Intelligent Systems Seminar Assignment 2

Mateja Cerina, Jaka Milavec

3.1.2022

Goal

The goal of the assignment was to inspect, transform and learn from text input. We were provided a pre-split collection of labeled documents related to detection of fake news. The task was to pre-process the documents, do feature construction, model of the documents, and evaluate them as either fake (0) or true (1).

```
library(CORElearn)
library(ggplot2)
library(tm)
library(wordcloud)
library(class)
library(caret)
library(e1071)
library(rpart)
library(randomForest)
library(ipred)
```

1 Pre-processing

Our first task was to clean the documents of possible noise, including, e.g., URL links, strange symbols etc. We decided to put everything in lowercase, remove punctuation, numbers and stopwords. We also removed URLs and stemmed the words. Finally we removed whitespaces and only left words with characters from the English alphabet in the documents.

```
test <- read.table(file="test_data.tsv", sep="\t",
                   header=TRUE, comment.char = "", quote = "")

test_corpus <- Corpus(VectorSource(test$text_a))
test_corpus <- tm_map(test_corpus, content_transformer(tolower))
test_corpus <- tm_map(test_corpus, removePunctuation)
test_corpus <- tm_map(test_corpus, removeNumbers)
test_corpus <- tm_map(test_corpus, removeWords, stopwords("english"))
removeURL <- function(x) gsub("http[[:alnum:]]*", "", x)
test_corpus <- tm_map(test_corpus, content_transformer(removeURL))
test_corpus <- tm_map(test_corpus, stemDocument)
removeSymbols <- function(x) gsub("[^ a-z]", "", x)
test_corpus <- tm_map(test_corpus, content_transformer(removeSymbols))
test_corpus <- tm_map(test_corpus, stripWhitespace)
```

