607 Project 4: eMail Filter

Jose Mawyin

11/1/2019

607 Project 4: eMail Filter using Naive Bayes Classifier

Outline

---

1. Introduction

This project will show how we can use *guided* machine learning to create a classifier able to sort spam and not-spam(ham) email. This is an example of guided machine learning because we will use a set of labeled data (ham/spam) to train our model.

In this classifier we will use a naive Bayes (*NB*) algorithm for our spam/ham detection. NB is based in the concept of *conditional probability* and uses a series of "predictor variables are conditionally independent of one another given the response value".

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}$$

where
P(A) is prior probability,
P(A|B) is posterior probability and its read as the probability of the event A happening given the event B.
P(B) is marginal likelihood
P(B|A) is likelihood
P(B|A) is likelihood
$P(B|A) \cdot P(A)$ could also be thought as joint probability, which denotes the probability of A intersection B; in other words, the probability of both event A and B happening together.

Our data will consist of a dataframe containing our response variable, a column with labels spam or ham. And, the text of a series of collected emails classified already to be either spam or ham as the predictor variables.

---

2. Data

We will source our data from a set of emails collected by the Apache Spam Assassin dataset that can be found at:

https://spamassassin.apache.org/old/publiccorpus/

2 different sets of ham and 2 different sets of spam data will be imported into R using the custom function "email.extract" shown below.

```
email.extract <- function(folder, type){
library(tm)
library(tm.plugin.mail)
library(stringr)
name<-VCorpus(DirSource(folder),list(reader=readMail))
name.length <- length(name)
print(name.length)
cured.name.df <- as.data.frame(matrix(ncol=5,nrow=name.length))
for (i in 1:name.length) {
  cured.name.df[i,1] <- name[[i]][["meta"]][["header"]][["From"]]
  cured.name.df[i,2] <- sub(".*\\<(.*)\\>.*", "\\1", cured.name.df[i,1], perl=TRUE) #Keep only  email
  cured.name.df[i,2]<- gsub("@(.+)$", "\\1", cured.name.df[i,2]) #Keep only  domain
  cured.name.df[i,3] <- name[[i]][["meta"]][["header"]][["Subject"]]
  cured.name.df[i,4] <- name[[i]][["content"]] %>% as.String()
  cured.name.df[i,5] <- type
}
colnames(cured.name.df) <- c("From","Domain","Subject","Content", "Type")
return(cured.name.df)
}
```

The imported datasets contained characters that did not transcode well into *UTF-8* which is the common encoding used when describing Unicode. Many of the steps used in the processing of the data needed before creater our classifier failed when dealing with non *UTF-8* characters.

The r-chunk code below shows the process used to make sure that all the characters in the text were properly transcoded into *UTF-8*.

```
ham <- rbind(ham.1, ham.2)
ham$Subject <- iconv(ham$Subject, "latin1", "UTF-8",sub='')
ham$Content <- iconv(ham$Content, "latin1", "UTF-8",sub='')

spam <- rbind(spam.1,spam.2)
spam$Subject <- iconv(spam$Subject, "latin1", "UTF-8",sub='')
spam$Content <- iconv(spam$Content, "latin1", "UTF-8",sub='')

spamham <- rbind(ham.1, ham.2,spam.1,spam.2)
spamham$Type <- as.factor(spamham$Type)
```

Figure 1: Frequent Words in the Content of Spam(Red)

The four *wordcloud* plots (Figure 1 to Figure 4) in this section show the 100 most comon words found in the Content and Subject of spam and ham email. We can notice some features noticeable from these plots:

- Content *wordcloud* +Many of the most common spam words seem to be HTML code embbeded in to the body of the email. +The most words in ham emails are usual typical words in the an email.
- Subject *wordcloud* +Frequent words in the subject of spam emails appear to be ad words such as free, money, business, etc. +Freqency words in the subject of ham eamils do not look at all as common words in a typical email. After checking a few of these ham email, it seems that a significant percentage of them were forwarded and a tag attached to the subject of the email. These tags appear as the most frequent words.

Before the following data preparation steps we will combine all the spam and ham datasets into a single set. The table below shows the number of emails per type and ratio of email.

