Seminar 2: Mining sequential data

Aljaž Potočnik

20 12 2021

Reading data

```
train_data <- read.csv(file='train_data.tsv', sep = '\t', header = TRUE)
train_labels <- as.factor(train_data$label)
train_documents <- train_data$text_a
test_data <- read.csv(file='test_data.tsv', sep = '\t', header = TRUE)
test_labels <- as.factor(test_data$label)
test_documents <- test_data$text_a
train_baseline <- max(table(train_labels))/sum(table(train_labels))
test_baseline <- max(table(test_labels))/sum(table(test_labels))
```

Pre-processing

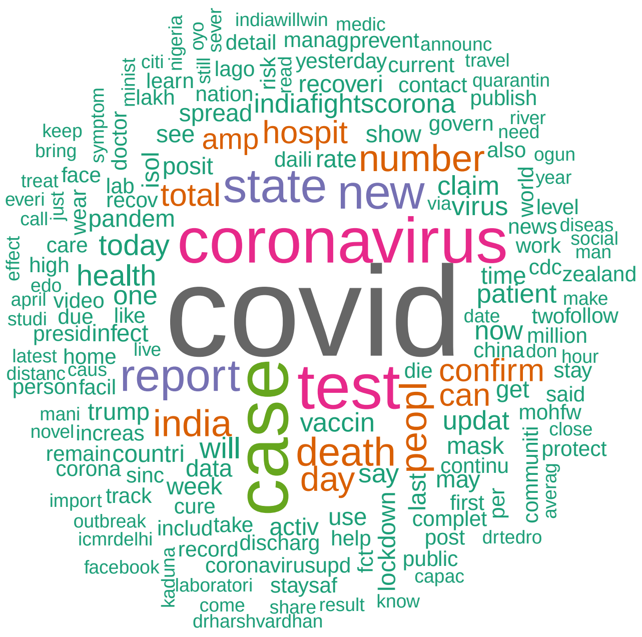
I created two corpuses(train and test) and firstly get rid of all terms containing url links and all special characters(non ASCII) then I transformed all terms to lowercase, removed numbers, and remove english stopwords, preformed stemming, and stripped them of white space so I got terms without noise. Firstly I didnt remove punctuation because it gave better results, but I think it is a little biased with @(twitter name) for example.

```
train_documents <- gsub("http*"," ", train_documents) ## url words
train_documents <- gsub("[^[:alnum:]]", " ", train_documents) ## non alpha numeric symbols
train_documents <- gsub("[^\u0001-\u007F]+|<U\\+\\w+>"," ", train_documents) ## non ASCII
corpus_train <- Corpus(VectorSource(train_documents))
corpus_train <- tm_map(corpus_train, content_transformer(tolower))
corpus_train <- tm_map(corpus_train, removePunctuation)
corpus_train <- tm_map(corpus_train, removeNumbers)
corpus_train <- tm_map(corpus_train, removeWords, stopwords('english'))
corpus_train <- tm_map(corpus_train, stemDocument)
corpus_train <- tm_map(corpus_train, stripWhitespace)

test_documents <- gsub("http*"," ", test_documents) ## url words
test_documents <- gsub("[^[:alnum:]]", " ", test_documents) ## non alpha numeric symbols
test_documents <- gsub("[^\u0001-\u007F]+|<U\\+\\w+>"," ", test_documents) ##non ASCII
corpus_test <- Corpus(VectorSource(test_documents))
corpus_test <- tm_map(corpus_test, content_transformer(tolower))
corpus_test  <- tm_map(corpus_test, removePunctuation)
corpus_test  <- tm_map(corpus_test, removeNumbers)
corpus_test <- tm_map(corpus_test, removeWords, stopwords('english'))
corpus_test <- tm_map(corpus_test, stemDocument)
corpus_test <- tm_map(corpus_test, stripWhitespace)
```

I preformed two visualizations of most frequent words in train corpus after pre-processing of data.

