Patrick Vacek
Final Project
MATH 3210

Evaluating the Effectiveness of a Combination of Multiple Classifiers

Executive Summary

My goal is to determine if voting (using a combination of multiple classifiers) is an effective means for classification. The principle is that each individual classification technique has certain strengths and weaknesses, and no matter which technique is applied, there will always be areas that that technique cannot accurately handle (Dunham 119-120). By combining several methods and giving each one vote, I can balance out the weaknesses of each individual method. Assuming that the strengths outweigh the weaknesses of each technique applied to the spam dataset, the voting should be able to weed out most incorrect classifications by an individual technique.

The dataset with which I will work is a dataset of spam. The dataset consists of 4601 instances, each representing an email and various statistics about the email. 54 of the 58 attributes for each instance measured the frequency of the appearance of certain words or characters in the email (which range from the generic "technology" to the personal "george", which is the name of the dataset's donor), three measured statistics about the appearance of capital letters in the email, and the final attribute was a predetermined class value (spam or not spam).

I chose five algorithms to apply to the dataset, and I gave each algorithm one vote for each data instance. Final classification was determined by simple majority of votes. Since the dataset has 4601 instances, I split it into two halves to create both a large training set and a large testing set. I trained each algorithm on the training set but applied the resulting model to the test set to find my final results. I will briefly explain each algorithm I used.

The Naïve Bayesian Classifier works by using Bayes' theorem to determine the probability that a given attribute will correspond to a given class (Dunham 52-53). For each instance, the probabilities of each attribute for a certain class are multiplied together to give a final probability of the whole instance belonging to the class (Dunham 86-87). The instance is then assigned to the class for which it has the highest probability (Ibid.).

C4.5 creates a decision tree using information entropy. The tree consists of leaf nodes that represent classes, internal nodes that represent attributes, and arcs that represent predicates that correspond to the parent attribute node (Dunham 92-93). The splitting attributes are selecting by finding the attributes with the greatest information gain, which is the difference in information needed to make a classification before and after the split (Dunham 98).

The K-Nearest Neighbor algorithm finds the distance between a given test instance's attributes and those of each training instance's (Han 314-315). The given test instance is assigned to the class that the majority of the closest k neighbors belong to. I chose k = 5.

The Multilayer Perceptron algorithm is a type of artificial neural network. Neural networks are based on a directed graph, and use the attributes of a data instance as inputs and the classes as output nodes (Dunham 103). Each instance runs through the directed graph and ends up with probability values in each final class node; whichever class has the highest probability is the class the instance is assigned to

(Ibid.). The number of hidden layers between the input and output and the number of nodes in each layer are variables, decided upon by the user, that depend on circumstances (Dunham 104). When propagating an instance through the network, an activation function of some sort (often sigmoidal) is applied at each node to create output values along each arc to propagate to the next layer of nodes (Dunham 105). During training of the network, some kind of algorithm using backpropagation must be applied to adjust for the errors throughout the network in assigning a training instance to a class (Dunham 106-109).

The Genetic Algorithm uses the principles of natural evolution to develop a more fit population (Han 316). The initial population is formed by creating a vector of values that represent a rule; the last element determines class and the other elements specify rules to be applied to data instances that ideally would determine if a given instance belongs to that class or not (Ibid.). In line with evolutionary theory, only the fittest population survive. The genetic algorithm uses some sort of fitness function to determine which members of the population survive and reproduce, but this fitness function should relate to the minimization of errors when using the vector's rules (Ibid.). During reproduction, the vectors can crossover and combine parts of each other to combine rulesets, and there can be an additional chance of mutation to create entirely new rules (Ibid.). After iterating through a certain number of generations or reaching a given fitness level across the population, the algorithm stops and applies the rules of the fittest population to the dataset (Ibid.).

