Everyone is talking about text analysis. Is it puzzling that this data source is so popular right now? Actually no. Most of our datasets rely on (hand-coded) textual information. Extracting, processing, and analyzing this oasis of information becomes increasingly relevant for a large variety of research fields.

**What is quanteda?**

In order to analyze text data, R has several packages available. In this blog post we focus on quanteda. quanteda is one of the most popular R packages for the **qu**antitative **an**alysis of **te**xtual **da**ta that is fully-featured and allows the user to easily perform natural language processing tasks. It offers an extensive documentation and is regularly updated. quanteda is most useful for preparing data that can then be further analyzed using unsupervised/supervised machine learning or other techniques. A combination with tidyverse leads to a more transparent code structure and offers a mere variety of useful areas that could not be addressed within the limited time of the workshop (e.g., scaling models, part-of-speech (POS) tagging, named entities, word embeddings, etc.).

There are also similar R packages such as tm, tidytext, and koRpus. [tm](https://cran.r-project.org/web/packages/tm/index.html) has simpler grammer but slightly fewer features, [tidytext](https://cran.r-project.org/web/packages/tidytext/index.html) is very closely integrated with dplyr and well-documented, and [koRpus](https://cran.r-project.org/web/packages/koRpus/index.html) is good for tasks

**How do we use quanteda?**

Most analyses in quanteda require three steps:

**1. Import the data**

The data that we usually use for text analysis is available in text formats (e.g., .txt or .csv files).

**2. Build a corpus**

After reading in the data, we need to generate a **corpus**. A corpus is a type of dataset that is used in text analysis. It contains “a collection of text or speech material that has been brought together according to a certain set of predetermined criteria” . These criteria are usually set by the researchers and are in concordance with the guiding question. For instance, if you are interested in analyzing speeches in the UN General Debate, these predetermined criteria are the time and scope conditions of these debates (speeches by countries at different points in time).

**3. Calculate a data frequency matrix (DFM)**

Another essential component for text analysis is a **data frequency matrix (DFM)**; also called **document-term matrix (DTM)**. These two terms are synonyms but quanteda refers to a DFM whereas others will refer to DTM. It describes how frequently terms occur in the corpus by counting single terms.  
To generate a DFM, we first split the text into its single terms (tokens). We then count how frequently each term (token) occurs in each document.

The following graphic describes visually how we turn raw text into a vector-space representation that is easily accessible and analyzable with quantitative statistical tools. It also visualizes how we can think of a DFM. The rows represent the documents that are part of the corpus and the columns show the different terms (tokens). The values in the cells indicate how frequently these terms (tokens) are used across the documents.



Figure 1: Model of a DFM

**Important things to remember about DFMs:**

* A **corpus is positional (string of words)** and a **DFM is non-positional (bag of words)**. Put differently, the order of the words matters in a corpus whereas a DFM does not have information on the position of words.
* A **token** is each individual word in a text (but it could also be a sentence, paragraph, or character). This is why we call creating a “bag of words” also **tokenizing text**. In a nutshell, a DFM is a very efficient way of organizing the frequency of features/tokens but does not contain any information on their position. In our example, the **features** of a text are represented by the columns of a DFM and aggregate the frequency of each **token**.
* In most projects you want **one corpus to contain all your data** and **generate many DFMs** from that.
* The **rows of a DFM** can contain **any unit** on which you can **aggregate documents**. In the example above, we used the single documents as the unit. It may also well be more fine-grained with sub-documents or more aggregated with a larger collection of documents.
* The **columns of a DFM** are **any unit** on which you can **aggregate features**. **Features** are extracted from the texts and quantitatively measurable. Features can be words, parts of the text, content categories, word counts, etc. In the example above, we used single words such as “united”, “nations”, and “peace”.

To showcase the three steps introduced above, we are using the [UN General Debate data by Mikhaylov, Baturo, and Dasandi](https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/0TJX8Y) dataset. There is also a [pre-processed version of the dataset](https://rdrr.io/github/quanteda/quanteda.corpora/man/data_corpus_ungd2017.html) accessible with quanteda.corpora.

How to access the UNGD data with quanteda.corpora

# Install package quanteda.corpora

Library("quanteda/quanteda.corpora")

# Load the dataset

quanteda.corpora::data\_corpus\_ungd2017

We will, however, mainly rely on the original dataset throughout the following explanations to match closely the regular workflow of textual data in R. If you want to replicate the steps, please download the data [here](https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/0TJX8Y) and unzip the zip file. Your global\_path should direct you to the text file folders.

In a first step, we need to load the necessary packages and read in the data.