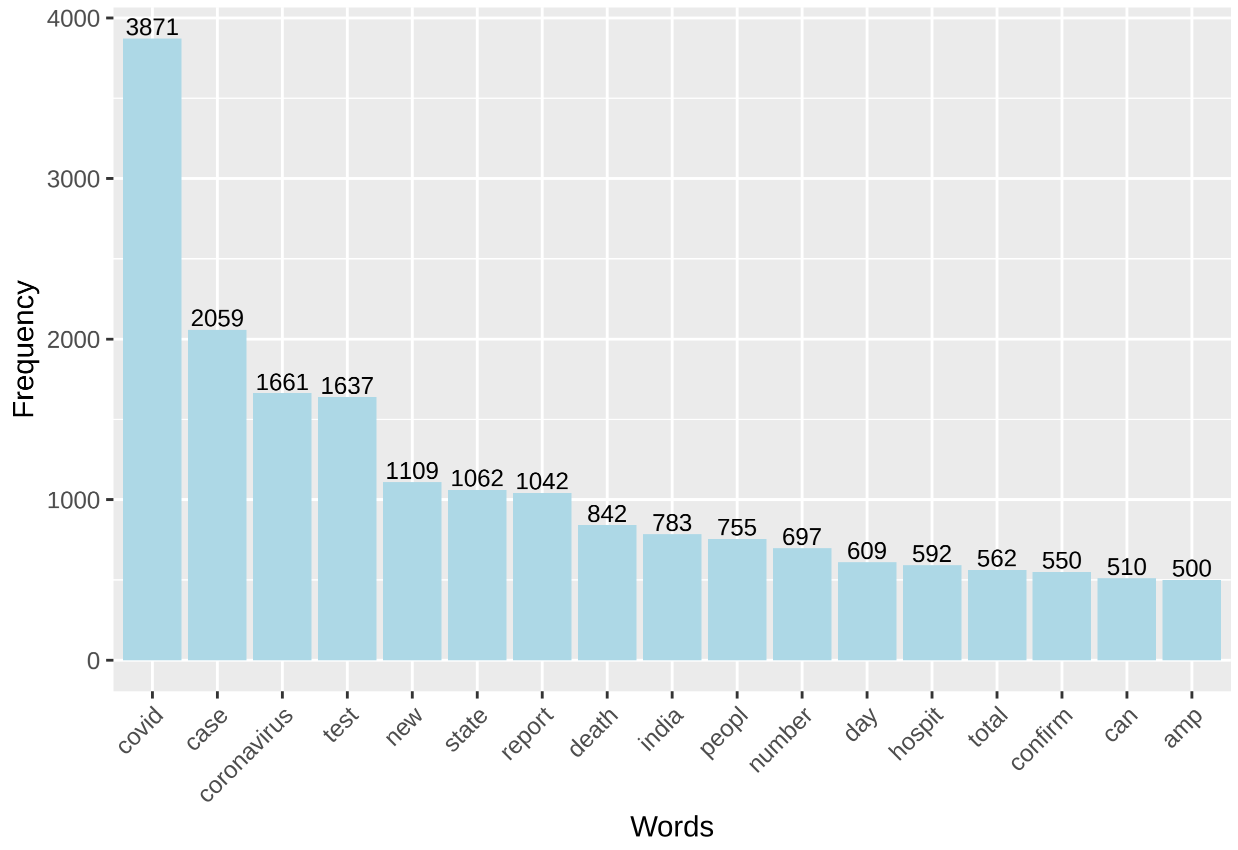
```r
tdm = TermDocumentMatrix(corpus_train)
termFrequency <- rowSums(as.matrix(tdm))
termFrequency <- subset(termFrequency, termFrequency >= 100)
wordFreq <- sort(rowSums(as.matrix(termFrequency)), decreasing=TRUE)
wordcloud(words=names(wordFreq), freq=wordFreq, min.freq = 100,
          random.order=F, colors=brewer.pal(8, "Dark2"))
```



