Text classification: Detection of spam messages

About the data set

The dataset that we use in this example is a preprocessed subset of the Ling-Spam Dataset. It is based on 960 real email messages from a linguistics mailing list. The dataset was originaly prepared for the Machine Learning course taught by Stanford Prof. Andrew Ng; it is downloaded from:
http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?course=MachineLearning&doc=exercises/ex6/ex6.html

The dataset is split into two subsets: a 700-email subset for training and a 260-email subset for testing; each subset contains 50% spam and 50% nonspam messages. The data is stored in several .txt files (each email message in a separate file) organized into 4 directories: spam-train, nonspam-train, spam-test, and nonspam-test. These directories are made available in the data/emails directory within this project.

Load the required R packages and utility functions

```
library(tm)

## Warning: package 'tm' was built under R version 3.4.3

library(caret)
library(class) # for kNN classifier

source("text_mining_utils.R")
```

Loading the traing and test sets

We'll start by loading the data that will be used for the training and testing of the classifier. First, get full names of the files (with email messages) to be used for the training set:

```
spam.train.files <- list.files(path = "data/emails/spam-train", full.names = TRUE)
#spam.train.files[1:10]

nonspam.train.files <- list.files(path = "data/emails/nonspam-train", full.names =
TRUE)
# nonspam.train.files[1:10]
```

Create a data frame for the training data; the data frame will have 3 columns (variables):

- message file path
- message label (spam / nonspam)
- message content (initially NA)

```
train.data <- data.frame(fpath = c(spam.train.files, nonspam.train.files),
                         label = c(rep("spam", times=length(spam.train.files)),
                                   rep("nonspam",
times=length(nonspam.train.files))),
                         text = NA, stringsAsFactors = FALSE)
str(train.data)
```

```
## 'data.frame':    700 obs. of  3 variables:
##  $ fpath: chr  "data/emails/spam-train/spmsga1.txt" "data/emails/spam-
train/spmsga10.txt" "data/emails/spam-train/spmsga100.txt" "data/emails/spam-
train/spmsga101.txt" ...
##  $ label: chr  "spam" "spam" "spam" "spam" ...
##  $ text : logi  NA NA NA NA NA NA ...
```

Read text from the training set of spam and nonspam messages, and use it to populate the *text* variable of the *train.data* dataframe:

```
for(i in 1:nrow(train.data)) {
  train.data$text[i] <- read.text(train.data$fpath[i])
}
# head(train.data)
```

We should just transform the *label* variable into a factor:

```
train.data$label <- as.factor(train.data$label)
summary(train.data$label)
```

```
## nonspam    spam
##     350     350
```

Now, load the test data and build a data frame that will be used for testing the classifier. The procedure is the same as the one we followed for the training data, so, we'll do all in one chunk:

```
spam.test.files <- list.files(path = "data/emails/spam-test", full.names = TRUE)
nonspam.test.files <- list.files(path = "data/emails/nonspam-test", full.names =
TRUE)

test.data <- data.frame(fpath = c(spam.test.files, nonspam.test.files),
                         label = c(rep("spam", times=length(spam.test.files)),
                                   rep("nonspam",
times=length(nonspam.test.files))),
                         text = NA, stringsAsFactors = FALSE)

for(i in 1:nrow(test.data)) {
  test.data$text[i] <- read.text(test.data$fpath[i])
}

test.data$label <- as.factor(test.data$label)

# head(test.data)
```

Data preparation (text preprocessing)

The loaded text is already pre-processed:

- stop-words have been removed
- numbers and punctuation have been removed
- text has been converted to lower case
- it has also been lemmatized
- all white spaces (tabs, newlines, spaces) have been trimmed to a single space character.

So, no need for pre-processing it here. We just need to create a corpus and then a Document Term Matrix.

Create a corpus

We will create the corpus using both training and testing data sets; later, when building a classifier, we will, of course, use just the training portion of the corpus. So, let's first merge the two data sets:

```
all.data <- rbind(train.data, test.data)
dim(all.data)
```

```
## [1] 960   3
```

We have 960 instances in total: first 700 are for training, the rest (260) for testing.

