

# CS6735 Programming Assignment

Jacob Smith

December 12, 2017

# 1 Introduction

This report details the experimental study conducted for the CS6735 programming assignment. Six machine learning algorithms were implemented. Each was evaluated on five datasets from the UCI Machine Learning Repository using 10 times 5-fold cross validation. The technical details of the implementation are described in this report and the results are presented and analyzed.

## 2 Objectives

1. Implement the following machine learning algorithms:
  - (a) ID3
  - (b) AdaBoost on ID3
  - (c) Random Forest
  - (d) Naïve Bayes
  - (e) AdaBoost on Naïve Bayes
  - (f) K-Nearest Neighbors
2. Evaluate the algorithms on the following datasets:
  - (a) Breast Cancer Data
  - (b) Car Data
  - (c) E Coli. Data
  - (d) Mushroom Data
3. Create a report of the experimental study, including:
  - (a) A description of the learning algorithms
  - (b) A description of the datasets
  - (c) The technical details of the implementation
  - (d) The design of the program
  - (e) A report and analysis of the experimental results

### 3 Algorithms

For this study, we used six different algorithms; however, only the five unique algorithms were implemented. The AdaBoost on ID3 and AdaBoost on Naïve Bayes algorithms did not need to be implemented independently.

#### 3.1 ID3

The Iterative Dichotomiser 3 (ID3) algorithm is used to efficiently create decision trees [3]. Initially, the algorithm as described by Thomas Mitchell [2] was implemented. ID3 begins by creating a root node with a set of categorical, labeled data  $S$ . The node iteratively finds the attribute  $a$  which introduces the largest information gain and the node splits on that attribute. Subsets  $S_1, \dots, S_n$  are used to create  $n$  children nodes where  $n$  is the amount of distinct values of attribute  $a$ . This continues recursively until one of several terminating conditions are met. For example, the process ends if there are no more possible subsets, the split is not statistically significant or the entropy of a node is 0. Furthermore, two early stopping conditions were implemented including a max tree level and a minimum number of samples per node.

After the ID3 algorithm was implemented, further enhancements were made to allow for continuous data to be used. These modifications were based off the C4.5 algorithm which was also developed by Ross Quinlan [1]. Given a continuous attribute  $a$ , we temporarily sort the data  $S$ . After this occurs, we iteratively determine the threshold to split the data that maximizes the information gain. To test potential thresholds, we find two sequential, distinct values of some attribute  $a$  and split at the midway point of the two values. This enhancement removes the need for discretization of data. Furthermore, this enhancement chooses the optimal threshold to split the data which improves classification accuracy.

#### 3.2 Naïve Bayes

The Naïve Bayes algorithm was also implemented as described by Thomas Mitchell [2]. This algorithm is a probabilistic classifier named due to its assumption of conditional independence amongst each attribute:

$$P(a_1, \dots, a_n | v_j) = \prod_{i=1}^n P(a_i | v_j)$$

and its application of Bayes theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Naïve Bayes is a fast learning algorithm that receives relatively good results despite its large bias. The algorithm inherently works with both categorical and continuous data. For this assignment, a Gaussian distribution was assumed for all continuous attributes. The final algorithm implemented was:

$$\hat{y} = \operatorname{argmax}_{k \in \{1, \dots, K\}} P(v_k) \prod_{i=1}^m p(a_i|v_k) \prod_{j=m+1}^n P(a_j|v_k)$$

where  $P$  is the probability,  $p$  is the Gaussian probability density function and  $K$  is the number of distinct classes. In this equation, the first  $m$  attributes are continuous whereas the later are categorical. During training, the dataset  $S$  is split by class into subsets  $S_1, \dots, S_n$  where  $n$  is the number of distinct classes. For each subset, we estimate the class probability  $P(v_k)$ . Subsequently, for each categorical attribute value  $a_i$ , we estimate  $P(a_i|v_k)$ . We are not able to immediately calculate  $p(a_i|v_k)$  for continuous attributes; however, we are able to determine the mean and standard deviation.

### 3.3 AdaBoost

AdaBoost is a boosting algorithm that was first introduced in 1995 by Freund Schapire. This type of ensemble learning uses a set of weak learners to reduce bias and become a strong learner. It solves many of the difficulties associated with previous boosting algorithms [4].

