# CS 181 Practical 2: Predicting Classifying Malicious Software

Meriton Ibrahimi
meritonibrahimicollege.harvard.edu
Kaggle: @meriton

March 9, 2019

## 1 Technical Approach

### Feature Engineering

To successfully classify malicious software from files of executable code, we need to extract and distinguish predictive features. Proper feature engineering is a mix of utilizing expert knowledge on the generated data with a bit of trial and error. We group our features for each executable file as follows:

- Endpoints - 30 binary features indicating the first and and last function calls
- Total - an integer valued feature identifying the total number of functions calls
- Functions counts - 102 integer valued features of the total number of calls for each function
- n-grams - 8225 binary features of n-grams of functions calls where $n \in 2, 3, 4$

In short, this is a massive number of features, far exceeding the number of observations (executable files). The idea behind subsetting these features is to find the set most predictive of the software class. Intuitively, it seems reasonable that perhaps software have distinctive "signatures" captured by the first three groups of features i.e. perhaps malicious software make a large numbers of specific feature calls. The n-grams are less intuitive but help to more holistically encompass the list of executable files by linking specific sequences of calls; however, the number of n-gram features is massive and needs to be trimmed to the most descriptive and powerfully predictive.

### Feature Selection

Random forests (RFs) are an extension of decision trees which work to reduce model error in two ways: 1) through boostrap aggregation (bagging) of multiple decision trees, RFs reduce variance in the model; 2) by randomly selecting features from which to choose potential splits at every node of the tree, RFs help to eliminate the greedy algorithmic search still present in bagged decision trees. As a result, RFs are among the most popular models in the machine learning community

for their performance, robustness, and ease of use. RFs, however, can also be used for feature selection. Consider a RF built of $D$ features with accuracy $\alpha$. Now, consider a different RF built on the same $D$ features but where feature $k \in 1..D$ is randomly permuted over the observations; say the latter RF achieves an accuracy $\beta$. If feature $k$ were a significantly important predictive feature, we would expect $\alpha - \beta > 0$. where the magnitude of the drop in accuracy is greater for more important features. For unimportant features, we would expect an analogous permutation and measure of change in accuracy to be zero. Thus, we can perform this methodology over all 8225 n-gram features to determine the most important features. Carrying out this feature selection over 3-fold cross validation where each RF is built on 300 enesmbled trees and where the maximum possible depth is 300, we identify the 50 most important n-gram features.