The spam dataframe contains 1896 files. While the ham dataframe contains 5051 files. The ratio of spam to ham is 0.38 ham spam 5051 1896

Figure 2: Frequent Words in the Content of Not-Spam(Blue)

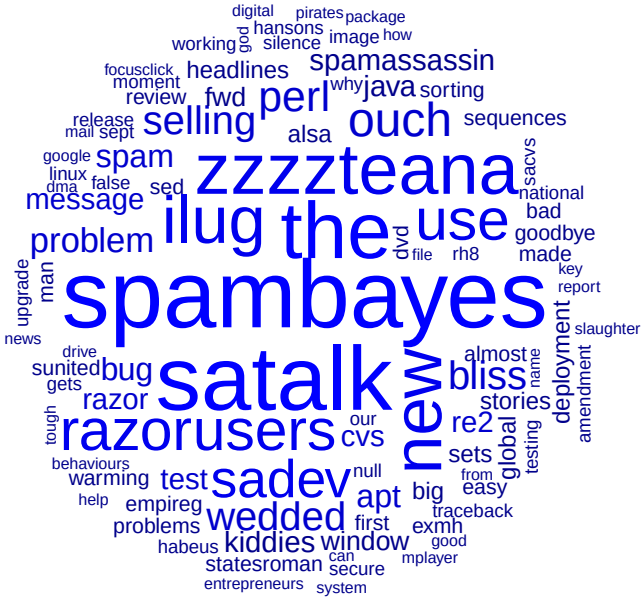

Figure 3: Frequent Words in the Subject of Not-Spam(Blue)

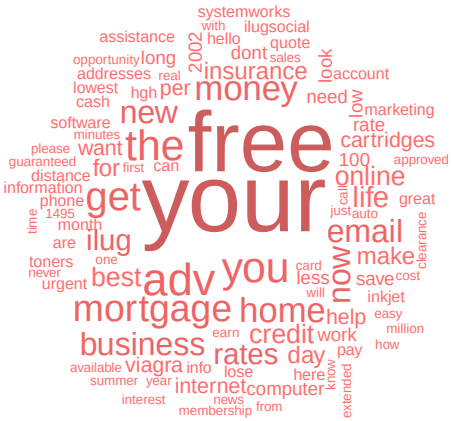

Figure 4: Frequent Words in the Subject of Spam(Red)

3. Data Preparation

The next step is to create a *DocumentTermMatrix* that describes the frequency of terms that occur in a collection of documents.

Before the creation of the *DocumentTermMatrix* we need to do the following:

Change the case of all the words to lower-case. Remove all numbers. *Remove all stop-words or words that are the most common words in a language. The tm package gives the option to remove english stop-words only.* Remove all punctuation signs. *Remove all extra spaces between words.

```
tic()
#email_corpus <- Corpus(VectorSource(spamham$Subject))
email_corpus_clean <- email_corpus %>%
    tm_map(content_transformer(tolower)) %>%
    tm_map(removeNumbers) %>%
    tm_map(removeWords, stopwords(kind="en")) %>%
    tm_map(removePunctuation) %>%
    tm_map(stripWhitespace)
email_dtm <- DocumentTermMatrix(email_corpus_clean)
email_dtm <- removeSparseTerms(email_dtm, 0.95) #Removing sparse data
toc()
```

```
## 13.174 sec elapsed
```

Now that we have our *DocumentTermMatrix* we remove those words that appear with low frequency in the emails.

4. Classifier Training

First, let's create a subset of spam/ham emails to train (75% of all emails) our NB classifier and a subset of emails to test (25% of all emails) the accuracy of the classifier.

```
train_index <- createDataPartition(spamham$Type, p=0.75, list=FALSE)

email_raw_train <- spamham[train_index,]
email_corpus_clean_train <- email_corpus_clean[train_index]
email_dtm_train <- email_dtm[train_index,]


email_raw_test <- spamham[-train_index,]
email_corpus_clean_test <- email_corpus_clean[-train_index]
email_dtm_test <- email_dtm[-train_index,]
```

Then we will create a dictionary of words that appear with a frequency of at least 5 times in the our dataset.

```
tic()
#Create a dictionary of words that appear at least 5 times in the emails.
email_dict <- findFreqTerms(email_dtm_train, lowfreq=5)

email_train <- DocumentTermMatrix(email_corpus_clean_train, list(dictionary=email_dict))
email_train <- email_train %>% apply(MARGIN=2, FUN=convert_counts)

email_test <- DocumentTermMatrix(email_corpus_clean_test, list(dictionary=email_dict))
email_test <- email_test %>% apply(MARGIN=2, FUN=convert_counts)

toc()
```

```
## 2.597 sec elapsed
```

```
paste("This is the dictionary of", length(email_dict), "terms that appear at least 5 times in the emails
```