The first visualization was made with wordcloud, and the following was made with ggplot for more numerical visualization.

```r
termFrequency2 <- subset(termFrequency, termFrequency >= 500)
wordFreq2 <- sort(rowSums(as.matrix(termFrequency2)), decreasing=TRUE)
w = as.data.frame(wordFreq2)
word_freq <- data.frame(rownames(w), w$wordFreq)

colnames(word_freq) <- c("words", "frequency")
ggplot (word_freq, aes(x = reorder(words, - frequency), y= frequency )) +
  geom_bar( stat = "Identity" , fill = "lightBlue" ) +
  geom_text( aes (label = frequency ) , vjust = - 0.20, size = 3 ) +
  xlab( "Words" ) +
  ylab( "Frequency" ) +
  theme ( axis.text.x = element_text ( angle = 45 , hjust = 1 ) )
```



After visualizations we see that corpus is about covid and covid related terms.

Feature construction

I transformed data into feature matrices using DocumentTerm matrix, rows are documents and columns are terms, and cells are weighted with term frequency–inverse document frequency (TfIdf), it gives us good information about how important is a word to a document and not only term frequency in which some words could carry more weight than they are important. After having this matrix I removed less frequent terms to reduce the dimensions of the document-term matrix.

I did similar for test data,with a slight difference now when we see new features(new terms), I added terms from train set which do not appear in test set, and assign them zero values, because some following models can not handle empty data. And I chose zero because it means that term appears in all documents, and it is ont important.

```r
train_tfidf <- DocumentTermMatrix(corpus_train, control = list(weighting=weightTfIdf))
train_tfidf <- removeSparseTerms(train_tfidf, sparse=0.99)
train_mat <- as.matrix(train_tfidf)
colnames(train_mat) <- make.names(colnames(train_mat), unique=TRUE)
colnames(train_mat)[ncol(train_mat)] <- 'y'
train <- as.data.frame(train_mat)
train$y <- train_labels

test_tfidf <- DocumentTermMatrix(corpus_test, control = list(weighting=weightTfIdf))
test_tfidf <- removeSparseTerms(test_tfidf, sparse=0.99)
test_mat <- as.matrix(test_tfidf)
colnames(test_mat) <- make.names(colnames(test_mat), unique=TRUE)
colnames(test_mat)[ncol(test_mat)] <- 'y'
```

```r
test <- as.data.frame(test_mat)
test$y <- test_labels

features_to_add <-setdiff(colnames(train), colnames(test))
test[features_to_add] = NA
test[is.na(test)] <- 0
```

Modeling

I used 4 models: decision tree, naive Bayes, KNN and ensemble method random forest with 200 trees. In Knn I used 3 for kinNN parameter which means it decides upon the only 3 closest neighbors. It worked best this way. Random forest model is not from the same library as others models.

```r
start_time <- Sys.time()
modelDT <- CoreModel(y ~ ., train, model="tree")
end_time <- Sys.time()
train_DT <-  difftime(end_time, start_time, units='secs')
start_time <- Sys.time()
modelNB <- CoreModel(y ~ ., train, model="bayes")
end_time <- Sys.time()
train_NB <-  difftime(end_time, start_time, units='secs')
start_time <- Sys.time()
modelKNN <- CoreModel(y ~ ., train, model="knn", kInNN = 3)
end_time <- Sys.time()
train_KNN <-  difftime(end_time, start_time, units='secs')
start_time <- Sys.time()
rf <- randomForest(y ~ ., train, ntree=200)
end_time <- Sys.time()
train_rf <-  difftime(end_time, start_time, units='secs')
```

Evaluation

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

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

F1 <- function(observed, predicted, CM)
{
  precision <- CM[2,2]/(CM[2,2] + CM[2,1])
  recall <- CM[2,2]/(CM[2,2] + CM[1,2])
  2*(precision * recall)/(precision + recall)
}

start_time <- Sys.time()
predDT <- predict(modelDT, test, type = "class")
end_time <- Sys.time()
test_DT <-  difftime(end_time, start_time, units='secs')
result.caDT <- CA(test$y, predDT)
confussionDT <- confusionMatrix(predDT, test$y, mode = "everything", positive="1")$table
result.f1DT <- F1(test$y, predDT, confussionDT)
```

