	a = bagged
4	121	5	58	b = boosted
0	144	28	49	c = sfo
0	259	76	310	d = tree

Table C.24: Confusion matrix for the J48 model applied to the test dataset.

a	b	c	d	
337	0	0	0	a = bagged
188	0	0	0	b = boosted
221	0	0	0	c = sfo
645	0	0	0	d = tree

Table C.25: Confusion matrix for the KStar model applied to the test dataset.

a	b	c	d	
2	134	126	75	a = bagged
4	61	63	60	b = boosted
0	57	123	41	c = sfo
10	290	168	177	d = tree

Table C.26: Confusion matrix for the LMT model applied to the test dataset.

a	b	c	d	
0	68	269	0	a = bagged
0	127	61	0	b = boosted
0	0	221	0	c = sfo
0	45	600	0	d = tree

Table C.27: Confusion matrix for the LWL model applied to the test dataset.

a	b	c	d	
0	0	337	0	a = bagged
0	0	188	0	b = boosted
0	0	221	0	c = sfo
0	0	645	0	d = tree

Table C.28: Confusion matrix for the Logistic model applied to the test dataset.

a	b	c	d	
7	102	52	176	a = bagged
1	134	9	44	b = boosted
0	107	38	76	c = sfo
16	292	174	163	d = tree

Table C.29: Confusion matrix for the REPTree model applied to the test dataset.

a	b	c	d	
0	108	26	203	a = bagged
0	137	5	46	b = boosted
0	118	37	66	c = sfo
0	425	59	161	d = tree

Table C.30: Confusion matrix for the Random Forest model applied to the test dataset.

a	b	c	d	
110	154	0	73	a = bagged
9	147	1	31	b = boosted
0	211	6	4	c = sfo
99	458	14	74	d = tree

Table C.31: Confusion matrix for the Random Tree model applied to the test dataset.

a	b	c	d	
0	312	25	0	a = bagged
0	181	7	0	b = boosted
0	182	39	0	c = sfo
0	603	42	0	d = tree

Table C.32: Confusion matrix for the SimpleKMeans model applied to the test dataset.

a	b	c	d	
0	30	25	282	a = bagged
0	66	7	115	b = boosted
0	0	39	182	c = sfo
0	20	42	583	d = tree

Table C.33: Confusion matrix for the Simple Logistic model applied to the test dataset.

C.3 Confusion Matrices for Binary Class Training Datasets

a	b	c	d	
0	366	0	0	a = bagged
0	3945	0	0	b = boosted
0	2009	0	0	c = sfo
0	1981	0	0	d = tree

Table C.34: Confusion matrix for the Canopy model applied to the training (binary classes) dataset.

a	b	c	d	
0	253	103	10	a = bagged
0	2752	931	262	b = boosted
0	26	1983	0	c = sfo
0	981	587	413	d = tree

Table C.35: Confusion matrix for the Decision Stump model applied to the training (binary classes) dataset.

a	b	c	d	
0	193	115	58	a = bagged
0	3086	709	150	b = boosted
0	263	1746	0	c = sfo
0	979	767	235	d = tree

Table C.36: Confusion matrix for the EM model applied to the training (binary classes) dataset.

a	b	c	d	
0	166	123	77	a = bagged
0	3057	817	71	b = boosted
0	1010	999	0	c = sfo
0	1143	704	134	d = tree

Table C.37: Confusion matrix for the Farthest First model applied to the training (binary classes) dataset.

a	b	c	d	
0	366	0	0	a = bagged
0	3832	0	113	b = boosted
0	2009	0	0	c = sfo
0	1700	0	281	d = tree

Table C.38: Confusion matrix for the Hierarchical Clusterer model applied to the training (binary classes) dataset.

a	b	c	d	
119	122	103	22	a = bagged
190	2438	986	331	b = boosted
0	0	2009	0	c = sfo
258	343	708	672	d = tree

Table C.39: Confusion matrix for the Hoeffding Tree model applied to the training (binary classes) dataset.

a	b	c	d	
136	70	39	121	a = bagged
28	3318	293	306	b = boosted
24	84	1876	25	c = sfo
24	246	279	1432	d = tree

Table C.40: Confusion matrix for the IBk model applied to the training (binary classes) dataset.

a	b	c	d	
294	10	38	24	a = bagged
12	3782	40	111	b = boosted
26	27	1915	41	c = sfo
17	128	61	1775	d = tree

Table C.41: Confusion matrix for the J48 model applied to the training (binary classes) dataset.

a	b	c	d	
121	113	41	91	a = bagged
43	3337	69	496	b = boosted
30	22	1895	62	c = sfo
11	410	51	1509	d = tree

Table C.42: Confusion matrix for the LMT model applied to the training (binary classes) dataset.

a	b	c	d	
0	155	108	103	a = bagged
0	2623	987	335	b = boosted
0	6	2003	0	c = sfo
0	490	899	592	d = tree

Table C.43: Confusion matrix for the LWL model applied to the training (binary classes) dataset.

a	b	c	d	
1	157	103	105	a = bagged
0	3104	401	440	b = boosted
0	282	1580	147	c = sfo
5	619	181	1176	d = tree

Table C.44: Confusion matrix for the Logistic model applied to the training (binary classes) dataset.

a	b	c	d	
259	15	49	43	a = bagged
13	3754	44	134	b = boosted
23	35	1900	51	c = sfo
34	153	58	1736	d = tree

Table C.45: Confusion matrix for the REPTree model applied to the training (binary classes) dataset.

