Using a Genetic Algorithm to Select Features for Spam Filtering

Jeremy Buchmann

December 13, 2002

Abstract

Unsolicited Bulk Email, commonly called Spam, has become a daily problem for email users. It is not only annoying and sometimes offensive, it also wastes time and bandwidth. In this paper, I attempt to use a genetic algorithm to select a near-optimal set of words for spam filtering, using a neural network as a classifier.

1 Introduction

1.1 Motivation

I want to reduce the number of words that a classifier must use to make a classification as spam or not spam. An optimal word list could potentially reduce the time it takes to make a classification and improve the accuracy.

1.2 Background

Goldberg[2] and Holland[1] have written the principle books on genetic algorithms.

Yang and Honavar[3] compared a genetic algorithm/neural network combination to a neural network in their performance of feature subset selection for many different data sets. The genetic algorithm/neural network combination always achieved better results. They compared their results to other algorithms for feature subset selection and their method was usually as good or better than the others. Their work gave me the idea for using a genetic algorithm to select the important words in email.

There has been relatively little research done in the area of spam filtering. Most of the work that has been done has focused on using Naive Bayesian classifiers[4, 5] or memory-based classifiers[4]. To my knowledge, this is the first attempt at using a genetic algorithm to assist a classifier in spam filtering.

Drewes[6] used a backpropogating neural network to filter spam email and reported a successful classification rate of 99% on a test set[1]. His work forms the basis of the neural network portion of this experiment.

[1] His email corpus was different than mine.

1.3 Methodology

A genetic algorithm is a search algorithm modeled after the biological process of evolution. A population of candidate solutions called individuals are evaluated by a fitness function, and the best are selected to continue to the next generation of individuals. Some or all of the individuals that advance to the next generation are combined with other individuals and mutated to form hybrid individuals. This process of selection, combination, and mutation continues until an individual is found that meets the requirements of the search.

To apply a genetic algorithm to spam feature selection, I constructed a word list containing all of the words present in the emails and used a genetic algorithm to select which words were important.

1.4 Summary of Results

While the genetic algorithm provided a better solution than a set of hand-picked words, it failed to outperform a set of words composed of the most common one hundred words.

1.5 Paper Outline

Section 2 describes genetic algorithms and neural networks, and how I applied a genetic algorithm to the problem of spam feature selection. Section 3 presents the results I obtained, and Section 4 concludes the paper with a summary of the results and a discussion of future work.

2 Methodology

2.1 Genetic Algorithms

A genetic algorithm is a search algorithm modeled after the biological process of evolution. A population of candidate solutions called individuals are evaluated by a fitness function, and the best are selected to continue to the next generation of individuals. Some or all of the individuals that advance to the next generation are combined with other individuals and mutated to form hybrid individuals. This process of selection, combination, and mutation continues until an individual is found that meets the requirements of the search.

Each individual is generally modeled as a vector (a one-dimensional array) called a chromosome. Each element of the vector is called an allele. A chromosome is often a binary vector for convenience purposes, but more complicated representations can be used as needed. The selection of an appropriate way to represent a solution as a chromosome is one of the challenges of using a genetic algorithm.

A fitness function is used to evaluate each individual. Each individual is given a fitness value that indicates how fit it is as a solution. The fitness value is used by the selection and crossover operators to determine which individuals move on to the next generation and which ones do not.

After each individual in the population has been evaluated by the fitness function, those with the highest fitness are selected to advance to the next generation, and/or operated on to produce new individuals. The most common operators are selection, crossover and mutation.

Selection is an operator that simply chooses the best individuals and moves them on to the next generation. These individuals provide future generations with the best solution that is currently known.

