Spam Filter: An Approach using Genetic Programming and Decision Trees

Aaron St.John, Cory Stuart, Michael West

Introduction and Problem Definition

Spam is the unwanted commercial email that seems to be clogging up everyone's inbox. Spam can range from emails selling prescription drugs, to advertisements for pornography, to emails trying to sell a way to "fire your boss" and start your own semi-legal business. Our problem was the design a classifier that could identify spam and filter it out without accidentally identifying legitimate email in.

For our project we chose to concentrate on the theoretical problem of identifying spam, rather than the software engineering problem of building a finished spam-filtering product. Therefore are goal is to design a classifier, but not to concentrate on integrating it with existing email clients or severs. This allowed us to concentrate of the computational intelligence part of the problem without getting bogged down in implementation details.

Another decision that was made early one was to use the ling-spam email corpus as a test data set. The ling-spam corpus is a collection of email from a moderated mailing list about the science of linguistics. This corpus is commonly used in spam research, and by using it we are able to more easily compare our results with the results that other researches obtained.

Model Research

The first step of the project was to make an overview of the state of the art in spam filtering to get an idea of the strengths and weaknesses of the available models, and

eventually decide which model was appropriate for our project. We took a two-pronged approach to researching models. First, we read academic papers on the subject of spam filtering. Second, we looked at existing anti-spam software that was currently available. This approach allowed us to see not only what techniques could be applied and were being researched, but also what techniques are being applied successfully.

In our research, we found that spam filtering is a very hot topic, and many different approaches were being used. For example, one popular spam filter, SpamAssassin, takes a multi-faceted approach. Each email is evaluated against a set of weighted heuristic rules (such as 'the email contains the word "free" or 'the header appears to be forged'). If a rule evaluates to true, its weight is added to the email's score. If the final score is above a certain threshold, the email is classified as spam. In this sense, SpamAssassin is using an artificial neuron as a classifier. However, the weights for SpamAssassin rules are usually optimized by hand, rather than by using a neuron training algorithm.

A more straightforward neural network can also be used to classify email. Several techniques exist for using neural network based classifier. The simplest, perhaps, is to use the presence or absence of a set of word as the input vector.

Another popular approach was Bayesian filtering of email. Bayesian filters learn the probably that a given feature (usually a word form the email) indicates either spam or non-spam using human classified email. These probabilities are then used to classify future email automatically.

Other simple techniques are also popular. For example, a user can employ a white list, allowing only email from known senders through, and deleting all mail from

unknown sender. This approach is obviously not optimal and can have a high rate of false positives. For example, if a friend changed his email address, he would no longer be on the white list, and thus his emails would be deleted. Contrasting the white list approach is the black list approach. Using a black list scheme, email sent from addresses appearing on the white list are classified as spam, and everything else is let through. This approach clearly has a high right of false negatives, and it would be very hard to generate a complete list of spammers.

Model Used

In the end, we chose to use a decision tree model. Decision trees are simple classifiers that are represented as trees. Each internal node contains a rule to test. Based on the results of the tests at each internal node, the algorithm follows a path down to a leaf node. Each leaf node is labeled with a class. When a leaf node is reached, the item being tested is classified according to the leaf node's label.

Decision trees can be easily adapted to text classification. Each internal node is given a word. If the text being classified contains the word specified by an internal node, the edge leading to 'true child' is followed. If the text does not contain the internal node's word, the edge leading to the 'false child' is followed. This is the type of decision tree we decided to use for our project.

After deciding to use a decision tree, it was necessary to decide how the decision tree would be generated. There are currently several algorithms used to generate decision trees based on a set of test data. One such algorithm we considered strongly was the C4.5 algorithm. The advantage of using C4.5 is that existing source code could be easily adapted for our project.

However, we reasoned that since genetic programming was designed to evolve trees, it could be used to evolve an optimal decision tree. Genetic programming is not commonly used with decision trees, and we felt it would be an innovative and interesting approach. In the end, using genetic programming to evolve decision trees was the approach we decided to take.

One advantage of using genetic programming is that several different criteria can be considered when generating a decision tree. For example, are program weighs the false positive rate of the decision tree more than the false negative rate, and also takes into account the size (and therefore efficiency) of a possible solution.

