Practical 2: Classifying Malicious Software

Team 16: WJKKKW
Xinyuan Wang wang_xinyuan@g.harvard.edu
Rui Zhao ruz466@g.harvard.edu
Yijun Zhou yijun_zhou@g.harvard.edu

1 Technical Approach

1.1 Feature Engineering

Since we only have training and testing XML files containing execution histories, we need to extract useful features first. We tried extracting all system calls in the training data set. We wrote a function "all_system_call_count" that found and built each system call ever appearing in any XML document of training set as a feature; 102 features were generated in total. We stored the numerical values in the feature matrix. This is an exhaust but most convenient way analyzing the system calls of each document. Since we have 3086 training examples, the number of features we got - 102 - could still be considered as a reasonable number.

We were not satisfied with the current 102 features, and wanted to add some keyword-based features containing information of the content of certain system calls. We did some research to find out the pattern of specific malware, and got two new features that might be helpful to our model.
a) exist_VBA_getusername: Find if keyword "VBA" exists in the "key" attribute of "open_key" system call. If so, find if "get_username" system call appears after "open_key" call. Set this feature as 1 if the above condition is satisfied, and set 0 otherwise. This feature could help us to filter the majority of "VB" class under our observation.
b) exist_Swizzor_0002DF: Find if the sequence "0002DF01-0000-0000-C000-000000000046" exists in the "key" attribute of "open_key" system call. Set this feature as 1 if it is satisfied, and set 0 otherwise. This feature could help us to filter the majority of "Swizzor" class under our observation.

1.2 Model Selection

During the process of model exploration, we tried out many methods, including Logistic Regression Classifier, Neural Network, Support Vector Classifier, Decision Tree, Ensemble methods including Random Forest Classifier, Extra Tree Classifier and Gradient Boosting Classifier, as well as Bayesian approach including Gaussian Naive Bayes. On the purpose of comparison, we used Categorization Accuracy as the criterion, which is the proportion of the number of correctly classified examples in the total number of examples. We randomly split the original training set into 70/30; 70% is the new training set and the remaining 30% is the new testing set. In the following, all models are trained on the new training set and evaluated on the new testing set.

1.2.1 Non-Bayesian Approach

- Logistic Regression Classifier:

  Logistic Regression Classifier is used to estimate the probability of a mutinomial response based on one or more predictor variables. Logistic Regression Classifier uses sigmoid function to map the value in the real space into [0,1] interval. We tuned the hyperparameter with 5-fold cross-validation with α from 10^{-4} to 10^{2} to determine an appropriate regularization penalty. At $\alpha = 0.1$, Logistic Regression Classifier had the best Categorization Accuracy of 0.85529. For this part, we used the Logistic Regression package in sklearn.linear_model.

- Neural Network:

  Neural Network consists of multiple non-linear hidden layers with several nodes on each layer. For each node it has a threshold function to determine if an input data could pass or not. Here we set the parameter solver to be lbfgs, which is an optimizer in the family of quasi-Newton methods. Using 5-fold crossing validation, we found the best parameter at alpha = 0.01, hidden_layer_sizes=(15, 17), random_state=5. MLP Classifier had the best Categorization Accuracy of 0.79590. For this part we used the package sklearn.neural_network.MLPClassifier.

- Support Vector Classifier:

  SVC is a maximize-margin based classifier. Given training data, with marked categories (in this case, 15 categories), SVC builds a model assigning new data to one category or others. An SVM model is a representative of the examples as points in space, mapped so that the examples of separate categories are divided by a clear gap that is as wide as possible. After cross-validation tuning, we got the best model at C = 1, with best Categorization Accuracy of 0.80886. For this part we used the package sklearn.svm.SVC.

- Ensemble Methods – Random Forest Classifier & Extra Tree Classifier:

  As in bootstrap aggregating, Random Forest(RF) and Extra Tree(ET) Classifier are meta estimators to build a number of decision trees on bootstrapped training samples. The only difference is that ET fits a number of randomized decision trees. Then averaging would be used to improve the predictive accuracy and control over-fitting. Compared with RF, additional randomness is added for ET during the process of selecting thresholds for randomly selected features to compute further splits of the decision tree. Extra Tree has the advantage of further reducing variance, but may sometimes resulting in increasing bias.

  We tuned the n_estimators from 5 to 30, and max_features from 20 to 100 to guarantee the effectiveness and efficiency. At n_estimators=25 and max_features=60, Random Forest Classifier has the best categorization accuracy of 0.89417. At n_estimators=25 and max_features=40, Extra Tree Classifier has the best Categorization Accuracy of 0.89093. Two models have very similar outcome. For this part we used the package sklearn.ensemble. RandomForestClassifier and sklearn.ensemble.ExtraTreesClassifier.

