# Regulatory Documents via Neural Nets - Documentation

*Sebastian Knigge*

*18 9 2019*

## Contents



# 1 Setup

Following libraries are used in the code:

```
library(dplyr)
library(tidytext)
library(pdftools)
library(tidyr)
library(stringr)
library(tidytext)
library(udpipe)
library(topicmodels)
library(ggplot2)
library(wordcloud)
library(tm)
library(SnowballC)
library(RColorBrewer)
library(RCurl)
library(XML)
library(openxlsx)
library(keras)
```

# 2 Import data

The Documents had to be preprocessed like for the LDA part. This first part - again - focusses on reading in the pdf documents.

```
# getting the right order
setwd('..')
documents <- read.xlsx("Docs_classes.xlsx")[,2]
classes <- read.xlsx("Docs_classes.xlsx")[,c(1,3)]
documents <- paste0(documents,".pdf")
```

Following functions are used to set up and analyze the pdfs.

```r
# getting the right directory
library(here)
setwd("../")
path <- getwd() %>%
  file.path("TextDocs")
setwd(path)
```

When cleaning up data, we have to take into account certain circumstances of the regulatory documents. For example, there are many formulas and technical abbreviations in the documents. Every variable, every estimator, and every index is included as a single word in the bag of words. These terms sometimes have a big influence on the documents, because they are very specific for individual documents and occur quite often. To avoid this, we exclude all mixed words with characters and numeric values, as well as all terms with special characters (e.g. Greek letters).

```r
read_pdf_clean <- function(document){
  # This function loads the document given per name
  # and excludes the stop words inclusive numbers
  pdf1 <- pdf_text(file.path(path, document)) %>%
    strsplit(split = "\n") %>%
    do.call("c",.) %>%
    as_tibble() %>%
    unnest_tokens(word,value) %>%
    # also exclude all words including numbers and special characters
    filter(grepl("^[a-z]+$", word))
  # load stopword library
  data(stop_words)
  # add own words to stop word library - here the numbers from 1 to 10
  new_stop_words <- tibble(word=as.character(0:9),
                           lexicon=rep("own",10)) %>%
                           bind_rows(stop_words)
  # stop words are excluded via anti_join
  pdf1 %>%
    anti_join(new_stop_words)
}

plot_most_freq_words <- function(pdf, n=7){
  # plots a bar plot via ggplot
  pdf %>% count(word) %>% arrange(desc(n)) %>% head(n) %>%
    ggplot(aes(x=word,y=n)) +
    geom_bar(stat="identity")+
    # no labels for x and y scale
    theme(axis.title.y=element_blank(),
          axis.title.x=element_blank())
}
```

Now we can read in all documents in a for loop:

```r
# inital set up for the corpus
pdf1 <- read_pdf_clean(documents[1])
corpus <- tibble(document=1, word=pdf1$word)
# adding the documents iteratively
for (i in 2:length(documents)){
  pdf_i <- read_pdf_clean(documents[i])
  corpus <- tibble(document=i, word=pdf_i$word) %>% bind_rows(corpus,.)
}
```

# 3 Apply ANN

```
dtm <- corpus %>% count(document, word, sort = TRUE) %>%
  select(doc_id=document, term=word, freq=n) %>%
  document_term_matrix()
c(N,M) %<-% dim(dtm)
```

We will reduce the dimensionality again (i.e. performing embedding).

```
dtm_embedding2 <- dtm %>%
  dtm_remove_lowfreq(minfreq = 2)
# ordering
dtm_embedding2 <- dtm_embedding2[order(as.numeric(rownames(dtm_embedding2))),]

dtm_embedding_all <- dtm %>%
  dtm_remove_lowfreq(minfreq=0)
# ordering
dtm_embedding_all <- dtm_embedding_all[order(as.numeric(rownames(dtm_embedding_all))),]

dtm_embedding_tfidf <- dtm %>%
  dtm_remove_tfidf(top=M*0.8)
# ordering
dtm_embedding_tfidf <- dtm_embedding_tfidf[order(as.numeric(rownames(dtm_embedding_tfidf))),]
```

```
# convert x matrix into a form such that it can be used for tensorflow
adjust_tensor_format <- function(classes, dtm){
  x_chapters <- apply(dtm, 1, function(x) as.matrix(x)) %>% t()
  # one hot encoding for the Groups (y)
  topics_categorical <- classes$Group %>% -1 %>%
    to_categorical()
  ret <- list(
    x=x_chapters,
    y=topics_categorical,
    topics=classes$Group
  )
  return(ret)
}
```

```
dtm_tensor_2 <- adjust_tensor_format(classes, dtm_embedding2)
dtm_tensor_all <- adjust_tensor_format(classes, dtm_embedding_all)
dtm_tensor_tfidf <- adjust_tensor_format(classes, dtm_embedding_tfidf)
```