The key insight of the AdaBoost algorithm is to maintain a weight value  $w_i$  for every sample within the dataset  $S$ . This weight increases when its sample is misclassified and decreases when its sample is correctly classified. The AdaBoost algorithm is easily implemented on weak learners who are parameterized by sample weights; however, neither Naïve Bayes or ID3 natively implement weighted training. Therefore, an alternative implementation was used where a subset of the training data is selected with replacement based on their associated weights. For example, if the weight values for two samples are  $< 1, 2 >$ , the second sample would be twice as likely to be chosen each time.

Initially, the weights are set to  $\frac{1}{n}$  where  $n$  is the amount of samples in the dataset  $S$ . For a predefined number of iterations, a new weak learner is trained on the a weighted subset of the training data. Predictions are made for each training sample and the error is calculated. A new value  $\alpha$  is calculated that represents the weight that will associated with the weak classifier.  $\alpha$  is also used to update the weights of each sample. To make

classification, there is a weighted vote between the weak learners. The sign of the summed votes is used to make the final classification as either  $+1$  or  $-1$ .

One limitation of the AdaBoost algorithm is its assumption of a binary classification. One solution includes training  $n$  AdaBoost models where  $n$  is the number of distinct classes. For each different model, we reduce one class to  $+1$  and the others to  $-1$ . To make classifications, we query each model and choose the class associated with the most positive prediction. This enhancement was initially implemented; however, it proved incredibly difficult to train these models with large datasets due to time and memory constraints.

As an alternative, the SAMME AdaBoost algorithm [5] was implemented. This algorithm which naturally manages multiclass classification problems. The only difference to the learning process is the addition of  $\log(|v| - 1)$ , where  $|v|$  is the number of classes, to the  $\alpha$  value. Typically, the error must be less than  $1/2$  in order for  $\alpha$  to be positive. With the SAMME modification,  $\alpha$  is positive when  $1 - error > 1/n$  [5].

### 3.3.1 AdaBoost on Naïve Bayes

Naïve Bayes is an inherently weak learner due to its assumption of conditional independence amongst attributes. This fact introduces a large amount of bias which makes Naïve Bayes an ideal candidate for the AdaBoost algorithm.

### 3.3.2 AdaBoost on ID3

The ID3 algorithm does not introduce a large amount of bias and may overfit if allowed to train to completion. Therefore, to implement AdaBoost on ID3, the depth of the trees was limited.

## 3.4 Random Forest

The random forest algorithm is a bagging algorithm which uses an ensemble of strong learners to reduce the effects of overfitting to the training set. This method contains similarities to the AdaBoost algorithm as described in section 3.3; however, no weights are used for random forest. For a predefined number of iterations, a subset of the of the training data is selected with replacement. The new training set is subsequently used to train a decision tree which is added to the *forest*. To make a classification, each tree in the forest votes and the most common class becomes the predicted class. As an extension of the random forest algorithm, weights may be given to each decision tree based on their classification accuracy; however, this extension was not implemented.

### 3.5 K-Nearest Neighbors

The k-nearest neighbors algorithm is an instance based learning algorithm which makes predictions by finding the  $k$  nearest neighbors of a sample  $x$  and using the most common class as the predicted class for that sample. This algorithm can be used with both categorical and continuous data. For this implementation, Euclidian distance was used for continuous variables and Hamming distance was used for categorical data. A weighted classifier may be used; however, this extension was not added as the  $k$  values were usually set to 1.

## 4 Data

Each algorithm described in section 3 was implemented on the following five datasets from the UCI Machine Learning Repository.

### 4.1 Breast Cancer Data

**Title:** Wisconsin Diagnostic Breast Cancer

**Date:** November 1995

**Number of Samples:** 569

**Number of Attributes:** 10

**Attribute Information:**

Description	Values	Type
Sample Code Number	id number	Discrete
Clump Thickness	1 - 10	Discrete
Uniformity of Cell Size	1 - 10	Discrete
Uniformity of Cell Shape	1 - 10	Discrete
Marginal Adhesion	1 - 10	Discrete
Single Epithelial Cell Size	1 - 10	Discrete
Bare Nuclei	1 - 10	Discrete
Bland Chromatin	1 - 10	Discrete
Normal Nucleoli	1 - 10	Discrete
Mitoses	1 - 10	Discrete

**Class Target:** Neoplasm

**Class Information:**

Description	Value	Count	Percentage
Benign	2	357	65.5%
Malignant	4	212	34.5%

**Missing Attributes:** 16

## 4.2 Car Data

**Title:** Car Evaluation Database

**Date:** June 1997

**Number of Samples:** 1728

**Number of Attributes:** 6

**Attribute Information:**

Description	Values	Type
buying	v-high, high, med, low	Discrete
maint	v-high, high, med, low	Discrete
doors	2, 3, 4, 5-more	Discrete
persons	2, 4, more	Discrete
lug_boot	small, med, big	Discrete
safety	low, med, high	Discrete