Given a test set of 2300 instances, 907 of which were spam and 1393 were not, my results are as follows:

	Naïve Bayes	C4.5	KNN	Perceptron	Genetic	Voting
Errors:	562	333	328	518	513	222
Error %:	0.24435	0.14478	0.14261	0.22522	0.22304	0.09652
False Positives:	532	225	159	488	318	201
False Positive %:	0.38191	0.16152	0.11414	0.35032	0.22828	0.14429
False Negatives:	30	108	169	30	195	21
False Negative %:	0.03308	0.11907	0.18633	0.03308	0.21499	0.02315

I evaluated each algorithm individually and then based on the voting classifications. The error value counts the number of incorrectly classified instances, and the error percentage is a calculation of the number of incorrect classifications divided by the total number of instances. The false positives value counts the number of instances that were not spam but incorrectly classified as such, and the false positive percentage is the number of false positives divided by the number of non-spam instances. The false negatives value is the number of instances that were spam but classified as not spam, and the false negative percentage is the number of false negatives divided by the number of spam instances.

Three of the algorithms misclassified over 500 instances and two misclassified only about 330, but voting on classification reduced the misclassification count to merely 222. This is a reduction in error of over 50% from the three less accurate algorithms and about 33% from the two more accurate ones. Voting may not be perfect, but it is clearly an improvement over any individual technique.

I included the numbers about false positives and negatives for two reasons. First, four of the algorithms were better at reducing false negatives than false positives, and two of them extremely so. This carried over into the voting, where there were a mere 21 false negatives. However, this carries into the second reason, which is that with a dataset of spam, our algorithm should aim to reduce false positives to an absolute minimum and thus be far less concerned about false negatives. An email system that rarely misses unwanted spam but considers around 20% of legitimate emails as spam can hardly be considered desirable. For other datasets, the reduction of false positives may not be as critical, and thus these results would imply that voting based on a combination of multiple classifiers is a more effective technique than utilizing just a single classifier.

Problem Description

My goal is to determine if voting (using a combination of multiple classifiers) is an effective means for classification. The principle is that each individual classification technique has certain strengths and weaknesses, and no matter which technique is applied, there will always be areas that that technique cannot accurately handle (Dunham 119-120). By combining several methods and giving each one vote, I can balance out the weaknesses of each individual method. Assuming that the strengths outweigh the weaknesses of each technique applied to the spam dataset, the voting should be able to weed out most incorrect classifications by an individual technique. However, it is possible that if several techniques fail in a similar manner, the strengths of the minority will be overshadowed.

To simplify the process, it is important to use an odd number of techniques and just two classes so that votes will not end in a tie. Therefore, I have chosen five algorithms to apply to a dataset, and I will give each algorithm one vote for each data instance. Final classification will be by simple majority of votes. The five algorithms I will use are a naïve Bayesian classifier, C4.5 (which uses entropy to create decision trees), K-Nearest Neighbors, a multilayer perceptron (a type of neural network using backpropagation), and a genetic algorithm (used to determine an optimal decision tree).

The dataset with which I will work is a dataset of spam culled from the inbox of George Forman, an employee of Hewlett-Packard Labs. Spam has a vague definition but is generally considered to be undesired email consisting of advertisements, chain letters, pornography, and the like (Hopkins). The dataset consists of 4601 instances, each representing an email and various statistics about the email. 48 of the 58 attributes are numeric values representing the number of times the given word appeared in the email out of all the words in the email (number of appearances divided by total number of words). Some are very generic, such as "you" or "address"; some are standard business terms, like "technology" and "meeting"; others are more typical of what one would associate with spam: "credit", "free", and so on. A few of the terms, such as "george" (the dataset's donor's name), "hpl" (an abbreviation of the donor's employer), and "650" (the donor's area code), are specifically related to the source of the dataset but may help for this personalized case. Six of the attributes are percentage frequencies of the appearance of certain characters (";", "(", "[", "!", "$", and "#") in the email, and three are statistics relating to the occurrence of capital letters: average length of uninterrupted sequences of capital letters, longest length of an uninterrupted sequence of capital letters, and the total number of capital letters in the email. The final attribute is the predetermined class of the instance (spam or not spam).

Analysis Technique

I will first explain each of the algorithms used and then explain how I used the algorithms to reach my results.