# Load all required packages

library(tidyverse) # Also loads dplyr, ggplot2, and haven

library(quanteda) # For NLP

library(readtext) # To read .txt files

library(stm) # For structural topic models

library(stminsights) # For visual exploration of STM

library(wordcloud) # To generate wordclouds

library(gsl) # Required for the topicmodels package

library(topicmodels) # For topicmodels

library(caret) # For machine learning

# Download data here:

# <https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/0TJX8Y>

# and unzip the zip file

# Read in data (.txt files)

global\_path <- "path/to/folder/UN-data/"

# We load the data (.txt files) from all subfolders (readtext can handle

# this without specification) and store them in the main UNGDspeeches

# dataframe. Beyond the speech text, this data also includes the

# meta-data from the text filenames and add variables for the country,

# UN session, and year.

# The code is based on <https://github.com/quanteda/quanteda.corpora/issues/6>

# and <https://github.com/sjankin/UnitedNations/blob/master/files/UNGD_analysis_example.Rmd>

# For the purpose of this blog post, we use the data from all sessions.

UNGDspeeches <- readtext(

paste0(global\_path, "\*/\*.txt"),

docvarsfrom = "filenames",

docvarnames = c("country", "session", "year"),

dvsep = "\_",

encoding = "UTF-8"

)

We can then proceed and generate a corpus.

mycorpus <- corpus(UNGDspeeches, country = "country")

# Assigns a unique identifier to each text

docvars(mycorpus, "Textno") <-

sprintf("%02d", 1:ndoc(mycorpus))

As we can see (by calling the object mycorpus), the corpus consists of 8,093 documents.

Output: mycorpus

mycorpus

Corpus consisting of 8,093 documents and 4 docvars.

With this data, we can already generate first descriptive statistics.

# Save statistics in "mycorpus.stats"

mycorpus.stats <- summary(mycorpus)

# And print the statistics of the first 10 observations

head(mycorpus.stats, n = 10)

## Text Types Tokens Sentences country session year Textno

## 1 ALB\_25\_1970.txt 1728 9078 256 ALB 25 1970 01

## 2 ARG\_25\_1970.txt 1425 5192 218 ARG 25 1970 02

## 3 AUS\_25\_1970.txt 1612 5690 270 AUS 25 1970 03

## 4 AUT\_25\_1970.txt 1340 4717 164 AUT 25 1970 04

## 5 BEL\_25\_1970.txt 1288 4786 207 BEL 25 1970 05

## 6 BLR\_25\_1970.txt 1427 6138 204 BLR 25 1970 06

## 7 BOL\_25\_1970.txt 1560 5613 225 BOL 25 1970 07

## 8 BRA\_25\_1970.txt 1333 4427 154 BRA 25 1970 08

## 9 CAN\_25\_1970.txt 728 1887 97 CAN 25 1970 09

## 10 CMR\_25\_1970.txt 928 3144 106 CMR 25 1970 10

In a next step, we can also calculate the **data frequency matrix**. To do so, first we need to generate tokens (tokens()). This includes removing the numbers (remove\_numbers), punctuations (remove\_punct), symbols (remove\_symbols), twitter characters such as @ and # (remove\_twitter), urls beginning with http(s) (remove\_url), and hyphens (remove\_hyphens). We further include the docvars from our corpus (include\_docvars).

# Preprocess the text

# Create tokens

token <-

tokens(

mycorpus,

remove\_numbers = TRUE,

remove\_punct = TRUE,

remove\_symbols = TRUE,

remove\_twitter = TRUE,

remove\_url = TRUE,

remove\_hyphens = TRUE,

include\_docvars = TRUE

)

Since the pre-1994 documents were scanned with OCR scanners, several tokens with combinations of digits and characters were introduced.

# Clean tokens created by OCR

token\_ungd <- tokens\_select(

token,

c("[\\d-]", "[[:punct:]]", "^.{1,2}$"),

selection = "remove",

valuetype = "regex",

verbose = TRUE

)

In the next step, we then create the **data frequency matrix**. We lower and stem the words (tolower and stem) and remove common stop words (remove=stopwords()). Stopwords are words that appear in texts but do not give the text a substantial meaning (e.g., “the”, “a”, or “for”). Since the language of all documents is English, we only remove English stopwords here. quanteda can also deal with stopwords from other languages.

mydfm <- dfm(token\_ungd,

tolower = TRUE,

stem = TRUE,

remove = stopwords("english")

)

We can also trim the text with dfm\_trim. Using the command and its respective specifications, we filter words that appear less than 7.5% and more than 90%. This rather conservative approach is possible because we have a sufficiently large corpus.

mydfm.trim <-

dfm\_trim(

mydfm,

min\_docfreq = 0.075,

# min 7.5%

max\_docfreq = 0.90,

# max 90%

docfreq\_type = "prop"

)

To get a look at the DFM, we now print their first 10 observations and first 10 features:

# And print the results of the first 10 observations and first 10 features in a DFM

head(dfm\_sort(mydfm.trim, decreasing = TRUE, margin = "both"),

n = 10,

nf = 10)

Document-feature matrix of: 10 documents, 10 features (6.0% sparse).

10 x 10 sparse Matrix of class "dfm"

features

docs problem session conflict council africa global resolut hope south situat

CUB\_34\_1979.txt 36 8 1 0 13 3 10 8 10 23

IRL\_39\_1984.txt 41 9 18 11 14 5 16 21 19 9

PAN\_37\_1982.txt 14 12 12 8 11 2 6 11 20 10

BFA\_29\_1974.txt 25 17 1 4 15 0 10 20 6 9

GRC\_43\_1988.txt 27 9 13 14 10 2 10 10 11 5

PRY\_38\_1983.txt 30 17 12 3 0 3 21 10 8 16

RUS\_31\_1976.txt 16 12 4 2 8 0 16 2 6 9

UGA\_30\_1975.txt 26 17 2 7 54 0 9 9 12 8

RUS\_32\_1977.txt 13 8 12 1 12 0 10 3 6 11

ALB\_28\_1973.txt 17 4 6 1 5 1 3 2 9 13

The sparsity gives us information about the proportion of cells that have zero counts.