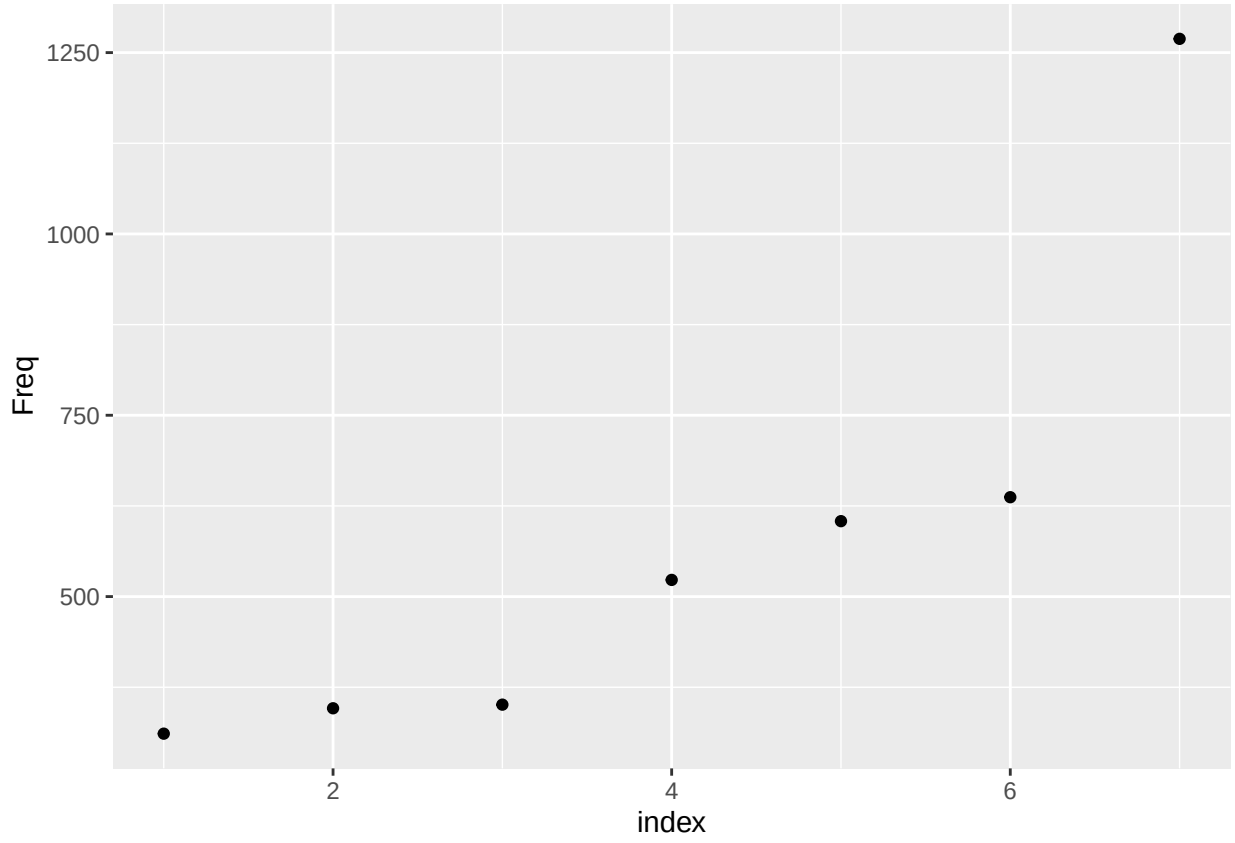
```
train <- read.table(file="train_data.tsv", sep="\t",
                    header=TRUE, comment.char = "", quote = "")

train_corpus <- Corpus(VectorSource(train$text_a))
train_corpus <- tm_map(train_corpus, content_transformer(tolower))
train_corpus <- tm_map(train_corpus, removePunctuation)
train_corpus <- tm_map(train_corpus, removeNumbers)
train_corpus <- tm_map(train_corpus, removeWords, stopwords("english"))
train_corpus <- tm_map(train_corpus, content_transformer(removeURL))
train_corpus <- tm_map(train_corpus, stemDocument)
train_corpus <- tm_map(train_corpus, content_transformer(removeSymbols))
train_corpus <- tm_map(train_corpus, stripWhitespace)
```

We converted the testing and training data into separate matrices and located the most frequent terms in the documents.

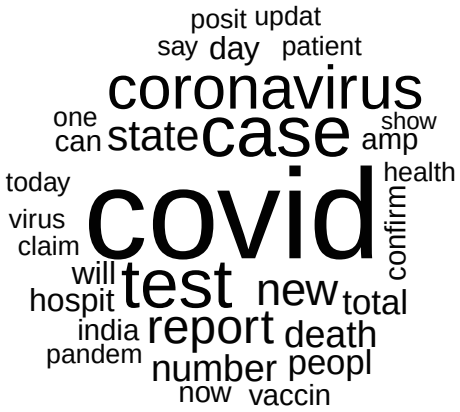
```
test_tdm <- TermDocumentMatrix(test_corpus)
findFreqTerms(test_tdm, lowfreq = 300)
```

```
## [1] "case"        "covid"       "new"         "report"      "state"
## [6] "test"        "coronavirus"
```

```
termFrequency <- rowSums(as.matrix(test_tdm))
termFrequency <- subset(termFrequency, termFrequency >= 300)
qplot(seq(length(termFrequency)), sort(termFrequency), xlab = "index", ylab = "Freq")
```