Naïve Bayesian Classifier

The first algorithm applied is the Naïve Bayesian Classifier. This algorithm uses Bayes' theorem to determine which class each instance is most likely to belong to. Bayes' theorem is mathematically as:

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

(Bayes' theorem). P(A) is the probability of A, here a particular class, being found in the entire sample (Dunham 52-53). P(B) is the probably that B, here a particular value of an attribute of the data instances, will be found in the sample (Ibid.). P(B|A) is the probability that B, a particular value of an attribute, will occur given A, a certain class (Ibid.). These three probabilities are easily found within a dataset just by counting. P(A|B), the desired output value, is the probability that B, an attribute value, belongs to A, a class (Ibid.). This probability is hard to find in the data without calculating the other probabilities. Thus, the likelihood of a value belonging to a particular class is found by multiplying the probability of the class having the value with the probability of the class occurring in the dataset and then dividing by the probability that the value appears in the dataset.

Then we must apply this information to each instance in the dataset. For each instance, we can calculate the probability that the instance belongs to a particular class. For each attribute that the instance has, we can calculate the likelihood of that value belonging to a class given Bayes' theorem, but if we then multiply each of these likelihoods for a particular class together, we can estimate the probability that the instance as a whole belongs to the class (Dunham 86-87). Whichever class has the highest probability is the class that the instance is assigned to (Ibid.).

I used the naïve Bayesian classifier found in the Weka application to implement this algorithm.

C4.5

The C4.5 technique is an improved version of the ID3 algorithm, both developed by Ross Quinlan and based upon decision trees. Decision trees use a tree structure based on predicates and classes to make determining each instance's class a simple process of going down the tree (Dunham 92-93). Each leaf node of the tree is a class, while internal nodes are attributes and the arcs between them are predicates based on the parent attribute (Ibid.). An instance's attribute values are compared with the predicates of the tree, and whichever leaf the instance ends up at is used to classify it (Ibid.).

ID3 uses the concept of information entropy to determine the attribute predicates (Dunham 97). Entropy is a measurement of disorder or uncertainty; if every data instance belonged to one class, there is no uncertainty involved and the entropy is thus zero (Ibid.). If there is uncertainty, then we need a certain amount of information to determine the class of any instance, and this information is represented by:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i)$$

(Han 286; Information entropy). In this equation, $p(x_i)$ represents the probability that an instance x belongs to a class i, and the sum of these probabilities for each class will equal one (since the instance must belong to a class) (Han 286; Dunham 98). Summing these probabilities multiplied by the logarithm of the same probabilities gives H(X), the amount of information required to classify x (Han 286). Ideally, a splitting attribute for a decision tree could divide a dataset into two halves, each of which contains all of the members of one of the classes. Since perfect splits are rare, ID3 actually determines splitting attributes by finding the attribute with the highest information gain, where gain is the difference in information needed to make a classification before and after the split (Dunham 98). ID3 determines which splits have the highest gain and makes a tree based on this. C4.5 follows the same concept but handles missing and continuous data better, prunes ineffective branches, and uses a slightly more complex splitting calculation (Dunham 100-101).

I used John Aleshunas's implementation of C4.5 based on Quinlan's original C code. I used most of the default values (weight factor of 2, 10 candidate trees, window size of 20, window increment of 20), but lowered the pruning factor to .1, since this seemed to both reduce the number of errors in the training data and also provide a rather small decision tree.

K-Nearest Neighbors
The K-Nearest Neighbor algorithm is fairly simple. Each training data instance is stored as a vector of its attributes, and then each testing instance calculates its distance to the training instances (Han 314-315). Each test instance chooses the closest k neighbors and uses the classes of the neighbors to vote on the class of the test instance (Ibid.). Closeness is usually determined by using Euclidean distance (the square root of the sum of the squared differences between each attribute in the training and test instance) (Ibid.).

I used the IBk implementation of this algorithm as found in the Weka application. I found that five neighbors ($k = 5$) provided the best results.

Multilayer Perceptron
The Multilayer Perceptron algorithm is a type of artificial neural network. Neural networks are based on a directed graph which use the attributes of a data instance as inputs and the classes as output nodes (Dunham 103). Each instance runs through the directed graph and ends up with probability values in each final class node; whichever class has the highest probability is the class the instance is assigned to (Ibid.). The number of hidden layers between the input and output and the number of nodes in each layer are variables, decided upon by the user, that depend on circumstances (Dunham 104). When propagating an instance through the network, an activation function of some sort (often sigmoidal) is applied at each node to create output values along each arc to propagate to the next layer of nodes (Dunham 105).