**Classification**

A researcher usually faces one of the following situations: **The categories are known beforehand** or **the categories are unknown**. If the researcher knows the categories, s/he can use automated methods to minimize the workload that is associated with the categorization of the texts. Throughout the workshop, two methods were presented: a **dictionary method** and a **supervised method**. If the researcher does not know the categories, s/he is likely to resort to **unsupervised machine learning**. The following section provides illustrative examples for both methods.

**Known categories**

**Known categories: Dictionaries**

**Dictionaries** contain lists of words that correspond to different categories. If we apply a dictionary approach, we count how often words that are associated with different categories are represented in each document. These dictionaries help us to classify (or categorize) the speeches based on the frequency of the words that they contain. Popular dictionaries are sentiment dictionaries.

To read in the dictionary, we use quanteda’s built-in function dictionary().

# load the dictionary with quanteda's built-in function

dict <- dictionary(file = "policy\_agendas\_english.lcd")

We apply this dictionary to filter the share of each country’s speeches on immigration, international affair and defence.

mydfm.un <- dfm(mydfm.trim, groups = "country", dictionary = dict)

[un.topics.pa](http://un.topics.pa) <- convert(mydfm.un, "data.frame") %>%

dplyr::rename(country = document) %>%

select(country, immigration, intl\_affairs, defence) %>%

tidyr::gather(immigration:defence, key = "Topic", value = "Share") %>%

group\_by(country) %>%

mutate(Share = Share / sum(Share)) %>%

mutate(Topic = haven::as\_factor(Topic))

In a next step, we can visualize the results with ggplot. This gives us a first impression of the distribution of the topics in the 2018 UN General Debate across countries.

[un.topics.pa](http://un.topics.pa) %>%

ggplot(aes(country, Share, colour = Topic, fill = Topic)) +

geom\_bar(stat = "identity") +

scale\_colour\_brewer(palette = "Set1") +

scale\_fill\_brewer(palette = "Pastel1") +

ggtitle("Distribution of PA topics in the UN General Debate corpus") +

xlab("") +

ylab("Topic share (%)") +

theme(axis.text.x = element\_blank(),

axis.ticks.x = element\_blank())



Figure 3: Distribution of PA topics in the UN General Debate corpus

We observe a relatively high share for both defence and international affairs whereas immigration receives fewer attention in the speeches.

**Known categories: Supervised machine learning – Naive Bayes (NB)**

We now turn to **supervised machine learning**. Similar to the dictionary approach explained above, **this method also requires some pre-existing classifications**. But in contrast to a dictionary, we now divide the data into a **training** and a **test dataset**. This follows the general logic of machine learning algorithms. The training data already contains the classifications and *trains* the algorithm (e.g., our Naive Bayes classifier) to predict the class of our speech based on the features that are given. A Naive Bayes classifier now calculates the probability for each class based on the features. It eventually goes for the class with the highest probability and selects this class as the corresponding category. It is **based on the Bayes theorem for conditional probability**. It can be formally written as:  
\[ P(A | B) = \frac{P(A) \* P(B | A)}{P(B)}\]  
In plain words, the probability of A is conditional on B.

Bayes’ theorem

* \(A\) and \(B\) are events
* \(P(A)\) and \(P(B)\) is the probability of observing \(A\) and \(B\) (respectively) independent from each other
* \(P(A) \neq 0\) and \(P(B) \neq 0\)
* \(P(A|B)\) is the **conditional probability** that \(A\) occurs when \(B\) is true  
  \[P(A|B) = \frac{P(A \cap B)}{P(B)}, if P(B) \neq 0\]
* \(P(B|A)\) is the **conditional probability** that \(B\) occurs when \(A\) is true  
  \[P(B|A) = \frac{P(B \cap A)}{P(A)}, if P(A) \neq 0\]
* And we also have the **joint probability** of $ P(A B) = P(B A)$ because

\[ \Longrightarrow P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)\]  
\[ \Longrightarrow P(A | B) = \frac{P(A \cap B)}{P(B)}\]  
\[ \Longrightarrow P(A | B) = \frac{\frac{P(A \cap B)}{P(B)}\*P(A)}{P(B)}\]  
\[ \Longrightarrow P(A | B) = \frac{P(B|A)\*P(A)}{P(B)}\]

**Why is Naive Bayes “naive”?** Naive Bayes is “naive” because of its **strong independence assumptions**. It assumes that all features are equally important and that all features are independent. If you think of n-grams and compare unigrams and bigrams, you can intuitively understand why the last assumption is a strong assumption. A unigram counts each word as a gram (“I” “like” “walking” “in” “the” “sun”) whereas a bigram counts two words as a gram (“I like” “like walking” “walking in” “in the” “the sun”).