**Class Target:** Car Quality

**Class Information:**

Description	Value	Count	Percentage
Unacceptable	unacc	1210	70.023%
Acceptable	acc	384	22.222%
Good	good	69	3.993%
Very Good	v-good	65	3.762%

**Missing Attributes:** None

## 4.3 E. Coli Data

**Title:** Protein Localization Sites

**Date:** September 1997

**Number of Samples:** 336

**Number of Attributes:** 8

**Attribute Information:**

Description	Values	Type
Sequence Name	Accession Number	Discrete
mcg	0.00 - 0.89	Continuous
gvh	0.16 - 1.00	Continuous
lip	0.48 - 1.00	Discrete
chg	0.50 - 1.00	Discrete
aac	0.00 - 0.88	Continuous
alm1	0.03 - 1.00	Continuous
alm2	0.00 - 0.99	Continuous

**Class Target:** Localization Site

**Class Information:**

Description	Value	Count	Percentage
Cytoplasm	cp	143	42.56%
Inner Membrane	im	77	22.92%
Periplasm	pp	52	15.48%
Inner Membrane Uncleavable	imU	35	10.42%
Outer Membrane	om	20	5.95%
Outer Membrane	omL	5	1.49%
Inner Membrane Lipoprotein	imL	2	0.59%
Inner Membrane Cleavable	imS	2	0.59%

**Missing Attributes:** None

**4.4 Letter Recognition Data**

**Title:** Letter Image Recognition Data

**Date:** January 1991

**Number of Samples:** 20000

**Number of Attributes:** 16

**Attribute Information:**



<b>Description</b>	<b>Values</b>	<b>Type</b>
x-box	0 - 9	Continuous
y-box	0 - 9	Continuous
width	0 - 9	Continuous
high	0 - 9	Continuous
onpix	0 - 9	Continuous
x-bar	0 - 9	Continuous
y-bar	0 - 9	Continuous
x2bar	0 - 9	Continuous
y2bar	0 - 9	Continuous
xybar	0 - 9	Continuous
x2ybr	0 - 9	Continuous
xy2br	0 - 9	Continuous
x-ege	0 - 9	Continuous
xegvy	0 - 9	Continuous
y-ege	0 - 9	Continuous
yegvx	0 - 9	Continuous

**Class Target:** Letter

**Class Information:**

<b>Description</b>	<b>Value</b>	<b>Count</b>	<b>Percentage</b>
Letter A	A	789	0.04%
Letter B	B	766	0.04%
Letter C	C	736	0.04%
Letter D	D	805	0.04%
Letter E	E	768	0.04%
Letter F	F	775	0.04%
Letter G	G	773	0.04%
Letter H	H	734	0.04%
Letter I	I	755	0.04%
Letter J	J	747	0.04%
Letter K	K	739	0.04%
Letter L	L	761	0.04%
Letter M	M	792	0.04%
Letter N	N	783	0.04%
Letter O	O	753	0.04%
Letter P	P	803	0.04%
Letter Q	Q	783	0.04%
Letter R	R	758	0.04%
Letter S	S	748	0.04%
Letter T	T	796	0.04%
Letter U	U	813	0.04%
Letter V	V	764	0.04%
Letter W	W	752	0.04%
Letter X	X	787	0.04%
Letter Y	Y	786	0.04%
Letter Z	Z	734	0.04%

**Missing Attributes:** None

## 4.5 Mushroom Data

**Title:** Mushroom Database

**Date:** 27 April 1987

**Number of Samples:** 8124

**Number of Attributes:** 22

**Attribute Information:**

<b>Description</b>	<b>Values</b>	<b>Type</b>
cap-shape	b,c,x,f,k,s	Discrete
cap-surface	f,g,y,s	Discrete
cap-color	n,b,c,g,r,p,u,e,w,y	Discrete
bruises?	t,f	Discrete
odor	a,l,c,y,f,m,n,p,s	Discrete
gill-attachment	a,d,f,n	Discrete
gill-spacing	c,w,d	Discrete
gill-size	b,n	Discrete
gill-color	k,n,b,h,g,r,o,p,u,e,w,y	Discrete
stalk-shape	e,t	Discrete
stalk-root	b,c,u,e,z,r,?	Discrete
stalk-surface-above-ring	f,y,k,s	Discrete
stalk-surface-below-ring	f,y,k,s	Discrete
stalk-color-above-ring	n,b,c,g,o,p,e,w,y	Discrete
stalk-color-below-ring	n,b,c,g,o,p,e,w,y	Discrete
veil-type	p,u	Discrete
veil-color	n,o,w,y	Discrete
ring-number	n,o,t	Discrete
ring-type	c,e,f,l,n,p,s,z	Discrete
spore-print-color	k,n,b,h,r,o,u,w,y	Discrete
population	a,c,n,s,v,y	Discrete
habitat	g,l,m,p,u,w,d	Discrete

