CS6735 Programming Assignment

Ashley Barkworth

1 Introduction

Machine learning algorithms are used for various problems, including classification. Classification is a supervised learning concept which categorizes a set of data into classes. It is a common machine learning problem and can be solved through many different supervised learning algorithms. This report presents a comparison between five supervised learning methods: decision trees, boosting, random forest, Naïve Bayes, and k-nearest neighbor.

The objectives of this study are as follows: evaluate five different learning algorithms on five different data sets using 10 times 5-fold cross validation. The rest of the report is outlined as follows. Section 2 provides an overview of the different learning algorithms. Section 3 discusses the details of the data sets used. Section 4 details the technical implementation of the algorithms. Section 5 shows the results of the cross-validation. Section 6 gives an analysis of the results. Finally, Section 7 provides a conclusion and future research directions.

2 Learning Algorithms

Below is an overview of the five learning algorithms I implemented and the parameters I considered. I tested the algorithms with different parametric combinations and chose the best ones for the final evaluation results.

2.1 ID3

The Iterative Dichotomiser 3 (ID3) algorithm is an algorithm developed by Ross Quinlan to generate decision trees. Decision trees break down data sets into smaller and smaller subsets based on attribute values, where the final decisions (i.e. classifications) are stored in the leaves. ID3 creates decision trees using a top-down greedy approach that considers the entire space of possible branches without any backtracking. Starting at the root node, it considers each attribute in the examples and partitions the data set into subsets containing examples with the same attribute value. It then computes the entropy of these subsets, where a perfectly homogeneous subset as an entropy of 0. This entropy is used to calculate the information gain for splitting the data set on this attribute and the attribute yielding the highest information gain are chosen to split the node. Each of the resulting split subsets are used to create child nodes, and ID3 then recursively splits each of the children.

For my implementation, I used information gain to split both discrete and continuous attributes. However, to split continuous attributes, I used information gain based on splitting the examples with the best pivot, an approach proposed by Quinlan in the C4.5 algorithm. To determine the pivot, I first sorted the attribute values in ascending order and evaluated each of the distinct midpoints between adjacent values by splitting the training examples into two groups: examples with an attribute value less than the midpoint and examples with an attribute value greater than the midpoint. I then calculated the resulting information gain from this split. The midpoint with the highest information gain was chosen as the pivot and the examples were split using the pivot as a threshold. Decision trees were allowed to grow to their full height. After growing the full decision tree, reduced-error pruning is performed by removing a node (one by one) and then checking the classification accuracy on a set of pruning examples. If the accuracy is increased, than the node is removed; otherwise, it recursively checks the child nodes.

2.2 Naïve Bayes

Naïve Bayes is a probabilistic algorithm that applies Bayes' theorem where

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

with the assumption of conditional independence between attributes, i.e.,

$$P(a_1, a_2, \ldots, a_n|v_j) = \Pi_{i=1}^{n} P(a_i|v_j)$$

Naïve Bayes does not generate accurate probabilities, but it still works quite well in classification problems for both discrete and continuous values.

To handle continuous values, I assumed that they were distributed according to a Gaussian distribution, and used the training data to compute distribution parameters- i.e., the mean and variance. The learned probability distribution is used during classification to compute the probability density for each class value. To handle missing values, I used Laplace smoothing. The posterior probabilities $P(A = a_i | V = v_j)$ are calculated using the formula

$$\frac{n_c + 1}{n + |A|}$$

where n_c is the number of examples where $a = a_i$ and $v = v_j$, n is the number of examples where $v = v_j$, and $|A|$ is the number of values that attribute A can take on.

2.3 AdaBoost

AdaBoost is a boosting algorithm first developed by Yoav Freund and Robert Schapire. It is an ensemble learning method that can be used with other learning algorithms to improve their performance. It reduces the bias of individual learners and trains weak learners into strong learners. AdaBoost works by maintaining a weighted distribution over the training examples. It trains a weak learner using a bootstrapped sample of the training data and then calculates its error when classifying the entire training data. The algorithm calculates values of alpha that are proportionate to the error of each weak learner, and increases the weights of incorrectly classified examples by multiplying this alpha by the current weights of incorrect examples. Subsequently, the weights of incorrectly classified examples increase and the weights of correctly classified examples decrease, causing the weak learners to focus on the incorrectly classified examples. For example, decision trees may focus on splitting examples with higher weights. The output of every weak learner is combined into a weighted sum, representing the final output of the boosted classifier.