Another advantage is that we can start with zero knowledge about what features indicate spam, and which indicate legitimate male. This means that the algorithm is free to choose criteria that may be nonnutritive for a human. This approach can also tailor the classifier for a given person's email. For example, a doctor may get several legitimate emails about Viagra, whereas for most people, emails about Viagra are often spam. Using our approach, the algorithm would learn that Viagra is not an indicator for spam in the doctor, but is an indicator for spam for another person's email.

Algorithm Used

When developing our solution, we found that the straightforward approach to genetic programming would not be sufficient for our project. Therefore, our algorithm has several additions to the basic approach to genetic programming. Our complete algorithm is described below.

First, the email messages from our test data set are loaded into two separate corpuses, one containing spam and one containing non-spam. Each email is parsed into a

list of words. Words appearing on a stop list of very common English words (such as ‘the’ or ‘and’) are ignored. By using a stop list we can assure that our algorithm does not waste time evaluating words that are very likely to appear in both spam and non-spam messages.

Next, a population of randomly generated decision trees is generated. Each internal node is assigned randomly chosen words from the email. Each leaf node is assigned a random Boolean value (true for the class ‘spam’ and false for the class ‘non-spam’). Psuedo-code for the exact algorithm used to generate a random decision tree is given in Appendix A.

The fitness of each solution in the population is then calculated. Each tree in tested against 100 randomly chosen spam emails and 100 randomly chosen non-spam emails. These tests generate the false negative ratio and the false positive ration, each a number form zero (no false negatives or positives) to one (all false negatives or positives). The fitness function is then calculated as follows:

$$f(t) = ((1 - falseNegativeRatio) * FALSE\_NEGATIVE\_WEIGHT + (1 - falsePositiveRatio) * FALSE\_POSITIVE\_WEIGHT + SIZE\_WEIGHT - (size^2) / MAX\_SIZE^2 * SIZE\_WEIGHT) + 0.0001) / (FALSE\_POSITIVE\_WEIGHT + FALSE\_NEGATIVE\_WEIGHT + SIZE\_WEIGHT + 0.0001))^5$$

The weights used in this function are:

```
FALSE_POSITIVE_WEIGHT = 1.6;
FALSE_NEGATIVE_WEIGHT = 1.0;
SIZE_WEIGHT = 0.2;
MAX_SIZE = 500;
```

As you can see, false positives are weighted more heavily than false negatives. When designing a spam filter, it is important to err on the side of caution. If a false negative occurs (a spam email is classified as legitimate email) it is simply an annoyance, and the user has to manually delete the email. However, if a false positive occurs (a piece of legitimate email is classified as spam), the user may lose important data. He or

she could lose an important email from a family member, or miss an important business email.

The size of the tree is taken into account so that more efficient trees are favored over larger, less efficient trees. Any tree with a size large than MAX_SIZE is automatically given a very low fitness score. This ensure that it will not be considered for breeding, as very large tree slow the algorithm down significantly while adding little value.

The fitness is then divided by the sum of the weights to produce a number between zero and one. The number is taken to the fifth power to help had more of a distinction between two trees that are both very good candidates.

Next, a breeding population of the fittest trees in generated using the roulette wheel algorithm. That is, if a possible solution's fitness is 12% of the population's total fitness, it has a 12% chance of being added to the breeding population at each iteration.

After generating a breeding population using the roulette wheel algorithm, several new random trees are also added to the breeding population. This is done so that the set of words that are considered does not become stale. By randomly adding new trees, we ensure that new words are constantly being added to he population.

Trees from the breeding population are then bred to generate a new working population of trees. Standard one-point tree crossover is used for this operation.

After generating a new working population, the population is subjected to mutation. Each tree in the population has a 75% chance of being mutated. Although this is a very high mutation rate, it produced better results in our tests than lower mutation rates. To mutate a tree, a random node is picked. If the randomly picked node is an

internal node, its word is changed to a new randomly picked word form the email corpuses. If a leaf node is picked, its value is toggled.

Next, the best solution from the previous generation is added into the new population. This insures that a promising solution does not become lost, a problem that was common in early tests.

