CS 199

HW2 - Data representation

March 4, 2014

Sam Laane `<laane2@illinois.edu>`
José Vicente Ruiz `<ruizcep2@illinois.edu>`

Contents

1 Comparing two classifiers on email spam

1.1 Implementation

```r
#!/usr/bin/env Rscript

# Install the random forest, cvTools and FNN packages.
#install.packages('randomForest')
#install.packages('cvTools')
#install.packages('FNN')

# Load the libraries.
library(randomForest)
library(cvTools)
library(FNN)

# Define functions.
printf <- function(...) invisible(print(sprintf(...)))

# Read the data.
spam <- read.csv('spambase.data', h=F)   # h=F (headers=FALSE)

# 10 fold cross validation (data divided in 10 parts randomly
   sampled)
accuracy <- 0
kaccuracy <- 0
folds <- cvFolds(nrow(spam), K=10)
for(i in 1:10) {
  # Print iteration.
  printf("Iterarion %d", i)

  # Separate train and test data.
  train <- spam[folds$subsets[folds$which != i], ]
  test  <- spam[folds$subsets[folds$which == i], ]

  # Random Forest classifier:
  #  - Create a random forest using the training data.
  clf <- randomForest(factor(train$V58) ~ ., data=train[,1:54],
     ntree=100)

  #  - Classify the test data with the random forest classifier.
  labels <- predict(clf, test[,1:54])

  #  - Get the number of correctly classified samples.
  hits <- length(labels[labels == test[,58]])
  accuracy <- accuracy + (hits / nrow(test))

  # Nearest Neighbours classifier:
  #  - Classify using fields 1 to 54 as it gets far better
     results than 1 to 57.
```

```
  klabels<- knn(train[1:54], test[1:54], train$V58, k = 1, prob =
      FALSE,
                algorithm=c("kd_tree"))

  #  - Get the number of correctly classified samples.
  khits <- length(klabels[klabels == test[,58]])
  kaccuracy <- kaccuracy + (khits /nrow(test))
}

# Print the resulting accuracies.
accuracy <- (accuracy / 10) * 100
kaccuracy <- (kaccuracy / 10) * 100

printf("Random forest accuracy: %.2f %%", accuracy)
printf("KNN accuracy: %.2f %%", kaccuracy)
```

1.2 Results

The *random forest* gets an accuracy of about 95%. This is slightly better than *k-nearest neighbor*, which has an accuracy of about 91%.

As for which is "better", while *random forests* have a higher accuracy they are also slower to run. In practice, we would go with *random forests* as the time to build then is not a big concern.

