Seminar 2: Mining sequential data

Intelligent Systems (IS)

December 31, 2021

Miha Godec

Vid Potočnik

Installing packages

```
install.packages(
  c(
    "tm",
    "SnowballC",
    "wordcloud",
    "proxy",
    "kernlab",
    "NLP",
    "openNLP",
    "ggplot2",
    "CORElearn",
    "ipred",
    "stringr"
  )
)
install.packages("openNLPmodels.en", repos = "http://datacube.wu.ac.at/", type =
                   "source
```

Reading the data

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




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

We save the binary labels showing if the document is fake news or not in a separate variable called `fake_news_train` and `fake_news_test`, for later use in **Feature construction**.

Pre-processing

The data, that we were provided with was noisy. It had strange symbols, URL links etc. Our task was to remove this noise, in order for the classification to be faster and more accurate.

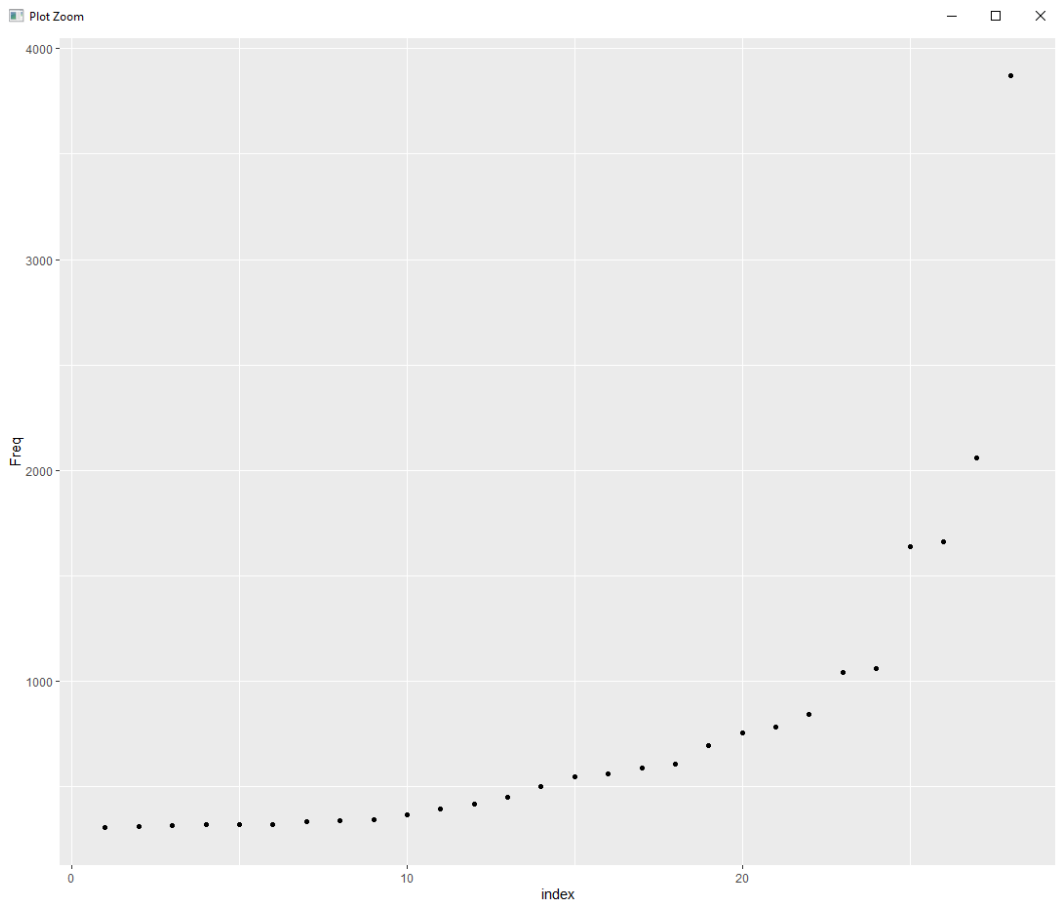
```
# Construct a corpus for a vector as input.
corpus_test <- Corpus(VectorSource(test_data))
corpus_train <- Corpus(VectorSource(train_data))

subSpace <-
  content_transformer(function(x, pattern)
    gsub(pattern, " ", x))

# Remove links
corpus_test = tm_map(corpus_test, subSpace, "https[^ ]*")
corpus_train = tm_map(corpus_train, subSpace, "https[^ ]*")

# Remove everything except letters
corpus_test <- tm_map(corpus_test, subSpace, "[^a-zA-Z]")
corpus_train <- tm_map(corpus_train, subSpace, "[^a-zA-Z]")

# Change letters to lower case
corpus_test <- tm_map(corpus_test, content_transformer(tolower))
corpus_train <- tm_map(corpus_train, content_transformer(tolower))

# Remove stopwords
corpus_test <-
  tm_map(corpus_test, removeWords, stopwords('english'))
corpus_train <-
  tm_map(corpus_train, removeWords, stopwords('english'))

# Read the custom stopwords list
conn = file("english.stop.txt", open = "r")
mystopwords = readLines(conn)
close(conn)

# Remove custom stopwords
corpus_test <- tm_map(corpus_test, removeWords, mystopwords)
corpus_train <- tm_map(corpus_train, removeWords, mystopwords)

# Stem words to retrieve their radicals
corpus_test <- tm_map(corpus_test, stemDocument)
```

```
corpus_train <- tm_map(corpus_train, stemDocument)

# Strip extra whitespace from text documents
corpus_test <- tm_map(corpus_test, stripWhitespace)
corpus_train <- tm_map(corpus_train, stripWhitespace)
```

