Malware Classification

This code snippet shows you how to train a Naive Bayes (NB) algorithm for malware classification. For this purpose, we can use different features to train our algorithm. This includes features that are obtained using both static code analysis and/or analysis of the dynamic behavior of the malware.

The dataset: We will use the dataset published at https://data.mendeley.com/datasets/w2w8gjsgnt/1 (http://dx.doi.org/10.17632/w2w8gjsgnt.1#file-6806d890-e13f-4644-abc2-630cca78216f). This data set contains a total of 1944 features that are obtained from the static and dynamic analysis of ~ 19400 malware samples including malware samples from APT attacks. We pre-processed the dataset by removing the first three columns and the last seven columns. Instead, we added a label column at the end of the dataset.

Machine learning (ML) task: We want to create an NB model to identify (classify) the type of malware, so we are solving a classification problem here.

```
myDataOrg <- read.csv("malware.csv", header=T)
dim(myDataOrg) # check dimensions of myData
```

```
## [1] 19457  1935
```

```
levels(myDataOrg$label) # check different levels (values) for each class
```

```
## [1] "Backdoor"  "OtherType" "Rootkit"   "Spyware"   "Trojan"    "Unknown"
## [7] "Worm"
```

Check the balance of the dataset : Class imbalance is a very common problem in cyber security datasets, and it is quite common that a 1:100000 ratio between classes (e.g. attack: normal) due to the scarcity of attack data. If the class imbalance occurred then it can be affected on model performance. We tried an NB model for the whole datset and accuracy was ~34%. Let's look at ratio between classes in our dataset.

```
print(table(myDataOrg$label))
```

```
##
##  Backdoor OtherType   Rootkit   Spyware    Trojan   Unknown      Worm
##        53      1275       789       709     11034      3957      1640
```

As you can see, our dataset is a hugely imbalance, especially Backdoor: Trojan. A mix of oversampling and undersampling methods could be utilised to balance the dataset, e.g., by increasing the size of Backdoor class and reducing the size of Trojan class. However, this can be resulted information lost in the larger class. Therefore, we will split the data set into two subsets and train two NB models separately (Ensamble technique). Of course, if NB doesn't work very well with one dataset, you can train another model (e.g. random forest) for that data set and combine NB with other model for better performance.

Note: It should be noted that NB woud not be the best option for this type of dataset, instead we recommend to try out some ensamble techniques. However, we will train NB model in this way as our goal in this post to show you how to train the NB model for this dataset.

```
myDataSet1<-rbind(subset(myDataOrg, label == "Backdoor"),subset(myDataOrg, label == "OtherType"),subset

myDataSet2<-rbind(subset(myDataOrg, label == "Trojan"),subset(myDataOrg, label == "Unknown"),subset(myDa
```

We split the original datset into two subsets. Let's continue with myDataSet1. You can follow the same approach for myDataSet2. The following code makes class sizes equal in myDataSet1. As mentioned you can try diffrent model for myDataSet2 as well.

```
sample.df <- function(df, n) df[sample(nrow(df), n,replace = T), , drop = F]

classSize<-500 # sample from each class size of 500 records
```

```r
myData<-rbind(sample.df(subset(myDataSet1, label == "Backdoor"), classSize),sample.df(subset(myDataSet1
```

Since we want to represent the presence or absence of a certain static/dynamic feature in the malware, we code our dataset as follows.

```r
convert_counts <- function(x) {
  x <- ifelse(x > 0, "Y", "N")
}
myData <- data.frame(apply(myData[,1:1934], MARGIN = 2,convert_counts),myData[1935] )
```

Creating training and validation datasets: We're going to follow the convention of 80/20 samples ratio to partition the dataset to the training and validation sets. We use the createDataPartition function from the caret package for this purpose.

```r
#install.packages("caret") #If the caret package is not installed on your system, uncomment this line t
set.seed(1234)
library(caret) #Loading the library
tr_index <- createDataPartition(myData$label, p=0.80, list=FALSE) # List of 80% of the rows
```

```
## Warning in createDataPartition(myData$label, p = 0.8, list = FALSE): Some
## classes have no records ( Trojan, Unknown, Worm ) and these will be ignored
```

```r
trainSet <- myData[tr_index,] # select 80% of the data for the trainSet
testSet <- myData[-tr_index,] # Select the remaining 20% of data for testSet
```

Building a NB classifier: Now we will train our NB classifier using the above trainSet. For this purpose, we will utilize e1071 package in R. Note that the priori probabilities can be computed using the following lines of code. In this case, the priori probabilities for all classes are the same.

```r
#install.packages("e1071") #If the e1071 package is not installed on your system, uncomment this line t
library(e1071)
NBclassfier <- naiveBayes(trainSet[,1:1934], trainSet$label) # train the model
```

Make predictions: Now let's apply the above model to assign labels for test cases in testSet. Then we create the confusion matrix, a table that is often used to describe the performance of a classifier.

```r
  testPrediction <- predict(NBclassfier, testSet[,1:1934]) # predict labels for test cases
  confusionMatrix(testPrediction, testSet$label) # Print confusion matrix
```

```
## Confusion Matrix and Statistics
##
##            Reference
## Prediction  Backdoor OtherType Rootkit Spyware Trojan Unknown Worm
##   Backdoor        89        15      10      12      0       0    0
##   OtherType        0        37      16      16      0       0    0
##   Rootkit          5        26      66      16      0       0    0
##   Spyware          6        22       8      56      0       0    0
##   Trojan           0         0       0       0      0       0    0
##   Unknown          0         0       0       0      0       0    0
##   Worm             0         0       0       0      0       0    0
##
## Overall Statistics
##
##                Accuracy : 0.62
##                  95% CI : (0.5704, 0.6678)
##     No Information Rate : 0.25
##     P-Value [Acc > NIR] : < 2.2e-16
```

```
##
##                   Kappa : 0.4933
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Backdoor Class: OtherType Class: Rootkit
## Sensitivity                   0.8900           0.3700         0.6600
## Specificity                   0.8767           0.8933         0.8433
## Pos Pred Value                0.7063           0.5362         0.5841
## Neg Pred Value                0.9599           0.8097         0.8815
## Prevalence                    0.2500           0.2500         0.2500
## Detection Rate                0.2225           0.0925         0.1650
## Detection Prevalence          0.3150           0.1725         0.2825
## Balanced Accuracy             0.8833           0.6317         0.7517
##                      Class: Spyware Class: Trojan Class: Unknown
## Sensitivity                  0.5600            NA             NA
## Specificity                  0.8800             1              1
## Pos Pred Value               0.6087            NA             NA
## Neg Pred Value               0.8571            NA             NA
## Prevalence                   0.2500             0              0
## Detection Rate               0.1400             0              0
## Detection Prevalence         0.2300             0              0
## Balanced Accuracy            0.7200            NA             NA
##                      Class: Worm
## Sensitivity                   NA
## Specificity                    1
## Pos Pred Value                NA
## Neg Pred Value                NA
## Prevalence                     0
## Detection Rate                 0
## Detection Prevalence           0
## Balanced Accuracy             NA
```