Bayesian poisoning

The purpose of this code snippet to demonstrate 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 Bayesin 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 a 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,
  stem=T
))
```

Lets use only terms which are 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 going to poison the test set! We're going to do 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, 1, 0)
}

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) # Print confusion matrix
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction ham spam
```

```
##       ham  713  226
##       spam  21   73
##
##                Accuracy : 0.7609
##                  95% CI : (0.7337, 0.7866)
##     No Information Rate : 0.7106
##     P-Value [Acc > NIR] : 0.0001623
##
##                   Kappa : 0.2705
##
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9714
##             Specificity : 0.2441
##          Pos Pred Value : 0.7593
##          Neg Pred Value : 0.7766
##              Prevalence : 0.7106
##          Detection Rate : 0.6902
##    Detection Prevalence : 0.9090
##       Balanced Accuracy : 0.6078
##
##        'Positive' Class : ham
##
```