However, even when the assumptions are not fully met, Naive Bayes still performs well.

A Naive Bayes is a **relatively simple classification algorithm** because it does not require much time and working capacity of your machine. To use a Naive Bayes classifier, we rely on quanteda’s built-in function [textmodel\_nb](http://quanteda.io/reference/textmodel_nb.html).

To perform the Naive Bayes estimation, we proceed with the following steps:

**1. We set up training and test data based on the corpus.**  
**2. Based on these two datasets, we generate a DFM.**  
**3. We train the algorithm by feeding in the training data and eventually use the test data for performance.**  
**4. We then check the performance (accuracy) of our results.**  
**5. And compare it with a random prediction.**

# load data

load("../newspaper.RData")

# transform variables

pred\_data$text <- as.character(pred\_data$text)

pred\_data$country <- as.character(pred\_data$country)

For simplicity, we keep only the USA, Great Britain, France, Brazil, and Japan.

pred\_data <- pred\_data %>%

dplyr::filter(country %in% c("us", "gb", "fr", "br", "jp")) %>%

dplyr::select(text, country)

head(pred\_data)

| **row** | **text** | **country** |
| --- | --- | --- |
| 1 | ’08 French champ Ivanovic loses to Safarova in 3rd. PARIS (AP) – Former French Open champion Ana Ivanovic lost in the third round Saturday, beaten 6-3, 6-3 by 23rd-seeded Lucie Safarova of the Czech Republic. | fr |
| 2 | Up to USD1,000 a day to care for child migrants. More than 57,000 unaccompanied children, mostly from Central America, have been caught entering the country illegally since last October, and President Barack Obama has asked for USD3.7 billion in emergency funding to address what he has called an ‘urgent humanitarian solution.’ ‘One of the figures that sticks in everybody’s mind is we’re paying about USD250 to USD1,000 per child,’ Senator Jeff Flake told reporters, citing figures presented at a closed-door briefing by Homeland Security Secretary Jeh Johnson. Federal authorities are struggling to find more cost-effective housing, medical care, counseling and legal services for the undocumented minors. The base cost per bed was USD250 per day, including other services, Senator Dianne Feinstein said, without providing details. | us |
| 3 | 1,400 gay weddings in England, Wales in first three months. Just over 1,400 gay couples tied the knot in the three months after same-sex marriage was allowed in England and Wales, figures out Thursday showed. The Office for National Statistics said 1,409 marriages took place between March 29 and June 30. ‘The novelty and significance of marriage becoming available led to an initial rush among same-sex couples wanting to be among the very first to assume the same rights and protection afforded to heterosexual couples,’ said James Brown, a partner at law firm JMW Solicitors. The figures will likely surge from December once civil partnerships can be converted into marriages. | gb |
| 4 | 1 dead after fan fighting in Brazil. SAO PAULO (AP) – Police say a 21-year-old man died after a confrontation between rival football fan groups in Brazil on Sunday. | br |
| 5 | 1 dead as plane with French tourists crashes in US. PAGE, Arizona (AP) – Authorities say a small plane carrying French tourists crashed while trying to land at an airport in Arizona, and one person was killed and another hospitalized. | fr |
| 6 | 1 US theory is someone diverted missing plane. WASHINGTON (AP) – A U.S. official says investigators are examining the possibility that someone caused the disappearance of a Malaysia Airlines jet with 239 people on board, and that it may have been ‘an act of piracy.’ | us |

We pre-process the data again. Our final corpus thus includes the newspaper headlines by country.

data\_corpus <- corpus(pred\_data, text\_field = "text")

In a first step, we need to define our training and our test dataset. Based on these two datasets, we generate a DFM. To do so, we apply similar general data pre-processing steps as discussed above.

# Set a seed for replication purposes

set.seed(68159)

# Generate random 10,000 numbers without replacement

training\_id <- sample(1:29542, 10000, replace = FALSE)

# Create docvar with ID

docvars(data\_corpus, "id\_numeric") <- 1:ndoc(data\_corpus)

# Get training set

dfmat\_training <-

corpus\_subset(data\_corpus, id\_numeric %in% training\_id) %>%

dfm(stem = TRUE)

# Get test set (documents not in training\_id)

dfmat\_test <-

corpus\_subset(data\_corpus,!id\_numeric %in% training\_id) %>%

dfm(stem = TRUE)

We can now check the distribution of the countries across the two DFMs:

print(prop.table(table(docvars(

dfmat\_training, "country"

))) \* 100)

br fr gb jp us

13.34 14.37 33.65 11.10 27.54

print(prop.table(table(docvars(

dfmat\_test, "country"

))) \* 100)

br fr gb jp us

13.70893 15.13151 33.02630 10.93542 27.19783

As we can see, the countries are equally distributed across both DFMs.

In a next step, we train the Naive Bayes classifier. Going back to the formula stated above, we know that A is conditional on B.

\[ P(A | B) = \frac{P(A) \* P(B | A)}{P(B)}\]

**A is what we want to know** (the country that is mainly addressed in each text) and **B is what we see** (the text). We can now proceed and replace A and B with the respective terms. This leads us to the next equation:

\[ P(Country | Text) = \frac{P(Country) \* P(Text | Country)}{P(Text)}\]

We can then proceed and train our algorithm using quanteda’s built-in function textmodel\_nb.