2 Filtering SMS spam

2.1 Implementation

```
#!/usr/bin/env Rscript

# Install required packages.
#install.packages('randomForest')
#install.packages('cvTools')
#install.packages('FNN')
#install.packages("tm")
#install.packages("SnowballC")

# Load the libraries.
library(randomForest)
library(cvTools)
library(FNN)
library(tm)
library(SnowballC)

# Define functions.
printf <- function(...) invisible(print(sprintf(...)))

# Read all the lines of the document and store them in a vector of characters.
msgChars <- readLines("Texts/SMSSpamCollection")


# Split by tabs since the labels is separated with a \t from the content.
msgList <- strsplit(msgChars, '\t')

# Transform the list into a matrix (do.class() applies the operation to every element in the list).
msgMatrix <- do.call(rbind, msgList)

# Get the labels and convert them to integer.
trueLabels <- (msgMatrix[,1]=="spam") + 0 # Add 0 to convert from logical to int.

# Remove non-ASCII words.
content <- c(msgMatrix[,2])
Encoding(content) <- "latin1"
content <- iconv(content, "latin1", "ASCII", sub="")

# Create the corpus data only the content of the mails (one document per line).
corpus <- Corpus(VectorSource(content))

# Remove punctuation symbols.
```

```
corpus <- tm_map(corpus, removePunctuation, preserve_intra_word_dashes=T)

# Everything to lower case.
corpus <- tm_map(corpus, tolower)

# Remove stop words (case sensitive).
corpus <- tm_map(corpus, removeWords, stopwords("english"))

# Stem the document words, i.e., reduce them to the root.
corpus <- tm_map(corpus, stemDocument, language="english")

# Remove extra whitespaces.
corpus <- tm_map(corpus, stripWhitespace)

# Build a document term matrix from the corpus.
dtm <- DocumentTermMatrix(corpus)

# Remove sparse terms to get a managable number of terms.
dtm <- removeSparseTerms(dtm, 0.98)

# Convert the document term matrix to a standard matrix.
freqMatrix <- as.data.frame( as.matrix(dtm) )

# Normalize the frequency matrix: 0 if absent, 1 if present.
spam <- (freqMatrix > 0) + 0 # Add 0 to convert from logical to int.

# Delete the header names to avoid collisions with R reserved words.
spam <- matrix(spam, ncol=ncol(spam), dimnames=NULL)

# 10 fold cross validation (data divided in 10 parts randomly sampled)
accuracy <- 0
kaccuracy <- 0
accuracyOfAssumingHam <- 0
folds <- cvFolds(nrow(spam), K=10)
for(i in 1:10) {
  # Print iteration.
  printf("Iterarion %d", i)

  # Separate train and test data and labels.
  trainData <- spam[folds$subsets[folds$which != i], ]
  testData  <- spam[folds$subsets[folds$which == i], ]
  trainLabels <- trueLabels[folds$subsets[folds$which != i]]
  testLabels  <- trueLabels[folds$subsets[folds$which == i]]

  # Random Forest classifier:
  #  - Create a random forest using the training data.
```

```
  clf <- randomForest(factor(trainLabels) ~ ., data=trainData,
     ntree=30)

  #  - Classify the test data with the random forest classifier.
  labels <- predict(clf, testData)

  #  - Get the number of correctly classified samples.
  hits <- length(labels[labels == testLabels])
  accuracy <- accuracy + (hits / nrow(testData))

  # Nearest Neighbours classifier:
  #  - Classify using fields 1 to 54 as it gets far better
     results than 1 to 57.
  klabels<- knn(trainData, testData, trainLabels, k = 1, prob =
     FALSE,
             algorithm=c("kd_tree"))

  #  - Get the number of correctly classified samples.
  khits <- length(klabels[klabels == testLabels])
  kaccuracy <- kaccuracy + (khits /nrow(testData))

  ahits <- nrow(testData) -sum(testLabels)
  accuracyOfAssumingHam <- accuracyOfAssumingHam + (ahits/nrow(
     testData))

}

# Print the resulting accuracies.
accuracy <- (accuracy / 10) * 100
kaccuracy <- (kaccuracy / 10) * 100
accuracyOfAssumingHam <- (accuracyOfAssumingHam/ 10) * 100

printf("Random forest accuracy: %.2f %%", accuracy)
printf("KNN accuracy: %.2f %%", kaccuracy)
printf("Assuming nothing is spam %.2f %%", accuracyOfAssumingHam)
```

2.2 Results

The preference of our classifier's depends on the size of our tokenized feature vector (see more below). With a size of 51 our *random forest* is again the most accurate. It's accuracy is about 95%.

K-nearest neighbour was very close with an accuracy of about 93%. The trivial filtering strategy of simply assuming everything is not spam has a consonant accuracy of 86.60%. As one can see our filters can out perform better than this simple strategy.

