Tutorial 7: Classification

Andreas Niekler, Gregor Wiedemann

2020-07-03

Contents

In this tutorial we show the use of supervised machine learning for text classification. The basic idea is to compute a model based on training data. Training data usually are hand-coded documents or text snippets associated with a specific category (class). From these texts, features (e.g. words) are extracted and associated with categories in the model. The model then can be utilized to categorize new texts.

We cover basic principles of the process such as cross-validation and feature engineering in the following steps:

1. Read text data
2. Read training data
3. Build feature matrix
4. Classify (LiblineaR)
5. K-fold cross validation
6. Optimize C
7. Optimize features: stopwords, bi-grams, stemming
8. Final classification

As data, again we use the "State of the Union"-addresses. But this time, we operate on paragraphs instead of documents. The file `data/sotu_paragraphs.csv` provides the speeches in the appropriate format. For each paragraph, we want to know whether it covers content related to **domestic or foreign affairs**.

Read paragraphs

As already known, we read the text source (21334 paragraphs from 231 speeches). For the moment, we apply very basic preprocessing.

```
options(stringsAsFactors = FALSE)
library(quanteda)

textdata <- read.csv("data/sotu_paragraphs.csv", sep = ";", encoding = "UTF-8")

corpus <- corpus(textdata$text, docnames = textdata$doc_id)

# Build a dictionary of lemmas
lemma_data <- read.csv("resources/baseform_en.tsv", encoding = "UTF-8")

# Create a DTM
corpus_token <- corpus %>%
  tokens(remove_punct = TRUE, remove_numbers = TRUE, remove_symbols = TRUE) %>%
  tokens_tolower()
```

Load training data

We provide 300 manually annotated example paragraphs as training data. In a CSV-file, the paragraph id and the corresponding category is stored.

```
# Read previously annotated training data
trainingData <- read.csv2("data/paragraph_training_data_format.csv", stringsAsFactors = T)
# Training data format
colnames(trainingData)
```

```
## [1] "ID"    "LABEL"
```

```
# Example paragraph Foreign Affairs
set.seed(13)
domestic_example <- sample(trainingData$ID[trainingData$LABEL == "DOMESTIC"], 1)
as.character(texts(corpus)[domestic_example])
```

```
## [1] "By the use of the State banks, which do not derive their charters from\nthe General Government a
```

```
foreign_example <- sample(trainingData$ID[trainingData$LABEL == "FOREIGN"], 1)
as.character(texts(corpus)[foreign_example])
```

```
## [1] "The International Atomic Energy Agency confirmed in the 1990s that Saddam\nHussein had an advanc
```

How is the ratio between domestic and foreign content in the training data?

```
classCounts <- table(trainingData[, "LABEL"])
print(classCounts)
```

```
##
## DOMESTIC  FOREIGN
##      209       91
```

```
numberOfDocuments <- nrow(trainingData)
```

For our first classification attempt, we create a Document-Term Matrix from the preprocessed corpus and use the extracted single words (unigrams) as features for the classification. Since the resulting DTM might contain too many words, we restrict the vocabulary to a minimum frequency.

```
# Base line: create feature set out of unigrams
# Probably the DTM is too big for the classifier. Let us reduce it
minimumFrequency <- 5

DTM <- corpus_token %>%
  dfm() %>%
  dfm_trim(min_docfreq = minimumFrequency, max_docfreq = Inf)

# How many features do we have?
dim(DTM)
```

```
## [1] 21334 10950
```

Classification

Now we build a linear classification model with the LiblineaR package. The package LiblineaR wraps around the open source library LIBLINEAR which provides a very fast implementations for two text classification algorithms: 1. Logistic Regression, and 2. Support Vector Machines (SVM) with a linear kernel. For both algorithms, different regularization methods exist (e.g. L1, and L2-regularization). The combination of algorithms and regularization can be controlled by the `type` parameter of the `LiblineaR` function. We stick to the default type, L2-regularized logistic regression (LR), since it usually achieves good performance in text classification.

First, we load the packages. Since Liblinear requires a special Sparse Matrix format, we also load the "SparseM" package and a conversion function which allows to convert `quanteda's dfm`-matrices into `SparseM`-matrices.

Then, we split the annotated data into a training set (80%) and a test set (20%) using a boolean selector. The expression assigned to `selector_idx` creates a boolean vector of length 300 containing a FALSE value in every fifths position. This selector is used to select to training set. Its inverted vector (!) is used to select the test set.

```
require(LiblineaR)
require(SparseM)
source("utils.R")

annotatedDTM <- DTM[trainingData[, "ID"], ]
annotatedDTM <- convertMatrixToSparseM(annotatedDTM)
annotatedLabels <- trainingData[, "LABEL"]

# split into training and test set
selector_idx <- rep(c(rep(TRUE, 4), FALSE), length.out = numberOfDocuments)
```

```r
trainingDTM <- annotatedDTM[selector_idx, ]
trainingLabels <- annotatedLabels[selector_idx]
testDTM <- annotatedDTM[!selector_idx, ]
testLabels <- annotatedLabels[!selector_idx]

# create LR classification model
model <- LiblineaR(trainingDTM, trainingLabels)
summary(model)
```

```
##            Length Class  Mode
## TypeDetail     1  -none- character
## Type           1  -none- numeric
## W          10951  -none- numeric
## Bias           1  -none- numeric
## ClassNames     2  factor numeric
## NbClass        1  -none- numeric
```