**Class Target:** Edibility

**Class Information:**

<b>Description</b>	<b>Value</b>	<b>Count</b>	<b>Percentage</b>
Edible	e	4208	51.80%
Poisonous	p	3916	48.20%

**Missing Attributes:** 2480 (all for attribute #11)

## 5 Technical Details

Missing attribute values within the breast cancer and mushroom datasets were simply treated as distinct categories. Attempts were initially made to replace the missing values; however, no accuracy improvements were observed. Furthermore, no discretization occurred as all base algorithms were able to handle continuous data. For each dataset, any sort of identification number or code attributes were removed. Within the E. Coli dataset, the lip and chg categorical attributes were also removed as they provided minimal

information. Furthermore, any string values were converted to integers. For example, the “v-high”, “high”, “med” and “low” attribute values of the car dataset would have been converted to 0, 1, 2, 3 respectively.

The following subsections outline the hyperparameters used for each algorithm. Various testing logic was used to find the optimal hyperparameters each dataset and algorithm.

### 5.1 ID3

Dataset	Minimum # of Samples
Breast Cancer	1
Car	1
E. Coli	2
Letter Recognition	1
Mushroom	1

### 5.2 Naïve Bayes

Dataset	Distribution
Breast Cancer	None
Car	None
E. Coli	Gaussian
Letter Recognition	Gaussian
Mushroom	None

### 5.3 AdaBoost on ID3

Dataset	Max Level	# of Learners	Proportion of Samples
Breast Cancer	1	100	0.50
Car	4	200	0.30
E. Coli	3	100	0.20
Letter Recognition	8	300	0.25
Mushroom	3	10	0.30

## 5.4 AdaBoost on Naïve Bayes

Dataset	Distribution	# of Learners	Proportion of Samples
Breast Cancer	None	100	0.10
Car	None	100	0.40
E. Coli	Gaussian	250	0.10
Letter Recognition	Gaussian	10	0.30
Mushroom	None	10	0.50

## 5.5 Random Forest

Dataset	# of Learners	Proportion of Samples
Breast Cancer	120	0.70
Car	200	0.60
E. Coli	250	0.60
Letter Recognition	100	0.60
Mushroom	10	0.50

## 5.6 K-Nearest Neighbors

Dataset	K	Distance
Breast Cancer	1	Hamming
Car	1	Hamming
E. Coli	1	Euclidian
Letter Recognition	1	Euclidian
Mushroom	1	Hamming

# 6 Project Design

A Java machine learning framework was first created to ease the implementation of the six learning algorithms. The framework uses Maven to build the project, JUnit for unit tests and slf4j to control logging. All work independently of the machine learning algorithms and may be completely removed without affecting the functionality of the code. The project is divided up into various packages which will be described in the following subsections.

## 6.1 jml

This is the root package of the project and contains several of the main classes that are used throughout the various sub-packages.

**Algorithm** is the interface that all algorithms implement. These objects fit a Model to a DataSet.

**Dataset** is an object which represents a dataset. It abstracts away from the data and provides several useful behaviors.

**Model** is an abstract class that all models implement. Objects which extend this class must be able to make a prediction given a new data instance.

**KFold** implements the k-fold cross validation algorithm. The object is parameterized by the number of folds and generates a Report given an Algorithm and Dataset.

**Report** is an object which summarizes the results of a particular algorithm on a particular dataset. A report contains a list of accuracies and can calculate information such as the mean accuracy of standard deviation.

**Util** is a utility object that contains several useful static methods. For example, Util contains methods to read from CSV files, convert primitive arrays to Lists and calculating information entropy.

## 6.2 tree

This package contains the objects associated with decision trees.

**ID3** is the implementation of the ID3 algorithm. It fits a DataSet to a tree of Nodes and creates an ID3Model. The algorithm is parameterized by the max tree level and minimum number of samples per Node.