The complication of neural networks comes from their training and adjustment. Since we know the class of each instance, we will use supervised learning to adjust arc weights of the network based on the output node values of each instance compared to the correct classification of the instance (Dunham 106-107). A certain error value can be calculated by finding the difference between the desired and actual output values, and this is usually multiplied by some learning rate constant to determine how much to change the values of the previous layer (Dunham 107-108). Backpropagation is the technique of applying a certain amount of these initial changes to layers further back in the network (Dunham 108-109). A simple method is to alter the layers closer to the input nodes in such a way to minimize the error values found in the layers closest to the output nodes (Ibid.).

Perceptrons use the same process, but each node (or perceptron) has multiple inputs and just one output (Dunham 112). A multilayer perceptron is thus a network of several perceptrons spread across multiple layers (Dunham 113). I used the multilayer perceptron implementation found in the Weka application. I used the default values (hidden layers determined by the number of attributes and classes divided by two, learning rate of .3, a limit of 500 training iterations, etc.).

Genetic Algorithm

The Genetic Algorithm uses the principles of natural evolution to develop a more fit population (Han 316). The initial population is formed by creating a vector of values that represent a rule; the last element determines class and the other elements specify rules to be applied to data instances that ideally would determine if a given instance belongs to that class or not (Ibid.). In line with evolutionary theory, only the fittest population survive. The genetic algorithm uses some sort of fitness function to determine which members of the population survive and reproduce, but this fitness function should relate to the minimization of errors when using the vector's rules (Ibid.). During reproduction, the vectors can crossover and combine parts of each other to combine rulesets, and there can be an additional chance of mutation to create entirely new rules (Ibid.). After iterating through a certain number of generations or reaching a given fitness level across the population, the algorithm stops and applies the rules of the fittest population to the dataset (Ibid.).

I used the free version of GATree, a commercial product. I used the default values (100 generations, population size of 100, .99 crossover probability, .1 mutation probability, etc.) except for the preference of smaller vs. more accurate trees. I found that smaller trees were only marginally less accurate than larger ones and thus decided that the simplicity would be preferable. (Many of these variables could not be changed beyond a certain limit or at all unless I bought the full version, which may have reduced the potential accuracy of the algorithm.)

Combining the Techniques

Since the dataset I chose is quite large (4601 instances), I split it in half to create a training set (2301 instances) and a test or holdout set (2300 instances). For each technique I used, I create a model using the training set and then applied it to the test set to gather my actual results. For each instance, I recorded how each technique classified the instance and then I used the results to vote on a final classification.

Assumptions

I have made the following assumptions while conducting my research:

1. Each algorithm does indeed have particular strengths and weaknesses, and that the weaknesses can be balanced out by using multiple techniques that do not share the same weaknesses.
2. The techniques I chose are different enough to be representative of the entire population of algorithms.
3. Diversity in types of algorithms is necessary for voting to operate ideally.
4. Five techniques are enough to vote effectively.
5. My classification models were not significantly overtrained or undertrained.

Results

Given a test set of 2300 instances, 907 of which were spam and 1393 were not, my results are as follows:

	Naïve Bayes	C4.5	KNN	Perceptron	Genetic	Voting
Errors:	562	333	328	518	513	222
Error %:	0.24435	0.14478	0.14261	0.22522	0.22304	0.09652
False Positives:	532	225	159	488	318	201
False Positive %:	0.38191	0.16152	0.11414	0.35032	0.22828	0.14429
False Negatives:	30	108	169	30	195	21
False Negative %:	0.03308	0.11907	0.18633	0.03308	0.21499	0.02315

I evaluated each algorithm individually and then based on the voting classifications. The error value counts the number of incorrectly classified instances, and the error percentage is a calculation of the number of incorrect classifications divided by the total number of instances. The false positives value counts the number of instances that were not spam but incorrectly classified as such, and the false positive percentage is the number of false positives divided by the number of non-spam instances. The false negatives value is the number of instances that were spam but classified as not spam, and the false negative percentage is the number of false negatives divided by the number of spam instances. Additional data about the training sets can be found in the appendix.