## 3.1 Splitting and Fitting

The following function splits the sample in an manner, such that each cluster is eually to its size represented in the test data and the validation data.

```
sample_cluster_wise <- function(data, test_ratio=0.2, val_ratio=0.2, seed=1234){
  X <- data$x; y <- data$y
  cluster=data$topics
  set.seed(seed)
  {
  # setting the absolute number of observations for the sample of each cluster
  n_test <- (table(cluster)*test_ratio) %>% floor()
```

```r
  n_val <- (table(cluster)*val_ratio) %>% floor()
  # function to get the correct sample indices for validation and test sample
  samp_ind <- function(i, n_list) which(cluster==i) %>% sample(n_list[i])
  test_indices <- unique(cluster) %>% sort() %>%
    sapply(function(i) samp_ind(i, n_test)) %>%
    unlist()
  val_indices <- unique(cluster) %>% sort() %>%
    sapply(function(i) samp_ind(i, n_val)) %>%
    unlist()
  }
  ret <- list(partial_x_train=X[-c(val_indices,test_indices),],
              partial_y_train=y[-c(val_indices,test_indices),],
              x_val=X[val_indices,], y_val = y[val_indices,],

              x_test=X[test_indices,], y_test = y[test_indices,])
  return(ret)
}
```

The following two functions are setting up the model and evaluate the goodness of fit.

```r
# The whole model is set up and trained within this function
set_up_n_fit <- function(split, books_n=n_books){
  # starting with 64 neurons and scaling it down to 46 in the
  # mid layer turned out to be a well predicting model
  model <- keras_model_sequential() %>%
    layer_dense(units=64, activation="relu", input_shape=ncol(split$partial_x_train)) %>%
    layer_dense(units=46, activation="relu") %>%
    # we want to classify for 7 categories
    layer_dense(units=7, activation="softmax")

  model %>% compile(
    optimizer="rmsprop",
    loss="categorical_crossentropy",
    metrics=c("accuracy"))

  history <- model %>% fit(
    split$partial_x_train,
    split$partial_y_train,
    # from experience the model tends to
    # overfit for more than 5 epochs
    epochs=5,
    batch_size=2^9,
    validation_data=list(split$x_val,split$y_val)
  )
  return(
    list(history=history,
         model=model))
}


# making a prediction on the test data and calculating the
# misspecification rate; we also want to save the true categories and the predicted ones
evaluate_model <- function(model_fit, y=split$y_test, x=split$x_test) {
  prediction <- model_fit %>% predict(x)
```

```
  pred <- apply(prediction, 1, which.max)
  true_value <- apply(y, 1, which.max)
  misspecified <- sum(!pred==true_value)/length(pred)
  ret <- list(misspecified=misspecified,
              # the function also dicloses the true and the predicted
              # values for exact evaluation, if needed
              pred=pred, true_value=true_value)
  return(ret)
}
```

Now we can start applying the model. Here we use a split of 20% validation-, 17% test- and 63% training-set. All 3 embeddings are evaluated. We use 35 simulations to test each embedding.

```
tim1 <- Sys.time()
n <- 35
results <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_2, seed=i*201, test_ratio = 0.17)
  results[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2 <- Sys.time()
```

```
u_2 <- tim2-tim1
results_2 <- results
misc.rate_2 <- results_2 %>% mean
```

```
tim1 <- Sys.time()
n <- 35
results <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_2, seed=i*201, test_ratio = 0.17)
  results[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2 <- Sys.time()
```

```
u_all <- tim2-tim1
results_all <- results
misc.rate_all <- results_all %>% mean
```

```
tim1 <- Sys.time()
n <- 35
results <- rep(NA,n)
for(i in 1:n){
  split <- sample_cluster_wise(dtm_tensor_tfidf, seed=i*201, test_ratio = 0.17)
  results[i] <- set_up_n_fit(split) %>% .$model %>%
    evaluate_model() %>% .$misspecified
}
tim2 <- Sys.time()
```

```
u_tfidf <- tim2-tim1
results_tfidf <- results
misc.rate_tfidf <- results_tfidf %>% mean
```

```
# overview
performance_matrix <- data.frame(freq2.embedding=c(misc.rate_2, u_2),
           all.embedding=c(misc.rate_all, u_all),
           tfidf=c(misc.rate_tfidf, u_tfidf))
rownames(performance_matrix) <- c("missc. rate", "time")
performance_matrix %>%  stargazer(summary=FALSE, header=F, title = "ANN for Reg. Docs.")
```

Table 1: ANN for Reg. Docs.

| | freq2.embedding | all.embedding | tfidf |
|---|---|---|---|
| missc. rate | 0.286 | 0.357 | 0.271 |
| time | 1.770 | 4.261 | 8.913 |