I implemented the AdaBoost algorithm on weak decision tree stumps that are generated with ID3 (without the reduced error pruning). I considered boosted trees after 1, 10, 25, 50, 100, 150, and 200 steps of boosting. Because the original AdaBoost algorithm is for classification problems with two classes (i.e. -1,1) and some of the data sets contained more than two classes, I implemented an adaptation of the AdaBoost algorithm called Stagewise Additive Modeling using a Multi-class Exponential loss function (SAMME) [1] which modifies the original AdaBoost algorithm by computing alpha values with

$$\alpha = \log \frac{1 - error}{error} + \log(K - 1)$$

where K is the total number of classes. In addition to boosting steps, I also considered different proportions for the size of the data set used for training weak learners, e.g., a proportion of 0.6 implies that the dataset used for training weak learners would be bootstrapped from the data set used for training AdaBoost and the size of this data set would be 0.6 of the size of the AdaBoost training data set. I considered proportions of $0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$, and 1.0.

2.4 Random Forest

Random Forest is another ensemble method that works by constructing multiple decision trees in parallel and outputting the most common class predicted from the individual trees. The first algorithm was developed by Tin Kam Ho in 1995, and an extension of the algorithm was developed by Leo Breiman and Adele Cutler in 2006, who coined the term ”random forests” for this method.

For my implementation, I considered forest sizes of 1, 10, 25, 50, 100, 150, and 200. For values of m, where m indicates the maximum number of attributes considered for each node split in ID3, I considered values of $m = 1, 2, 4, 6, \ldots, \frac{3}{4}|A|$ where $|A|$ is the total number of attributes in the dataset. Like with boosting, I also considered different proportions for the size of the data set used for training weak learners, and considered proportions of $0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9$, and 1.0.

2.5 K-Nearest Neighbors

The k-nearest neighbors (KNN) algorithm is an instance based learning method that classifies examples by computing the k closest neighbors and outputting the most common class among these neighbors as the predicted class. The training time for this algorithm is trivial - it simply stores the training examples. However, the classification time can be very long.

For this study, I considered values of $k = 1, 2, 5, 10, 15$, and 20. I used Euclidean distance as the metric for determining the closest neighbors for continuous attributes. For discrete attributes, I used Hamming distance, where the distance between two examples is the total number of mismatched attribute value between the two. For example, the Hamming distance between [LOW,MED,4,MORE,SMALL,HIGH,GOOD] and [LOW,HIGH,4,MORE,BIG,HIGH,UNACC] is 3.

3 Data Sets

I compared the algorithms on 5 classification problems. The Wisconsin Diagnostic Breast Cancer (Breast Cancer), Car Evaluation (Car), Ecoli (Ecoli), Letter Recognition (Letter) and Mushroom (Mushroom) data sets are from the UCI repository [2]. See Table 1 for the characteristics of these data sets. (Note that the Breast Cancer data set from D2L is the original 1992 version, which is the one I used for this study. The one that is linked in the programming assignment description PDF is the 1995 version and is not the one used for this study.)

3.1 Breast Cancer Data Set

Title: Wisconsin Diagnostic Breast Cancer
Date: July 1992
Number of Samples: 699
Number of Attributes: 10
Attribute Information: See Table 1.

TABLE 1
Attribute information for Breast Cancer Data Set

Description	Values	Type
Sample	ID number	Discrete
Code	1-10	Discrete
Number	1-10	Discrete
Clump	1-10	Discrete
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: Diagnosis
Class Information: See Table 2.

TABLE 2
Class information for Breast Cancer Data Set

Description	Value	Count	Percentage
Benign	2	458	65.5%
Malignant	4	241	34.5%

Missing Attribute values: 16

3.2 Car Data Set

Title: Car Evaluation Database
Date: June 1997
Number of Samples: 1728
Number of Attributes: 6
Attribute Information: See Table 3.

TABLE 3
Attribute information for Car Data Set

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

Class Target: Car Quality
Class Information: See Table 4.
Missing Attribute values: None

TABLE 4
Class information for Car Data Set

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

3.3 Ecoli Data Set

Title: Protein Localization Sites
Date: September 1997
Number of Samples: 336
Number of Attributes: 8
Attribute Information: See Table 5.

TABLE 5
Attribute information for Ecoli Data Set

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

Class Target: Localization Site
Class Information: See Table 6.