We tested plenty of different configurations of data and figured out that this one was optimal for the classifiers we used. We removed all URL links, everything except letters (and lower-cased them), stop-words from the `tm` library and also removing the custom stop-words from the list provided at lectures was useful. Retrieving radicals from the data also improved the classification accuracy of our model. At last we removed the extra white-space.

Term frequencies

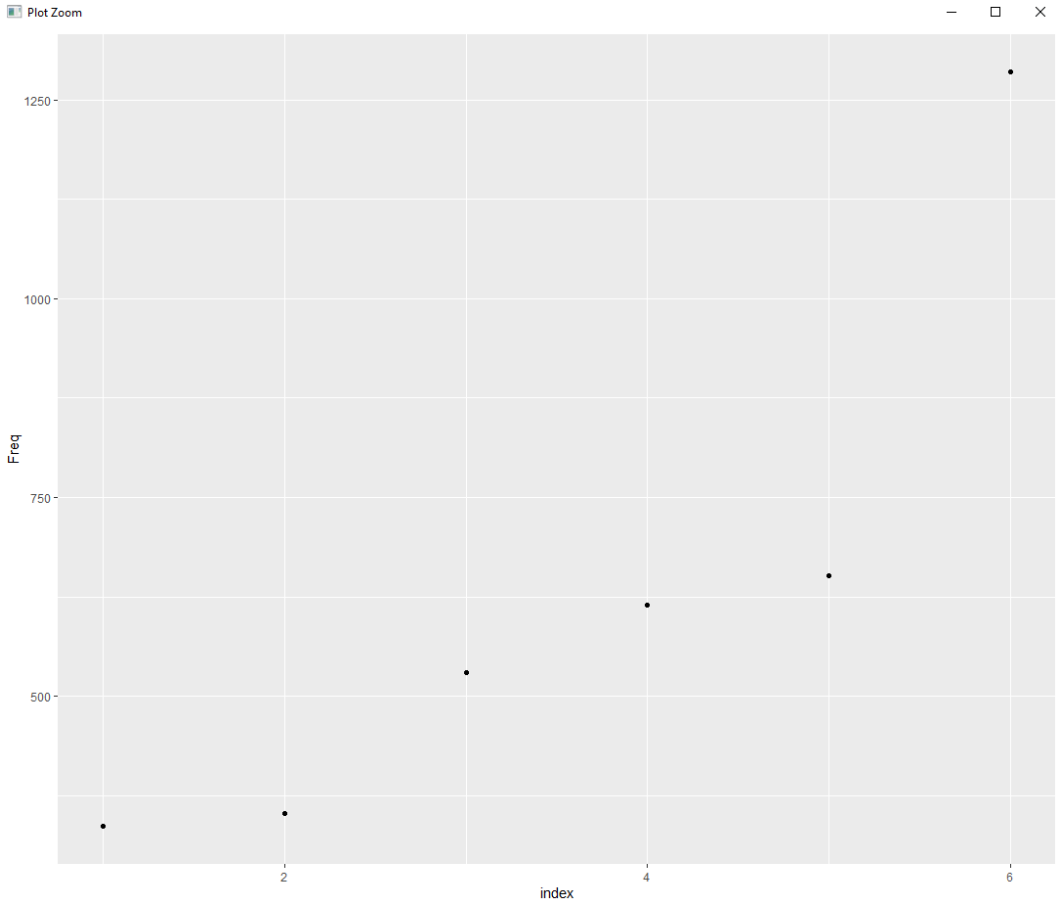
Graph 1: most common terms in the train data




```
    show             rate           infect             posit
     306              313              316              321
   claim           pandem            virus indiafightscorona
     323              324              336              338
 patient            updat           vaccin           health
     345              371              400              418
   today              amp          confirm            total
     450              500              550              562
  hospit              day           number            peopl
     592              610              697              755
   india            death           report            state
     811              842             1042             1062
    test      coronavirus             case            covid
    1637             1661             2059             4020
```