```
start_time <- Sys.time()
predNB <- predict(modelNB, test, type="class")
end_time <- Sys.time()
test_NB <-  difftime(end_time, start_time, units='secs')
result.caNB <- CA(test$y, predNB)
confussionNB <- confusionMatrix(predNB, test$y, mode = "everything", positive="1")$table
result.f1NB <- F1(test$y, predNB, confussionNB)

start_time <- Sys.time()
predKNN <- predict(modelKNN, test, type="class")
end_time <- Sys.time()
test_KNN <-  difftime(end_time, start_time, units='secs')
result.caKNN <- CA(test$y, predKNN)
confussionKNN <- confusionMatrix(predKNN, test$y, mode = "everything", positive="1")$table
result.f1KNN <- F1(test$y, predKNN, confussionKNN)

start_time <- Sys.time()
rf.predicted <- predict(rf, test, type = "class")
end_time <- Sys.time()
test_rf <-  difftime(end_time, start_time, units='secs')
result.caRF <- CA(test$y,rf.predicted)
confussionRF <- confusionMatrix(rf.predicted, test$y, mode = "everything", positive="1")$table
result.f1RF <- F1(test$y, rf.predicted, confussionRF)
```

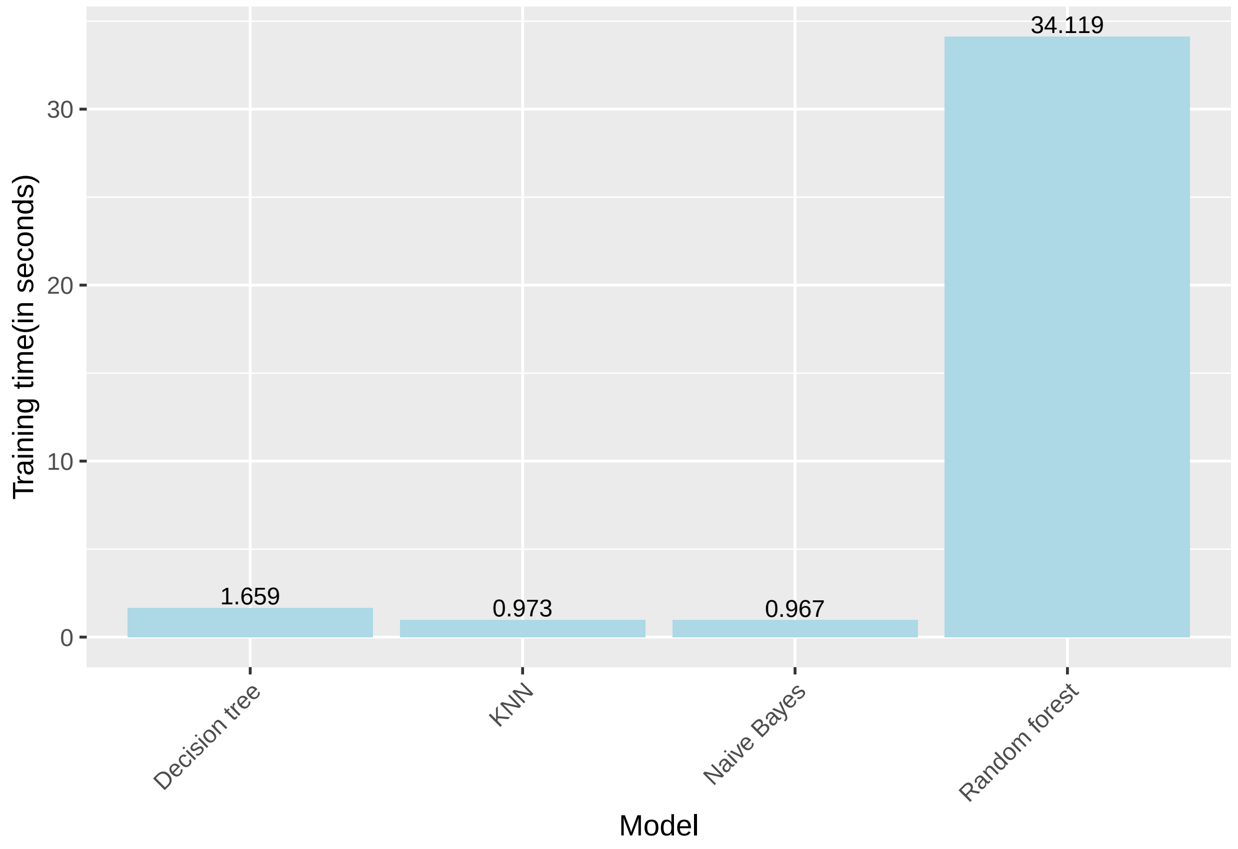
Decision tree and naive Bayes worked pretty similar, more features gave better results. I tried and change the dimension of document-term matrix with removing more of less frequent terms, but it did not gave me good results except for KNN that gave better results. As expected the best model was random forest which similarly as Bayes and decision tree worked better with more features. Important parameter was the number of trees generated in forest and till around 200 the accuracy was getting better but what was more than that did not really improve the accuracy. We can see some visualizations of performance and accuracy down.

