

CS 199

# HW2 - Data representation

March 4, 2014

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

## Contents

<b>1</b>	<b>Comparing two classifiers on email spam</b>	<b>2</b>
1.1	Implementation . . . . .	2
1.2	Results . . . . .	3
<b>2</b>	<b>Filtering SMS spam</b>	<b>4</b>
2.1	Implementation . . . . .	4
2.2	Results . . . . .	6
<b>3</b>	<b>Playing with tokenization</b>	<b>7</b>
3.1	Implementation . . . . .	7
3.2	Results . . . . .	10

# 1 Comparing two classifiers on email spam

## 1.1 Implementation

```
1  #!/usr/bin/env Rscript
2
3  # Install the random forest, cvTools and FNN packages.
4  #install.packages('randomForest')
5  #install.packages('cvTools')
6  #install.packages('FNN')
7
8  # Load the libraries.
9  library(randomForest)
10 library(cvTools)
11 library(FNN)
12
13 # Define functions.
14 printf <- function(...) invisible(print(sprintf(...)))
15
16 # Read the data.
17 spam <- read.csv('spambase.data', h=F)    # h=F (headers=FALSE)
18
19 # 10 fold cross validation (data divided in 10 parts randomly
    sampled)
20 accuracy <- 0
21 kaccuracy <- 0
22 folds <- cvFolds(nrow(spam), K=10)
23 for(i in 1:10) {
24   # Print iteration.
25   printf("Iteration %d", i)
26
27   # Separate train and test data.
28   train <- spam[folds$subsets[folds$which != i], ]
29   test  <- spam[folds$subsets[folds$which == i], ]
30
31   # Random Forest classifier:
32   # - Create a random forest using the training data.
33   clf <- randomForest(factor(train$V58) ~ ., data=train[,1:54],
        ntree=100)
34
35   # - Classify the test data with the random forest classifier.
36   labels <- predict(clf, test[,1:54])
37
38   # - Get the number of correctly classified samples.
39   hits <- length(labels[labels == test[,58]])
40   accuracy <- accuracy + (hits / nrow(test))
41
42   # Nearest Neighbours classifier:
43   # - Classify using fields 1 to 54 as it gets far better
        results than 1 to 57.
```

```
44   klabels<- knn(train[1:54], test[1:54], train$V58, k = 1, prob =  
      FALSE,  
45               algorithm=c("kd_tree"))  
46  
47   # - Get the number of correctly classified samples.  
48   khits <- length(klabels[klabels == test[,58]])  
49   kaccuracy <- kaccuracy + (khits /nrow(test))  
50 }  
51  
52 # Print the resulting accuracies.  
53 accuracy <- (accuracy / 10) * 100  
54 kaccuracy <- (kaccuracy / 10) * 100  
55  
56 printf("Random forest accuracy: %.2f %%", accuracy)  
57 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

```
1  #!/usr/bin/env Rscript
2
3  # Install required packages.
4  #install.packages('randomForest')
5  #install.packages('cvTools')
6  #install.packages('FNN')
7  #install.packages("tm")
8  #install.packages("SnowballC")
9
10 # Load the libraries.
11 library(randomForest)
12 library(cvTools)
13 library(FNN)
14 library(tm)
15 library(SnowballC)
16
17 # Define functions.
18 printf <- function(...) invisible(print(sprintf(...)))
19
20 # Read all the lines of the document and store them in a vector
    of characters.
21 msgChars <- readLines("Texts/SMSSpamCollection")
22
23
24 # Split by tabs since the labels is separated with a \t from the
    content.
25 msgList <- strsplit(msgChars, '\t')
26
27 # Transform the list into a matrix (do.call() applies the
    operation to every element in the list).
28 msgMatrix <- do.call(rbind, msgList)
29
30 # Get the labels and convert them to integer.
31 trueLabels <- (msgMatrix[,1]=="spam") + 0 # Add 0 to convert from
    logical to int.
32
33 # Remove non-ASCII words.
34 content <- c(msgMatrix[,2])
35 Encoding(content) <- "latin1"
36 content <- iconv(content, "latin1", "ASCII", sub="")
37
38 # Create the corpus data only the content of the mails (one
    document per line).
39 corpus <- Corpus(VectorSource(content))
40
41 # Remove punctuation symbols.
```