TABLE 6
Class information for Ecoli Data Set

Description	Value	Count	Percentage
Cytoplasm	cp	143	42.56%
Inner Membrane	im	77	22.92%
Perisplasm	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 Attribute values: None

3.4 Letter Recognition Data Set

Title: Letter Image Recognition Data
Date: January 1991
Number of Samples: 20000
Number of Attributes: 16
Attribute Information: See Table 7.
Class Target: Letter
Class Information: See Table 8.
Missing Attribute values: None

3.5 Mushroom Data Set

Title: Mushroom Database
Date: April 1987
Number of Samples: 8124
Number of Attributes: 22

TABLE 7
Attribute information for Letter Recognition Data Set

Description	Values	Type
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

TABLE 8
Class information for Letter Recognition Data Set

Description	Value	Count	Percentage
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 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	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%

Attribute Information: See Table 9.
Class Target: Edibility
Class Information: See Table 10.
Missing Attribute values: 2480 (attribute 11)

TABLE 9
Attribute information for Mushroom Data Set

Description	Values	Type
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	p,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

TABLE 10
Class information for Mushroom Data Set

Description	Value	Count	Percentage
Edible	e	4208	51.80%
Poisonous	p	3916	48.20%

4 Technical Implementation

4.1 Pre-processing

I applied several pre-processing measures to the data sets. I removed the first column in the Breast Cancer and Ecoli data sets since they were ID/Sequence number columns. In addition, I removed the gvh and lip columns from the Ecoli data set, since the values for these columns are all 0.48 and 0.50, respectively, and thus do not provide any added information for classification. To handle missing values in the Mushroom and Ecoli data sets, I assigned the most frequent value for that attribute from the dataset.

I did not discretize the decimal values in the Ecoli data set. Instead, I handled continuous values differently than discrete values in the implementations of the algorithms, as described previously in **Section 2.1**. For the Letter data set, I interpreted the integer attributes values as continuous values.

4.2 Program Structure

I implemented the five learning algorithms in Java and used IntelliJ as an IDE. The program's entry point is the `Main` class, which runs 10 times 5-fold validation on the five learning algorithms with their optimal parameters for each of the five datasets. For each dataset and algorithm, the program prints to the console the average and standard deviation of the accuracies obtained. (NOTE: The program multiples the averages and standard deviation by 100 before printing, e.g. an accuracy of 0.853 is printed as 85.3. This report shows the values without multiplying.)

Each algorithm is represented by a class which implements a `Classifier` interface containing two methods:

1) **train()** takes the training dataset as input and learns the corresponding hypothesis (e.g. decision tree, posterior probabilities) or, in the case of KNN, stores the training examples
2) **classify()** takes a single test example as input and returns the predicted class for this example given its attribute values

For algorithms with parameters (i.e. k-nearest neighbors, AdaBoost, Random Forest) each class included them in their constructors. The parameters used were the ones mentioned in Section 2 and a series of tests were performed to choose the optimal parameters.

Data sets are represented by a `Dataset` class, which stores a 2D String array of the original data that is parsed from the .data files as well as:

- the total number of attributes (including the target attribute)
- the target attribute column index (depending on whether the target attribute is the first or last column)
- the type of dataset (i.e. discrete or continuous)
- a 2D array storing the unique values for each attribute
- a list of the unique class (i.e. target attribute) values.

There are two other important class variables: **rowIndices** stores a list of the indices from the original data set and is useful when creating subsets. For example, when splitting attributes in the ID3 algorithm, a new `Dataset` instance is created for each attribute value. These new `Dataset` instances reference a copy of the original data rather than create a new 2D array for the subset of examples for better performance. Instead, these subsets store the row indices from the original data where the example's attribute value matches. This is also used for creating training and test subsets from the original data during cross-validation. The other important class variable is **entropy**, which stores the entropy of a dataset that is used in the ID3 algorithm.

The cross-validation is implemented in a `KFold` class. Each `KFold` instance has a value for k (for this study, $k = 5$), as well as a data set and an algorithm to perform cross-validation on. It splits the data set into k folds and performs the cross-validation, which yields k different accuracies. There is a private class within `KFold` called `Split`, where each `Split` instance has an integer array of train indices and test indices for training and testing, respectively.

There are different packages in the project. Of note, the *ensemble* package contains the `AdaBoost` and `Random Forest` classes containing the implementations for those two respective algorithms. The *trees* package contains different classes used for generating the decision trees. Namely, it contains an `ID3` class which implements the aforementioned `Classifier` interface and a `TreeNode` class that performs the splitting for the decision tree. `Children` implements the `Iterable` interface to process every child node of a parent. `Children` is an abstract class extended by `ContinuousChildren` and `DiscreteChildren`, which contain the implementation for each child depending on the data set type. In addition, the `Pruning` class performs reduced-error pruning on the resulting decision tree after ID3 completes. The *bayes* package contains the classes for implementing Naïve Bayes. `NaiveBayes` implements the `Classifier` interface, `Attribute` is an interface that computes and returns the conditional probabilities for each attribute, and `ClassSummary` contains the class probability as well as a list of `Attribute` instances storing the conditional probabilities of each attribute value given that class. `Attribute` is implemented by `Discrete` and `Continuous` depending on the data set type.