The model created by the LiblineaR function can now be utilized to predict the labels of the test set. Then we compare the result of the automatic classification to our known labels to determine the accuracy of the process.

```r
classification <- predict(model, testDTM)
predictedLabels <- classification$predictions
contingencyTable <- table(predictedLabels, testLabels)
print(contingencyTable)
```

```
##                testLabels
## predictedLabels DOMESTIC FOREIGN
##        DOMESTIC       24       5
##        FOREIGN        21      10
```

```r
accuracy <- sum(diag(contingencyTable)) / length(testLabels)
print(accuracy) # share of correctly classified paragraphs
```

```
## [1] 0.567
```

The accuracy of 0.567 appears moderate for a first try. But how does it actually relate to a base line? Think of the imbalanced class proportions in our training set. Let us create a pseudo classification as base line, in which we do not classify at all, but simply assume the label "DOMESTIC" or "FOREIGN" for each paragraph.

We further employ a function called `F.measure` which gives more differentiated measures than simple accuracy (`A`) to determine the classification quality. The F.measure (`F`) is the harmonic mean of Precision (`P`) and Recall (`R`) (see https://en.wikipedia.org/wiki/Precision_and_recall#Definition_.28classification_context.29 for Details).

```r
# Create pseudo classification
pseudoLabelsDOM <- factor(rep("DOMESTIC", length(testLabels)), levels(testLabels))
pseudoLabelsFOR <- factor(rep("FOREIGN", length(testLabels)), levels(testLabels))

# Evaluation of former LR classification with F-measures
F.measure(predictedLabels, testLabels, positiveClassName = "DOMESTIC")
```

```
##      P      R      S      F      A   Pos.
##  0.828  0.533  0.667  0.649  0.567 45.000
```

```r
F.measure(predictedLabels, testLabels, positiveClassName = "FOREIGN")
```

```
##      P      R      S      F      A   Pos.
##  0.323  0.667  0.533  0.435  0.567 15.000
```

```r
# Evaluation of pseudo classification with F-measures
F.measure(pseudoLabelsDOM, testLabels, positiveClassName = "DOMESTIC")
```

```
##      P      R      S      F      A   Pos.
##  0.750  1.000  0.000  0.857  0.750 45.000
```

```r
F.measure(pseudoLabelsFOR, testLabels, positiveClassName = "FOREIGN")
```

```
##     P     R     S     F     A  Pos.
##  0.25  1.00  0.00  0.40  0.25 15.00
```

This little experiment shows that depending on the definition of our positive class, the accuracy is either 25% or 75% if not classifying at all. In both cases the *specificity* (`S`), the true negative rate, is zero. From this, we can learn two things:

1. If classes in training/test sets are imbalanced, accuracy might be a misleading measurement. Other measure should be considered additionally.
2. To utilize accuracy and F-measure in a meaningful way, the less frequent class should be defined as POSITIVE class (FOREIGN in our case).

K-fold cross validation

To evaluate a classifier, the training data can be divided into training and test data. The model learns on the former and is evaluated with the latter. In this procedure, we unfortunately lose the test data to learn from. If there is little training data available, the k-fold cross-validation is a more suitable procedure.

For this, training data is split into e.g. K = 10 parts. Then k-1 parts are used for training and 1 part is used for testing. This process is repeated k times, with another split of the overall data set for testing in each iteration.

The final result is determined from the average of the quality of the k runs. This allows a good approximation to the classification quality, including all training data.

The `get_k_fold_logical_indexes` function introduced below returns a logical vector for the fold `j` for cross-validation. It splits a training data record of the size `n` into `k` folds. The resulting vector and its negated vector can be used for easy training data / test data selection.

```r
get_k_fold_logical_indexes <- function(j, k, n) {
  if (j > k) stop("Cannot select fold larger than nFolds")
  fold_lidx <- rep(FALSE, k)
  fold_lidx[j] <- TRUE
  fold_lidx <- rep(fold_lidx, length.out = n)
  return(fold_lidx)
}

# Example usage
get_k_fold_logical_indexes(1, k = 10, n = 12)
```

```
##  [1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
```

```r
get_k_fold_logical_indexes(2, k = 10, n = 12)
```

```
##  [1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