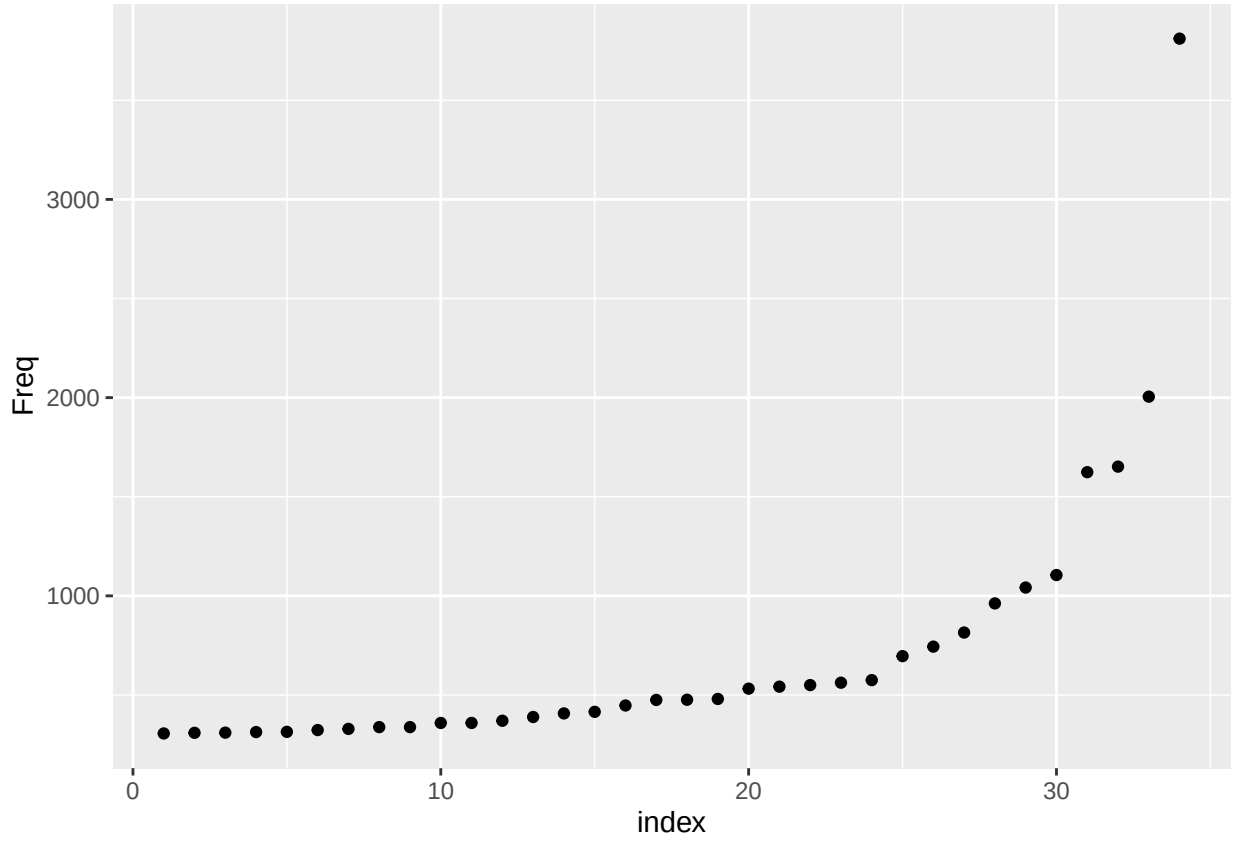
```
mat <- as.matrix(test_tdm)
wordFreq <- sort(rowSums(mat), decreasing = TRUE)
grayLevels <- gray((wordFreq + 10) / (max(wordFreq) + 10))
wordcloud(words = names(wordFreq), freq = wordFreq, min.freq = 100,
          random.order = F, colors = grayLevels)
```