# Train naive Bayes

# The function takes a DFM as the first argument

model.NB <-

textmodel\_nb(dfmat\_training, docvars(dfmat\_training, "country"), prior = "docfreq")

# The prior indicates an assumed distribution.

# Here we choose how frequently the categories occur in our data.

dfmat\_matched <-

dfm\_match(dfmat\_test, features = featnames(dfmat\_training))

The command summary(model.NB) gives us the results of our prediction. Click unfold to see the results.

Code: summary(model.NB)

summary(model.NB)

Call:

textmodel\_nb.dfm(x = dfmat\_training, y = docvars(dfmat\_training,

"country"), prior = "docfreq")

Class Priors:

(showing first 5 elements)

br fr gb jp us

0.1334 0.1437 0.3365 0.1110 0.2754

Estimated Feature Scores:

' 08 french champ ivanov lose to safarova in 3rd . pari

br 0.08949 0.1679 0.007791 0.07636 0.08195 0.08707 0.09989 0.1474 0.1214 0.34322 0.1176 0.008863

fr 0.14212 0.2778 0.940061 0.37912 0.20345 0.16469 0.13803 0.3659 0.1473 0.24345 0.1296 0.947706

gb 0.43332 0.1532 0.032008 0.24397 0.14963 0.49395 0.35845 0.1346 0.3540 0.17905 0.3321 0.032363

jp 0.06507 0.1184 0.004396 0.10771 0.28900 0.06580 0.10287 0.1040 0.1007 0.06916 0.1059 0.002778

us 0.27000 0.2827 0.015743 0.19285 0.27598 0.18850 0.30077 0.2482 0.2766 0.16512 0.3148 0.008290

( ap ) - former open champion ana lost the third round

br 0.16080 0.24091 0.16100 0.1611 0.09034 0.17116 0.12708 0.08142 0.08045 0.1101 0.08964 0.30833

fr 0.14221 0.14820 0.14219 0.1363 0.17312 0.18636 0.24110 0.13475 0.16643 0.1350 0.17178 0.19535

gb 0.34860 0.28303 0.34870 0.3403 0.45720 0.36654 0.51497 0.07433 0.38923 0.3603 0.39194 0.24189

jp 0.09123 0.08125 0.09126 0.0922 0.04583 0.07443 0.05684 0.22969 0.07943 0.1005 0.07653 0.04757

us 0.25715 0.24661 0.25685 0.2702 0.23351 0.20152 0.06002 0.47982 0.28446 0.2940 0.27010 0.20686

saturday , beaten 6-3 by 23rd-seed

br 0.14287 0.1078 0.19323 0.21226 0.09731 0.1955

fr 0.19921 0.1384 0.20558 0.40985 0.12900 0.3236

gb 0.44845 0.3437 0.45359 0.21530 0.33256 0.1785

jp 0.09248 0.1099 0.07787 0.08317 0.11636 0.1379

us 0.11699 0.3001 0.06972 0.07942 0.32477 0.1646

To better understand how well we did, we can also generate two frequency tables for right and wrong predictions.

prop.table(table(predict(model.NB) == docvars(dfmat\_training, "country"))) \* 100

FALSE TRUE

2.71 97.29

To check if this result indicates a good performance, we compare it with a random result. We randomize the list of countries (and keep the overall frequency distribution of our countries constant) to allow our random algorithm a legitimate chance for a correct classification.

prop.table(table(sample(predict(model.NB)) == docvars(dfmat\_training, "country"))) \* 100

FALSE TRUE

76.63 23.37

As we can see from the the summarized table below, our Naive Bayes classifier clearly outperforms a random algorithm.

|  | **Naive Bayes** | **Random** |
| --- | --- | --- |
| False | 2.71 | 76.63 |
| True | 97.29 | 23.37 |

We are likely to increase our accuracy even more by pre-processing our text data.

A confusion matrix helps us assess how well our algorithm performed. It shows us the prediction for all five countries in contrast to the actual class that is given by our data. For example, we predict 2580 articles as belonging to Great Britain that actually belong to Great Britain. However, we also predict 39 articles as British articles while they are actually French. Overall, when we look at the diagonal, we see that most predictions correctly classify the articles and that our algorithm performs well.

actual\_class <- docvars(dfmat\_matched, "country")

predicted\_class <- predict(model.NB, newdata = dfmat\_matched)

tab\_class <- table(actual\_class, predicted\_class)

tab\_class

predicted\_class

actual\_class br fr gb jp us

br 2580 6 50 6 37

fr 39 2697 124 11 86

gb 24 52 6072 24 282

jp 33 9 31 1945 119

us 56 32 194 66 4967

We store our confusion matrix in an object because we need it later to visualize the results.

confusion <- confusionMatrix(tab\_class, mode = "everything")