Now, we can create the corpus:

```
corpus <- Corpus(VectorSource(all.data$text))
```

Create a Document Term Matrix

Next, we create a Document Term Matrix (DTM) using words of length 2+ and term frequency (TF) weighting scheme (the default):

```
dtm <- DocumentTermMatrix(x = corpus,
                          control = list(wordLengths = c(2,Inf),
                                         weighting = weightTfIdf))
inspect(dtm)
```

```
## <<DocumentTermMatrix (documents: 960, terms: 22744)>>
## Non-/sparse entries: 149194/21685046
## Sparsity           : 99%
## Maximal term length: 74
## Weighting          : term frequency - inverse document frequency (normalized)
(tf-idf)
## Sample             :
##      Terms
## Docs  adult click com free        http    language our sex site
##   352     0     0   0    0 0.000000000 0.000000000   0   0    0
##   486     0     0   0    0 0.000000000 0.001850028   0   0    0
##   516     0     0   0    0 0.026949908 0.000000000   0   0    0
##   525     0     0   0    0 0.048717141 0.000000000   0   0    0
```

```
##   532     0     0   0    0 0.000000000 0.000000000   0   0    0
##   616     0     0   0    0 0.006529101 0.000000000   0   0    0
##   647     0     0   0    0 0.000000000 0.000000000   0   0    0
##   650     0     0   0    0 0.009279455 0.000000000   0   0    0
##   899     0     0   0    0 0.057574803 0.000000000   0   0    0
##   922     0     0   0    0 0.005579937 0.000000000   0   0    0
##      Terms
## Docs   university
##   352 0.000000000
##   486 0.002003405
##   516 0.034995654
##   525 0.000000000
##   532 0.010344627
##   616 0.000000000
##   647 0.000000000
##   650 0.000000000
##   899 0.000000000
##   922 0.007245796
```

We have very high number of words (almost 23K), and very sparse DTM (99%). So, we should better remove the sparse terms:

```
dtm.reduced <- removeSparseTerms(dtm, sparse = 0.975)
dtm.reduced

## <<DocumentTermMatrix (documents: 960, terms: 1268)>>
## Non-/sparse entries: 90327/1126953
## Sparsity           : 93%
## Maximal term length: 15
## Weighting          : term frequency - inverse document frequency (normalized)
(tf-idf)
```

This looks better: 1268 words preserved out of the original set of 22,744 words (~6%); the overall sparsity is reduced to 93%, and the max term length is reduced to 15 characters (from 74)

Since we want to use DTM for classification purposes, we need to transform it into a 'simple' matrix that can be passed to a function for building a classifier:

```
dtm.final <- as.matrix(dtm.reduced)
dim(dtm.final)

## [1]  960 1268
```

Split the matrix into training and test parts

```
training.dtm <- dtm.final[1:nrow(train.data),]
test.dtm <- dtm.final[(nrow(train.data)+1):nrow(dtm.final),]
```

Build kNN classifier

To execute the KNN classification in R, we will use the knn() f. from the *class* package (already loaded). We need to provide the knn() f. with the following data:

- training data with no labels,
- test data with no labels,
- labels for the training set,
- number of neighbours to consider (parameter k)

Initially, we will simply guess the number of neighbours (k), and later on we will apply a more systematic approach to determine the best value for k.

```
k <- 5
train.labels <- train.data$label
set.seed(2612)
knn.fit <- knn(train = training.dtm,
               test = test.dtm,
               cl = train.labels,
               k = k)
```

Create the confusion matrix

```
test.labels <- test.data$label
conf.mat <- table(Predictions = knn.fit,
                  Actual = test.labels)
conf.mat
```

```
##             Actual
## Predictions nonspam spam
##     nonspam     127   16
##     spam          3  114
```

Out of 260 observations in the test set, we have 16 false negatives (FN), and 3 false positives (FP). In this case, FPs are emails that are not spam but were classified as spam, whereas FNs are spam emails that were recognized as not spam. So, we should try to reduce FP (not spam that ends up in the Spam folder), even at the expense of (slightly) increasing FN (spam messages that end up in Inbox); in other words, we should be ready to 'traid' some recall for higher precision.

Compute evaluation measures:

```
knn1.eval <- compute.eval.measures(conf.mat)
knn1.eval
```

```
##  accuracy precision    recall        F1
##    0.9269    0.9769    0.8881    0.9304
```

Now, intead of guessing, we'll cross-validate kNN models with different values for k, and see which value fo k gives the best performance. Then, we'll use the *test.set* to test the model that proves to be the best.