[1] "This is the dictionary of 247 terms that appear at least 5 times in the emails and that will be used determine if an email is spam or ham."

```
paste(email_dict, collapse = "   ")
```

[1] "able actually add address already also always another anyone anything around available back best better bit body borderd build business call can case center change check click code color come company computer contact contenttransferencoding contenttype course current data date day days different div done easy either else email end enough error even ever every file files find first following font form found free full future geek get getting give going good got great group head heaven help helvetica high home hours however html httplists-freshrpmsnetmailmanlistinforpmlist httpwwwnewsisfreecomclick img information instead internet just keep know last least less let life like line link linux list little long look looking lot made mail mailing make making many may maybe message messages meta might million money much must name need network never new news next nothing now number offer old one online order original page part people per pfont phone place please point possible probably problem problems program public put read real really receive received receiving remove removed reply right rpmlist rpmlistfreshrpmsnet run running said say see seems seen send sent sep september server service set sfnet simply since site size sized software someone something spam special sponsored start state still stuff subject supplied support sure system table take tell textplain thanks thing

things think though time times today true try trying two unsubscribe url use used user users using version want way web week welcome well width widthd will wish within without work working works world wrote year years yes yet"

Now we can use the parsed email subject text as our predictor variables to train a **NB** classifier and the Type of email (spam/ham) as the response variable. This classifer will use the following parameters as well:

usekernel parameter allows us to use a kernel density estimate for continuous variables versus a guassian density estimate.

adjust allows us to adjust the bandwidth of the kernel density (larger numbers mean more flexible density estimate).

fL allows us to incorporate the Laplace smoother.

```
ctrl <- trainControl(method="cv", 10)
set.seed(12358)
email_model2 <- train(email_train, email_raw_train$Type, method="nb",
                      tuneGrid=data.frame(.fL=1, .usekernel=FALSE, .adjust = seq(0, 5, by = 1)),
                  trControl=ctrl)
email_model2
```

```
## Naive Bayes
##
## 5211 samples
##  247 predictor
##    2 classes: 'ham', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 4691, 4690, 4689, 4690, 4690, 4690, ...
## Resampling results across tuning parameters:
##
##   adjust  Accuracy   Kappa
##   0       0.9318709  0.8230248
##   1       0.9318709  0.8230248
##   2       0.9318709  0.8230248
##   3       0.9318709  0.8230248
##   4       0.9318709  0.8230248
##   5       0.9318709  0.8230248
##
## Tuning parameter 'fL' was held constant at a value of 1
## Tuning
##  parameter 'usekernel' was held constant at a value of FALSE
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were fL = 1, usekernel = FALSE
##  and adjust = 0.
```

---

5. Email Classification Using Content

Our NB classifier model was trined using email content data. Let's see how does it perform predicting wether emails are either spam or ham.

```
email_predict2 <- predict(email_model2, email_test)
cm2 <- confusionMatrix(email_predict2, email_raw_test$Type, positive="spam")
cm2
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction  ham spam
##       ham  1229   70
##       spam   33  404
##
##                Accuracy : 0.9407
##                  95% CI : (0.9285, 0.9513)
##     No Information Rate : 0.727
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.8468
##
##  Mcnemar's Test P-Value : 0.0003894
##
##             Sensitivity : 0.8523
##             Specificity : 0.9739
##          Pos Pred Value : 0.9245
##          Neg Pred Value : 0.9461
##              Prevalence : 0.2730
##          Detection Rate : 0.2327
##    Detection Prevalence : 0.2517
##       Balanced Accuracy : 0.9131
##
##        'Positive' Class : spam
##
```

The prediction outputs a series of performance parameters in the form of a *Confusion Matrix* that is a table used to describe the performance of a classification model or "classifier".

Confusion Matrix

The confusion matrix is a two by two table that contains four outcomes produced by a binary classifier. Various measures, such as error-rate, accuracy, specificity, sensitivity, and precision, are derived from the confusion matrix.

Sensitivity (Recall or True positive rate)

Sensitivity (SN) is calculated as the number of correct positive predictions divided by the total number of positives. It is also called recall (REC) or true positive rate (TPR). The best sensitivity is 1.0, whereas the worst is 0.0.

$\text{SN} = \frac{\text{TP}}{\text{TP+FN}} = \frac{\text{TP}}{\text{P}}$

Specificity (True negative rate)

Specificity (SP) is calculated as the number of correct negative predictions divided by the total number of negatives. It is also called true negative rate (TNR). The best specificity is 1.0, whereas the worst is 0.0.

