Bayesian poisoning

The purpose of this code snippet is to demonstrate to the reader how a Naive Bayes (NB) classifier is vulnerable to Bayesian poisoning attacks. Bayesian poisoning is a technique used by email spammers to compromise the effectiveness of Bayesian-based spam filters by adding non-spamming (ham) like words at the end of a spam message. As a result, the spam filter considers the message to be legitimate - a Type II statistical error. We use the R language in this implementation.

The dataset: We utilise the dataset at https://www.kaggle.com/venky73/spam-mails-dataset for this purpose.

Machine learning (ML) task: Our ML task in this problem would be a classification. First, we develop a Bayesian-based spam filter using the above email repository (raw emails) and then perform a Bayesian poisoning attack to bypass the security control. To this end, we poison the test data into the spam filter and bypass the security check.

```
set.seed(12345) # Set the seed value so that the same result is achieved in each replication

#install.packages('tm') # Install the required libraries if your system does not have them
#install.packages('wordcloud')
#install.packages('e1071')
#install.packages('gmodels')
#install.packages('SnowballC')

library(tm) # Load the necessary libraries
```

```
## Loading required package: NLP
```

```
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(e1071)
library(gmodels)
library(SnowballC)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'ggplot2'

## The following object is masked from 'package:NLP':
##
##     annotate
```

Load the dataset to the R environment, and remove the term “Subject:” as it’s common in each message. There is no point of keeping common constant terms like “Subject:” as they don’t have discrimination power.

```
myData <- read.csv("spam_ham_dataset.csv") # read the dataset
myData<-myData[,c("label","text")] # select two columns
myData$text<-substring(myData$text,9) # removing "Subject:"
```

Check the distributions of words (a.k.a document terms) in each class. If the term distributions are different, we can use term frequencies in a message as features to classify the message using an ML model.

```
spamMsg<-subset(myData,label=="spam")
hamMsg<-subset(myData,label=="ham")
# wordcloud(spamMsg$text,scale=c(4,.5),min.freq=5) # plot if the word apperas more than 5 times in the
# wordcloud(hamMsg$text,scale=c(4,.5),min.freq=5) # plot if the word apperas more than 5 times in the h
```

As we can see in the above output, term distributions are different, so we can train a ML model (NB in our case) to classify our messages.

Building an NB based spam classifier: Now we will train our NB classifier using the above dataset. We will utilize e1071 package in R for this purpose. To this end, we need to follow following steps,

step 1. We need to create a document term matrix (DTM) - a matrix that describes the frequency of terms occured in our message collection.

```
myCorpus <- VCorpus(VectorSource(myData$text))

myDTM <- DocumentTermMatrix(myCorpus, control = list(
  tolower=T,
  removeNumbers=T,
  removePunctuation=T,
  stopwords = TRUE,
  stem=T
))
```

Let's use only terms which have frequency >=5 as features in our model building.

```
freqWords <- findFreqTerms(myDTM,5)
myDTM <- myDTM[,freqWords]
```

step 2. Split the dataset into train and test sets. Later we are going to poison the test set! We're going to use an 80/20 partitioning for the training and testing. We'll use the createDataPartition function from the caret package for this purpose.

```
tr_index <- createDataPartition(myData$label, p=0.80, list=FALSE) # List of 80% of the rows
trainSet <- myData[tr_index,] # select 80% of the data for the trainSet
testSet <- myData[-tr_index,] # Select the remaining 20% of data for testSet
```

Since we are creating our NB model with the DTM entries, we should obtain corresponding DTM entries for the data in trainSet and testSet.

```
myDTMTrain <- myDTM[tr_index,]
myDTMTest <- myDTM[-tr_index,]
```

```
convert_counts <- function(x) {
  x <- ifelse(x > 0, "Y", "N")
}

myDTMTrainNew <- apply(myDTMTrain, MARGIN = 2,convert_counts)
myDTMTestNew <- apply(myDTMTest, MARGIN = 2, convert_counts)
```

```
msgClassifier <- naiveBayes(myDTMTrainNew, trainSet$label)
testPredict <- predict(msgClassifier, myDTMTestNew)
```

```
confusionMatrix(testPredict, testSet$label, positive = "spam") # Print confusion matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction ham spam
##       ham  704   67
##       spam  30  232
##
##                Accuracy : 0.9061
##                  95% CI : (0.8867, 0.9232)
##     No Information Rate : 0.7106
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.763
##
##  Mcnemar's Test P-Value : 0.0002569
##
##             Sensitivity : 0.7759
##             Specificity : 0.9591
##          Pos Pred Value : 0.8855
##          Neg Pred Value : 0.9131
##              Prevalence : 0.2894
##          Detection Rate : 0.2246
##    Detection Prevalence : 0.2536
##       Balanced Accuracy : 0.8675
##
##        'Positive' Class : spam
##
```

Bayesian poisoning: Let's poison the test data to bypass the above spam filter. To this end, randomly selected non-spam messages are ammended at the end of spam messages in our test dataset.

```
### Poison the test data
spamTestCases <- subset(testSet,label=="spam")
hamTestCases <- subset(testSet, label=="ham")
hamMsgInmyData<-subset(myData, label=="ham")
hamMsg2ammend<-hamMsgInmyData[sample(nrow(hamMsgInmyData), nrow(hamTestCases)), ]
hamTestCases$text <-paste(hamTestCases$text,hamMsg2ammend$text,sep = " ")
poisTestData<-rbind(hamTestCases,spamTestCases)
myData[-tr_index,]<-poisTestData
```

Create the new DTM with poison data and remove low frequency words

```
myCorpusPoisoned <- VCorpus(VectorSource(myData$text))

myDTMPoisoned <- DocumentTermMatrix(myCorpus, control = list(
  tolower=T,
  removeNumbers=T,
  removePunctuation=T,
  stopwords = T,
  stem=T
))

freqWords <- findFreqTerms(myDTMPoisoned,5)
myDTMPoisoned <- myDTMPoisoned[,freqWords]

myDTMTestPoisoned <- myDTMPoisoned[-tr_index,] # new test set, do not need train set as we use the same


myDTMTestNewPoisoned <- apply(myDTMTestPoisoned, MARGIN = 2, convert_counts) # convert to categorical
```

```
testPredictPoisoned <- predict(msgClassifier, myDTMTestNewPoisoned)
```

```
confusionMatrix(testPredictPoisoned, testSet$label, positive = "spam") # Print confusion matrix for poi
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction ham spam
##       ham  704   67
##       spam  30  232
##
##                Accuracy : 0.9061
##                  95% CI : (0.8867, 0.9232)
##     No Information Rate : 0.7106
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.763
##
##  Mcnemar's Test P-Value : 0.0002569
##
##             Sensitivity : 0.7759
##             Specificity : 0.9591
##          Pos Pred Value : 0.8855
##          Neg Pred Value : 0.9131
##              Prevalence : 0.2894
##          Detection Rate : 0.2246
##    Detection Prevalence : 0.2536
##       Balanced Accuracy : 0.8675
##
##        'Positive' Class : spam
##
```

As you can see in the confusion matrix, the misclassification of spam as ham (false negative) increases from 67 to 220. Adding more hap like words at the end of spam messages attacker can increase the false negative number further.