**ID3Model** is a trained ID3 model. It uses the trained decision tree nodes to make a classification given a new sample.

**Node** is a node within a decision tree. It contains the logic to choose the attribute which maximizes information gain.

**Children** is an abstract class which both ContinuousChildren and DiscreteChildren implement. This allows for the abstraction of most of the logic associated with splitting and classifying using continuous and discrete attributes.

**ContinuousChildren** are the children of a Node that split on a continuous attribute.

**DiscreteChildren** are the children of a Node that split on a discrete attribute.

### 6.3 ensemble

This package contains the objects associated with ensemble learning.

**AdaBoost** is the implementation of the SAMME AdaBoost algorithm. This algorithm is able to handle both binary and multiclass classification problems and is parameterized by the number of weak learners, proportion of samples and base learning algorithm.

**AdaBoostModel** is a trained AdaBoost model. It classifies a new instance using a weighted vote mechanism and returns the most likely class.

**RandomForest** is the implementation of the random forest algorithm. This algorithm is parameterized by the number of strong learners, proportion of samples and base learning algorithm.

**RandomForestModel** is a trained random forest model. It classifies a new instance using a voting mechanism and returns the most common classification.

### 6.4 exceptions

This package contains the common exceptions which occur the learning process.

**AttributeException** is the runtime exception thrown when an error arises due to the attribute types (continuous, discrete).

**DataException** is the runtime exception thrown when there is an error associated with the data.

**FileException** is the runtime exception thrown when there is an issue with the data file, particularly while reading CSV files.

**PredictionException** is the runtime exception thrown when an error occurs during classification.

### 6.5 math

This package contains objects associated with mathematics.

**MathException** is the runtime exception thrown when an error occurs during a mathematical operation.

**Matrix** is an object which represents a matrix. It contains useful operations for manipulating matrices such as retrieving, modifying and adding rows and columns.

**Tuple** is a tuple object which uses Generics to hold a pair of objects. This object is useful for moving associated objects throughout the code.

**Util** is a utility class that contains several useful mathematical operations such as logarithmic and exponential functions.

**Vector** is an object representation of a vector. Similar to the Matrix class, it contains several useful operation for manipulating its values. Furthermore, several vector arithmetic operations are implemented such as multiplication, addition and the dot product. The Vector class abstracts away from the underlying data representation. Although it uses a List of Doubles as the data structure, it may be treated as a vector of integers if desired.

### 6.5.1 distance

This package contains distance functions for the KNN classifier.

**Distance** is an interface which all distance functions must implement.

**Euclidian** is an object which represents the Euclidian distance function  $\sqrt{\sum_{i=1}^n (q_i - p_i)^2}$ . This object is used to calculate the Euclidian distance between two continuous Vectors while classifying an new instance in a KNNModel.

**Hamming** is an object which represents the Hamming distance function  $\sum_{i=1}^n f_i(q, p)$  where  $f_i(q, p) = 0$  if  $q_i = p_i$  and 1 otherwise. This object is used to calculate the Hamming distance between two discrete Vectors while classifying an new instance in a KNNModel.

### 6.5.2 distribution

This package contains distribution implementations for the Naïve Bayes classifier.

**Distribution** is an interface which all distribution objects must implement.

**Gaussian** An object which represents a Gaussian distribution. It is used by the Naïve Bayes classifier for continuous attributes.

## 6.6 bayes

This package contains all of the objects associated with Naïve Bayes.

**Attribute** is an interface which the Continuous and Discrete attribute types must implement.



**BayesException** is a runtime exception thrown when no Distribution is supplied and there are continuous attributes within the DataSet.

**ClassSummary** is the object created during the NaiveBayes learning process. It is parameterized by a class probability  $P(v_k)$  and a List of Attributes so that it can calculate  $P(v_k|x)$ .

**Continuous** represents a continuous attribute. It is parameterized by a mean, standard deviation and Distribution. Given a new value, the probability  $p(a_i|v_k)$  is calculated.

**Discrete** represents a discrete attribute. It is parametrized by a map of attribute values to conditional probabilities. Given a new value, the conditional probability  $P(a_j|v_k)$  is found in the map.

**NaïveBayes** is the implementation of the Naïve Bayes algorithm. It is optionally parameterized by a Distribution. The algorithm create a ClassSummary for each distinct class.

**NaïveBayesModel** is a trained Naïve Bayes model. It classifies new samples by calculating the conditional probability of each class  $P(v_k|x)$  and returning the most likely classification.