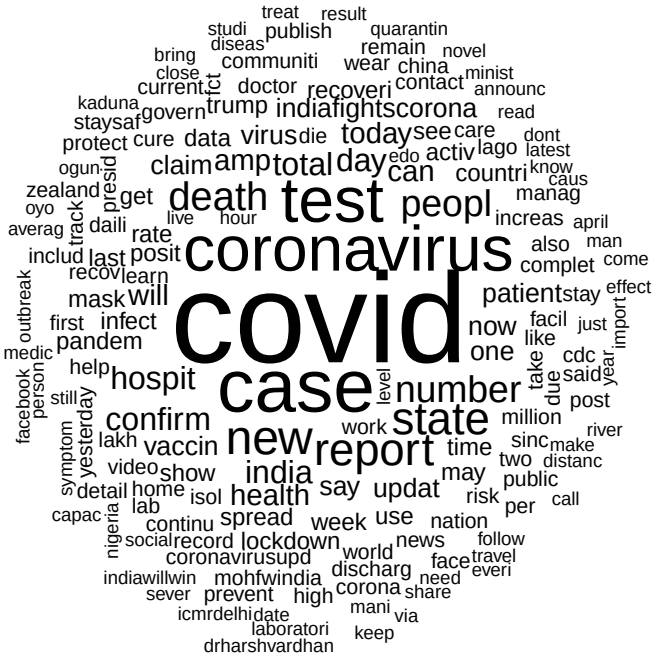
```r
train_tdm2 <- TermDocumentMatrix(train_corpus)
findFreqTerms(train_tdm2, lowfreq = 300)
```

```
##  [1] "coronavirus"       "death"             "covid"
##  [4] "number"            "one"               "state"
##  [7] "will"              "health"            "india"
## [10] "indiafightscorona" "hospit"            "peopl"
## [13] "posit"             "test"              "case"
## [16] "total"             "can"               "now"
## [19] "report"            "updat"             "rate"
## [22] "confirm"           "virus"             "vaccin"
## [25] "today"             "amp"               "show"
## [28] "pandem"            "say"               "day"
## [31] "infect"            "patient"           "new"
## [34] "claim"
```

```r
termFrequency2 <- rowSums(as.matrix(train_tdm2))
termFrequency2 <- subset(termFrequency2, termFrequency2 >= 300)
qplot(seq(length(termFrequency2)), sort(termFrequency2), xlab = "index", ylab = "Freq")
```



```
mat2 <- as.matrix(train_tdm2)
wordFreq2 <- sort(rowSums(mat2), decreasing=TRUE)
grayLevels2 <- gray((wordFreq2 + 10) / (max(wordFreq2) + 10))
wordcloud(words = names(wordFreq2), freq=wordFreq2, min.freq = 100,
          random.order = F, colors = grayLevels2)
```



From the most frequent words in the documents we can observe that they are about the ongoing pandemic, coronavirus, covid, the vaccine, most affected countries and the effects of the pandemic. We can also see that the size of the train data is bigger than the test data by a big margin.

2 Feature construction

To convert the documents into feature matrices suitable for learning, we use the `DocumentTermMatrix()` function. First we convert the training set and since there are a lot of sparse entries, we use the `removeSparseTerms()` function to remove them. We then use the result of this function when making a document-term matrix for the testing set. This ensures that we only use the words that appear in the training set. We can see that both document-term matrices have the same number of columns, i.e. terms, and a different number of rows, i.e. number of documents, which correspond to our training and testing set files.