Confusion Matrix and Statistics

predicted\_class

actual\_class br fr gb jp us

br 2580 6 50 6 37

fr 39 2697 124 11 86

gb 24 52 6072 24 282

jp 33 9 31 1945 119

us 56 32 194 66 4967

Overall Statistics

Accuracy : 0.9344

95% CI : (0.9309, 0.9379)

No Information Rate : 0.3311

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.914

Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

Class: br Class: fr Class: gb Class: jp Class: us

Sensitivity 0.9444 0.9646 0.9383 0.94786 0.9046

Specificity 0.9941 0.9845 0.9708 0.98902 0.9752

Pos Pred Value 0.9630 0.9121 0.9408 0.91015 0.9345

Neg Pred Value 0.9910 0.9940 0.9695 0.99385 0.9632

Precision 0.9630 0.9121 0.9408 0.91015 0.9345

Recall 0.9444 0.9646 0.9383 0.94786 0.9046

F1 0.9536 0.9376 0.9396 0.92862 0.9193

Prevalence 0.1398 0.1431 0.3311 0.10500 0.2810

Detection Rate 0.1320 0.1380 0.3107 0.09953 0.2542

Detection Prevalence 0.1371 0.1513 0.3303 0.10935 0.2720

Balanced Accuracy 0.9692 0.9745 0.9546 0.96844 0.9399

To display our confusion matrix visually, we could either produce a heatmap or a confusion matrix.

We first plot a heatmap using the code below.

Code: Heatmap using ggplot2

# Save confusion matrix as data frame

confusion.data <- as.data.frame(confusion[["table"]])

# Reverse the order

level\_order\_y <-

factor(confusion.data$actual\_class,

level = c('us', 'jp', 'gb', 'fr', 'br'))

ggplot(confusion.data,

aes(x = predicted\_class, y = level\_order\_y, fill = Freq)) +

xlab("Predicted class") +

ylab("Actual class") +

geom\_tile() + theme\_bw() + coord\_equal() +

scale\_fill\_distiller(palette = "Blues", direction = 1) +

scale\_x\_discrete(labels = c("Brazil", "France", "Great \n Britain", "Japan", "USA")) +

scale\_y\_discrete(labels = c("USA", "Japan", "Great \n Britain", "France", "Brazil"))



Figure 4: Contingency table

Code: Confusion matrix

# Generate a data matrix from the confusion table

dat <- data.matrix(confusion$table)

# Change order of column names

order.columns <- c(5, 4, 3, 2, 1)

dat <- dat[order.columns,]

par(mgp = c(1.5, .3, 0))

# Plot

plot(

0,

0,

# Type of plotting symbol

pch = "",

# Range of x-axis

xlim = c(0.5, 5.5),

# Range of y-axis

ylim = c(0.5, 6.5),

# Suppresses both x and y axes

axes = FALSE,

# Label of x-axis

xlab = "Predicted class",

# Label of y-axis

ylab = "Actual class",

)

# Write a for-loop that adds the bubbles to the plot

for (i in 1:dim(dat)[1]) {

symbols(

c(1:dim(dat)[2]),

rep(i, dim(dat)[2]),

circle = sqrt(dat[i,] / 9000 / pi),

add = TRUE,

inches = FALSE,

fg = brewer.pal(sqrt(dat[i,] / 9000 / pi), "Blues"),

bg = brewer.pal(sqrt(dat[i,] / 9000 / pi), "Blues")

)

}

axis(

1,

col = "white",

col.axis = "black",

at = c(1:5),

label = colnames(dat)

)

axis(

2,

at = c(1:5),

label = rownames(dat),

las = 1,

col.axis = "black",

col = "white"

)

# Add numbers to plot

for (i in 1:5) {

text(c(1:5), rep(i, 5), dat[i,], cex = 0.8)

}



Figure 5: Contingency table

Both figures show us that our prediction performs well for all countries but it performs particularly well for the USA and Great Britain. Darker colors show a higher frequency in both plots, the contingency table also indicates a greater frequency with the size of the bubbles.

**Unknown categories**

**Unknown categories: Unsupervised machine learning – Latent semantic analysis (LSA)**

The next section addresses how to analyze texts with unknown categories. **Latent Semantic Analysis (LSA)** evaluates documents and seeks to find the underlying meaning or concept of these documents. If each word only had one meaning, LSA would have an easy job. However, oftentimes, words are ambiguous, have multiple meanings or are synonyms. One example from our corpus is “may” – it could be a verb, a noun for a month, or a name. To overcome this problem, LSA essentially compares how often words appear together in one document and then compares this across all other documents. By grouping words with other words, we try to identify those words that are semantically related and eventually also get the true meaning of ambiguous words.  
More technically, LSA is a useful technique for aligning feature distributions to an n-dimensional space. LSA can (among other things) be used to compare similarity of documents/documents grouped by some variable.

1. **Documents are non-positional** (“bag of words”). The “bag of words” approach assumes that the order of the words does not matter. What matters is only the frequency of the single words.
2. **Concepts are** understood as **patterns of words** where certain words often go together in similar documents.
3. **Words only have one meaning** given the contexts surrounding the patterns of words.