The caret package will be used to find the optimal parameter (k) value through cross validation. First, define cross-validation (cv) parameters; we'll do 10-fold cross-validation:

```
numFolds = trainControl( method = "cv", number = 10 )
```

Then, define the range of k values to examine in the cross-validation. We'll take odd numbers between 3 and 25 - recall that in case of binary classification, it is recommended to choose an odd number for k

```
kGrid = expand.grid(.k = seq(from = 3, to = 25, by = 2))
```

Train the model through cross-validation

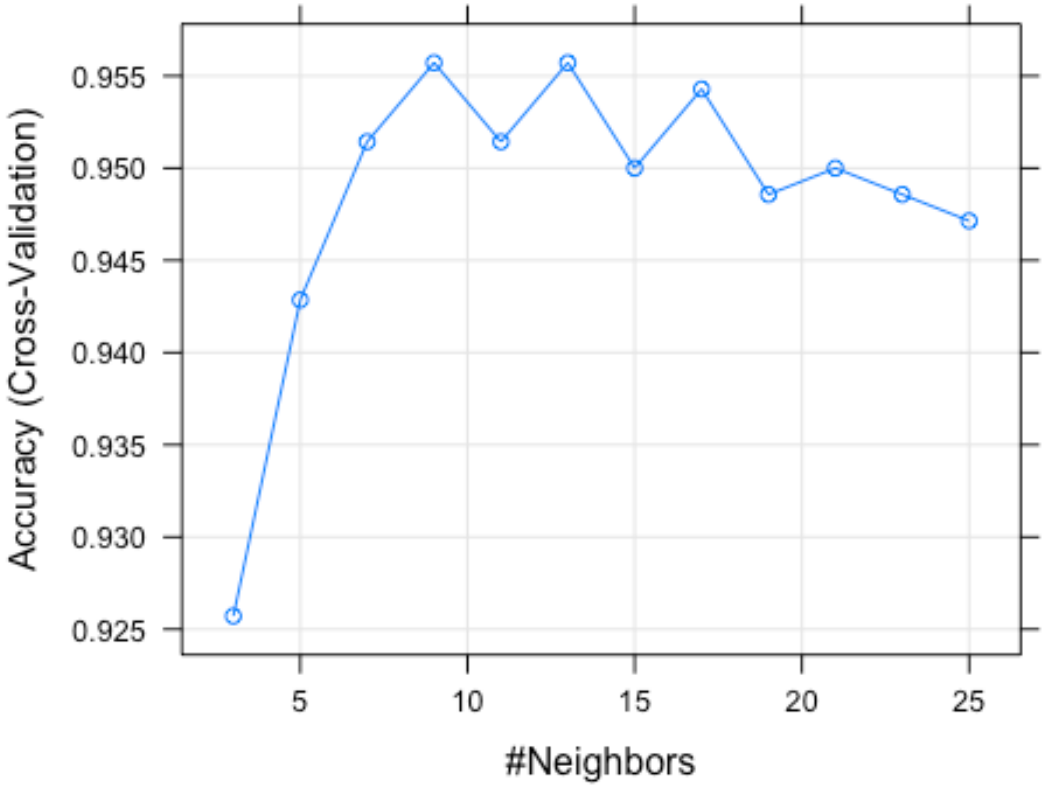
```
set.seed(2612)
knn.cv <- train(x = training.dtm,
                y = train.labels,
                method = "knn",
                trControl = numFolds,
                tuneGrid = kGrid)
```

Examine the obtained results for different values of k:

```
knn.cv
```

```
## k-Nearest Neighbors
##
##  700 samples
## 1268 predictors
##    2 classes: 'nonspam', 'spam'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 630, 630, 630, 630, 630, 630, ...
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa
##    3  0.9257143  0.8514286
##    5  0.9428571  0.8857143
##    7  0.9514286  0.9028571
##    9  0.9557143  0.9114286
##   11  0.9514286  0.9028571
##   13  0.9557143  0.9114286
##   15  0.9500000  0.9000000
##   17  0.9542857  0.9085714
##   19  0.9485714  0.8971429
##   21  0.9500000  0.9000000
##   23  0.9485714  0.8971429
##   25  0.9471429  0.8942857
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was k = 13.
```

```
plot(knn.cv)
```



For k=13, we get the best value for all the examined metrics. So, we choose k=13 as the number of neighbours.

```
knn.fit2 <- knn(train = training.dtm,
                test = test.dtm,
                cl = train.labels,
                k = 13)

conf.mat2 <- table(Predicted = knn.fit2, Actual = test.labels)
conf.mat2
```

```
##           Actual
## Predicted nonspam spam
##   nonspam     126   12
##   spam          4  118
```

We reduced the number of FNs, but slightly increased the number of FPs.

```
knn2.eval <- compute.eval.measures(conf.mat2)
knn2.eval
```

```
##  accuracy precision    recall        F1
##    0.9385    0.9692    0.9130    0.9403
```

Compare the evaluation measures obtained for the two models:

```
data.frame(rbind(knn1.eval, knn2.eval), row.names = c("k=5", "k=13"))
```

```
##      accuracy precision recall     F1
## k=5    0.9269    0.9769 0.8881 0.9304
## k=13   0.9385    0.9692 0.9130 0.9403
```

We've managed to improve all the metrics except for precision.