Bayesian learning

Borja Ruiz Amantegui

February 21st 2018

Contents

1 Data preparation

1.1 Cleaning data

In order to work with our dataset we must first convert it to a corpus which lets us use functions from the **tm** library.

```
corpus <- Corpus(VectorSource(sms$text))
```

We can now use the converted data and transform it depending on our needs. We decide to:

1. Convert all letters to lowercase.
2. Remove punctuations.
3. Remove numbers.
4. Remove most used words in english language in order have only pertinent information.
5. Remove extra white spaces.

```
clean_corpus <- tm_map(corpus, tolower)
clean_corpus <- tm_map(clean_corpus, removePunctuation)
clean_corpus <- tm_map(clean_corpus, removeNumbers)
clean_corpus <- tm_map(clean_corpus, removeWords, stopwords("en"))
clean_corpus <- tm_map(clean_corpus, stripWhitespace)
inspect(clean_corpus[1:3])
```

```
## <<SimpleCorpus>>
## Metadata:  corpus specific: 1, document level (indexed): 0
## Content:  documents: 3
##
## [1] go jurong point crazy available bugis n great world la e buffet cine got amore wat
## [2] ok lar joking wif u oni
## [3] free entry wkly comp win fa cup final tkts st may text fa receive entry questionstd txt ratetcs
```

1.2 Train/test

We create two new variables including indices: one for those who are spam; and another for those being ham.

```
spam_indices <- which(sms$type == "spam")
ham_indices <- which(sms$type == "ham")
```

In order to compute the error rates we must divide data into training and testing. So the predictions can be done from a dataset independent from the one that trained the model.

```
sms_train <- sms[1:4169,]
sms_test <- sms[4170:5559,]
corpus_train <- clean_corpus[1:4169]
corpus_test <- clean_corpus[4170:5559]
```

1.3 Appearance filtering

Since we must predict categories depending the text contained, we must create a matrix from the document and divide it into training and testing sets. Each row will represent a message received and each column a word. Each cell will who the amount of appearances of such word in the text received.

```r
sms_dtm <- DocumentTermMatrix(clean_corpus)
inspect(sms_dtm[1:4, 30:35])
```

```
## <<DocumentTermMatrix (documents: 4, terms: 6)>>
## Non-/sparse entries: 6/18
## Sparsity           : 75%
## Maximal term length: 7
## Weighting          : term frequency (tf)
## Sample             :
##     Terms
## Docs already dun early txt win wkly
##    1       0   0     0   0   0    0
##    2       0   0     0   0   0    0
##    3       0   0     0   1   1    1
##    4       1   1     1   0   0    0
```

```r
sms_dtm_train <- sms_dtm[1:4169,]
sms_dtm_test <- sms_dtm[4170:5559,]
```

We now have to filter. Since most words appearing wont correlate because of its low appearance in the messages, we decide to filter those words that don't appear at least 5 times.

```r
five_times_words <- findFreqTerms(sms_dtm_train, 5)
length(five_times_words)
```

```
## [1] 1235
```

```r
five_times_words[1:5]
```

```
## [1] "available" "bugis"     "cine"      "crazy"     "got"
```

And now create the document term matrixes, for training and testing, already filtered by the words appearance. And in order to finish the data preparation we created a function to convert the count of the words into a "YES" or "NO" label to determine its appearance with this format.

```r
sms_dtm_train <- DocumentTermMatrix(corpus_train, control=list(dictionary = five_times_words))
sms_dtm_test <- DocumentTermMatrix(corpus_test, control=list(dictionary = five_times_words))

convert_count <- function(x){
  y <- ifelse(x > 0, 1,0)
  y <- factor(y, levels=c(0,1), labels=c("No", "Yes"))
  y
}


sms_dtm_train <- apply(sms_dtm_train, 2, convert_count)
sms_dtm_train[1:4, 30:35]
```

```
##     Terms
## Docs goes nah  think though usf  back
##    1 "No" "No" "No"  "No"   "No" "No"
##    2 "No" "No" "No"  "No"   "No" "No"
##    3 "No" "No" "No"  "No"   "No" "No"
##    4 "No" "No" "No"  "No"   "No" "No"
```