3 Playing with tokenization

3.1 Implementation

```r
#!/usr/bin/env Rscript

# Install required packages.
#install.packages('randomForest')
#install.packages('cvTools')
#install.packages('FNN')
#install.packages("tm")
#install.packages("SnowballC")

# Load the libraries.
library(randomForest)
library(cvTools)
library(FNN)
library(tm)
library(SnowballC)

# Define functions.
printf <- function(...) invisible(print(sprintf(...)))

# Read all the lines of the document and store them in a vector of characters.
msgChars <- readLines("Texts/SMSSpamCollection")


# Split by tabs since the labels is separated with a \t from the content.
msgList <- strsplit(msgChars, '\t')

# Transform the list into a matrix (do.class() applies the operation to every element in the list).
msgMatrix <- do.call(rbind, msgList)

# Get the labels and convert them to integer.
trueLabels <- (msgMatrix[,1]=="spam") + 0 # Add 0 to convert from logical to int.

# Remove non-ASCII words.
content <- c(msgMatrix[,2])
Encoding(content) <- "latin1"
content <- iconv(content, "latin1", "ASCII", sub="")

# Create the corpus data only the content of the mails (one document per line).
corpus <- Corpus(VectorSource(content))

# Remove punctuation symbols.
```

```
corpus <- tm_map(corpus, removePunctuation, preserve_intra_word_dashes=T)

# Everything to lower case.
corpus <- tm_map(corpus, tolower)

# Remove stop words (case sensitive).
corpus <- tm_map(corpus, removeWords, stopwords("english"))

# Stem the document words, i.e., reduce them to the root.
corpus <- tm_map(corpus, stemDocument, language="english")

# Remove extra whitespaces.
corpus <- tm_map(corpus, stripWhitespace)

# Build a document term matrix from the corpus.
dtm <- DocumentTermMatrix(corpus)

termCount = c(".94", ".95",".97",".98",".99")
termCount
spam <- new.env()
accuracy<- new.env()
kaccuracy <- new.env()

for (v in termCount){
    # Remove sparse terms to get a managable number of terms.
    dtmtmp <- removeSparseTerms(dtm, as.numeric(v))
    # Convert the document term matrix to a standard matrix.
    freqMatrix <- as.data.frame( as.matrix(dtmtmp) )
    # Normalize the frequency matrix: 0 if absent, 1 if present.
    spamt <- (freqMatrix > 0) + 0 # Add 0 to convert from logical to int.
    # Delete the header names to avoid collisions with R reserved words.
    spam[[v]] <- matrix(spamt, ncol=ncol(spamt), dimnames=NULL)
    accuracy[[v]] <- 0
    kaccuracy[[v]] <- 0
}

# 10 fold cross validation (data divided in 10 parts randomly sampled)
accuracyOfAssumingHam <- 0
folds <- cvFolds(nrow(spam), K=10)
for(i in 1:10) {
    # Print iteration.
    printf("Iterarion %d", i)
    for (v in termCount){
        # Separate train and test data and labels.
        trainData <- spam[[v]][folds$subsets[folds$which != i], ]
        testData  <- spam[[v]][folds$subsets[folds$which == i], ]
```

```
        trainLabels <- trueLabels[folds$subsets[folds$which != i]]
        testLabels  <- trueLabels[folds$subsets[folds$which == i]]

        # Random Forest classifier:
        #  - Create a random forest using the training data.
        clf <- randomForest(factor(trainLabels) ~ ., data=trainData, ntree=30)

        #  - Classify the test data with the random forest classifier.
        labels <- predict(clf, testData)

        #  - Get the number of correctly classified samples.
        hits <- length(labels[labels == testLabels])
        accuracy[[v]] <- accuracy[[v]] + (hits / nrow(testData))

        # Nearest Neighbours classifier:
        #  - Classify using fields 1 to 54 as it gets far better results than 1 to 57.
        klabels<- knn(trainData, testData, trainLabels, k = 1, prob = FALSE,
                      algorithm=c("kd_tree"))

        #  - Get the number of correctly classified samples.
        khits <- length(klabels[klabels == testLabels])
        kaccuracy[[v]] <- kaccuracy[[v]] + (khits /nrow(testData))
    }
        ahits <- nrow(testData) -sum(testLabels)
        accuracyOfAssumingHam <- accuracyOfAssumingHam + (ahits/nrow(testData))

}

accuracyOfAssumingHam <- (accuracyOfAssumingHam/ 10) * 100

for (v in termCount){
    # Print the resulting accuracies.
    accuracy[[v]] <- (accuracy[[v]] / 10) * 100
    kaccuracy[[v]] <- (kaccuracy[[v]] / 10) * 100

    printf("Random forest accuracy: %.2f %%", accuracy[[v]])
    printf("KNN accuracy: %.2f %%", kaccuracy[[v]])
    printf("Assuming nothing is spam %.2f %%", accuracyOfAssumingHam)
    print(ncol(spam[[v]]))
    print(nrow(spam[[v]]))
}
```

3.2 Results

The resulting accuracies with different vector sizes (VS), obtained varying the restriction level on the sparsity of the terms, have been summarized in the following table:

Filtering strategy	**VS=6**	**VS=7**	**VS=24**	**VS=51**	**VS=139**
Random forest 100 trees	87.80	88.34	91.66	95.07	96.54
Nearest neighbor kd-tree	83.85	84.30	89.59	93.43	94.39
Ham trivial filtering	86.60	86.60	86.60	86.60	86.60