From this graph and list of terms we can see frequencies of the most common terms (frequency more than 300) in the train data.

Graph 2: most common terms in the test data



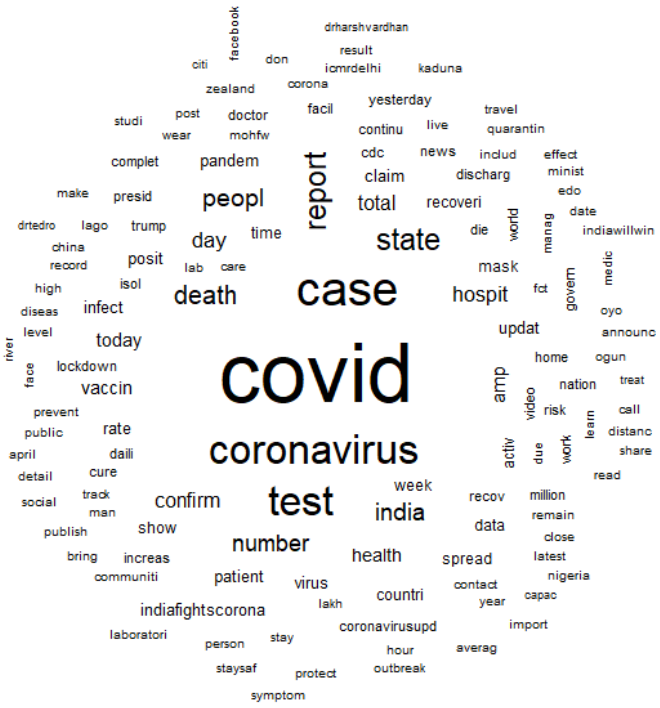

```
state      report   coronavirus        test        case       covid
  336         353           530         615         652        1341
```

From this graph and list of terms we can see frequencies of the most common terms (frequency more than 300) in the test data. We noticed that the most common words are the same in both the test and the train data-set (however not in the same order of frequencies)

This is code for **Graph 1** and **Graph 2**.