```
42 corpus <- tm_map(corpus, removePunctuation, preserve_intra_word_
    dashes=T)
43
44 # Everything to lower case.
45 corpus <- tm_map(corpus, tolower)
46
47 # Remove stop words (case sensitive).
48 corpus <- tm_map(corpus, removeWords, stopwords("english"))
49
50 # Stem the document words, i.e., reduce them to the root.
51 corpus <- tm_map(corpus, stemDocument, language="english")
52
53 # Remove extra whitespaces.
54 corpus <- tm_map(corpus, stripWhitespace)
55
56 # Build a document term matrix from the corpus.
57 dtm <- DocumentTermMatrix(corpus)
58
59 # Remove sparse terms to get a manageable number of terms.
60 dtm <- removeSparseTerms(dtm, 0.98)
61
62 # Convert the document term matrix to a standard matrix.
63 freqMatrix <- as.data.frame( as.matrix(dtm) )
64
65 # Normalize the frequency matrix: 0 if absent, 1 if present.
66 spam <- (freqMatrix > 0) + 0 # Add 0 to convert from logical to
    int.
67
68 # Delete the header names to avoid collisions with R reserved
    words.
69 spam <- matrix(spam, ncol=ncol(spam), dimnames=NULL)
70
71 # 10 fold cross validation (data divided in 10 parts randomly
    sampled)
72 accuracy <- 0
73 kaccuracy <- 0
74 accuracyOfAssumingHam <- 0
75 folds <- cvFolds(nrow(spam), K=10)
76 for(i in 1:10) {
77     # Print iteration.
78     printf("Iterarion %d", i)
79
80     # Separate train and test data and labels.
81     trainData <- spam[folds$subsets[folds$which != i], ]
82     testData <- spam[folds$subsets[folds$which == i], ]
83     trainLabels <- trueLabels[folds$subsets[folds$which != i]]
84     testLabels <- trueLabels[folds$subsets[folds$which == i]]
85
86     # Random Forest classifier:
87     # - Create a random forest using the training data.
```

