Practical 2: Classifying Malicious Software

Gunnar Plunkett, Jeb King, and John LaVelle
glplunkett@college.harvard.edu, jking@college.harvard.edu, johnlavelle@college.harvard.edu
jebking, gunplun, johnlavelle

March 10, 2019

1 Technical Approach

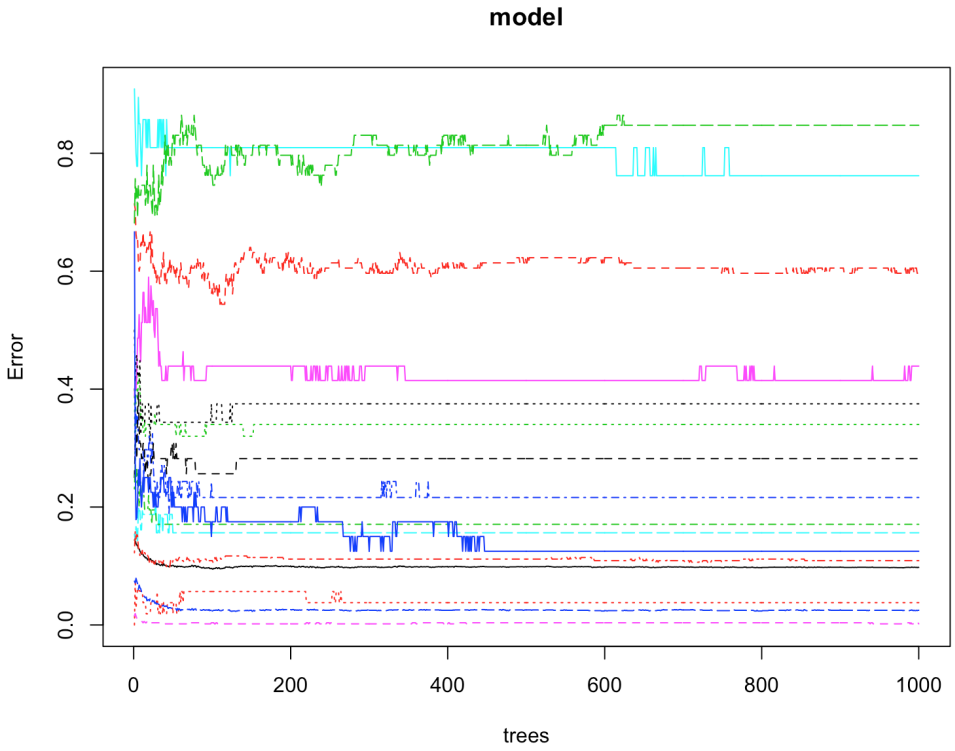
We ultimately decided to use a Random Forest Classifier because of its resistance to overfitting. The RFC uses bagging to classify the data; that is, it uses an ensemble of B decision trees, assumed to be overfitting the data, and averages, i.e. takes the majority, of their resulting predictions to form a decision boundary. Each decision tree takes as data a random subset, with replacement, of n different points $\{(\mathbf{x}_i, y_i)\}_{i\in[N]_b^n} \subseteq \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ from the training data set as well as a random subset of m features $S_b \subseteq S$ to analyze and from which produce a predictor f_b. This gives us the random forest predictor:

$$\hat{f}(x^*|\{(\mathbf{x}_i, y_i)\}_{i=1}^N) = \text{mode}_{b\in[B]}(f_b(x^*|\{(\mathbf{x}_i, y_i)\}_{i\in[N]_b^n}, S_b))$$

Every decision tree is built up to a maximum depth, forcing decisions based on attributes to try to maximize the gain from the entropy of the target attribute, in our case the type of virus. Entropy E is calculated using the following two equations, where C is the set of c virus categories p_i is the proportion of training data labeled C_i in a given category, X is a set of categories for a feature ins S_b, and $P(c)$ is the proportion of data in one of those categories:

$$E(T) = \sum_{i=1}^{c} -p_i \log_2 p_i$$
$$E(T, X) = \sum_{c\in X} P(c)E(c)$$

Every decision tree node based on a category T will end up with children based on X that maximize the formula $E(T) - E(T, X)$, i.e. information gain.