Three of the algorithms misclassified over 500 instances and two misclassified only about 330, but voting on classification reduced the misclassification count to merely 222. This is a reduction in error of over 50% from the three less accurate algorithms and about 33% from the two more accurate ones. Voting may not be perfect, but it is clearly an improvement over any individual technique.

I included the numbers about false positives and negatives for two reasons. First, four of the algorithms were better at reducing false negatives than false positives, and two of them extremely so. This carried over into the voting, where there were a mere 21 false negatives. However, this carries into the second reason, which is that with a dataset of spam, our algorithm should aim to reduce false positives to an absolute minimum and thus be far less concerned about false negatives. An email system that rarely misses unwanted spam but considers around 20% of legitimate emails as spam can hardly be considered desirable. For other datasets, the reduction of false positives may not be as critical, and thus these results would imply that voting based on a combination of multiple classifiers is a more effective technique than utilizing just a single classifier.

Issues

It is quite possible that several of the techniques could have the same weaknesses and thus could create a larger weakness in the grand voting scheme. Also, in trying to balance out weaknesses, I may have balanced out and nullified the strengths of certain techniques and thus reduced their helpfulness. Some of the algorithms I used probably could have been more fine-tuned to this dataset to reduce the number of errors (most obviously in the case of the genetic algorithm), although such a reduction would likely in turn reduce the number of errors in voting and still maintain my results as valid. As mentioned in the Results section, the number of false positives would render my results fairly unusable for actual spam classification, although with data less sensitive to false positives, these results would hold more strongly. I also cannot be entirely confident that voting can be applied to a dataset with more than two classes with ease.