```
train_times <- round(c(train_DT[[1]], train_NB[[1]], train_KNN[[1]], train_rf[[1]]),3)
test_times <- round(c(test_DT[[1]], test_NB[[1]], test_KNN[[1]], test_rf[[1]]),3)
f1_acc <- round(c(result.f1DT, result.f1NB, result.f1KNN, result.f1RF, 0.956, 0.939, 0.929, 0.911, 0.81
ca_acc <- round(c(result.caDT, result.caNB, result.caKNN, result.caRF, 0.957, 0.939, 0.929, 0.911, 0.81
models_str <- c("Decision tree", "Naive Bayes", "KNN", "Random forest")
train_time_frame <- data.frame(models_str, train_times)
test_time_frame <- data.frame(models_str, test_times)
models_all_acc <- c("Decision tree", "Naive Bayes", "KNN", "Random forest", "autoML", "MPNet + LR", "ch
                    ,"word + LR", "doc2vec + LR", "majority")

f1_frame <- data.frame(models_all_acc, f1_acc)
ca_frame <- data.frame(models_all_acc, ca_acc)
ca_frame <- ca_frame[order(ca_frame$ca_acc), ]
f1_frame <- f1_frame[order(f1_frame$f1_acc), ]
```

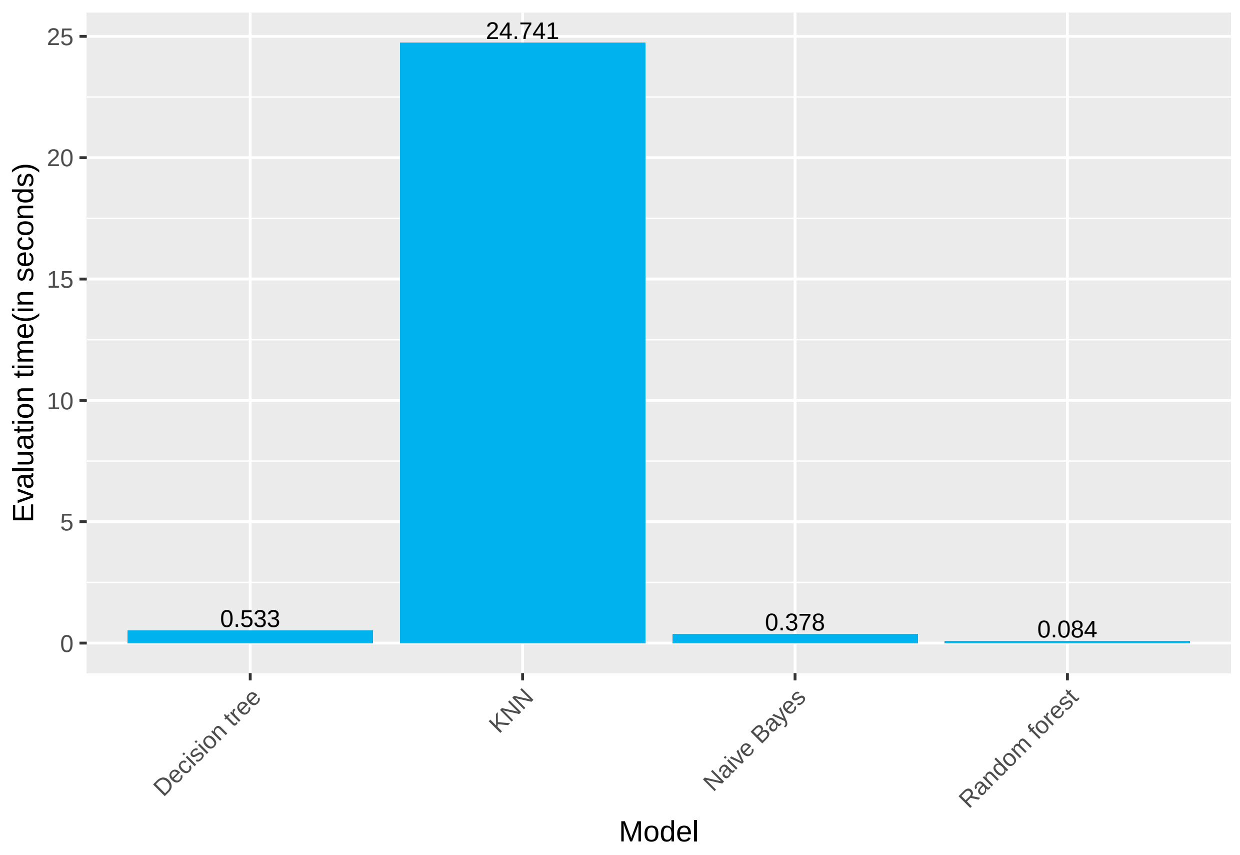
The plot of training times of my models in seconds. Random forest as expected takes the longest to train, because it has to generate 200 decision trees. The other three models are trained much faster.