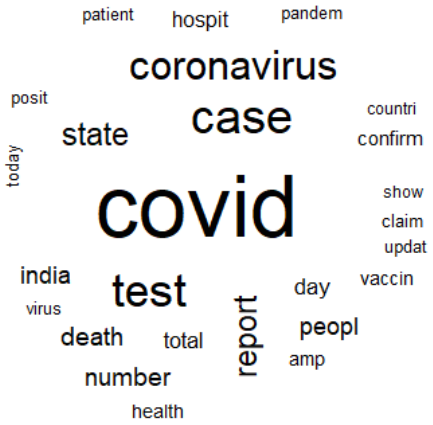
```
# rows = terms, columns = documents
tdm_train <- TermDocumentMatrix(corpus_train)
tdm_test <- TermDocumentMatrix(corpus_test)


# The plot shows the most frequent words in the train matrix
termFrequency_train <- rowSums(as.matrix(tdm_train))
termFrequency_train <-
  subset(termFrequency_train, termFrequency_train >= 300)
qplot(seq(length(termFrequency_train)),
      sort(termFrequency_train),
      xlab = "index",
      ylab = "Freq")

# The plot shows the most frequent words in the test matrix
termFrequency_test <- rowSums(as.matrix(tdm_test))
termFrequency_test <-
  subset(termFrequency_test, termFrequency_test >= 300)
qplot(seq(length(termFrequency_test)),
      sort(termFrequency_test),
      xlab = "index",
      ylab = "Freq")
```

Graph 3: wordcloud representation of train data



This wordcloud shows most common terms in the train data. Tags are single words, and the importance of each tag is shown with font size.

Graph 4: wordcloud representation of test data



This wordcloud shows most common terms in the test data. Tags are single words, and the importance of each tag is shown with font size.

This is code for **Graph 3** and **Graph 4**.

```
library(wordcloud)

mat_train <- as.matrix(tdm_train)
wordFreq_train <- sort(rowSums(mat_train), decreasing = TRUE)
grayLevels_train <-
  gray((wordFreq_train + 10) / (max(wordFreq_train) + 10))
wordcloud(
  words = names(wordFreq_train),
  freq = wordFreq_train,
  min.freq = 100,
  random.order = F,
  colors = grayLevels_train
)

mat_test <- as.matrix(tdm_test)
wordFreq_test <- sort(rowSums(mat_test), decreasing = TRUE)
grayLevels_test <-
  gray((wordFreq_test + 10) / (max(wordFreq_test) + 10))
wordcloud(
  words = names(wordFreq_test),
  freq = wordFreq_test,
  min.freq = 100,
  random.order = F,
```

```
  colors = grayLevels_test
)
```

Feature construction

```
# Constructs a document-term matrix (rows = documents, columns = terms)
dtm_train <-
  DocumentTermMatrix(corpus_train, control = list(weighting = weightTfIdf))
dtm_train <- removeSparseTerms(dtm_train, sparse = 0.99)
train_mat <- as.matrix(dtm_train)
train_df <- as.data.frame(train_mat)
train_df <- cbind(train_df, fake_news_train)
names(train_df)[ncol(train_df)] <- "fake_news_train"


dtm_test <-
DocumentTermMatrix(corpus_test,
                   control = list(dictionary = Terms(dtm_train), weighting = weightTfIdf))
test_mat <- as.matrix(dtm_test)
train_names <- names(train_df)
test_mat <- test_mat[, train_names]
test_df <- as.data.frame(test_mat)
test_df <- cbind(test_df, fake_news_test)
names(test_df)[ncol(test_df)] <- "fake_news_test"
```

We created the document-term matrix using `control = list(weighting = weightTfIdf` so that instead of term occurrences it consists of the frequencies. That turned out to be better than just using the number of term occurrences.

We had to use the function `removeSparseTerms` on the train matrix, as the size of it was too large to calculate any classification. In the configuration above, the function removes all terms that are not in more than 99% of documents.

The final space we used for learning was a dataframe, where rows represent documents and columns represent different terms. Also, we joined the `fake_news_train` label to the dataframe, as it was useful for the next step **Modeling**.