When removing sparse terms we used values `0.90` for kNN and `0.99` for the other classifiers. We did this because setting a lower value gave us a better accuracy when running the kNN classifier. In other examples the value `0.99` gave us a better accuracy versus processing time ratio.

```
train_dtm1kNN.tfidf <- DocumentTermMatrix(train_corpus,
                                          control = list(weighting = weightTfIdf))
train_dtmS1kNN.tfidf <- removeSparseTerms(train_dtm1kNN.tfidf, 0.90)

mat1kNN.mat <- as.matrix(train_dtmS1kNN.tfidf)

test_dtm2kNN.tfidf <- DocumentTermMatrix(test_corpus,
                                         control = list(
                                           dictionary=Terms(train_dtmS1kNN.tfidf),
                                                      weighting = weightTfIdf))
mat2kNN.mat <- as.matrix(test_dtm2kNN.tfidf)

test_label <- test$label
train_label <- train$label

test_datakNN <- cbind(mat2kNN.mat, test_label)
names(test_datakNN)[ncol(test_datakNN)] <- "Label"

train_datakNN <- cbind(mat1kNN.mat, train_label)
names(train_datakNN)[ncol(train_datakNN)] <- "Label"
```

```
train_dtm1.tfidf <- DocumentTermMatrix(train_corpus,
                                       control = list(weighting = weightTfIdf))

train_dtmS1.tfidf <- removeSparseTerms(train_dtm1.tfidf, 0.99)
mat1.mat <- as.matrix(train_dtmS1.tfidf)

test_dtm2.tfidf <- DocumentTermMatrix(test_corpus,
                                      control = list(dictionary=Terms(train_dtmS1.tfidf),
                                                     weighting = weightTfIdf))
mat2.mat <- as.matrix(test_dtm2.tfidf)

test_label <- test$label
train_label <- train$label

test_data <- cbind(mat2.mat, test_label)
names(test_data)[ncol(test_data)] <- "Label"

train_data <- cbind(mat1.mat, train_label)
names(train_data)[ncol(train_data)] <- "Label"
```

3 Modeling

For modeling we used four machine learning classifiers (kNN, SVM, Random Forrest and Naive Bayes) and one ensemble method (Bagging). We trained the models on the training data and produced the predictions for the test instances.

```
r1 <- which(names(test_datakNN)=="Label")
r2 <- which(names(train_datakNN)=="Label")

predictedKNN <- knn(train_datakNN[, -r1], test_datakNN[, -r2],
                    train_datakNN[, r1], k = 100)
observed <- test_label
```

```
svm.train <- train_data
svm.test <- test_data

model.svm <- train(as.factor(train_label) ~., data = svm.train, method = "svmLinear")
predictedSVM <- predict(model.svm, svm.test, type = "raw")
```

```
rf <- randomForest(as.factor(train_label) ~., data = train_data, ntree = 700)
predictedRF <- predict(rf, newdata = test_data)
```

```
bayes <- CoreModel(as.factor(train_label) ~., data = as.data.frame(train_data),
                   model = "bayes")
predictedNB <- predict(bayes, as.data.frame(test_data), type = "class")
```

```
bag <- bagging(as.factor(train_label) ~., data = as.data.frame(train_data), nbagg = 50)
predictedBAG <- predict(bag, test_data, type = "class")
```

4 Evaluation

For evaluation we computed the Classification accuracy and F1 value of the models above. We visualized the comparison of the models in a table and a bar plot.

```
CA <- function(observed, predicted)
{
  t <- table(observed, predicted)

  sum(diag(t)) / sum(t)
}
```