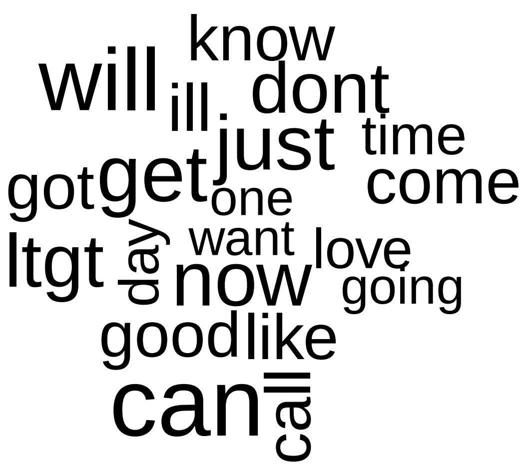
```r
sms_dtm_test <- apply(sms_dtm_test, 2, convert_count)
sms_dtm_test[1:4, 30:35]
```

```
##     Terms
## Docs second side  takes wait  year  actually
##    1 "No"   "No"  "No"  "No"  "No"  "No"
##    2 "No"   "No"  "No"  "No"  "No"  "No"
##    3 "Yes"  "Yes" "Yes" "Yes" "Yes" "No"
##    4 "No"   "No"  "No"  "No"  "No"  "Yes"
```

1.4 Wordcloud

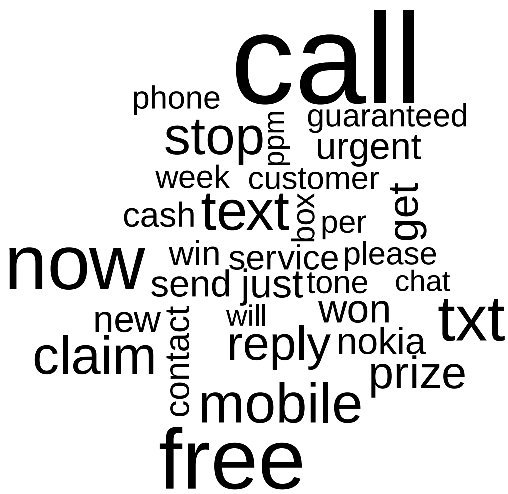
We can also plot wordclouds to obtain insight in the words found most commonly for *spam* or *ham*. We can now see the 10 most common words for *ham* labels:

```
wordcloud(clean_corpus[ham_indices], max.words=20, scale=c(3,.5))
```



And for *spam* labels:

```
wordcloud(clean_corpus[spam_indices], min.freq=40)
```



2 Model Usage

2.1 Naives Bayes

We now train a naive bayes classifier with our data. And make predictions for our testing set. Finally compare the predictions of our trained classifier with the real types of the observations with a table. And compute the error rate.

```
classifier <- naiveBayes(sms_dtm_train, sms_train$type)
class(classifier)
```

```
## [1] "naiveBayes"
```

```
predictions <- predict(classifier, newdata=sms_dtm_test)
table(predictions, sms_test$type)
```

```
##
## predictions  ham spam
##        ham  1204   22
##        spam    5  159
```

```
error <- mean(predictions!=sms_test$type);error
```

```
## [1] 0.01942446
```

2.2 Naive bayes with laplace smoother

And repeat the process but with the laplace smoother set to 1 in order to compare results.

```
B.clas <- naiveBayes(sms_dtm_train, sms_train$type,laplace = 1)
class(B.clas)
```

```
## [1] "naiveBayes"
```

```r
B.preds <- predict(B.clas, newdata=sms_dtm_test)
table(B.preds, sms_test$type)
```

```
##
## B.preds  ham spam
##    ham  1205   30
##    spam    4  151
```

```r
error <- mean(B.preds!=sms_test$type);error
```

```
## [1] 0.02446043
```