Finally, the process starts over again from calculating the new population's fitness rate. The loop continues until a predetermined number of generations are computed, at which point the best tree is saved, and the program exits.

Numerical Results

We tested the program using several different values for the constants, and determined that a population of 75 trees with 15 new random trees added every generation, and a 75% mutation chance was the best mix of speed and accuracy. We calculated 100 generations during each test. Our tests showed that our method toward spam filtering does work, although perhaps not as well as we had hoped.

Numerically, the evolved trees had a 14.91% false positive rate and a 19.77% false negative rate. Most commercially available anti-spam packages, however, boast false negative rates less than 5% and false positive rates less than 1%. The trees evolved in our tests had an average size of 74.4 nodes. Complete statistics for our tests appear in Appendix B.

Suggested improvements

Looking at the numbers, it's obvious that there is room for improvement in our program. One of the most obvious ways to improve the program would be to incorporate

basic optimizations designed to work with genetic programming. For example, our program tends to produce very different trees with each run. This leads me to believe that using subpopulations could help dramatically improve the performance of our algorithms.

Another fairly obvious area is to use other, better-established algorithms to generate the decision trees. Perhaps a hybrid solution would work well. For example, rather than adding random trees at each step of the evolution process, trees generated using the C4.5 algorithm could be added.

Other improvements could be made by better utilizing the trees that are generated, rather than by generating better trees. For example, each piece of email could be evaluated against 3 different decision trees. The email would only be declared spam if all three trees concurred that it was spam. Alternatively, the email could be classified, as spam if two out of three trees classified it is spam. The decision trees could also be used in conjunction for other programs. For example, the decision tree's classification of an email could be used as a SpamAssassin rule.

Conclusions

Although our numerical results are not as good as they could be, I believe they are encouraging, and show that using genetically evolved decision trees for spam filtering has a place in the software world. I believe with work, our project could be made into a powerful anti-spam solution.

There are several advantages to our current approach. For example, it allows a spam filter to become highly specialized and filtering a specific person's mail. When doing tests, I was surprised to that the word 'university' was often determined to be a

very strong indicator that a word was not spam. While most email that I get containing to word ‘university’ is spam, trying to sell me a diploma at a non-accredited school. However, the many of the users of the Linguist email list where university professors, so the word ‘university’ did indeed tend to indicate that a message was legitimate.

Another advantage is that our solution can be very easily adapted to giving mail arbitrary classifications (i.e., business email vs. personal email) so long as an appropriate training set is provided. It is also very easy to expose the knowledge contained in a decision tree, and once a decision tree is generated, email can be classified very quickly.

In conclusion, I believe that although our approach in not perfect, it does offer a new and valid way to classify spam. From our research, the evolution of decision trees is a little explored area, which offers promising results with further research.

Appendix A: Psuedo-Code to Generate a Random Decision Tree

```
size = 0;
Stack s = new Stack();
head = new InternalNode(Random Word);

s.push(head);

while(s in not empty) {
    InternalNode n = s.pop();

    //set the false child
    if(random() < 0.5 && size < 500)
    {
        n.setFalseChild(new InternalNode(Random Word));
        s.push(n.getFalseChild());
        size++;
    }
    else
        n.setFalseChild(new LeafNode(Random Boolean Value));

    //set the truechild
    if(random() < 0.5 && size < 500)
    {
```

```
            n.setTrueChild(new InternalNode(Random Word));
            s.push(n.getTrueChild());
            size++;
        }
        else
            n.setTrueeChild(new LeafNode(Random Boolean Value));
    }
```

Appendix B: Numeric testing results

Test #	Tree Size	False Negatives	False Positives	Fitness
Test #1	15	21.41%	6.47%	0.5475
Test #2	21	5.61%	21.39%	0.4639
Test #3	83	44.28%	4.52%	0.3575
Test #4	37	3.12%	39.43%	0.2589
Test #5	135	39.92%	1.82%	0.4227
Test #6	23	4.37%	20.48%	0.4905
Test #7	33	3.74%	24.71%	0.4311
Test #8	117	4.16%	15.34%	0.5697
Test #9	137	42.83%	3.86%	0.3700
Test #10	143	28.27%	11.11%	0.3932
Averages	**74.4**	**19.77%**	**14.91%**	**0.4305**