```r
get_k_fold_logical_indexes(3, k = 10, n = 12)
```

```
##  [1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Now we run 1) splitting of the annotated data, 2) model computation and testing in one for-loop.

```r
k <- 10
evalMeasures <- NULL
for (j in 1:k) {
  # create j-th boolean selection vector
  currentFold <- get_k_fold_logical_indexes(j, k, nrow(trainingDTM))

  # select training data split
  foldDTM <- annotatedDTM[!currentFold, ]
  foldLabels <- annotatedLabels[!currentFold]

  # create model
  model <- LiblineaR(foldDTM, foldLabels)

  # select test data split
  testSet <- annotatedDTM[currentFold, ]
  testLabels <- annotatedLabels[currentFold]

  # predict test labels
  predictedLabels <- predict(model, testSet)$predictions

  # evaluate predicted against test labels
  kthEvaluation <- F.measure(predictedLabels, testLabels, positiveClassName = "FOREIGN")

  # combine evaluation measures for k runs
  evalMeasures <- rbind(evalMeasures, kthEvaluation)
}
# Final evaluation values of k runs:
print(evalMeasures)
```

```
##                   P     R     S     F     A Pos.
## kthEvaluation 0.333 0.400 0.600 0.364 0.533   10
## kthEvaluation 0.737 0.875 0.643 0.800 0.767   16
## kthEvaluation 0.333 0.429 0.739 0.375 0.667    7
## kthEvaluation 0.467 0.778 0.619 0.583 0.667    9
## kthEvaluation 0.333 0.571 0.652 0.421 0.633    7
## kthEvaluation 0.312 0.714 0.522 0.435 0.567    7
## kthEvaluation 0.471 0.727 0.526 0.571 0.600   11
## kthEvaluation 0.500 0.833 0.792 0.625 0.800    6
## kthEvaluation 0.526 1.000 0.550 0.690 0.700   10
## kthEvaluation 0.385 0.625 0.636 0.476 0.633    8
```

```r
# Average over all folds
print(colMeans(evalMeasures))
```

```
##     P     R     S     F     A  Pos.
## 0.440 0.695 0.628 0.534 0.657 9.100
```

Accuracy is around 0.66%, F-measure is around 0.53%. Let's try some approaches to optimize the automatic classification.

Optimization

These first tries have clarified how to utilize and evaluate machine learning functions for text in R. Now we concentrate on optimization strategies to get better results from the automatic classification process.

C-Parameter

For a linear classification model, the cost parameter (C-parameter) is the most important parameter to tweak (for other SVM kernels such as the radial or polynomial kernel there are other important parameters which influence the shape of the kernel function). The **C-parameter** determines the cost of classifications on the training data during training.

High values of C lead to a high costs of misclassification. The decision boundary which the classifier learns, will try to avoid any misclassification. But, values too high can lead to an overfitting of the model. This means, it adapts well to the training data, but classification will more likely fail on new test data.

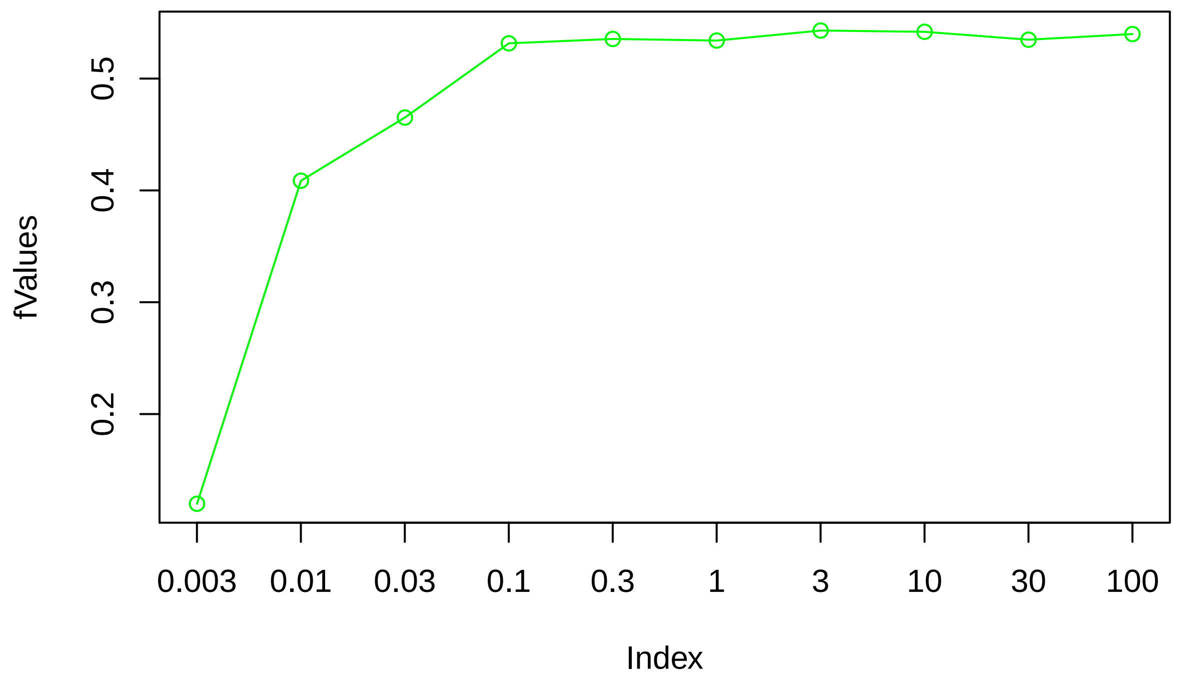
Low values of C lead to less strict decision boundaries, which accepts some misclassifications. Such a model might generalize better on unseen data. But in the end, there is now exact method to determine a good C-value beforehand. It rather is an empirical question. To choose an optimal C-value, we simply try from a range of values, run k-fold-cross-validation for each single value and decide for the C which resulted in the best classification accuracy / F-measure. This is realized in the following for-loop, which utilizes the function `k_fold_cross_validation`. The function (have a look into `F.measure.R`) simply wraps the code for cross-validation, we used above.

```r
cParameterValues <- c(0.003, 0.01, 0.03, 0.1, 0.3, 1, 3 , 10, 30, 100)
fValues <- NULL

for (cParameter in cParameterValues) {
  print(paste0("C = ", cParameter))
  evalMeasures <- k_fold_cross_validation(annotatedDTM, annotatedLabels, cost = cParameter)
  fValues <- c(fValues, evalMeasures["F"])
}
```

```
## [1] "C = 0.003"
## [1] "C = 0.01"
## [1] "C = 0.03"
## [1] "C = 0.1"
## [1] "C = 0.3"
## [1] "C = 1"
## [1] "C = 3"
## [1] "C = 10"
## [1] "C = 30"
## [1] "C = 100"
```

```
plot(fValues, type="o", col="green", xaxt="n")
axis(1,at=1:length(cParameterValues), labels = cParameterValues)
```



```
bestC <- cParameterValues[which.max(fValues)]
print(paste0("Best C value: ", bestC, ", F1 = ", max(fValues)))
```

```
## [1] "Best C value: 3, F1 = 0.542941816588523"
```