1.2.2 Non-Bayesian Approach

- Gaussian Naive Bayes:

  Naive Bayes method is a supervised learning algorithm based on applying Bayes theorem. It is called naive because it assumes independence among the features. Naive Bayes classifiers

Model	Categorization Accuracy
Logistic Regression	0.85529
Neural Network	0.79589
SVM	0.80885
SVM with prior	0.79806
Decision Tree	0.85745
Random Forest	0.89416
Random Forest with prior	0.87904
Extra Tree	0.89092
Gradient Boosting	0.87041
Gaussian Naive Bayes	0.28077

Table 1: Classifier models with corresponding categorization accuracy.

sometimes work quite well in some real-world situations like spam filtering. Meanwhile, Naive Bayes classifiers can be extremely fast compared to more sophisticated methods. However, naive Bayes is usually a bad estimator with poor probability outputs. Here we use the Gaussian Naive Bayes approach, which assumes the class-conditional distribution is Gaussian distribution. We found out the best Categorization Accuracy of 0.28078, which is not as good as other fitted models. For this part we used the package sklearn.naive_bayes.GaussianNB.

- Others:

  We also tried to incorporate the prior information of the distribution of malware classes in the training data into some other models, by setting the parameter class_weight to be the corresponding proportions of each class. Through doing this, the Categorization Accuracy of SVM decreased from 0.80886 to 0.79806, the Categorization Accuracy of Random Forest decreased from 0.89417 to 0.87905. The result shows that the prior information may not be helpful for the model fitting. The reason could be that the prior distribution for the train set is different from that for the test set.

2 Results

We have created and submitted a set of predictions on kaggle. After trying both malware-oriented feature engineering and model selection, our final method gives reasonably good performance. The tunning plot of n_estimators and max_features w.r.t Categorization Accuracy for Random Forest Classifier (the best model) are shown in Figure 1. The summary of our results can be seen in Table 1.

3 Discussion

We started with extracting features from the train set. The first came to our mind is to analyze system calls (i.e. the tags inside the XML documents). Usually the attacker would pretend to be a normal software but embeds malicious code that could bypass perimeter defenses and later do illegal things on victim's system. Therefore, we believe the appearance and frequency of some system calls would definitely contain useful information. We extracted all system calls that ever shows up in the training files and got 102 features in total.

After the basic feature extracting step, we began the model exploration process. We started with Logistic Regression, with the categorization accuracy of 0.85529, which is a relatively high

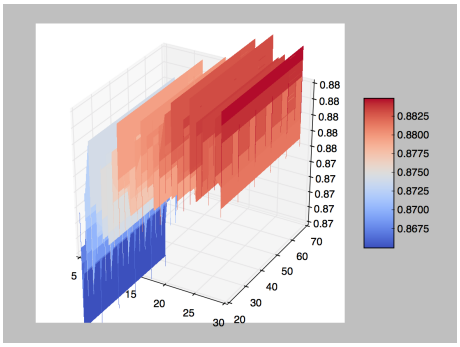

Figure 1: Tuning of Random Forest with best accuracy at n_estimator = 25, max_features = 60

score. We thought the result might be better if there is more flexibility, so we then implemented the multilayer perceptron (MLP) algorithm, which is a feedforward artificial neural network model. After the tuning process, the accuracy for MLP is 0.79589. From a different perspective, we thought about trying out with Support Vector Classifier, which classifies different classes using max-margin method. We got accuracy of 0.80885. To improve the result, we tried to use Decision tree as well. At the optimal parameter, we got the accuracy of 0.85745, which is the highest accuracy among all the models we have so far.

All the methods above are simple classifiers. So we started examining some ensemble methods. We first tried Random Forest, which builds many decision trees through bootstrapping data. The advantage of Random forest is to decrease variance without increasing bias. The final accuracy turned out to be 0.89416, which was the highest so far. Next we carried out Extra Tree, which has the advantage of further reducing variance by adding additional randomness, but at an expense of bias increasing. The accuracy here is 0.89092, which is not so good as Random Forest. Besides bagging methods, we also tried out Gradient Boosting, which produced a prediction model in the form of an ensemble of comparatively weak prediction models, with the accuracy of 0.87041.

All the models above did not take Bayesian approach into account. Next, we started to try Bayesian models and included class prior for some classifiers. As shown in table 1, we can see that Gaussian Naive Bayes had a low accuracy of 0.28077. We also tried Random Forest and Support Vector Classifier with class_weight parameter equals to class prior. Both outputs are worse than non-Bayesian approach. Therefore, Random Forest Tree without Prior became our final model.

After running all the above models using the basic 102 features, we turned our direction to feature engineering. We noticed that we didn't have any content-based features, so we tried to dig in that. After some research, we understood characteristics of the top two commonly occurring malware - Swizzor and VB. We got two new features based on that as explained in **1.1**, so we reran the models with 102 features + 2 new features individually and then the whole 104 features. However, our test accuracy actually dropped a little in these cases. Therefore, we decided not to use the two new features.

In short, we extracted all system calls in the training set to generate 102 numerical features. We explored some non-Bayesian models as well as Bayesian models. It turned out that the Random Forest model without class prior after tuning performs the best. We also did some feature engineering, but later decided not use the newly found features since they were not helpful. We finally submitted our non-Bayesian Random Forest model with the 102 features to Kaggle, and got score 0.82263 which beats the two baselines.