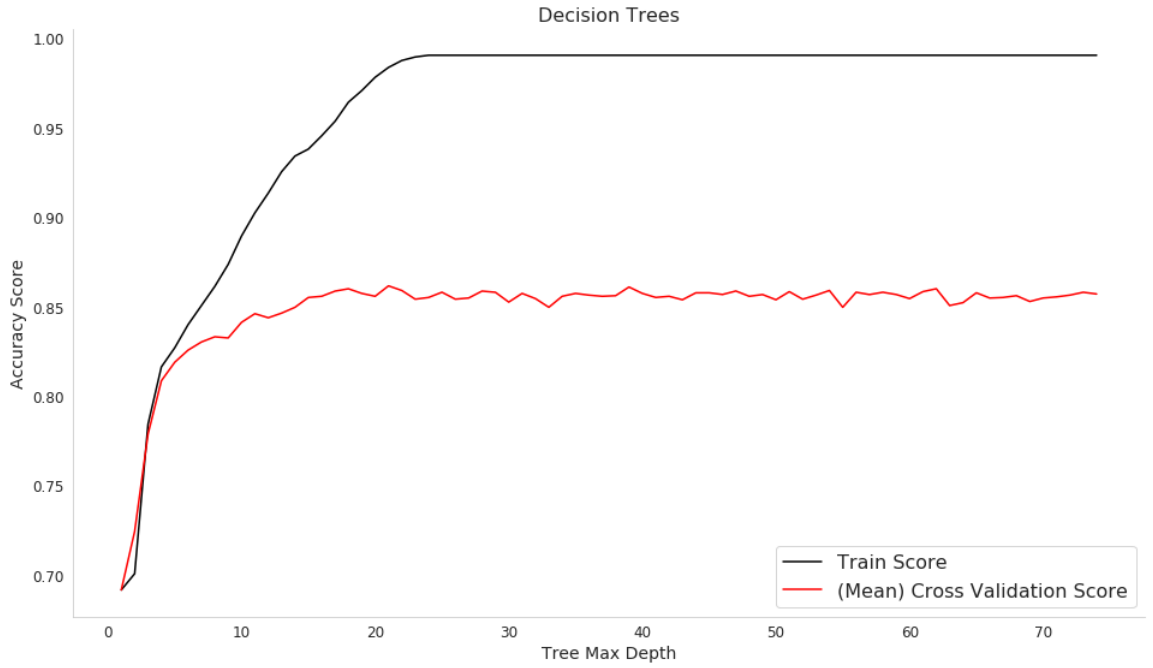
## Modeling and Model Selection

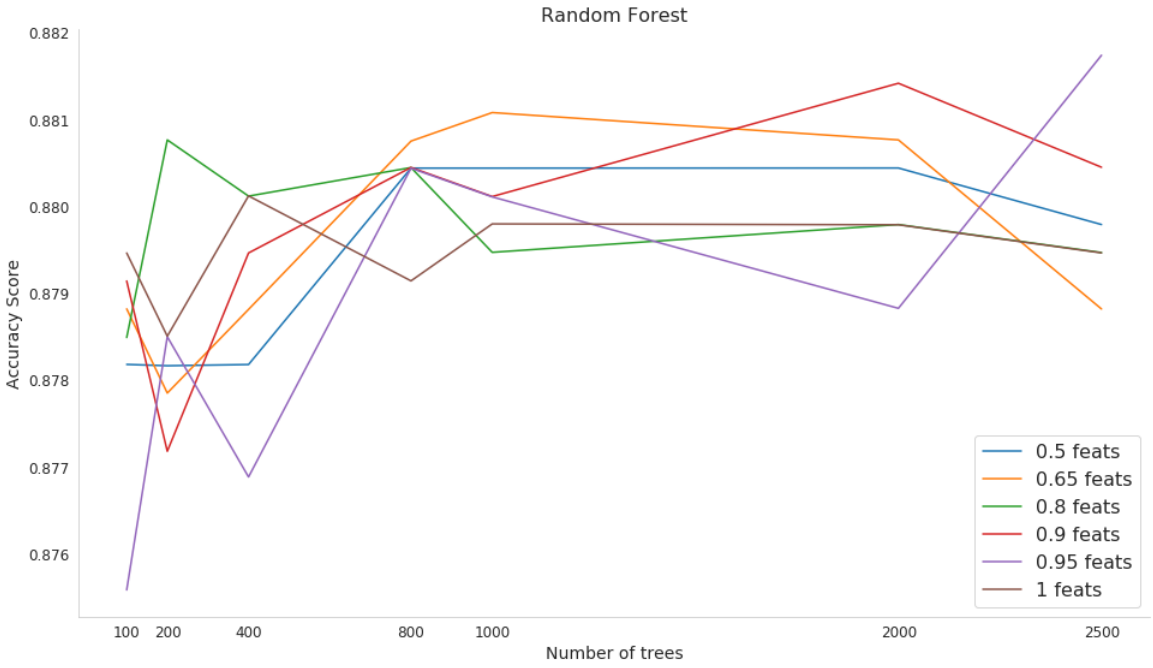
While we trimmed down to the number of features significantly with the above approach - from a total of 8,358 (=30+1+102+8,225) to 183 (=30+1+102+50) - the number of parameters is still fairly large, so we should consider regularization and cross validation where and when appropriate.