From the empirical test, we can obtain C = 3 as optimal choice for the cost parameter. On the current training data set with the current features it achieves 0.543 F-score.

Optimized Preprocessing

Not only the classification model has parameters which can be optimized to improve the results. More important are the features used for classification. In our preprocessing chain above, we extracted single types and transformed them into lower case. We now add different preprocessing steps and check on the results. To get an optimal cost parameter for each new feature set, we wrapped the code for C-parameter optimization into the `optimize_C` function.

Stop word removal

Stop words often do not contribute to the meaning of a text. For the decision between DOMESTIC and FOREIGN affairs, we do not expect any useful information from them. So let's remove them and if it improves the classifier.

```
stopwords_extended <- readLines("resources/stopwords_en.txt", encoding = "UTF-8")

# preprocessing
corpus_token_sw <- corpus %>%
  tokens(remove_punct = TRUE, remove_numbers = TRUE, remove_symbols = TRUE) %>%
  tokens_tolower() %>%
  tokens_remove(pattern = stopwords_extended)

print(paste0("1: ", substr(paste(corpus_token_sw[4963],collapse = " "), 0, 400), '...'))
```

```
## [1] "1: legislation extend colony newfoundland articles treaty washington 8th day protocol effect si
```

```
minimumFrequency <- 5

# Compute DTM
DTM <- corpus_token_sw %>%
  dfm() %>%
  dfm_trim(min_docfreq = minimumFrequency,max_docfreq = Inf)

# How many features do we have?
dim(DTM)
```

```
## [1] 21334 10448
```

```
# run cross validation
annotatedDTM <- convertMatrixToSparseM(DTM[trainingData[, "ID"], ])
best_C <- optimize_C(annotatedDTM, annotatedLabels)
```

```
## [1] "C = 0.003"
## [1] "C = 0.01"
## [1] "C = 0.03"
## [1] "C = 0.1"
## [1] "C = 0.3"
## [1] "C = 1"
## [1] "C = 3"
## [1] "C = 10"
## [1] "C = 30"
## [1] "C = 100"
## [1] "Best C value: 0.03, F1 = 0.647465370415286"
```

```
k_fold_cross_validation(annotatedDTM, annotatedLabels, cost = best_C)
```

```
##     P     R     S     F     A  Pos.
## 0.596 0.725 0.800 0.647 0.780 9.100
```

Bigrams

Now let us see, if the use of bigrams, i.e. concatenations of sequences of two words can improve the result. Bigrams, and larger n-Grams can capture important sequential contexts from texts such as negation, at least to a certain extent. For instance, "is not funny" will result in the features "is_not" and "not_funny".