```
ggplot (train_time_frame,  aes(x=models_str, y= train_times )) +
  geom_bar( stat = "Identity" , fill = "lightBlue" ) +
  geom_text( aes (label = train_times ) , vjust = - 0.20, size = 3 ) +
  xlab( "Model" ) +
  ylab( "Training time(in seconds)" ) +
  theme ( axis.text.x = element_text ( angle = 45 , hjust = 1 ) )
```



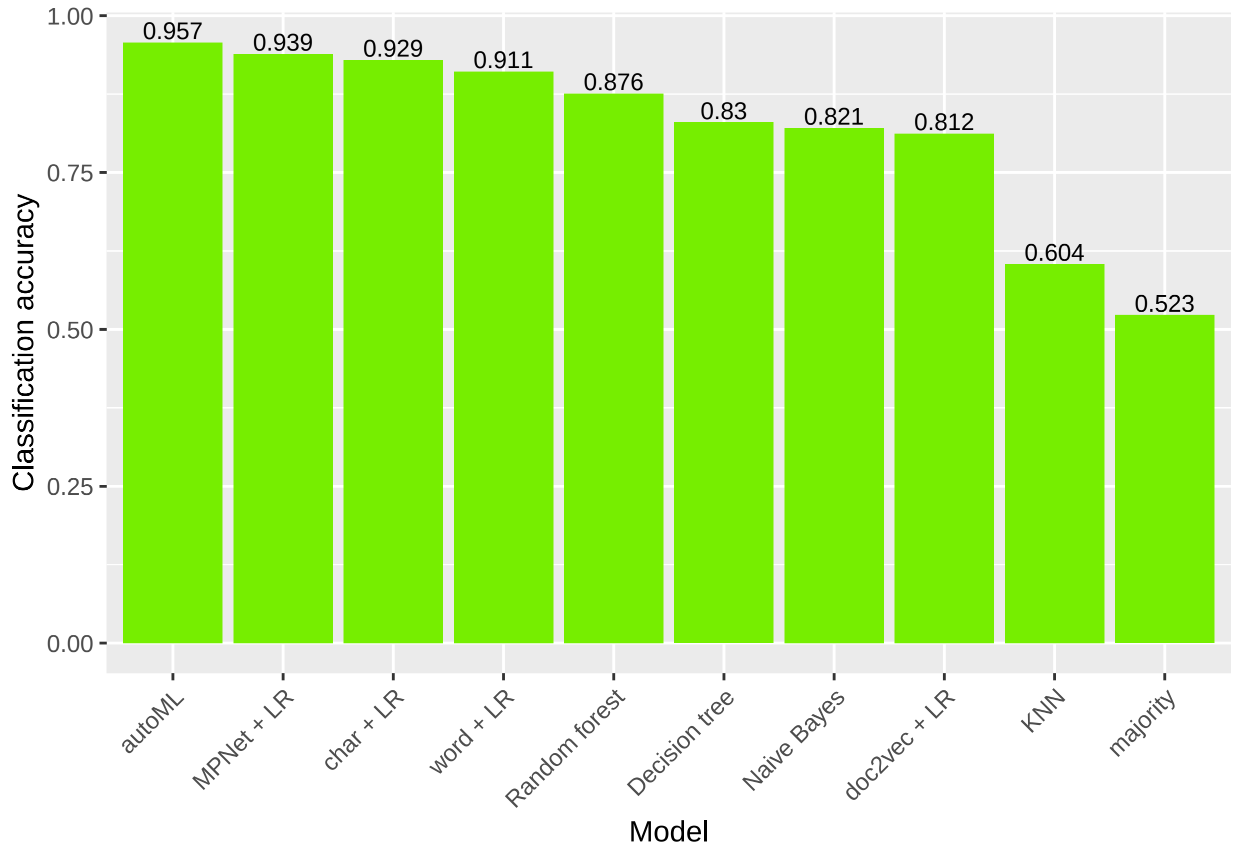
The KNN takes the longest to evaluate, which is expected because it is calculating distance to every node to determine label of node, others models are pretty quick. It was pretty interesting to me that random forest has faster evaluation time than decision tree, but I tried randomForest from COREmodel library it worked slower than decision tree which is expected but at the end I decided for this randomForest model because it is faster.