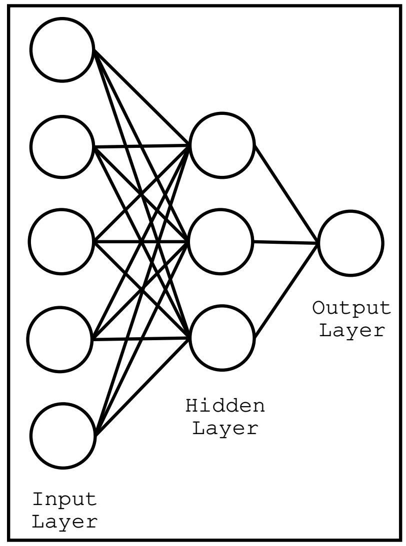

Figure 1: A three-layer neural network.

Crossover is an operator that, given two individuals, mixes their alleles to produce two new individuals. A common way of mixing is single-point crossover. Single-point crossover works by choosing one point in the vectors and swapping their alleles before and after that point to produce two new individuals. This technique can be extended to N-point crossover by choosing N points and swapping before and after all N points.

Mutation is another common operator that, given one individual, makes a small change in a random location in the chromosome to produce a new individual. Mutation provides genetic diversity beyond what is possible with crossover.

2.2 Neural Networks

An artificial neural network (simply called a neural network in this paper) is a highly parallel data structure that models, at a high level, the functioning of a biological brain. The neural network is composed of layers of nodes called neurons. Each neuron has an input, a threshold, and output weight. There are typically three or more layers; an input layer, one or more hidden layers, and an output layer. The input layer takes input from the feature vectors. The hidden layer receives weighted values from the input layer, determines if they reach a threshold value, and if so, sends the values through its own weights to output nodes or further hidden layers. The output nodes sum the incoming values and provide an output value that represents the classification of the feature vector. See Figure 1 for a diagram of a neural network.

To train a neural network, you quantify and normalize the values in a feature vector and copy those values into the input nodes of the neural network. Then, you copy the output value of that feature vector into the output node of the neural network. Finally, you tell the neural network to train itself, and it adjusts its weights and thresholds to match the input with the output. This is done for every feature vector in the training set. This process is then repeated until the neural network has adjusted its weights and thresholds enough to perform well. Each repetition is called an epoch.

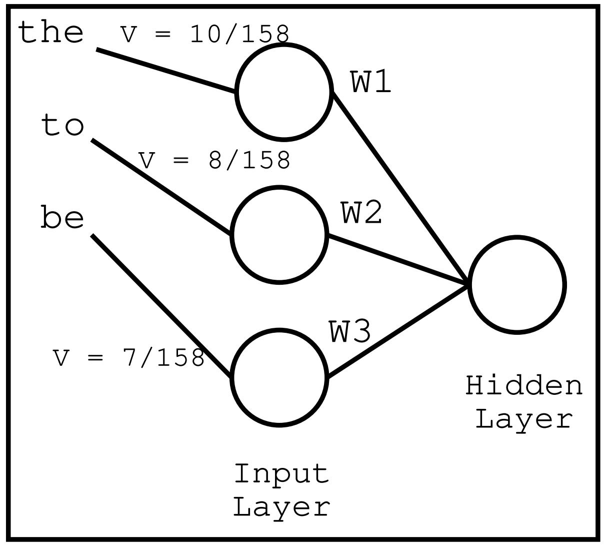

Figure 2: A small example portion of the system used to train and test the neural network.

2.3 Using a Neural Network as a Spam Filter

I constructed a list of words that contained every word in every email used in the experiment. This master word list was then lemmatized and ordered by the frequency of the words. The word that appeared the most was at the top of the list. Each entry in the master word list corresponded to an input node in a neural network. I chose a set of emails to be a training set and the rest were used as a testing set. To train the neural network, each word in each email is located in the master word list. Then, a value V is calculated as the number of times the word appears in the email divided by the number of words in the email. V is then loaded into the input node which corresponds to the location of the word in the master word list. When the training phase finished, I tested the neural network with the same procedure.