We ultimately fit a number of models. For each model, we performed 3-fold cross validation to determine optimal hyperparameters and regulation strength - when applicable. Further, for each model, if applicable, we carried out cross validation twice, once without class weights and once with class weights. Models we fit include: Logistic Regression (LR), Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), Decision Trees (DT), Random Forest, Adaboost,and XGBoost.

We started with LR, LDA, and QDA, but found that these models were not flexible enough for the constraints of the data. The data is very sparse and these latter models are fairly ill-defined in this space, particularly LR. As a result, we moved onto tree based models which offer considerable more flexibility. We do so cognizant of the fact that flexibility comes with the need to address and prevent overfitting.

For our dive into tree based models, we began with the basic DT model. In cross validating for optimal depth, we noticed that performance flattens out after a depth of 20. A deep DT model will achieve low bias in training, but because it is deep, it is also prone to being overfit and will produce considerable variability for the model. As a result, cross-validating will show when increasing depth no longer reduces error (and when it may even begin to increase error). At a depth of 20, we notice that the DT model is overfit; thus, it has low bias and high variance. We aim to reduce the error by considering a RF model. As mentioned, a RF looks to improve upon a DT model by maintaining the low bias of individual overfit trees but reducing the variance present by an average of boostrapped and ideally uniqulely different trees, thereby decreasing overall error. For the RF model, we optimize of the number of trees (rarely, however, is more trees worse) and the percentage of features from which to randomly choose at each split.




A RF model uses uses an ensemble of trees to build a low bias, low variance model by reducing the variance of low bias, high variance base models. In Boosting, we take the opposite approach. By using an ensemble of high bias, low variance individual tress built consecutively on top of the previous tree's error, Boosting looks to build a low bias, low variance final model. In examining the DT models, we notice that a DT model of maximum depth equal to 3 or less is an ideal candidate for the base learner of the Boosting model as a high bias, low variance model. From there, we performed cross validation to determine the optimal learning rate between models and

| Model | Accuracy via 3-Fold CV |
|---|---|
| LDA | 61.86 |
| QDA | 83.28 |
| Decision Tree | 85.51 |
| Random Forest | 88.17 |
| AdaBoost (Base Tree Depth=1 | 75.25 |
| AdaBoost( Base Tree Depth=3 | 81.83 |
| XGBoost | 88.73 |

Table 1

how many iterations of the base learner we should have - i.e. how much we should adjust weights of missclassified points and how many trees to build on top of one another.

We finally move over to eXtreme Gradient Boosting (XGBoost). This falls under gradient boosting like the previous Adaboost, but XGBoost has a number of improvements over traditional gradient boosting algorithms. Namely, XGBoost has built in tree pruning to tackle the greedy algorithmic nature of trees, is faster, and has build in regularization. This means that XGBoost is incredibly flexible, but this also means that we have a number of hyperparameters to tune. We need to cross validate for the number of trees to use, regularization strength, learning rate, and maximum depth of trees.

## 2 Results

Results of all our top-performing models (calculated on 3-fold cross validation of the training set) are displayed in Table 1. We see that our best model is the XGBoost followed closely by our best RF model. We see that the better performing tree models strictly out compete the less flexible linear and quadratic models. Intuitively, this makes sense as the features and feature space are large, sparse, and complex. We submitted several predictions to Kaggle. Unfortunately, our best performing model in XGBoost did not finish cross validation for tuning hyperparameters before the deadline. As a result, our best performing Kaggle submission was our best RF moodel. That RF model achieved a public leaderboard score of 81.63% before jumping 30 places to 82.06% on the private leaderboard showing our cross validation and parameter tuning worked to prevent high variance while maintaining relatively low bias.

## 3 Discussion

We present our thought process throughout this report. Ultimately, we take the approach of starting with relatively simple models before iteratively moving towards more complex and flexible models. We do so in an intuitive fashion. In need of a more flexible model over LR, LDA, and QDA, we move over to a base DT model. We see relatively low bias in this latter model, but we continue to build on top of the DT model to achieve models with not only low bias but also low variance.