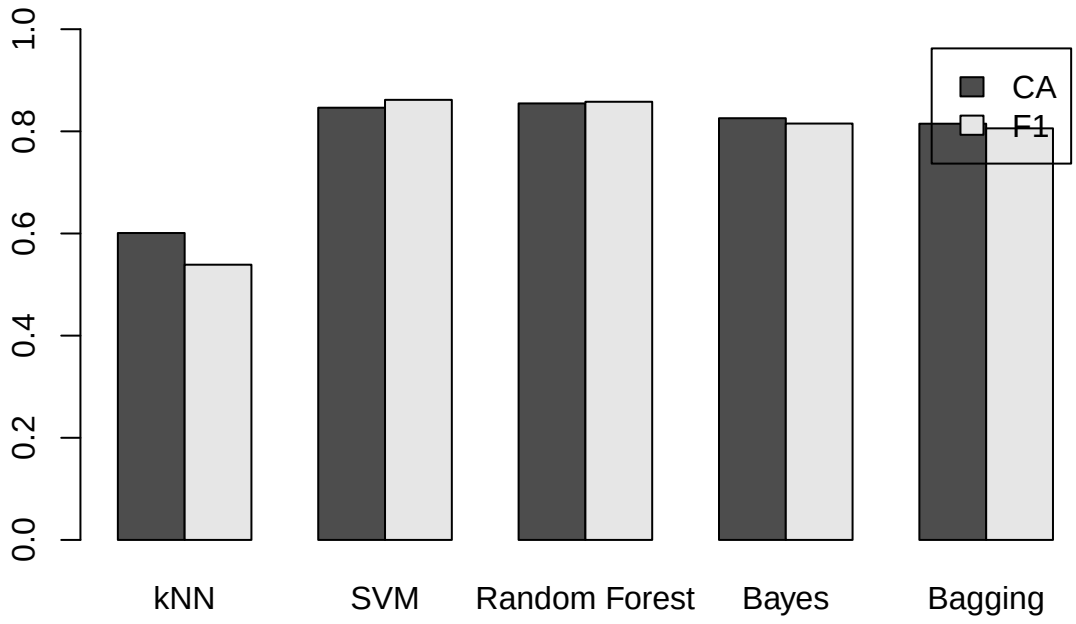
```
knnCA <- CA(observed, predictedKNN)
svmCA <- CA(observed, predictedSVM)
rfCA <- CA(observed, predictedRF)
nbCA <- CA(observed, predictedNB)
bagCA <- CA(test_label, predictedBAG)
```

```
actual <- test_label

confusionMatrixKNN <- confusionMatrix(factor(predictedKNN), factor(actual),
                                      mode = "everything", positive = "1")
confusionMatrixSVM <- confusionMatrix(factor(predictedSVM), factor(actual),
                                      mode = "everything", positive = "1")
confusionMatrixRF <- confusionMatrix(factor(predictedRF), factor(actual),
                                     mode = "everything", positive = "1")
confusionMatrixNB <- confusionMatrix(factor(predictedNB), factor(actual),
                                     mode = "everything", positive = "1")
confusionMatrixBAG <- confusionMatrix(factor(predictedBAG), factor(actual),
                                      mode = "everything", positive = "1")

knnF1 <- confusionMatrixKNN$byClass["F1"]
svmF1 <- confusionMatrixSVM$byClass["F1"]
rfF1<- confusionMatrixRF$byClass["F1"]
nbF1 <- confusionMatrixNB$byClass["F1"]
bagF1 <- confusionMatrixBAG$byClass["F1"]
```

```
library(knitr)

tab <- matrix(c(knnCA, knnF1, svmCA, svmF1, rfCA, rfF1, nbCA, nbF1, bagCA, bagF1),
              ncol = 2, byrow = TRUE)
colnames(tab) <- c("Accuracy", "F1")
rownames(tab) <- c("kNN", "SVM", "Random Forest", "Bayes", "Bagging")
tab <- as.table(tab)

kable(tab)
```

	Accuracy	F1
kNN	0.6009346	0.5388769
SVM	0.8462617	0.8617066
Random Forest	0.8546729	0.8579260
Bayes	0.8257009	0.8152551
Bagging	0.8149533	0.8058824

```
barplot(t(tab), ylim = c(0,1.0), legend = c("CA", "F1"), beside = TRUE)
```



From the accuracy and F1 scores we can observe that our kNN classifier achieved the lowest score, while the rest of them produced fairly equal results. The SVM classifier gave us a good accuracy and F1 score, but it took the longest to compute.

Changing different hyperparameters, such as setting the number of neighbours considered from the default `k = 1` to `k = 100`, gave us a slightly better accuracy and F1 score in kNN. Using a higher number of trees, `ntree = 700` instead of the default `ntree = 500`, in the random forest classifier also produced a better result. Similarly choosing a higher number of bags, in our case `50`, in the ensemble method bagging also resulted in a higher accuracy and F1 score. Meanwhile changing the kernel shape of the SVM classifier from linear to radial gave us a worse accuracy and F1 score.