5 Results

5.1 Performance Metrics

The two metrics used for evaluating the learning algorithms were average accuracy and standard deviation. The accuracy is computed by

$$\frac{\text{Number of correct predictions}}{\text{Total number of predictions}}$$

and the accuracy falls between 0.0 and 1.0.

For each data set-algorithm combination, I applied 10 times 5-fold cross-validation. To compute the averages and standard deviations, I stored the individual accuracies (50 in total) and then computed the average and standard deviation over these accuracies.

5.2 Results by Performance Metric

Table 2 shows the average accuracy as well as the standard deviation for each learning algorithm and data set. The best results for each data set are bolded.

TABLE 11
Average accuracy ± standard deviation for each learning algorithm by dataset

Algorithm	Breast Cancer	Car	Ecoli	Letter	Mushroom
ID3	0.957 ± 0.019	$\mathbf{0.929 \pm 0.014}$	0.833 ± 0.041	0.816 ± 0.0067	$\mathbf{1.0 \pm 0.0}$
Random Forest	$\mathbf{0.991 \pm 0.0071}$	0.917 ± 0.012	0.765 ± 0.042	0.808 ± 0.0082	0.997 ± 0.0012
AdaBoost on ID3	0.955 ± 0.016	0.700 ± 0.024	0.649 ± 0.053	0.124 ± 0.014	0.985 ± 0.0028
Naïve Bayes	0.973 ± 0.010	0.851 ± 0.016	0.838 ± 0.045	0.642 ± 0.0078	0.954 ± 0.0068
k-nearest neighbors	0.953 ± 0.016	0.776 ± 0.024	$\mathbf{0.852 \pm 0.038}$	$\mathbf{0.956 \pm 0.0073}$	$\mathbf{1.0 \pm 0.0}$

6 Analysis

As the results show, there is no universally optimal learning algorithm. Different data sets were classified best by different algorithms, such as Random Forest, decision trees, and k-nearest neighbor. Furthermore, each algorithm performed relatively poorly on at least one data set. For example, Random Forest performed relatively poorly on the Ecoli data set, Naïve Bayes performed relatively poorly on the Mushroom data set, and k-nearest neighbor performed relatively poorly on the the Car data set. The k-nearest neighbor performed very well overall. It had the best performance for three of the five data sets. The other algorithms that were optimal for at least one of the data sets were decision trees (ID3) and Random Forest.

I found that implementing reduced-error pruning significantly improved the performance of decision trees. For example, before pruning, DT yielded accuracies of around 0.86 and 0.78 for the Car and Letter data sets, respectively. This reduced-error pruning resolved the issue of overfitting.

On the whole, boosting performed the worst out of any learning algorithm. One outlier in the results is AdaBoost's performance against the Letter data set, where its classification performance is just about 4 times better than randomly assigning a letter for each example. More investigation is required to determine the cause of this poor performance.

In terms of parameters, I found that increasing the number of boosting steps for AdaBoost and the number of trees for Random Forest generally increased the classification accuracy. For Random Forest, increasing the value of m (i.e. the number of attributes to consider at each node split) increased the accuracy as well when m is very small (e.g. $m = 1, 2$). However, when m is not very small, lower values of m often performed better compared to higher values. For example, using a value of $m = 4$ for the Ecoli data set performed better than using $m = 6$. I found that increasing the proportion of training examples sampled for training the weak learners in the ensemble methods had mixed results, i.e., it increased, decreased, or had no effect on accuracy. Lastly, the optimal values of k for the k-nearest neighbor algorithm were $2, 1, 1, 10$, and 1 for the Car, Mushroom, Letter, Ecoli, and Breast Cancer data sets, respectively.

The most surprising result to me was the very good performance of the k-nearest neighbors algorithm. It often significantly outperformed the other learning algorithms, despite its simple classification logic.

7 Conclusion

This paper compared five different learning algorithms by performing cross-validation against five different data sets. The results indicate that while certain learning algorithms like Random Forest and k-nearest neighbor perform fairly well across all data sets, there is no universally best algorithm for each classification problem. In addition, configuring the parameters of different algorithms can improve their results. Given the importance of classification problems in machine learning, it is important to study the current algorithms used for classification to determine the best performing algorithms or to find ways to improve their classification accuracy. In the future, I would like to compare the reduced-error pruning that I implemented with other post pruning methods. In addition, further investigation into the poor results for boosting would be beneficial.

References

[1] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.
[2] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml