Study of High-Dimensional Classification

Siddarth Viswanathan

Summary

Code which can be added or removed from software to harm the intended use of a software system are considered malicious executables. It is therefore important for the correct functioning of software systems to efficiently detect malicious executables especially since thousands of new malicious executables are designed each year. Using R and a dataset of 373 labeled executables each with 531 described attributes we study the performance of four types of malicious executable classifiers and find the support vector machine the most successful algorithm for generalizing to new data.

Introduction

Malicious executables are often used to destroy the capacity and information of computer systems thereby making them unusable. Malicious executables also can be used to gather information without the knowledge of the user. It is vital in modern society to be able to limit the damage done by malicious executables and for this purpose many technologies have been succesful for malicious executable detection [1]. Continued study of malicious executable detection is required since eight to ten malicious programs are designed daily each with their own signature [2].

An important aspect of malicious executables is that since new signatures are always possible and being discovered we must always consider that the number of covariates on the executables increases more quickly than the number of executables. Whether the new executable is designed by a machine or by a human the analyst hoping to detect malicious executables must study the problem with the advantage given to the innovator; this implies that the possible attributes of the executables grows much quicker than the number of executables. This observation suggests the use of high-dimensional classification algorithms for malicious executable detection.

Data Exploration

We train high-dimensional binary classifiers using data from [3] which consists of 373 executables each with 531 attributes. Each of the executables is given a label as -1 (malicious) or +1 (non-malicious). The labeled data consists of 301 malicious executables and 72 non-malicious executables.

The 373 row dataset contains no missing values. Each of the 531 attributes are 0,1-binary covariates. Figures 1 and 2 provide histograms of the proportion of 1s over the 531 columns and the proportion of 1s over the 373 rows.

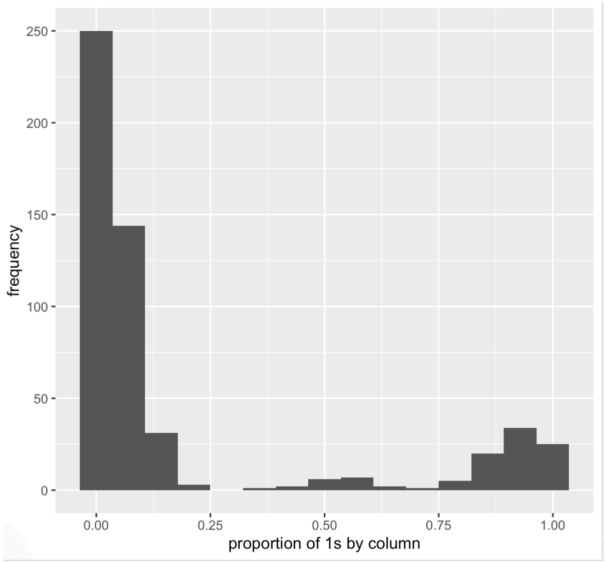

Figure 1: Most columns contain very few 1s while a few columns contain almost all 1. There are many characteristics not shared among the executables and a few characteristics that are shared.

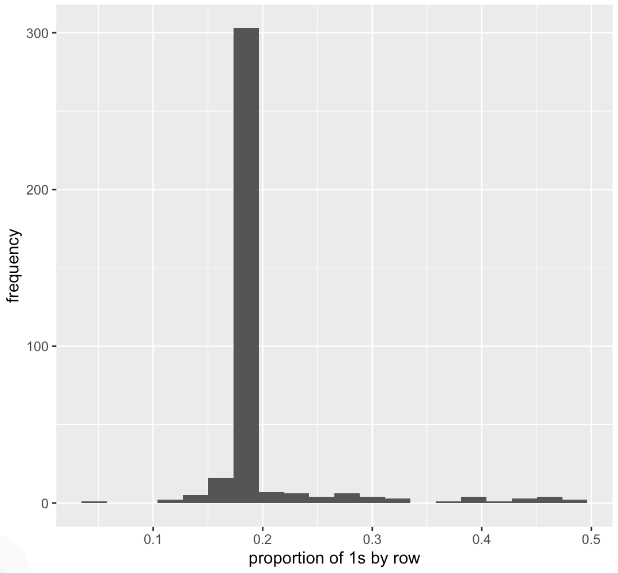

Figure 2: Most rows contain around 20% 1s with the rest of the values as 0. A few rows contain almost all 0s. The most populated rows contain around 50% 1s.

Figures 3 and 4 show a value-heatmap over all the 531 covariates and a correlation-heatmap between the 502 covariates which were not fully 1s or 0s.

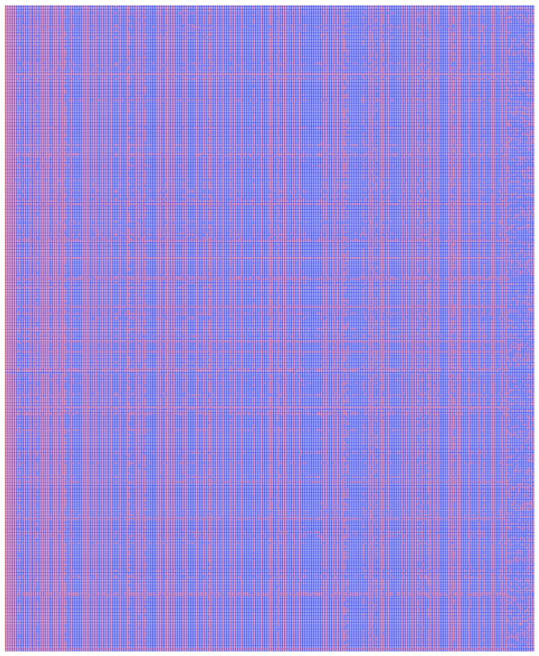
Figure 3: Covariate value heatmap (red=0, blue=1).

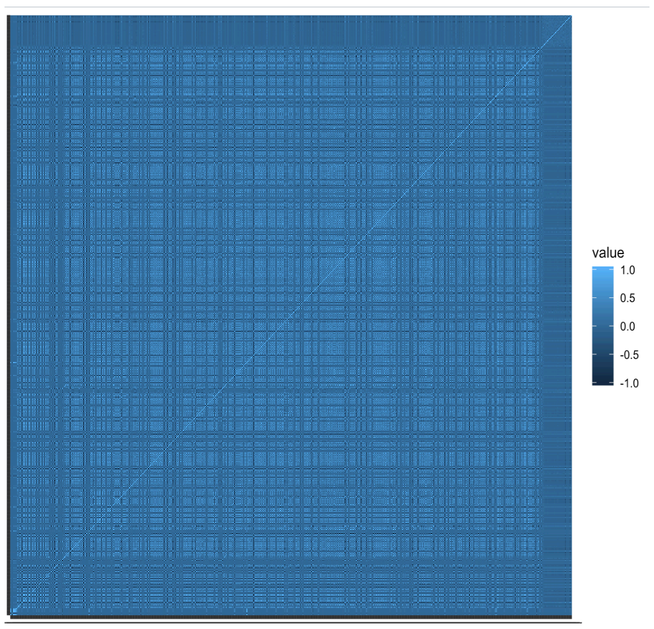

Figure 4: Covariate correlation heatmap. The darker the lower and more negative the correlation.

Both these figures were made over randomly shuffled data in preparation for cross-validation. The dependency structure revealed from the heatmaps does not provide much intuition toward our modeling strategy.

High-Dimensional Classifiers

Classification is a supervised learning method that can become difficult with relatively large numbers of attributes. Some classification algorithms such as Random Forests [4] or Support Vector Machines [5] are known to be reasonably robust to high-dimensionality. But for other algorithms the main problem with high-dimensionality is that classification errors can accumulate when there are too many noisy features, requiring dimension reduction [6]. Sparse latent discriminant analysis [7] and the logistic lasso [8] are therefore also useful high-dimensional classification algorithms.

We implement Random Forests (with 600 and 1000 trees), SVM, Sparse LDA, and logistic Lasso to classify malicious executables.

The random forest works by training a classification tree over repeatedly bootstrapped aggregated samples. By taking the dominant vote over all the classification trees we can make predictions [4].

The support vector machine assumes that the data can be separated linearly by some hyperplane which is a more appropriate assumption in high-dimensional spaces. By minimizing a distance ||w|| between two hyperplanes *w* satisfying *w*x-b= -1*, *w*x-b = 1* where x is the data we find the optimal hyperplane. The optimal hyperplane is fully determined by the few data points closest to the hyperplane, therefore SVM is generally useful in high-dimensional settings [5].

The logistic lasso [8] solves the standard lasso equation

$$\hat{\boldsymbol{\beta}}_{\text{Lasso}} = \text{argmin}_{\beta}(||\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}||_2^2/n + \lambda||\boldsymbol{\beta}||_1),$$

over a binary factor response which will send many regression coefficients to 0 due to the L1-norm used.

Sparse LDA [7] solves the linear programming problem:

$$\hat{\boldsymbol{\beta}}^{\text{LPD}} = \text{argmin}_{\boldsymbol{\beta}} \sum_{j=1}^{p} \|\boldsymbol{\beta}\|_1, \quad \text{s.t.} \left\|\hat{\boldsymbol{\Sigma}}\boldsymbol{\beta} - (\hat{\mu}_2 - \hat{\mu}_1)\right\|_{\infty} \leq \lambda_n.$$

where the Bayes rule classification vector is given a linear constraint.

Classifier Implementations and Comparison

The potentially dangerous nature of malicious executables means that the most important metric for classifier comparison is the number of false negatives. If an executable is malicious but the classifier claims it is non-malicious then the classifier fails. The second most important metric for classifier comparison is the number of false positives. If an executable is non-malicious but is classified as malicious this can lead to inefficiencies since potentially useful code will not be allowed into the system.

The method for choosing the best classifier is therefore to select the test data classifier with zero false negatives and the lowest number of false positives given the least amount of training data. We can control the amount of training data used by slowly increasing the number of cross-validations. Table 1 summarizes the number of false positives and false negatives across each k-fold cross-validation. For LASSO [9] we use cross-validations to gather the appropriate lambda. For SVM [10] we use a linear kernel. For RF [11] we use 600 and 1000 trees. For HDDA [12] we run across all possible models and choose the model with highest BIC.

	# False Negatives					**# False Positives**				
k (cv)	LASSO	HDDA	SVM	RF600	RF1000	LASSO	HDDA	SVM	RF600	RF1000
2	0	0	**0**	**0**	**0**	2	0	**1 (.003)**	**1 (.003)**	1
3	0	0	**0**	**0**	**0**	8	0	**3 (.005)**	**4 (.007)**	3
4	0	0	**0**	**0**	**0**	20	1	**6 (.007)**	**7 (.008)**	8
5	8	6	**0**	**0**	**0**	23	0	**9 (.008)**	**12 (.01)**	11
6	8	22	**0**	**0**	**0**	29	0	**11 (.007)**	**21 (.01)**	18
7	8	72	**0**	**0**	**0**	61	3	**15 (.008)**	**23 (.01)**	19
8	8	13	**0**	**0**	**0**	58	63	**58 (.03)**	**81 (.04)**	74
9	16	284	**0**	**0**	**0**	57	6	**29 (.01)**	**57 (.02)**	60
10		220	**0**	**0**	**0**		13	**31 (.01)**	**66 (.02)**	68
11			**0**	**0**	**0**			**129 (.04)**	**185 (.06)**	187
12			**0**	**0**	**0**			**31 (.01)**	**66 (.02)**	187
13			**0**	**0**	**0**			**83 (.02)**	**172 (.05)**	161

Table 1: # FN and #FP over k-cv for 5 classifiers. The missing values for LASSO and HDDA arise since both classifiers fail when the number of positive training samples is too low.

From Table 1 the Lasso and high-dimensional discriminant analysis classifiers yield many false negatives as k increases and should be disregarded. SVM and both Random Forests yield no false negatives with increasing k. Lasso and HDDA also yield many false positives when compared to SVM and the Random Forests. A comparison of SVM with the Random Forests shows that the number of false positive is always less than or equal to. The (1-precision) value is also provided for SVM and RF600 false positives to show that in general the classifiers have a very low proportion of false positives. Therefore by our classifier comparison criteria SVM is the algorithm that provides the best classification generalization.

A further inspection of the number of important variables in the logistic Lasso and RF provides intuition as to the impact of sparsity on this classification problem. For all cross-validation k-values the number of predictors used the Lasso is between 12 and 13 indicating a vast dimension reduction. Figure 5, a representative RF importance variable plot of the 40 most important variables, shows that the random forest also sees around 12 variables which stand out in importance. However the random forest does not only use these 12 variables and includes many other variables each of which is still relatively important.

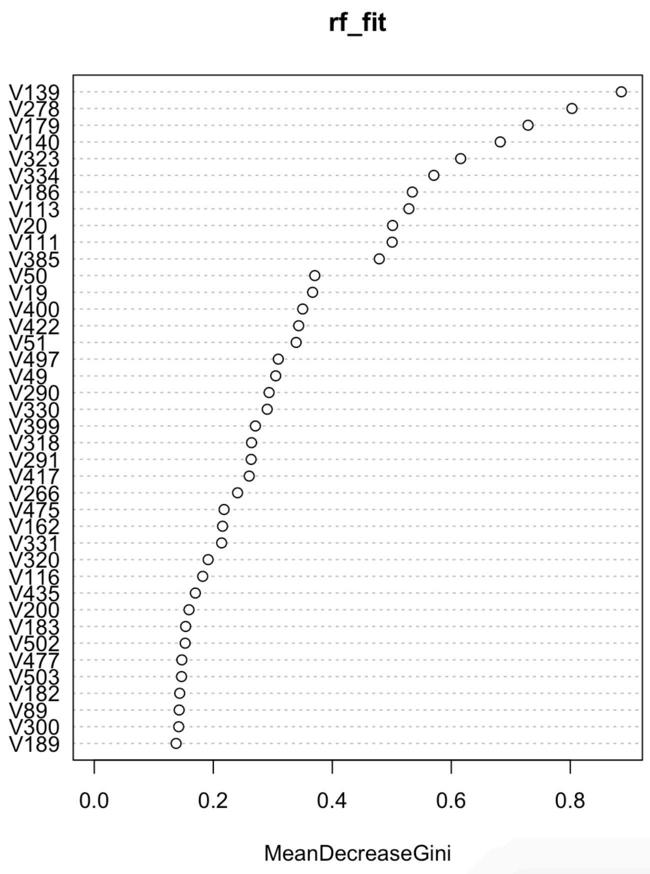

Figure 5: representative RF importance plot. Some variables stand out in importance, but many more are still important.

It is intriguing that the RF and SVM classifiers are performing so strongly as shown by there being no false negatives and very few false positives. Both SVM and RF are known to deal well when the number of attributes is larger than the number of observations. RF begins to fail when the number of attributes becomes very large while the SVM continues to perform well in high-dimensional settings due to only a relatively small number of support vectors influencing the hyperplane computation. However as the difference between the number of attributes and number of observations grows we expect that the classification methods specifically designed for sparse data such as LASSO and HDDA will begin to outperform RF and SVM.

References

[1] Kolter, J., Maloof, M. *Learning to detect and classify malicious executables in the wild*. Journal of Machine Learning Research 7 (2006) 2721-2744.

[2] M. G. Schultz, E. Eskin, F. Zadok and S. J. Stolfo, "Data mining methods for detection of new malicious executables," *Proceedings 2001 IEEE Symposium on Security and Privacy. S&P 2001*, Oakland, CA, USA, 2001, pp. 38-49.

[3] Rumao, P. *Detect Malacious Executable (Antivirus) Dataset*. UCI Machine Learning Repository.

[4] Breiman, L. *Random Forests*. Machine Learning 45 (1): 5-32. 2001.

[5] Steinwart, Ingo; Christmann, Andreas (2008). *Support Vector Machines*. New York: Springer.

[6] Fan, J., Fan, Y., Wu, Y. *High-dimensional Classification*.

[7] Cai, T., Liu, W. *Adaptive thresholding for sparse covariance matrix estimation*. Journal of the American Statistical Association, 106, 2011.

[8] Tibshirani, R. Regression shrinkage and selection via the lasso. *J. Royal. Statist. Soc. B.,* 58.1, 267-288 1996.

[9] Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, 33(1), 1–22.

[10] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel and Friedrich Leisch (2019). e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien. R package version 1.7-3. https://CRAN.R-project.org/package=e1071

[11] A. Liaw and M. Wiener (2002). Classification and Regression by randomForest. R News 2(3), 18--22

[12] Laurent Berge, Charles Bouveyron, Stephane Girard (2012). HDclassif: An R Package for Model-Based Clustering and Discriminant Analysis of High-Dimensional Data. Journal of Statistical Software, 46(6), 1-29.

R Code

```
rm(list=ls())

setwd('/Users/siddarthviswanathan/Documents/temple/semester2/dimension/project2/')

library("feather")
library("ggplot2")
library("d3heatmap")
library("glmnet")
library('HDclassif')
library('e1071')
library('randomForest')
library('reshape2')
########
## 0. Create train and test data
########

df_x <- as.data.frame(t(read_feather('featureVectors.feather')))
df_y <- as.data.frame(t(read_feather('trainingLabels.feather')))

set.seed(400)

shuffled_indices <- sample(nrow(df_x))
df_x <- df_x[shuffled_indices,]
df_y <- df_y[shuffled_indices,]

##########
## 1. summarize distribution and dependence structure of predictors
##########

sum(is.na(df_x)) # no missing values

dim(df_x) # 372 531

test1 <- c()
test2 <- c()
for(i in 1:ncol(df_x)){
  test1 <- append(test1, as.numeric(names(table(df_x[,i]))))
  test2 <- append(test2, sum(df_x[,i])/nrow(df_x))
}

# table(test1) reveals all columns contain only 0s and 1s

# plot of proportion of ones by column
qplot(test2, bins=15, xlab='proportion of 1s by column', ylab='frequency')

# find number of ones by row

test3 <- c()
for(j in 1:nrow(df_x)){
  test3 <- append(test3, sum(df_x[j,])/ncol(df_x))
}

# plot of proportion of 1s by column
qplot(test3, bins=10, xlab='proportion of 1s by row', ylab='frequency')

## heat map

d3heatmap(as.matrix(df_x), Rowv=FALSE, Colv=FALSE, col = c("blue", "red"), dendogram="none", cexRow=0)


## a correlation heat map cannot be computed since many columns have
# all 1s or 0s

# top 20 rows by 1s
test3[order(test3, decreasing=T)[1:20]]

# top 20 columns by 1s
test2[order(test2, decreasing=T)[1:50]]

## correlation matrix plot with full columns removed
corr_df_x <- df_x[vapply(df_x, function(x) length(unique(x)) > 1, logical(1L))]

cormat <- round(cor(corr_df_x),3)
melted_cormat <- melt(cormat)
ggplot(data=melted_cormat, aes(x=Var1, y=Var2, fill=value))+geom_tile()
############
## 2. comparison of classifications
## use cross validations and precision recall etc
##########

k <- 8
split_rows <- split(c(1:nrow(df_x)), ceiling(seq_along(c(1:nrow(df_x)))/(nrow(df_x)/k)))

### logistic lasso with k-fold cross validation, always around 13 variables used

results_tables <- list()

for(i in 1:length(split_rows)){
  train_df_x <- df_x[split_rows[[i]],]
  test_df_x <- df_x[-split_rows[[i]],]

  train_df_y <- df_y[split_rows[[i]]]
  test_df_y <- df_y[-split_rows[[i]]]

  lassofit <- glmnet(x=as.matrix(train_df_x), y=as.factor(train_df_y), family='binomial', alpha=1)
  fit.cv <- cv.glmnet(x=as.matrix(df_x),y=as.factor(df_y), family='binomial', alpha=1)
  predict_test_y <- predict(lassofit, as.matrix(test_df_x), s=fit.cv$lambda.1se, type='class')


  results_tables[[i]] <- table(actual=test_df_y, predicted=as.numeric(predict_test_y))

  b <- as.matrix(coef(fit.cv))
  print(length(rownames(b)[b != 0]))
}

Reduce("+", results_tables)



### high dimensional discriminant analysis with k-fold cv
k <- 7
split_rows <- split(c(1:nrow(df_x)), ceiling(seq_along(c(1:nrow(df_x)))/(nrow(df_x)/k)))

results_tables <- list()

for(i in 1:length(split_rows)){
  train_df_x <- df_x[split_rows[[i]],]
  test_df_x <- df_x[-split_rows[[i]],]

  train_df_y <- df_y[split_rows[[i]]]
  test_df_y <- df_y[-split_rows[[i]]]

  hddfit <- hdda(data=as.matrix(train_df_x), cls=train_df_y, model='ALL')
  predict_test_y <- predict(hddfit, as.matrix(test_df_x), type='class')

  results_tables[[i]] <- table(actual=test_df_y, predicted=as.numeric(predict_test_y$class))
}

Reduce("+", results_tables)



### SVM- classification, linear kernel, with k-fold cv
results_tables <- list()

for(i in 1:length(split_rows)){
  train_df_x <- df_x[split_rows[[i]],]
  test_df_x <- df_x[-split_rows[[i]],]

  train_df_y <- df_y[split_rows[[i]]]
  test_df_y <- df_y[-split_rows[[i]]]

  svmfit <- svm(x=as.matrix(train_df_x), y=as.factor(train_df_y), kernel='linear', scale=FALSE)
  predict_test_y <- predict(svmfit, as.matrix(test_df_x), type='class')

  results_tables[[i]] <- table(actual=test_df_y, predicted=as.numeric(predict_test_y))
}
```

```
Reduce("+", results_tables)


### random forest w/600 trees, with k-fold cv

k <- 8
split_rows <- split(c(1:nrow(df_x)), ceiling(seq_along(c(1:nrow(df_x)))/(nrow(df_x)/k)))

results_tables <- list()

for(i in 1:length(split_rows)){
  train_df_x <- df_x[split_rows[[i]],]
  test_df_x <- df_x[-split_rows[[i]],]

  train_df_y <- df_y[split_rows[[i]]]
  test_df_y <- df_y[-split_rows[[i]]]

  rf_fit <- randomForest(x=as.matrix(train_df_x), y=as.factor(train_df_y),ntree=1000)
  predict_test_y <- predict(rf_fit, as.matrix(test_df_x))

  #varImpPlot(rf_fit,type=2, n.var=40)

  results_tables[[i]] <- table(actual=test_df_y, predicted=as.numeric(predict_test_y))

}

#head(sort((VI_F=importance(rf_fit)), decreasing=T), 45)

Reduce("+", results_tables)
```