```
ggplot(test_time_frame,  aes(x= models_str, y= test_times )) +
  geom_bar( stat = "Identity" , fill = "deepskyblue2" ) +
  geom_text( aes (label = test_times ) , vjust = - 0.20, size = 3 ) +
  xlab( "Model" ) +
  ylab( "Evaluation time(in seconds)" ) +
  theme ( axis.text.x = element_text ( angle = 45 , hjust = 1 ) )
```



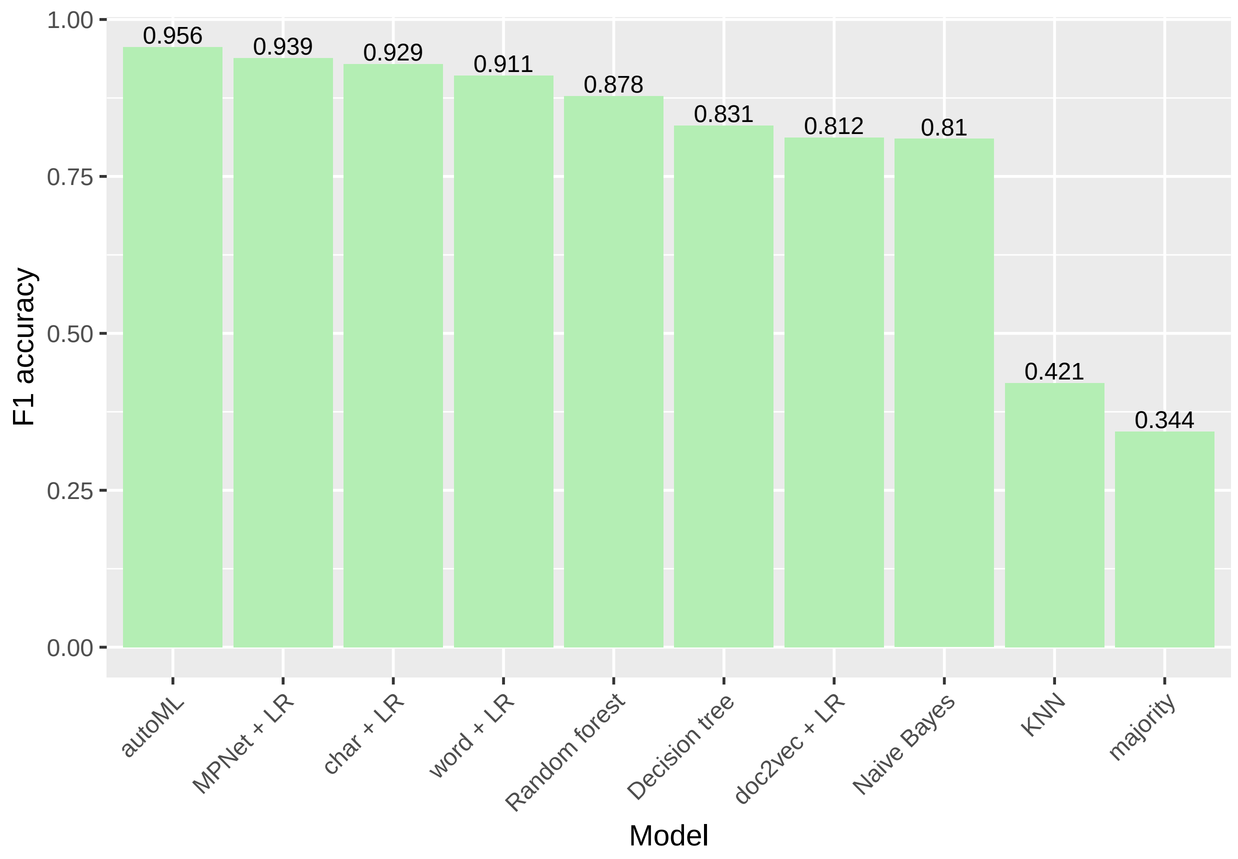
My best model which would be the random forest regarding classification accuracy achieves around 87% of accuracy, in bottom plot we can see where it puts us on benchmark.

```
ggplot (ca_frame, aes(x = reorder(models_all_acc, - ca_acc), y= ca_acc )) +
  geom_bar( stat = "Identity" , fill = "chartreuse2" ) +
  geom_text( aes (label = ca_acc ) , vjust = - 0.20, size = 3 ) +
  xlab( "Model" ) +
  ylab( "Classification accuracy" ) +
  theme ( axis.text.x = element_text ( angle = 45 , hjust = 1 ) )
```



My best model which would be the random forest regarding f1 accuracy achieves around 88% of accuracy, in bottom plot we can see where it puts us on benchmark. In the f1 accuracy is the most seen difference in accuracy of KNN depending on number of neighboring nodes considered in prediction, the best is with 3 neighbors.

```
ggplot (f1_frame, aes(x = reorder(models_all_acc, - f1_acc), y= f1_acc )) +
  geom_bar( stat = "Identity" , fill = "darkseagreen2" ) +
  geom_text( aes (label = f1_acc ) , vjust = - 0.20, size = 3 ) +
  xlab( "Model" ) +
  ylab( "F1 accuracy" ) +
  theme ( axis.text.x = element_text ( angle = 45 , hjust = 1 ) )
```


F1 accuracy
1.00
0.75
0.50
0.25
0.00
0.956
0.939
0.929
0.911
0.878
0.831
0.812
0.81
0.421
0.344
autoML
MPNet + LR
char + LR
word + LR
Random forest
Decision tree
doc2vec + LR
Naive Bayes
KNN
majority
Model