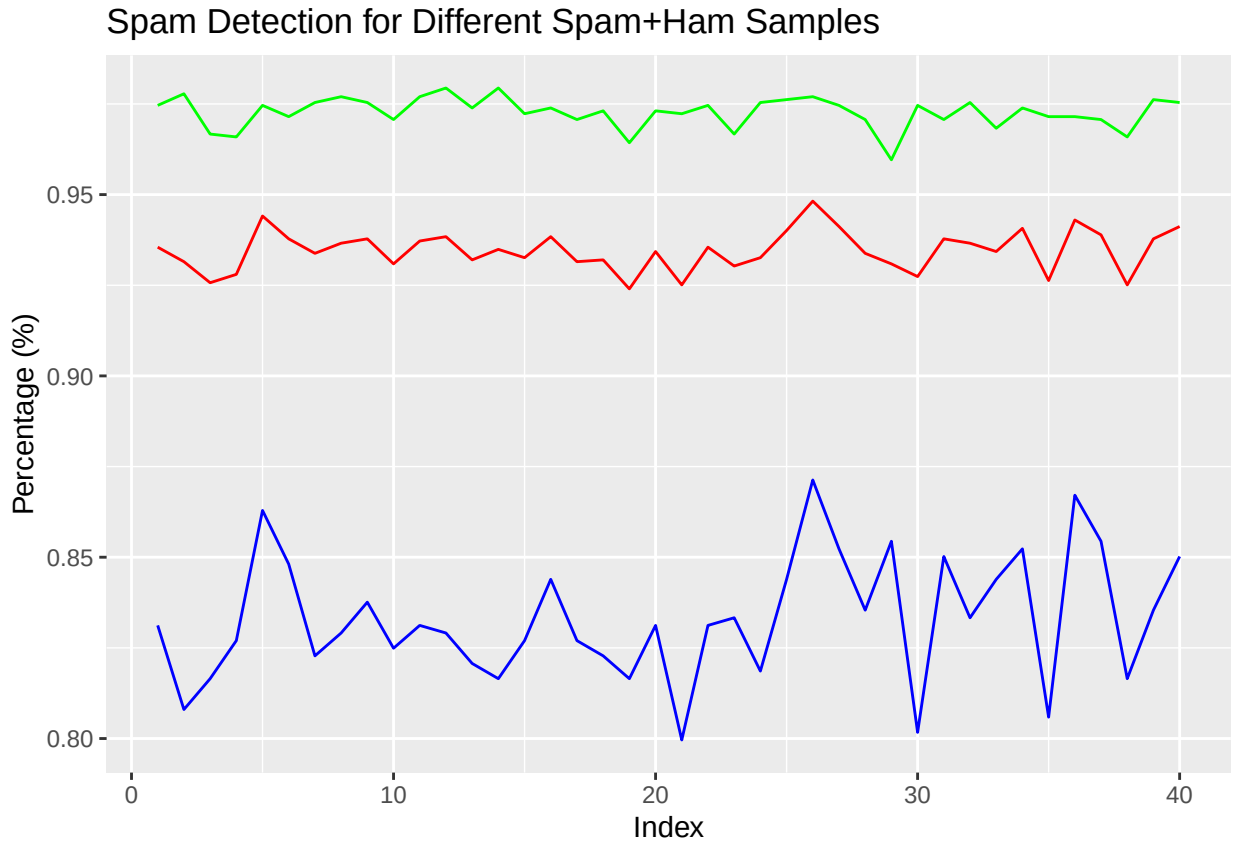

Figure 5: Accuracy(Red), Sensitivity(Blue) and Specificity(Green)

$\text{SP} = \frac{\text{TN}}{\text{TN+FP}} = \frac{\text{TN}}{\text{N}}$

The traces in Figure 5 show how the performance parameters described the previous page change when we use our NB model to classify different subsets of spam+ham emails.

After 40 differet samples:

Average Accuracy is: 0.934595 Average Sensitivity is: 0.8331225 Average Specificity is: 0.9726975

The *NB* classifier performs well in separating spam/ham in different email subsets with an average accuracy of 0.93652. The sensitivity is a bit lower at 0.837705 while the Specificity is the highest at 0.9736275.