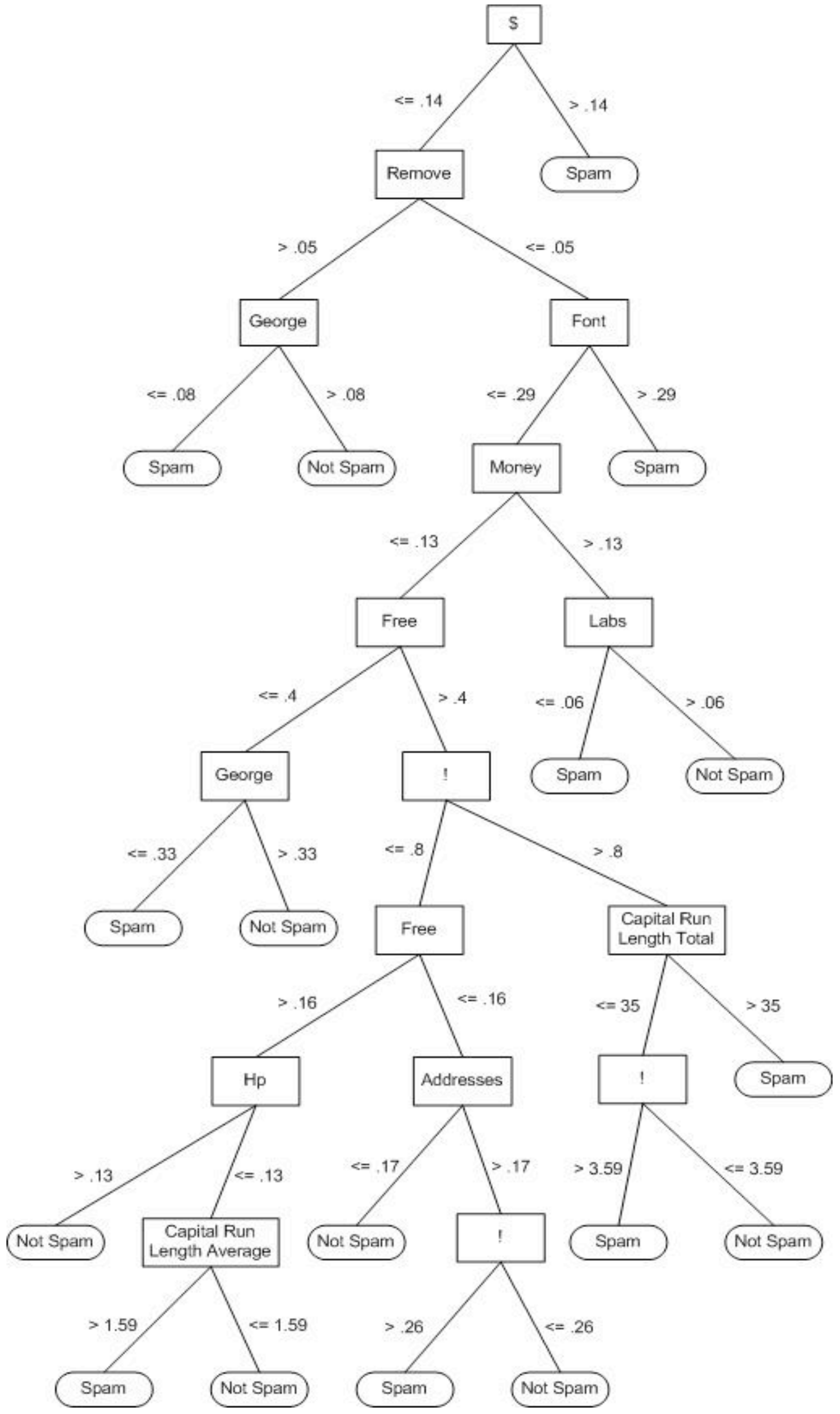
Appendix

I also evaluated the results of applying voting to the output from the training sets. Given a training set of 2301 instances, 906 of which were spam and 1395 were not, my results are as follows:

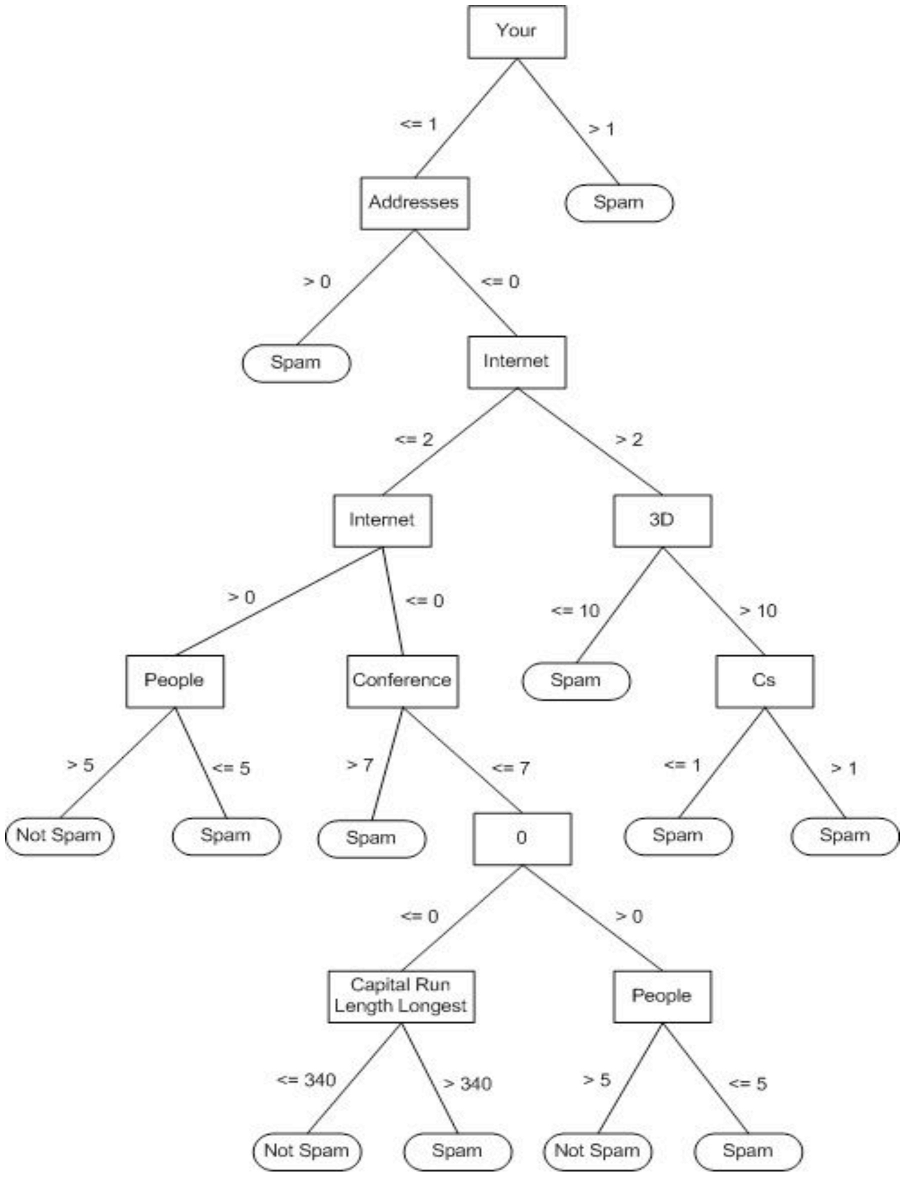
	Naïve Bayes	C4.5	KNN	Perceptron	Genetic	Voting
Errors:	337	195	145	270	537	141
Error %:	0.14646	0.08475	0.06302	0.11734	0.23338	0.06128
False Positives:	159	118	109	130	327	92
False Positive %:	0.11398	0.08459	0.07814	0.09319	0.23441	0.06595
False Negatives:	178	77	36	140	210	49
False Negative %:	0.19647	0.08499	0.03974	0.15453	0.23179	0.05408