To extract n-gram features, the `tokens_ngrams()` of `quanteda` accepts a list of integers, e.g. `1:2` to create unigram and bigram features.

```
corpus_token_bigrams <- corpus %>%
  tokens(remove_punct = TRUE, remove_numbers = TRUE, remove_symbols = TRUE) %>%
  tokens_tolower() %>%
  tokens_remove(pattern = stopwords_extended) %>%
  tokens_ngrams(1:2)

print(paste0("1: ", substr(paste(corpus_token_bigrams[4963], collapse = " "), 0, 400), '...'))
```

```
## [1] "1: legislation extend colony newfoundland articles treaty washington 8th day protocol effect si
```

```
# Compute DTM
DTM <- corpus_token_bigrams  %>%
  dfm() %>%
  dfm_trim(min_docfreq = minimumFrequency, max_docfreq = Inf)

# How many features do we have now?
dim(DTM)
```

```
## [1] 21334 22800
```

```
# How do they look like?
sample(colnames(DTM), 10)
```

```
##  [1] "unshaken"               "increase_national"     "republicans"
##  [4] "government_earnest"     "relations_maintained"  "gained_strength"
##  [7] "great_powers"           "electricity"           "exercise_control"
## [10] "struggle_independence"
```

```
# run cross validation
annotatedDTM <- convertMatrixToSparseM(DTM[trainingData[, "ID"], ])
best_C <- optimize_C(annotatedDTM, annotatedLabels)
```

```
## [1] "C = 0.003"
## [1] "C = 0.01"
## [1] "C = 0.03"
## [1] "C = 0.1"
## [1] "C = 0.3"
## [1] "C = 1"
## [1] "C = 3"
## [1] "C = 10"
## [1] "C = 30"
## [1] "C = 100"
## [1] "Best C value: 0.03, F1 = 0.644608227558143"
```

```r
k_fold_cross_validation(annotatedDTM, annotatedLabels, cost = best_C)
```

```
##     P     R     S     F     A  Pos.
## 0.591 0.725 0.795 0.645 0.777 9.100
```

In this case, bi-gram and tri-gram features seem to not contribute positively to the classification result.

Minimum feature frequency

Up to this point, we dropped features occurring less than five times in our data. Let's see if we include more features by increasing the minimum frequency.

```r
# More features
minimumFrequency <- 2

DTM <- corpus_token_sw  %>%
  dfm() %>%
  dfm_trim(min_docfreq = minimumFrequency, max_docfreq = Inf)

dim(DTM)
```

```
## [1] 21334 16689
```

```r
annotatedDTM <- convertMatrixToSparseM(DTM[trainingData[, "ID"], ])
best_C <- optimize_C(annotatedDTM, annotatedLabels)
```

```
## [1] "C = 0.003"
## [1] "C = 0.01"
## [1] "C = 0.03"
## [1] "C = 0.1"
## [1] "C = 0.3"
## [1] "C = 1"
## [1] "C = 3"
## [1] "C = 10"
## [1] "C = 30"
## [1] "C = 100"
## [1] "Best C value: 0.03, F1 = 0.651274894224809"
```

```r
k_fold_cross_validation(annotatedDTM, annotatedLabels, cost = best_C)
```

```
##     P     R     S     F     A  Pos.
## 0.602 0.725 0.804 0.651 0.783 9.100
```

It seems that feeding more features into the classifier has a little positive effect on the result.

Lemmatization

As a last method, we utilize lemmatization to unify different variants of the same semantic form (such as *nation* and *nations*).

```r
corpus_token_lemma <- corpus %>%
  tokens(remove_punct = TRUE, remove_numbers = TRUE, remove_symbols = TRUE) %>%
  tokens_tolower() %>%
  tokens_replace(lemma_data$inflected_form, lemma_data$lemma, valuetype = "fixed") %>%
  tokens_remove(pattern = stopwords_extended) %>%
  tokens_ngrams(1)

print(paste0("1: ", substr(paste(corpus_token_lemma[4963], collapse = " "), 0, 400), '...'))
```

```
## [1] "1: legislation extend colony newfoundland article treaty washington 8th day protocol effect sig
```

```r
minimumFrequency <- 2

DTM <- corpus_token_lemma  %>%
  dfm() %>%
  dfm_trim(min_docfreq = minimumFrequency, max_docfreq = Inf)

dim(DTM)
```

```
## [1] 21334 11523
```

```r
# run cross validation
annotatedDTM <- convertSlamToSparseM(DTM[trainingData[, "ID"], ])
best_C <- optimize_C(annotatedDTM, annotatedLabels)
```

```
## [1] "C = 0.003"
## [1] "C = 0.01"
## [1] "C = 0.03"
## [1] "C = 0.1"
## [1] "C = 0.3"
## [1] "C = 1"
## [1] "C = 3"
## [1] "C = 10"
## [1] "C = 30"
## [1] "C = 100"
## [1] "Best C value: 0.03, F1 = 0.69005439005439"
```

```r
k_fold_cross_validation(annotatedDTM, annotatedLabels, cost = best_C)
```

```
##     P     R     S     F     A  Pos.
## 0.641 0.767 0.817 0.690 0.807 9.100
```

Each individual approach to optimize our text features for classification has some effect on the results. It takes some time to engineer an optimal feature set.