```

88   clf <- randomForest(factor(trainLabels) ~ ., data=trainData,
89                        ntree=30)
89
90   # - Classify the test data with the random forest classifier.
91   labels <- predict(clf, testData)
92
93   # - Get the number of correctly classified samples.
94   hits <- length(labels[labels == testLabels])
95   accuracy <- accuracy + (hits / nrow(testData))
96
97   # Nearest Neighbours classifier:
98   # - Classify using fields 1 to 54 as it gets far better
99     results than 1 to 57.
100  klabels<- knn(trainData, testData, trainLabels, k = 1, prob =
101                FALSE,
102                algorithm=c("kd_tree"))
103
104  # - Get the number of correctly classified samples.
105  khits <- length(klabels[klabels == testLabels])
106  kaccuracy <- kaccuracy + (khits /nrow(testData))
107
108  ahits <- nrow(testData) -sum(testLabels)
109  accuracyOfAssumingHam <- accuracyOfAssumingHam + (ahits/nrow(
110    testData))
111
112  # Print the resulting accuracies.
113  accuracy <- (accuracy / 10) * 100
114  kaccuracy <- (kaccuracy / 10) * 100
115  accuracyOfAssumingHam <- (accuracyOfAssumingHam/ 10) * 100
116
117  printf("Random forest accuracy: %.2f %%", accuracy)
118  printf("KNN accuracy: %.2f %%", kaccuracy)
119  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

```
1  #!/usr/bin/env Rscript
2
3  # Install required packages.
4  #install.packages('randomForest')
5  #install.packages('cvTools')
6  #install.packages('FNN')
7  #install.packages("tm")
8  #install.packages("SnowballC")
9
10 # Load the libraries.
11 library(randomForest)
12 library(cvTools)
13 library(FNN)
14 library(tm)
15 library(SnowballC)
16
17 # Define functions.
18 printf <- function(...) invisible(print(sprintf(...)))
19
20 # Read all the lines of the document and store them in a vector
  # of characters.
21 msgChars <- readLines("Texts/SMS SpamCollection")
22
23
24 # Split by tabs since the labels is separated with a \t from the
  # content.
25 msgList <- strsplit(msgChars, '\t')
26
27 # Transform the list into a matrix (do.call() applies the
  # operation to every element in the list).
28 msgMatrix <- do.call(rbind, msgList)
29
30 # Get the labels and convert them to integer.
31 trueLabels <- (msgMatrix[,1]=="spam") + 0 # Add 0 to convert from
  # logical to int.
32
33 # Remove non-ASCII words.
34 content <- c(msgMatrix[,2])
35 Encoding(content) <- "latin1"
36 content <- iconv(content, "latin1", "ASCII", sub="")
37
38 # Create the corpus data only the content of the mails (one
  # document per line).
39 corpus <- Corpus(VectorSource(content))
40
41 # Remove punctuation symbols.
```



```
42 corpus <- tm_map(corpus, removePunctuation, preserve_intra_word_
    dashes=T)
43
44 # Everything to lower case.
45 corpus <- tm_map(corpus, tolower)
46
47 # Remove stop words (case sensitive).
48 corpus <- tm_map(corpus, removeWords, stopwords("english"))
49
50 # Stem the document words, i.e., reduce them to the root.
51 corpus <- tm_map(corpus, stemDocument, language="english")
52
53 # Remove extra whitespaces.
54 corpus <- tm_map(corpus, stripWhitespace)
55
56 # Build a document term matrix from the corpus.
57 dtm <- DocumentTermMatrix(corpus)
58
59 termCount = c(".94", ".95", ".97", ".98", ".99")
60 termCount
61 spam <- new.env()
62 accuracy<- new.env()
63 kaccuracy <- new.env()
64
65 for (v in termCount){
66     # Remove sparse terms to get a managable number of terms.
67     dtmtmp <- removeSparseTerms(dtm, as.numeric(v))
68     # Convert the document term matrix to a standard matrix.
69     freqMatrix <- as.data.frame( as.matrix(dtmtmp) )
70     # Normalize the frequency matrix: 0 if absent, 1 if present.
71     spamt <- (freqMatrix > 0) + 0 # Add 0 to convert from logical
        to int.
72     # Delete the header names to avoid collisions with R reserved
        words.
73     spam[[v]] <- matrix(spamt, ncol=ncol(spamt), dimnames=NULL)
74     accuracy[[v]] <- 0
75     kaccuracy[[v]] <- 0
76 }
77
78 # 10 fold cross validation (data divided in 10 parts randomly
    sampled)
79 accuracyOfAssumingHam <- 0
80 folds <- cvFolds(nrow(spam), K=10)
81 for(i in 1:10) {
82     # Print iteration.
83     printf("Iterarion %d", i)
84     for (v in termCount){
85         # Separate train and test data and labels.
86         trainData <- spam[[v]][folds$subsets[folds$which != i], ]
87         testData <- spam[[v]][folds$subsets[folds$which == i], ]
```

```

88     trainLabels <- trueLabels[folds$subsets[folds$which != i
      ]]
89     testLabels  <- trueLabels[folds$subsets[folds$which == i
      ]]
90
91     # Random Forest classifier:
92     # - Create a random forest using the training data.
93     clf <- randomForest(factor(trainLabels) ~ ., data=
      trainData, ntree=30)
94
95     # - Classify the test data with the random forest
      classifier.
96     labels <- predict(clf, testData)
97
98     # - Get the number of correctly classified samples.
99     hits <- length(labels[labels == testLabels])
100    accuracy[[v]] <- accuracy[[v]] + (hits / nrow(testData))
101
102    # Nearest Neighbours classifier:
103    # - Classify using fields 1 to 54 as it gets far better
      results than 1 to 57.
104    klabels<- knn(trainData, testData, trainLabels, k = 1,
      prob = FALSE,
105                algorithm=c("kd_tree"))
106
107    # - Get the number of correctly classified samples.
108    khits <- length(klabels[klabels == testLabels])
109    kaccuracy[[v]] <- kaccuracy[[v]] + (khits /nrow(testData)
      )
110  }
111    ahits <- nrow(testData) -sum(testLabels)
112    accuracyOfAssumingHam <- accuracyOfAssumingHam + (ahits/
      nrow(testData))
113
114 }
115
116 accuracyOfAssumingHam <- (accuracyOfAssumingHam/ 10) * 100
117
118 for (v in termCount){
119   # Print the resulting accuracies.
120   accuracy[[v]] <- (accuracy[[v]] / 10) * 100
121   kaccuracy[[v]] <- (kaccuracy[[v]] / 10) * 100
122
123   printf("Random forest accuracy: %.2f %%", accuracy[[v]])
124   printf("KNN accuracy: %.2f %%", kaccuracy[[v]])
125   printf("Assuming nothing is spam %.2f %%",
      accuracyOfAssumingHam)
126   print(ncol(spam[[v]]))
127   print(nrow(spam[[v]]))
128 }

```

### 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
<i>Random forest 100 trees</i>	87.80	88.34	91.66	95.07	96.54
<i>Nearest neighbor kd-tree</i>	83.85	84.30	89.59	93.43	94.39
<i>Ham trivial filtering</i>	86.60	86.60	86.60	86.60	86.60