The results of the training set are similar to those of the test set, but these results are generally better. This is to be expected, since the algorithms were fitted to this set and then applied to the test set. However, the genetic algorithm performs worse on the training set than the test set, and voting produced fewer false negatives in the test set than it did in the training set.

I have also provided graphical representations of the decision trees created by the C4.5 and genetic algorithms. Here is the tree generated by the C4.5 algorithm:



Below is the tree generated by the genetic algorithm. Notice the lack of similarity of splitting attributes between the two trees. More importantly, the genetic algorithm produced some presumably questionable results (and multiple attempts at reusing the algorithm consistently yielded the same tree). Note that the third-highest splitting attribute is "Internet", for which a value greater than 2 enters a subtree of two internal nodes and three leaves – but each of the leaves has the same result! Although this could be a result of using the free version of the GATree product, I am surprised that the algorithm did not prune that subtree into a single leaf node. Perhaps without the limitations of the free version, the algorithm would have produced better results throughout my research.



References

Aleshunas, John, and Ross Quinlan (2007). *C4.5 Decision Tree Induction*. Retrieved May 1, 2008, from http://mercury.webster.edu/aleshunas/MATH%203210/MATH%203210%20Source %20Code%20and%20Executables.html.

Bayes' theorem (2008). In *Wikipedia, The Free Encyclopedia*. Retrieved May 6, 2008, from http://en. wikipedia.org/w/index.php?title=Bayes%27_theorem&oldid=209271193.

Dunham, Margaret H. (2003). *Data mining: Introductory and advanced topics*. Upper Saddle River, NJ: Pearson Education, Inc.

Frank, Eibe, et al. (2007). *Weka 3: Data Mining Software in Java*. The University of Waikato. Retrieved April 26, 2008, from http://www.cs.waikato.ac.nz/ml/weka/.

GATree (2006). AHEADRM.com. Retrieved April 22, 2008, from http://www.gatree.com/index.html.

Han, Jiawei, and Micheline Kamber (2001). *Data mining: Concepts and Techniques*. San Francisco, CA: Morgan Kaufmann Publishers.

Hopkins, Mark, et al. (1999). *SPAM E-mail database*. Hewlett-Packard Labs. Retrieved April 22, 2008, from ftp://ftp.ics.uci.edu/pub/machine-learning-databases/spambase/.

Information entropy (2008). In *Wikipedia, The Free Encyclopedia*. Retrieved May 6, 2008, from http://en.wikipedia.org/w/index.php?title=Information_entropy&oldid=208004419.