For the next example, we go back to the UN General Assembly speech data set.

corpus.un.sample <- corpus\_sample(mycorpus, size = 500)

We again follow the cleaning steps described above.

Code: Data pre-processing steps

# Create tokens

token\_sample <-

tokens(

corpus.un.sample,

remove\_numbers = TRUE,

remove\_punct = TRUE,

remove\_symbols = TRUE,

remove\_twitter = TRUE,

remove\_url = TRUE,

remove\_hyphens = TRUE,

include\_docvars = TRUE

)

# Clean tokens created by OCR

token\_ungd\_sample <- tokens\_select(

token\_sample,

c("[\\d-]", "[[:punct:]]", "^.{1,2}$"),

selection = "remove",

valuetype = "regex",

verbose = TRUE

)

dfmat <- dfm(token\_ungd\_sample,

tolower = TRUE,

stem = TRUE,

remove = stopwords("english")

)

And then run the textmodel\_lsa command.

mylsa <- textmodel\_lsa(dfmat, nd = 10)

One interesting question would be: How similar are the USA and Russia? Each dot represents a country-year observation. The USA are colored blue, Russia is colored red, and all other countries are grey.

# We need the "stringr" package for the following command

sources <-

str\_remove\_all(rownames(mylsa$docs), "[0-9///'.\_txt]")

sources.color <- rep("gray", times = length(sources))

sources.color[sources %in% "USA"] <- "blue"

sources.color[sources %in% "RUS"] <- "red"

plot(

mylsa$docs[, 1:2],

col = alpha(sources.color, 0.3),

pch = 19,

xlab = "Dimension 1",

ylab = "Dimension 2",

main = "LSA dimensions by subcorpus"

)



Figure 6: Distribution of PA topics in the UN General Debate corpus

On Dimension 2 we do not really observe a difference between documents from the US and Russia while we do see a topical divide on Dimension 1.

# create an LSA space; return its truncated representation in the low-rank space

tmod <- textmodel\_lsa(dfmat[1:10, ])

# matrix in low\_rank LSA space

tmod$matrix\_low\_rank[, 1:5]

presid particular pleasur convey solemn

SYR\_25\_1970.txt 9 3.000000e+00 1.000000e+00 1.000000e+00 1.000000e+00

LBN\_36\_1981.txt 8 6.000000e+00 3.574198e-15 4.361095e-15 1.000000e+00

BLZ\_61\_2006.txt 1 2.319256e-13 -3.872892e-15 1.630640e-15 3.945845e-15

IDN\_58\_2003.txt 1 4.000000e+00 5.224885e-13 1.000000e+00 4.128642e-15

BOL\_62\_2007.txt 16 1.000000e+00 9.374921e-13 -1.542308e-13 1.865500e-14

ESP\_59\_2004.txt 1 1.000000e+00 1.096398e-12 1.000000e+00 -4.602221e-14

BEL\_46\_1991.txt 3 3.000000e+00 1.059028e-12 -1.910607e-13 -2.341183e-14

MLT\_56\_2001.txt 2 -2.442491e-14 9.686462e-13 1.000000e+00 -3.266137e-14

NAM\_53\_1998.txt 6 3.000000e+00 1.407991e-12 3.701761e-13 6.992887e-15

MWI\_60\_2005.txt 3 1.384795e-13 5.593955e-13 -1.283140e-13 -1.853978e-14

We now fold the queries into the space generated by dfmat[1:10,] and return its truncated versions of its representation in the new low-rank space.

pred <- predict(tmod, newdata = dfmat[1:10, ])

pred$docs\_newspace

4 x 10 Matrix of class "dgeMatrix"

[,1] [,2] [,3] [,4] [,5] [,6] [,7]

PAK\_46\_1991.txt -0.3854266 -0.04484373 0.1220358 -0.1098016 0.13608271 0.20572826 0.06882769

PRY\_73\_2018.txt -0.2258879 -0.10037933 0.1089027 -0.1457731 -0.01366223 0.06686449 -0.05633359

ITA\_37\_1982.txt -0.4558678 -0.06585580 0.1772039 -0.1827881 0.11242120 0.13485329 -0.05808548

KAZ\_61\_2006.txt -0.2305213 -0.05878778 0.1231484 -0.2164697 0.18169974 0.08266246 -0.04646643

[,8] [,9] [,10]

PAK\_46\_1991.txt 0.07593185 -0.12207856 -0.0150233

PRY\_73\_2018.txt -0.08488202 -0.06309237 -0.1158875

ITA\_37\_1982.txt 0.12460691 -0.06632978 -0.1762249

KAZ\_61\_2006.txt 0.08702912 -0.07348411 -0.1685055

**Unknown categories: Unsupervised machine learning – Latent Dirichlet Allocation (LDA)**

Both Latent Dirichlet Allocation (LDA) and Structural Topic Modeling (STM) belong to **topic modelling**. Topic models find patterns of words that appear together and group them into topics. The researcher decides on the number of topics and the algorithms then discover the main topics of the texts without prior information, training sets or human annotations.

It is a model that describes how the documents in a dataset were created. We assign an arbitrary number of topics (K) where each topic is a distribution over a fixed vocabulary. Each document is considered as a collection of words, one for each of K topics. It also follows the “bag of words” approach that considers each word in a document separately.