a	b	c	d	
298	21	28	19	a = bagged
12	3790	30	113	b = boosted
15	28	1930	36	c = sfo
13	90	30	1848	d = tree

Table C.46: Confusion matrix for the Random Forest model applied to the training (binary classes) dataset.

a	b	c	d	
297	16	26	27	a = bagged
12	3785	35	113	b = boosted
23	39	1903	44	c = sfo
16	111	33	1821	d = tree

Table C.47: Confusion matrix for the Random Tree model applied to the training (binary classes) dataset.

a	b	c	d	
0	194	129	43	a = bagged
0	2810	806	329	b = boosted
0	494	1310	205	c = sfo
0	804	776	401	d = tree

Table C.48: Confusion matrix for the SimpleKMeans model applied to the training (binary classes) dataset.

a	b	c	d	
2	161	103	100	a = bagged
0	3058	431	456	b = boosted
0	271	1579	159	c = sfo
5	631	195	1150	d = tree

Table C.49: Confusion matrix for the Simple Logistic model applied to the training (binary classes) dataset.

C.4 Confusion Matrices for Binary Class Test Datasest

a	b	c	d	
0	337	0	0	a = bagged
0	188	0	0	b = boosted
0	221	0	0	c = sfo
0	645	0	0	d = tree

Table C.50: Confusion matrix for the Canopy model applied to the test (binary classes) dataset.

a	b	c	d	
0	212	125	0	a = bagged
0	136	52	0	b = boosted
0	7	214	0	c = sfo
0	319	326	0	d = tree

Table C.51: Confusion matrix for the Decision Stump model applied to the test (binary classes) dataset.

a	b	c	d	
0	218	119	0	a = bagged
0	133	55	0	b = boosted
0	161	60	0	c = sfo
0	466	179	0	d = tree

Table C.52: Confusion matrix for the EM model applied to the test (binary classes) dataset.

a	b	c	d	
0	120	217	0	a = bagged
0	26	162	0	b = boosted
0	0	221	0	c = sfo
0	225	420	0	d = tree

Table C.53: Confusion matrix for the Farthest First model applied to the test (binary classes) dataset.

a	b	c	d	
0	337	0	0	a = bagged
0	188	0	0	b = boosted
0	221	0	0	c = sfo
0	645	0	0	d = tree

Table C.54: Confusion matrix for the Hierarchical Clusterer model applied to the test (binary classes) dataset.

a	b	c	d	
0	258	0	79	a = bagged
0	161	5	22	b = boosted
0	179	0	42	c = sfo
0	554	67	24	d = tree

Table C.55: Confusion matrix for the Hoeffding Tree model applied to the test (binary classes) dataset.

a	b	c	d	
0	45	202	90	a = bagged
0	91	34	63	b = boosted
0	17	183	21	c = sfo
0	150	391	104	d = tree

Table C.56: Confusion matrix for the IBk model applied to the test (binary classes) dataset.

a	b	c	d	
7	111	71	148	a = bagged
12	104	19	53	b = boosted
0	41	132	48	c = sfo
6	238	194	207	d = tree

Table C.57: Confusion matrix for the J48 model applied to the test (binary classes) dataset.

a	b	c	d	
86	119	15	117	a = bagged
25	100	2	61	b = boosted
0	92	22	107	c = sfo
140	302	37	166	d = tree

Table C.58: Confusion matrix for the LMT model applied to the test (binary classes) dataset.

a	b	c	d	
0	68	269	0	a = bagged
0	127	61	0	b = boosted
0	0	221	0	c = sfo
0	45	600	0	d = tree

Table C.59: Confusion matrix for the LWL model applied to the test (binary classes) dataset.

a	b	c	d	
86	0	147	104	a = bagged
83	0	88	17	b = boosted
85	0	0	136	c = sfo
109	0	240	296	d = tree

Table C.60: Confusion matrix for the Logistic model applied to the test (binary classes) dataset.

a	b	c	d	
6	97	174	60	a = bagged
2	99	58	29	b = boosted
15	49	139	18	c = sfo
28	214	320	83	d = tree

Table C.61: Confusion matrix for the REPTree model applied to the test (binary classes) dataset.

a	b	c	d	
0	129	28	180	a = bagged
0	119	4	65	b = boosted
0	156	27	38	c = sfo
0	435	22	188	d = tree

Table C.62: Confusion matrix for the Random Forest model applied to the test (binary classes) dataset.

a	b	c	d	
57	74	38	168	a = bagged
4	116	11	57	b = boosted
0	105	51	65	c = sfo
75	273	81	216	d = tree

Table C.63: Confusion matrix for the Random Tree model applied to the test (binary classes) dataset.

a	b	c	d	
0	316	21	0	a = bagged
0	179	9	0	b = boosted
0	189	32	0	c = sfo
0	553	92	0	d = tree

Table C.64: Confusion matrix for the SimpleKMeans model applied to the test (binary classes) dataset.

a	b	c	d	
115	186	5	31	a = bagged
26	99	5	58	b = boosted
0	209	12	0	c = sfo
219	378	27	21	d = tree

Table C.65: Confusion matrix for the Simple Logistic model applied to the test (binary classes) dataset.

Appendix D

Digital Appendix Index

The digital appendix for this document contains three files:
`Datasets.tar.gz`: Contains dataset files that were created for this project.
`ExternalAppendices.pdf`: Appendices that were too long to include in the actual document.
`SisyphiniML.tar.gz`: Contains all of the source code developed for this project.