Further, testing different features must be done for each new task / language individually, since there is no "one-size fits all" approach to this.

GENERAL ADVISE: For this tutorial we utilized a rather small training set of 300 examples, 91 of them in the positive class. Better classification accuracy can be expected, if more training data is available. Hence, instead of spending too much time on feature optimization, it will probably be a better idea to invest into generation of more training data first.

Final classification

We now apply our best classification model to the entire data set, to determine the occurrence of FORGEIN/DOMESTIC affairs related content in each single speech.

```r
# Final classification
annotatedDTM <- convertMatrixToSparseM(DTM[trainingData[, "ID"], ])

# C parameter tuning
best_C <- optimize_C(annotatedDTM, annotatedLabels)
```

```
## [1] "C = 0.003"
## [1] "C = 0.01"
## [1] "C = 0.03"
## [1] "C = 0.1"
## [1] "C = 0.3"
## [1] "C = 1"
## [1] "C = 3"
## [1] "C = 10"
## [1] "C = 30"
## [1] "C = 100"
## [1] "Best C value: 0.03, F1 = 0.69005439005439"
```

```r
# final classification
final_model <- LiblineaR(annotatedDTM, annotatedLabels, cost = best_C)
final_labels <- predict(final_model, convertSlamToSparseM(DTM))$predictions
table(final_labels) / sum(table(final_labels))
```

```
## final_labels
## DOMESTIC  FOREIGN
##    0.594    0.406
```

We see that the classifier puts the majority of the around 21,000 paragraphs into the DOMESTIC category. We can visualize the result as a bar chart with ggplot2. For better readability

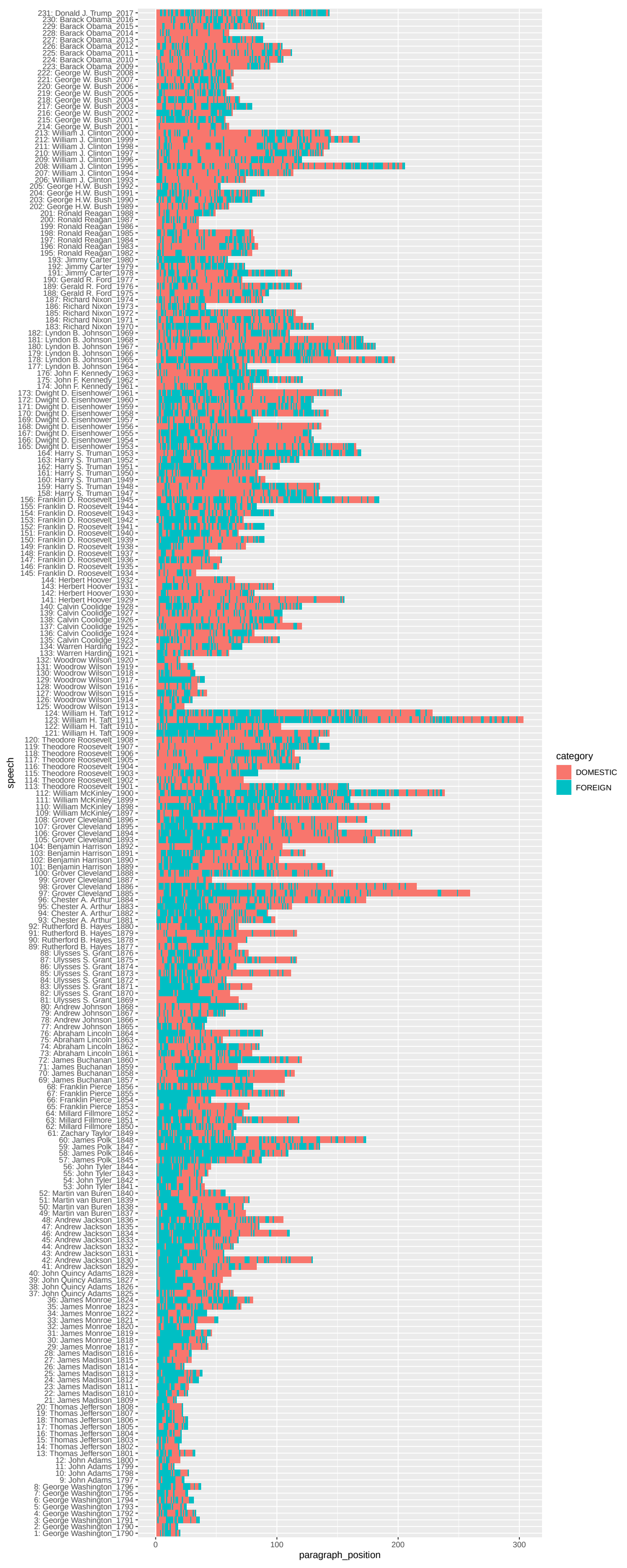
```r
speech_year <- substr(textdata$date, 0, 4)
speech_id <- textdata$speech_doc_id
paragraph_position <- unlist(sapply(table(speech_id), FUN = function(x) {1:x}))
presidents_df <- data.frame(
  paragraph_position = paragraph_position,
  speech = paste0(speech_id, ": ", textdata$president, "_", speech_year),
  category = final_labels
)

# preserve speech order in chart by using factors
presidents_df$speech <- factor(presidents_df$speech, levels = unique(presidents_df$speech))

# Remove two very long speeches to beautify the plot (you can also skipt this)
presidents_df <- presidents_df[!grepl("Carter_1981|Truman_1946", presidents_df$speech), ]

# plot classes of paragraphs for each speech as tile
require(ggplot2)
```

```r
ggplot(presidents_df, aes(x = speech, y = paragraph_position, fill = category)) +
  geom_tile(stat="identity") + coord_flip()
```


