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.

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

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.

```r
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:"
```

Let's check the word distributions (a.k.a. document terms) in each class. To this end we use word clouds, in which the size of the word is proportional to its frequency in the text. If the word clouds are different, we can conclude that words that appear frequently in spams differ from words that appear in hams. Therefore, we can use words (features) in a text message to classify them as ham or spam.

```r
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 word cloud outputs, the term distributions differ between spam and ham, so we can train an ML model (in our case, NB) by using words as features.

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 each message in our collection.

```r
myCorpus <- VCorpus(VectorSource(myData$text)) # create a corpus

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

DTM is usually a sparse matrix as most of its entries are filled with zeros. Let's use only terms which have frequency >=5 as features in our model building. To this end we use the findFreqTerms () function in R as follows. So, we can reduce the number of columns in our DTM matrix to 8615, which would be a manageable size in our model building.

```r
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.

```r
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.

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

Since we want to represent the presence or absence of a certain word (feature) in a particular message, we code our DTM as follows.

```r
convert_counts <- function(x) {
  x <- ifelse(x > 0, "T", "F")
}
myDTMTrainNew <- apply(myDTMTrain, MARGIN = 2,convert_counts)
myDTMTestNew <- apply(myDTMTest, MARGIN = 2, convert_counts)
```

Step 3: Building the NB based spam filter.

```r
NBbasedSpamFilter <- naiveBayes(myDTMTrainNew, trainSet$label) # train the model
testPredictMsgLabel <- predict(NBbasedSpamFilter, myDTMTestNew) # predict labels for test cases
```

Create the confusion matrix, a table that is often used to describe the performance of a classifier.

```r
confusionMatrix(testPredictMsgLabel, 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") # select spams in test cases
hamTestCases <- subset(testSet, label=="ham") # select ham in test cases
hamMsgInmyData<-subset(myData, label=="ham") # select ham in our original dataset
hamMsg2ammend<-hamMsgInmyData[sample(nrow(hamMsgInmyData), nrow(spamTestCases)), ] # select number of h
spamTestCases$text <-paste(spamTestCases$text,hamMsg2ammend$text,sep = " ") # ammend selected ham at th
poisTestData<-rbind(hamTestCases,spamTestCases) # create the poisoned test set
myData[-tr_index,]<-poisTestData # replace the test entries in original dataset with poisoned test case
```

Create the new DTM with poisoned data and select high frequency words. Note that we have to create the DTM and split it as we did with original data in the above.

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

myDTMPoisoned <- DocumentTermMatrix(myCorpusPoisoned, 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, note that we don't need  a training set

myDTMTestNewPoisoned <- apply(myDTMTestPoisoned, MARGIN = 2, convert_counts) # Using the same convert_c
```

Predict labels for poisoned test cases, note that we use the same NBbasedSpamFilter trained above.

```
poisonedTestPredictMsgLabel <- predict(NBbasedSpamFilter, myDTMTestNewPoisoned)
```

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

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction ham spam
##       ham  641  256
##       spam  93   43
##
##                Accuracy : 0.6621
##                  95% CI : (0.6324, 0.691)
##     No Information Rate : 0.7106
##     P-Value [Acc > NIR] : 0.9997
##
##                   Kappa : 0.0204
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.14381
##             Specificity : 0.87330
##          Pos Pred Value : 0.31618
##          Neg Pred Value : 0.71460
##              Prevalence : 0.28945
##          Detection Rate : 0.04163
##    Detection Prevalence : 0.13166
##       Balanced Accuracy : 0.50855
##
##        'Positive' Class : spam
##
```

As you can see in the confusion matrix, the misclassification of spam as ham (false negative) increases from 67 to 256. By adding more ham-like words to the end of spam messages, the attacker can further increase the false negative number.