Modeling

We decided to use 3 machine learning classifiers (nNN, naive Bayes, support-vector machine) and 2 ensemble methods (random forest, bagging), plus the majority classifier for reference.
We trained our models on the `train_df` dataframe using the `fake_news_train` label which is a column in the dataframe. Then we produced the predictions based on the test dataframe `test_df` and built a simple table showing the results.

Random forest (ensemble method):

```
library(CORElearn)

cm.rf <-
  CoreModel(fake_news_train ~ ., data = train_df, model = "rf")
rf.predicted <- predict(cm.rf, test_df, type = "class")
rf.observed <- test_df$fake_news_test
rf <- table(rf.observed, rf.predicted)
```

Naive Bayes:

```
library(CORElearn)

cm.nb <-
  CoreModel(fake_news_train ~ ., data = train_df, model = "bayes")
nb.predicted <- predict(cm.nb, test_df, type = "class")
nb.observed <- test_df$fake_news_test
nb <- table(nb.observed, nb.predicted)
```

kNN:

```
library(class)

train_r <- which(names(train_df) == "fake_news_train")
test_r <- which(names(test_df) == "fake_news_test")

predicted <-
  knn(train_df[, -train_r], test_df[,-test_r], train_df$fake_news_train, k =
        5)
```

```
observed <- test_df$fake_news_test
knn <- table(observed, predicted)
```

Support-vector machine:

```
library(caret)
library(e1071)

set.seed(112233)
library(parallel)
# Calculate the number of cores
no_cores <- detectCores() - 1

library(doParallel)
# create the cluster for caret to use
cl <- makePSOCKcluster(no_cores)
registerDoParallel(cl)

# svm with a linear kernel
model.svm <- train(as.factor(fake_news_train) ~ .,
                   data = train_df,
                   method = "svmLinear")

svm.predicted <- predict(model.svm, test_df, type = "raw")
svm.observed <- test_df$fake_news_test
svm <- table(svm.observed, svm.predicted)
```

For support-vector machine we used multi-core computation as it takes a long time using only one core. It still took a much longer time to complete than any other classifier we used.

Bagging (ensemble method):

```
library(ipred)

bagging <- bagging(fake_news_train ~ ., train_df, nbagg = 20)
bag.predicted <- predict(bagging, test_df, type = "class")$class
bag.observed <- test_df$fake_news_test
bag <- table(bag.observed, bag.predicted)
```

Majority classifier:

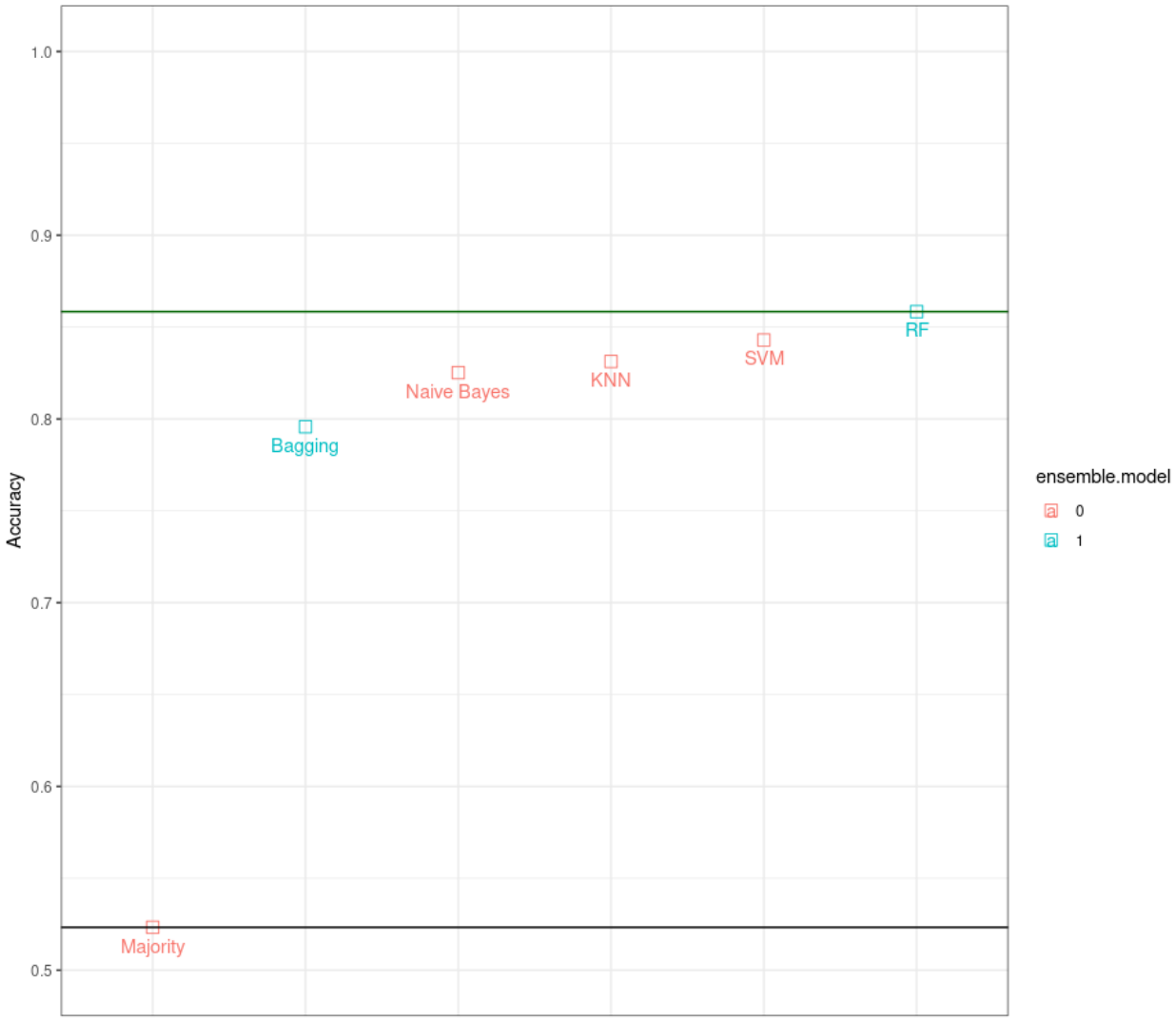
```
maj.train <- table(train_df$fake_news_train)
maj.train

maj.class <- 0
if (maj.train[1] < maj.train[2]) {
  maj.class <- 1
}

maj.observed <- test_df$fake_news_test
maj.predicted <- rep(maj.class, length(maj.observed))
maj.predicted

maj <- table(maj.observed, maj.predicted)
```

Evaluation:

Below, is a code block, which shows the implementation of the classification accuracy and F1 score.

We have calculated both values for every algorithm.

```
precision <- function(table) {
  table[1, 1] / sum(table[, 1])
}

recall <- function(table) {
  table[1, 1] / sum(table[1,])
}

ca <- function(table){
  sum(diag(table)) / sum(table)
}

f1 <- function(recall, precision) {
  2 * ((precision * recall) / (precision + recall))
}
```




```
  performances         f1   algo.names   ensemble
1    0.5233645  0.6871166     Majority          0
2    0.7957944  0.8018141      Bagging          1
3    0.8252336  0.8352423  Naive Bayes          0
4    0.8313084  0.8332564          KNN          0
5    0.8429907  0.8231579          SVM          0
6    0.8584112  0.8586094           RF          1
```

As we can see from the table, all of our implemented models beat the majority classifier by at least 20%. The best method is the **Random forest** method with approximately 86% classification accuracy.

If we compare out models, to some of the pre-computed baselines, we can see that many of them still beat our model, but we have managed to get a better accuracy than at least doc2vec + LR model.

We have tried different hyperparameter configurations, because some models took way too long to compute. So we had to do a lot of parameter tuning, in order to be time efficient.