231: Donald J. Trump_2017
230: Barack Obama_2016
229: Barack Obama_2015
228: Barack Obama_2014
227: Barack Obama_2013
226: Barack Obama_2012
225: Barack Obama_2011
224: Barack Obama_2010
223: Barack Obama_2009
222: George W. Bush_2008
221: George W. Bush_2007
220: George W. Bush_2006
219: George W. Bush_2005
218: George W. Bush_2004
217: George W. Bush_2003
216: George W. Bush_2002
215: George W. Bush_2001
214: George W. Bush_2001
213: William J. Clinton_2000
212: William J. Clinton_1999
211: William J. Clinton_1998
210: William J. Clinton_1997
209: William J. Clinton_1996
208: William J. Clinton_1995
207: William J. Clinton_1994
206: William J. Clinton_1993
205: George H.W. Bush_1992
204: George H.W. Bush_1991
203: George H.W. Bush_1990
202: George H.W. Bush_1989
201: Ronald Reagan_1988
200: Ronald Reagan_1987
199: Ronald Reagan_1986
198: Ronald Reagan_1985
197: Ronald Reagan_1984
196: Ronald Reagan_1983
195: Ronald Reagan_1982
193: Jimmy Carter_1980
192: Jimmy Carter_1979
191: Jimmy Carter_1978
190: Gerald R. Ford_1977
189: Gerald R. Ford_1976
188: Gerald R. Ford_1975
187: Richard Nixon_1974
186: Richard Nixon_1973
185: Richard Nixon_1972
184: Richard Nixon_1971
183: Richard Nixon_1970
182: Lyndon B. Johnson_1969
181: Lyndon B. Johnson_1968
180: Lyndon B. Johnson_1967
179: Lyndon B. Johnson_1966
178: Lyndon B. Johnson_1965
177: Lyndon B. Johnson_1964
176: John F. Kennedy_1963
175: John F. Kennedy_1962
174: John F. Kennedy_1961
173: Dwight D. Eisenhower_1961
172: Dwight D. Eisenhower_1960
171: Dwight D. Eisenhower_1959
170: Dwight D. Eisenhower_1958
169: Dwight D. Eisenhower_1957
168: Dwight D. Eisenhower_1956
167: Dwight D. Eisenhower_1955
166: Dwight D. Eisenhower_1954
165: Dwight D. Eisenhower_1953
164: Harry S. Truman_1953
163: Harry S. Truman_1952
162: Harry S. Truman_1951
161: Harry S. Truman_1950
160: Harry S. Truman_1949
159: Harry S. Truman_1948
158: Harry S. Truman_1947
156: Franklin D. Roosevelt_1945
155: Franklin D. Roosevelt_1944
154: Franklin D. Roosevelt_1943
153: Franklin D. Roosevelt_1942
152: Franklin D. Roosevelt_1941
151: Franklin D. Roosevelt_1940
150: Franklin D. Roosevelt_1939
149: Franklin D. Roosevelt_1938
148: Franklin D. Roosevelt_1937
147: Franklin D. Roosevelt_1936
146: Franklin D. Roosevelt_1935
145: Franklin D. Roosevelt_1934
144: Herbert Hoover_1932
143: Herbert Hoover_1931
142: Herbert Hoover_1930
141: Herbert Hoover_1929
140: Calvin Coolidge_1928
139: Calvin Coolidge_1927
138: Calvin Coolidge_1926
137: Calvin Coolidge_1925
136: Calvin Coolidge_1924
135: Calvin Coolidge_1923
134: Warren Harding_1922
133: Warren Harding_1921
132: Woodrow Wilson_1920
131: Woodrow Wilson_1919
130: Woodrow Wilson_1918
129: Woodrow Wilson_1917
128: Woodrow Wilson_1916
127: Woodrow Wilson_1915
126: Woodrow Wilson_1914
125: Woodrow Wilson_1913
124: William H. Taft_1912
123: William H. Taft_1911
122: William H. Taft_1910
121: William H. Taft_1909
120: Theodore Roosevelt_1908
119: Theodore Roosevelt_1907
118: Theodore Roosevelt_1906
117: Theodore Roosevelt_1905
116: Theodore Roosevelt_1904
115: Theodore Roosevelt_1903
114: Theodore Roosevelt_1902
113: Theodore Roosevelt_1901
112: William McKinley_1900
111: William McKinley_1899
110: William McKinley_1898
109: William McKinley_1897
108: Grover Cleveland_1896
107: Grover Cleveland_1895
106: Grover Cleveland_1894
105: Grover Cleveland_1893
104: Benjamin Harrison_1892
103: Benjamin Harrison_1891
102: Benjamin Harrison_1890
101: Benjamin Harrison_1889
100: Grover Cleveland_1888
99: Grover Cleveland_1887