Figure 2 provides a visual representation of how the system works. The examples for V are shown in fractional form to indicate the method used to calculate V. For example, the word "the" was in the email 10 times, and the total number of words in the email was 158. The output weights are represented by W1, W2, and W3.

With many emails, the master word list can be very large ($>$ 25,000 words). This is too many input nodes, so I chose the most common N words, where N was approximately 3,000. Using a backpropogating neural network implementation with N=3,000, I achieved approximately 90% classification accuracy after 300 epochs.

2.4 Applying a Genetic Algorithm to the Feature Subset Selection Problem

Ninety percent is a pretty good classification rate, but can we make it better? In theory, yes, because those most common 3,000 words include words like "the", "to", "be", "and", "a", and "of", which probably don't provide the neural network with any meaningful values. To choose the words that provide meaningful values, we must perform feature subset selection.

Feature subset selection is computationally difficult. There are 2^n possible combinations of attributes, which makes exhaustive search undesirable for non-trivial problems. Genetic algorithms are good at quickly approaching a decent solution in high-dimensional space, and therefore should be good candidates for selecting favorable subsets.

The genetic algorithm works by supplying a subset of one hundred words that become the master word list for the neural network. If a word in an email is not in the genetic algorithm word list, then its input node is left with a value of 0. Each individual of the population is composed of a vector of integers of length one hundred. Each value is an index into the master word list, therefore each value represents one word. The word list composed from each individual is used as the master word list for the neural network to train with.

A problem with this representation is that duplicates in the chromosome can cause sub-optimal behavior in the neural network. For each set of duplicates, the real number of words the neural network has available is decreased by one. To prevent duplicates from occurring, I wrote a routine into the crossover function that checks the chromosome for duplicates. If any duplicates are found, one of the duplicates is replaced by a randomly generated number. Since this injects new diversity into the population, I didn't use a mutation operator.

3 Results and Analysis

For my first attempt at applying a genetic algorithm to this problem, I used a binary string chromosome of length n, where n was less than the number of words from all the emails. Each value in the chromosome corresponded to a word in the master word list, starting with the most common word. A value of 1 indicated that the word was in the master word list, and a value of 0 indicated it was not. Using n=6,000, I ran four separate instances of a simple genetic algorithm[2] for approximately thirty generations without any improvement in classification accuracy. The run time for each of these instances was approximately 24 hours.

Lacking any positive results, I changed my encoding to that described in Section 2.4. This encoding produced some mildly positive results. Figure 3 shows the minimum, average, and maximum fitnesses for the first 53[3] generations. You can see that all three fitnesses increased very mildly.

You may notice that the fitness curves in Figure 3 don't show the typical logarithmic-looking curve that genetic algorithms tend to exhibit. There are several possible explanations for this. First, the genetic algorithm only completed fifty-three generations, which may not have been enough time to converge on a decent solution. Second, the space from which the genetic algorithm was choosing the one hundred words may have been too large.

To create a benchmark for the genetic algorithm, I tested two other methods of selecting words. The first method was to manually select one hundred words based on my knowledge of spam. The second method was to select the one hundred most common words. In all cases, a list of one hundred words was used to provide input values to the neural network. The results of all three methods are shown in Figure 4.

As you can see in the figure, the most common one hundred words proved to be the most accurate method of word selection with a classification accuracy of approximately 70%. The genetic algorithm came in second with a classification accuracy of approximately 62%, and surprisingly, manual selection did no better than chance. Even though the most common words ("the", "to", "be", etc) seem to be too common to use for classification purposes, they provide a measurable amount of classification power. The genetic algorithm selected its one hundred words from the most common 4,096 words. In retrospect, this may have provided too large of a range for the genetic algorithm to exploit quickly enough.

[2] Goldberg's[2] canonical genetic algorithm.
[3] Time has permitted me to only use 53 generations worth of data.