## 6.7 neighbors

This package contains all of the objects associated with k-nearest neighbors.

**KNN** is the implementation of the k-nearest neighbors algorithm. The learning process simply consist of providing the parameters to the KNNModel.

**KNNModel** is a trained KNN classifier. It is parameterized by the number of neighbors  $k$  and a DataSet. It classifies new samples by finding the  $k$  closest neighbors and returning the most common classification.

**KNNException** is the exception thrown when  $k$  exceeds the DataSet sample count.

## 7 Results

The ID3 algorithm generally outperformed the Naïve Bayes algorithm as seen in Table 1. This is especially true when considering the letter recognition dataset. One possibility for this discrepancy is the type of distribution used for continuous attributes. No other probability density functions were experimented with. Furthermore, the letter recognition dataset likely expresses high correlation amongst the attributes. However, the two algorithms perform almost identically on the E Coli. and breast cancer datasets.

Algorithm	Breast Cancer	Car	Letter	E. Coli	Mushroom
<b>ID3</b>	97.994%	90.634%	87.930%	79.678%	99.998%
<b>Naïve Bayes</b>	97.663%	81.940%	64.570%	80.556%	95.952%
<b>AdaBoost on ID3</b>	98.715%	93.295%	97.050%	86.918%	99.828%
<b>AdaBoost on Naïve Bayes</b>	97.826%	91.367%	59.540%	78.803%	99.744%
<b>Random Forest</b>	98.585%	91.175%	93.120%	84.021%	99.865%
<b>K-Nearest Neighbors</b>	98.050%	89.763%	93.923%	87.779%	99.826%

Table 1: The accuracies of the six implemented algorithms on the five UCI Machine Learning Repository datasets. Results were calculated using 10 times 5-fold cross validation.

Algorithm	Breast Cancer	Car	Letter	E. Coli	Mushroom
<b>ID3</b>	1.465%	4.061%	3.392%	8.952%	0.017%
<b>Naïve Bayes</b>	1.678%	3.373%	1.357%	11.199%	2.128%
<b>AdaBoost on ID3</b>	1.216%	3.143%	0.989%	6.908%	0.293%
<b>AdaBoost on Naïve Bayes</b>	1.394%	1.987%	0.716%	8.802%	0.652%
<b>Random Forest</b>	1.597%	5.171%	2.622%	8.325%	0.702%
<b>K-Nearest Neighbors</b>	1.118%	3.967%	2.284%	8.089%	0.433%

Table 2: The standard deviations of the six implemented algorithms on the five UCI Machine Learning Repository datasets. Results were calculated using 10 times 5-fold cross validation.

The AdaBoost algorithms generally maintained or improved the accuracy as their base models. For example, AdaBoost on ID3 greatly improved the classification accuracy for the E. Coli and letter recognition datasets while maintaining similar performance on the other datasets. Additionally, AdaBoost on Naïve Bayes improved the classification accuracy on the car and mushroom datasets while maintaining similar accuracies on the others. One exception to this is the letter recognition dataset where AdaBoost on Naïve Bayes scored 10% lower than regular Naïve Bayes. The random forest algorithm also improved the classification accuracy of ID3; however, it generally underperformed AdaBoost on ID3.

I believe the classification accuracy for each algorithm, dataset pair could be improved given more time for hyperparameter tuning, especially the ensemble algorithms. The larger datasets proved extremely difficult for experimentation as they required much longer training times. When experimenting with various hyperparameters for the letter dataset, it often took longer than 10 minutes to complete one 5-fold cross validation iteration. Although this is not an extremely long time, tuning became much more difficult. Optimization was considered after the initial algorithm implementations; however, training time was not drastically reduced.

The most surprising results received were from the k-nearest neighbors algorithm. It almost always outperformed the other algorithms and nearly received the highest accuracy for each. The amount of neighbors was set to 1 for each dataset. This was an unexpected result of the hyperparameter search. The most prominent flaw of the k-nearest neighbors algorithm was the extremely long classification times, especially for the larger datasets.

## References

- [1] Badr HSSINA, Abdelkarim MERBOUHA, Hanane EZZIKOURI, and Mohammed ERRITALI. A comparative study of decision tree id3 and c4.5. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 2014.
- [2] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.
- [3] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1(1):81–106, March 1986.
- [4] Robert E. Schapire. A brief introduction to boosting. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence - Volume 2, IJCAI'99*, pages 1401–1406, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.
- [5] Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie. Multi-class adaboost. 2006.