98: Grover Cleveland_1886
97: Grover Cleveland_1885
96: Chester A. Arthur_1884
95: Chester A. Arthur_1883
94: Chester A. Arthur_1882
93: Chester A. Arthur_1881
92: Rutherford B. Hayes_1880
91: Rutherford B. Hayes_1879
90: Rutherford B. Hayes_1878
89: Rutherford B. Hayes_1877
88: Ulysses S. Grant_1876
87: Ulysses S. Grant_1875
86: Ulysses S. Grant_1874
85: Ulysses S. Grant_1873
84: Ulysses S. Grant_1872
83: Ulysses S. Grant_1871
82: Ulysses S. Grant_1870
81: Ulysses S. Grant_1869
80: Andrew Johnson_1868
79: Andrew Johnson_1867
78: Andrew Johnson_1866
77: Andrew Johnson_1865
76: Abraham Lincoln_1864
75: Abraham Lincoln_1863
74: Abraham Lincoln_1862
73: Abraham Lincoln_1861
72: James Buchanan_1860
71: James Buchanan_1859
70: James Buchanan_1858
69: James Buchanan_1857
68: Franklin Pierce_1856
67: Franklin Pierce_1855
66: Franklin Pierce_1854
65: Franklin Pierce_1853
64: Millard Fillmore_1852
63: Millard Fillmore_1851
62: Millard Fillmore_1850
61: Zachary Taylor_1849
60: James Polk_1848
59: James Polk_1847
58: James Polk_1846
57: James Polk_1845
56: John Tyler_1844
55: John Tyler_1843
54: John Tyler_1842
53: John Tyler_1841
52: Martin van Buren_1840
51: Martin van Buren_1839
50: Martin van Buren_1838
49: Martin van Buren_1837
48: Andrew Jackson_1836
47: Andrew Jackson_1835
46: Andrew Jackson_1834
45: Andrew Jackson_1833
44: Andrew Jackson_1832
43: Andrew Jackson_1831
42: Andrew Jackson_1830
41: Andrew Jackson_1829
40: John Quincy Adams_1828
39: John Quincy Adams_1827
38: John Quincy Adams_1826
37: John Quincy Adams_1825
36: James Monroe_1824
35: James Monroe_1823
34: James Monroe_1822
33: James Monroe_1821
32: James Monroe_1820
31: James Monroe_1819
30: James Monroe_1818
29: James Monroe_1817
28: James Madison_1816
27: James Madison_1815
26: James Madison_1814
25: James Madison_1813
24: James Madison_1812
23: James Madison_1811
22: James Madison_1810
21: James Madison_1809
20: Thomas Jefferson_1808
19: Thomas Jefferson_1807
18: Thomas Jefferson_1806
17: Thomas Jefferson_1805
16: Thomas Jefferson_1804
15: Thomas Jefferson_1803
14: Thomas Jefferson_1802
13: Thomas Jefferson_1801
12: John Adams_1800
11: John Adams_1799
10: John Adams_1798
9: John Adams_1797
8: George Washington_1796
7: George Washington_1795
6: George Washington_1794
5: George Washington_1793
4: George Washington_1792
3: George Washington_1791
2: George Washington_1790
1: George Washington_1790
speech
0
100
200
300
paragraph_position
category
DOMESTIC
FOREIGN

Can you see how DOMESTIC affairs related content gets more important over the course of centuries? Also the position of FOREIGN policy statements changes around the turn from the 19th to 20th century from the begginning of a speech to more dispersed positions thoughout the speech, and finally a tendency to rather place them at the end of speeches.

Optional exercises

1. Divide the training data into a 80% training set and 20% test set. Train a classifier on the training set and evaluate the performance on the test set. As performance measure use **Krippendorff'a alpha** and **Cohen's Kappa** as provided in the `irr` package for R.
2. Use the `proba = T` parameter for the `predict` method during the final classification to evaluate on label probabilities instead of discrete label decisions. Use the output of probabilities for the label "FOREIGN" to classify paragraphs as "FOREIGN" only if the label probability is greater than 60%. Visualize the result. What can you observe compared to the previous plot (decision boundary around 50%)?