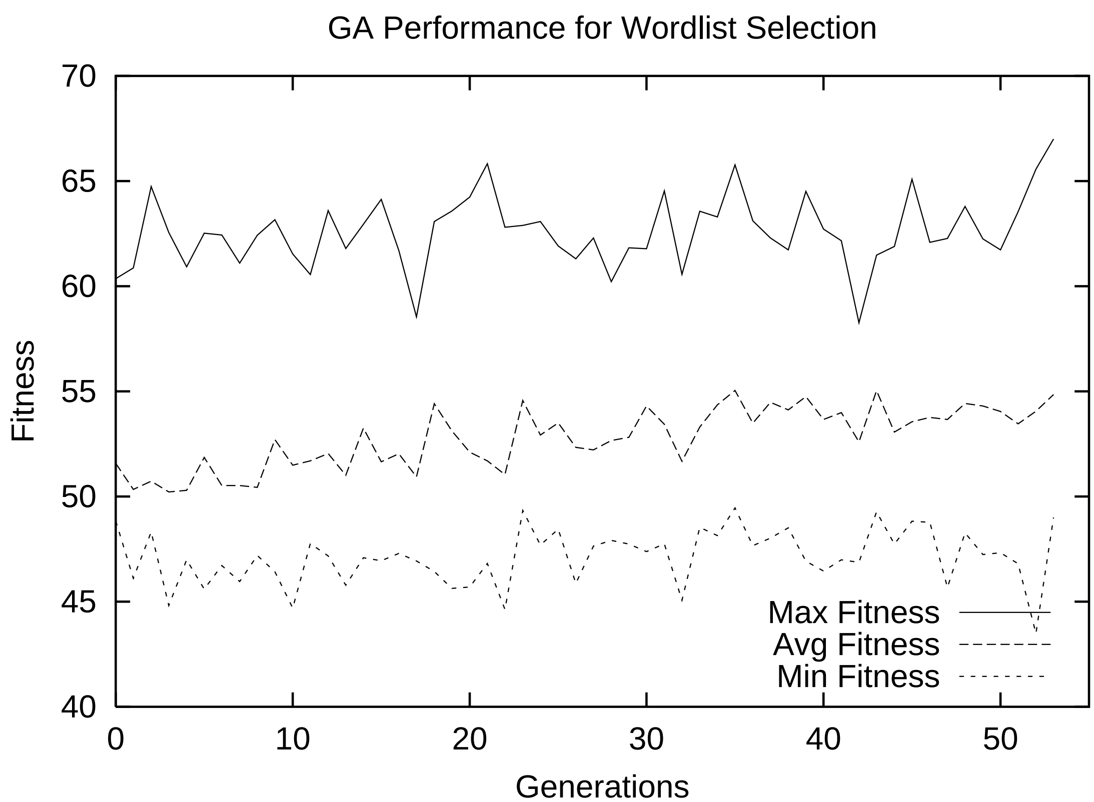

Figure 3: The minimum, average, and maximum fitness values.