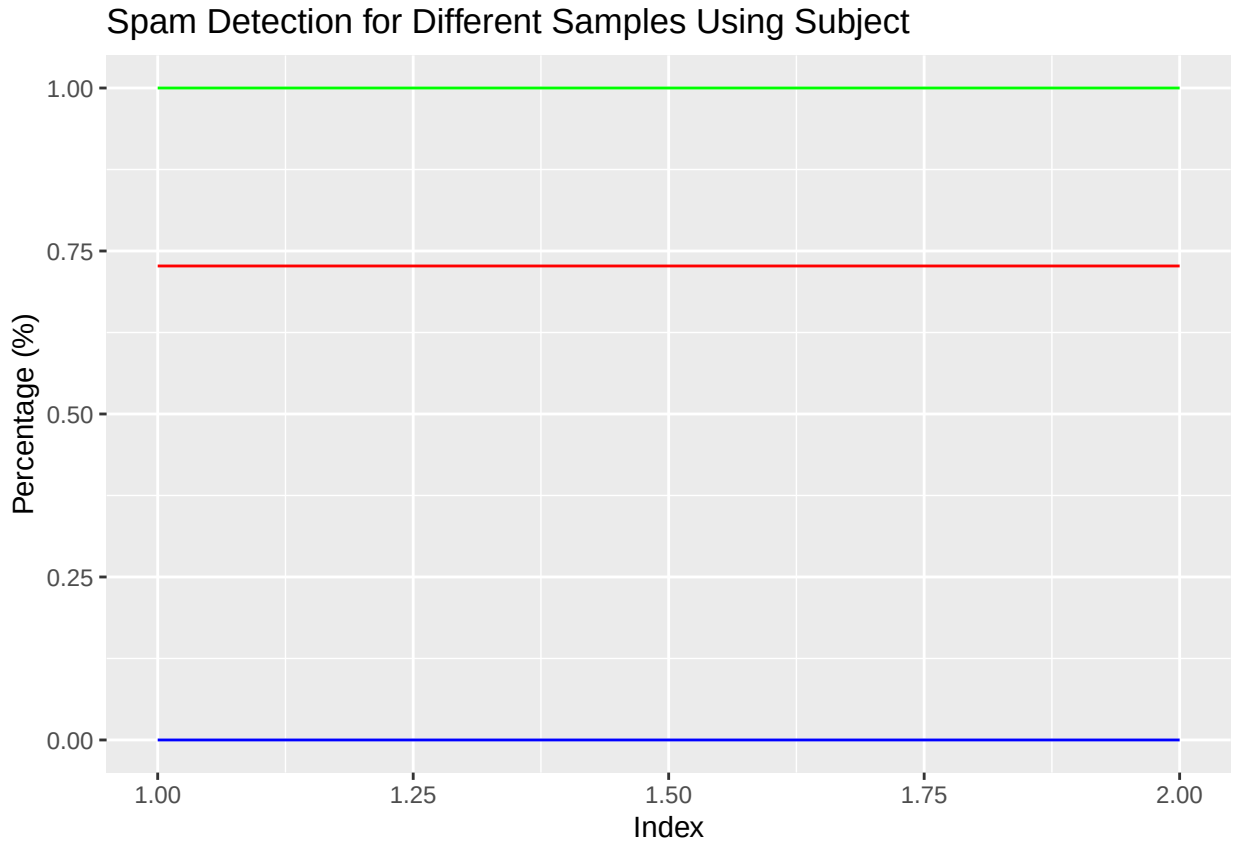

Figure 6: Accuracy(Red), Sensitivity(Blue) and Specificity(Green)

6. Email Classification Using Subject

How well does the NB classifier that we created work with much a different set of spam/ham data? We can use the subject information of the spam/ham dataset to see if we can use the same NB classifier to sort emails.

After 2 differet samples:

Average Accuracy is: 0.727 Average Sensitivity is: 0 Average Specificity is: 1

The performance is horrible. The text information in the Content of our emails has a completely different set of response of predictor variables as compared to the Subject of an emails. The text information of an email Subject is too short and has a different identifying fingerprints.

7. Concluding Remarks

We have shown how the Naive Bayes algorithm can be used to sort text documents of two response variables (spam/ham). There are other more complicated techniques of machine learning available but *NB* performs well despite its simplicity.

We have also seen how fine-tuned is the NB classifier that we trained to one particular type of data. In this case, the classifier excels in detecting spam/ham using Content data but fails completely when using the same classifier with a related dataset (Subject data).

---

8. Useful Links

Naive Bayes Classifier

Basic evaluation measures from the confusion matrix

A List of Available Models in Caret

Supervised classification with text data