Hyperparameter selection: We tuned our hyper-parameter for the amount of trees with several different values. See Table 2. We just went ahead and tired a few different values on different levels of magnitude, and decided to stay lower since the accuracy tended to get worse or not improve at all after 250 trees. The plot included above reveals the error based on different factors (categories) predicted by the random forest model. It doesn't look very great on some factors, as there was not very much improvement after around 100 trees. You can see the downward slope in the 0 to 50 range of the dashed blue factor line and the solid black line. There were a few factors that our model did not know how to predict well at all (the lines near the top).

We started with the simple staff-provided code and decided we needed more features: a simple and perhaps good feature set would include the total amount of system calls for each type of system call. Adding in these features allowed us to attain a Kaggle public score of 0.81149 and a private score of 0.81578. These features would work well since they take into account more of the environment of the virus executable. Although we did not account for extremely specific features like how many times a certain call was made immediately after another specific kind of call, the total amount still performed well enough to get us to #31 on the leaderboard.

2 Results

See Table 1 for Results. After getting validation scores on the three different models we used, we ended up only submitting the set of predictions from our Random Forest Classifier with 500 trees. This submission performed reasonably well (0.81149 accuracy) even when compared to the

Model	Acc.
Bigram Baseline	0.77368
Most Frequent Baseline	0.40621
Naive Bayes*	0.59
Logistic Regression*	0.69
Random Forest Classification	0.81149

Table 1: Results from our different models as well as the Kaggle baselines*Accuracy computed using 90-10 validation.

# of Trees	Acc.
10	0.9061489
50	0.9288026
100	0.9126214
250	0.9190939
500	0.9093851
1000	0.9126214
2000	0.9093851

Table 2: Validation scores based on the amount of trees in the Random Forest model.

approximately 0.91 accuracy it received on 90-10 validation. Notably, we received 0.81578 accuracy for our final RFC model on the public data, indicating minimal over-fitting since there was a <1% difference in accuracy. This would seem to indicate that at approximately 0.81 accuracy, the biggest problem with the model's prediction is overfitting.

3 Discussion

We needed to classify malware.

The data set given was a bunch of XML files describing system calls made by executable files (virus and non-virus), so at first we needed to find a way to turn these files into features for a datum.

For all of these models, we divided the provided into 90% training data, 10% evaluation data.

The first approach we took was a naive bayes' regression, on just the number of system calls taken (also using the prior distribution of the viruses, as taken from the given material). The results were really bad, with 100% of the guesses being just the 8th virus, which was the most frequent virus in the prior distribution. From here we thought "how do we add more features to the model?" We weren't certain that we could add features in the long-term and have them be uncorrelated with one another (a requirement of the Naive Bayes Distribution), so we decided to switch to a multinomial logistic regression, in order to better our long-term modelling prospects.

So, we added more features to the featureset: the amount of system calls made for each type of

system calls. To do this, we wrote another feature-extraction function that would work with the given starter code. Very simply, it counted the amount of times each system call was made (for example, load-image or set-value).

A small fix that was completed was to remove the subject id and the sequential id from the feature set (these inadvertently could affect the model by throwing in dummy data that really didn't have anything to do with malware: I remember in lecture that there was a model for predicting an illness that ended up performing well just because the ID number of the patients ended up correlating well with the disease status, even though this was incorrect).

When we ran this featureset on the multinomial logistic regression, it did improve, but not by much. When we looked at the feature-set that we added, we realized that there were a lot of areas that had a lot of zeroes (meaning that in large parts of nearly every single sample many commands were not being used). Because of these "holes" in the data, we thought that this might might mean that bagging the data would dramatically improve the results we received.

So, we switched to a random forest categorization model, still using the same feature set as the previous logistic regression, the counts of every system call being used, as well as the total number of system calls used overall. Initial tests of this found enormous improvement on the evaluation data. So, we ran a test on the number of trees to tune that hyperparameter and we found that the appropriate number of trees was 100.

Because of the model's performance on our own evaluation data and the Kaggle data was relatively high (around 90% on our evaluation data, and around 81% on the kaggle data), we think that this solution is a good solution.