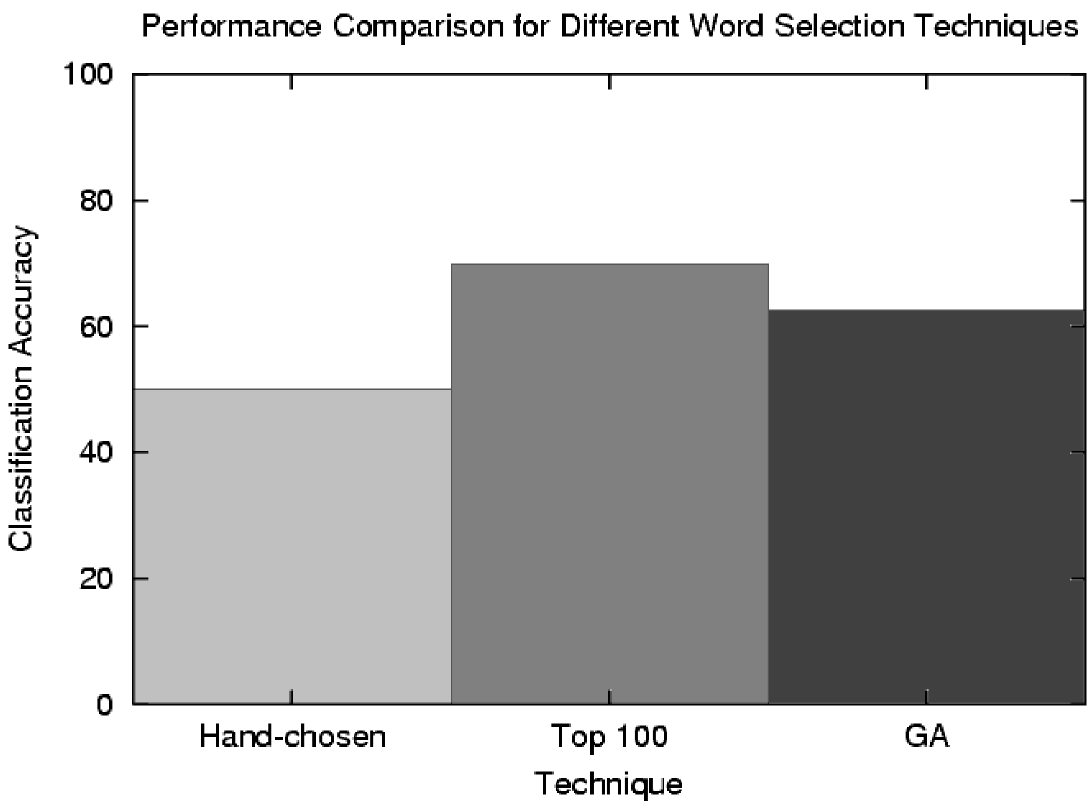

Figure 4: The classification accuracy for all three methods of word selection.

4 Conclusions

The genetic algorithm was able to make a measurable improvement in the classification accuracy of the neural network. Unfortunately, the improvement wasn't as good as just using the most common one hundred words. Despite the lack of good results, I still believe this technique can work. Previous work ([3]) leads me to believe that genetic algorithms can improve the classification of email. There are dozens of variables and further refinements of the technique that can be done. These may provide the ultimate answer.

The first change I would make would be to reduce the number of words the genetic algorithm could choose from. Two or three thousand might make a good number.

Another change that could be made is to change the neural network code. The current code is a straightforward backpropogating neural network which does the job, but can be very slow. In [3], the authors used a neural network called DistAI, which uses a different (and potentially faster) learning algorithm.

The type of classifier itself could also be changed. A Support Vector Machine (SVM) may work well as an email classifier since SVMs are known to work well in high-dimensional spaces. An SVM might make a good choice if it could be configured to learn quickly enough.

References

[1] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor: The University of Michigan Press.

[2] Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, New York.

[3] Yang, G. and Honavar, V. (1998). *Feature Subset Selection Using a Genetic Algorithm.* IEEE Intelligent Systems, 13:44-49.

[4] I. Androutsopoulos, G. Paliouras, V. Karkaletsis, G. Sakkis, C. Spyropoulos, and P. Stamatopoulos. (2000) *Learning to filter spam e-mail: A Comparison of a Naive Bayesian and a Memory-based Approach.* In Workshop on Machine Learning and Textual Information Access, 4th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD).

[5] Sahami, M., S. Dumais, D. Heckerman, and E. Horvitz. (1998). *A Bayesian Approach to Filtering Junk E-Mail.* Learning for Text Categorization - Papers from the AAAI Workshop, pages 55-62, Madison Wisconsin. AAAI Technical Report WS-98-05.

[6] Drewes, R. (2002). An Artificial Neural Network Spam Classifier. `http://interstice.com/drewes/cs676/spam-nn/`