To calculate the LDA models, we need to load the package topicmodels. If you use this package for the first time on your machine, you need to execute a specific sequence of commands, detailed in the code below.

Code: Installing the package topicmodels

# 1) Install GSL

# We first need to make sure that GSL (e.g. 'brew install gsl' in the terminal) is installed on our machine

# 2) Install gsl

# Then we proceed and choose either of the following commands:

# A)

install.packages("gsl")

# or B)

install.packages(

"<https://cran.rstudio.com/src/contrib/gsl_2.1-6.tar.gz>",

repos = NULL,

method = "libcurl"

)

# 3) Load the gsl package

library(gsl)

# 4) Install topicmodels

# Here we choose again either of the following commands:

# A)

install.packages("topicmodels")

# or B)

install.packages(

"<https://cran.r-project.org/src/contrib/topicmodels_0.2-8.tar.gz>",

repos = NULL,

method = "libcurl"

)

# 5) Load the topicmodels package

library(topicmodels)

# Now you are all set for the following models.

For the LDA, we again first trim our DFM. As above, the command dfm\_trim trimms the text. This allows us to filter words that appear less than 7.5% and more than 90%.

mydfm.un.trim <-

dfm\_trim(

mydfm,

min\_docfreq = 0.075,

# min 7.5%

max\_docfreq = 0.90,

# max 90%

docfreq\_type = "prop"

)

We then assign an arbitrary topic number and convert the trimmed DFM to a topicmodels object.

# Assign an arbitrary number of topics

topic.count <- 15

# Convert the trimmed DFM to a topicmodels object

dfm2topicmodels <- convert(mydfm.un.trim, to = "topicmodels")

Eventually, we can calculate the LDA model with quanteda’s LDA() command.

lda.model <- LDA(dfm2topicmodels, topic.count)

Output for lda.model

lda.model

A LDA\_VEM topic model with 15 topics.

as.data.frame(terms(lda.model, 6))

| **topic1** | **topic2** | **topic3** | **topic4** | **topic5** | **topic6** | **topic7** | **topic8** | **topic9** | **topic10** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| nuclear | cooper | problem | trade | america | arab | arab | global | oper | war |
| weapon | council | interest | problem | american | small | palestinian | cooper | negoti | today |
| republ | reform | hope | product | latin | nuclear | israel | social | conflict | now |
| soviet | global | possibl | per | respect | issu | territori | order | problem | power |
| relat | effect | now | economi | law | pacif | isra | futur | confer | live |
| disarma | process | even | increas | principl | weapon | resolut | common | south | mani |

| **topic11** | **topic12** | **topic13** | **topic14** | **topic15** |
| --- | --- | --- | --- | --- |
| global | terror | african | deleg | africa |
| sustain | council | africa | session | south |
| climat | terrorist | republ | problem | independ |
| chang | iraq | conflict | concern | african |
| challeng | resolut | democrat | great | namibia |
| goal | law | elect | republ | struggl |

How similar are the fifteen topics? This question is particularly interesting because it allows us to (possibly) cluster homogeneous topics. To get a better idea of our LDA model and about the similarity among the different topics, we can plot our results using the following chunck of code. dist and hclust are standard R commands that allow us to calculate the similarity.

lda.similarity <- as.data.frame(lda.model@beta) %>%

scale() %>%

dist(method = "euclidean") %>%

hclust(method = "ward.D2")

par(mar = c(0, 4, 4, 2))

plot(lda.similarity,

main = "LDA topic similarity by features",

xlab = "",

sub = "")

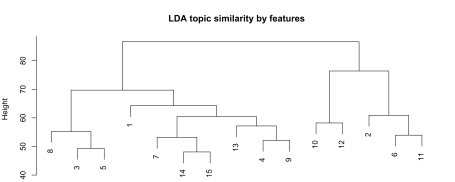


Figure 7: LDA topic similarity by features

The plot is called dendogram and visualizes a hierarchial clustering. The x-axis gives you the topics and the clusters of these topics. Put differently, it gives you information on the smilarity of the topics. On the y-axis, we see the dissmilarity (or distance) between our fifteen topics.

**Unknown categories: Unsupervised machine learning – Structural topic models (STM)**

We again trim our dfm with the command dfm\_trim.

mydfm.un.trim <-

dfm\_trim(

mydfm,

min\_docfreq = 0.075,

# min 7.5%

max\_docfreq = 0.90,

# max 90%

docfreq\_type = "prop"

)

We then assign the number of topics arbitrarily.

topic.count <- 25 # Assigns the number of topics

And eventually convert the DFM (with convert()) and calculate the STM (with stm()).

# Calculate the STM

dfm2stm <- convert(mydfm.un.trim, to = "stm")

model.stm <- stm(

dfm2stm$documents,

dfm2stm$vocab,

K = topic.count,

data = dfm2stm$meta,

init.type = "Spectral"

)

To get a first insight, we print the terms that appear in each topic.

as.data.frame(t(labelTopics(model.stm, n = 10)$prob))

| **V1** | **V2** | **V3** | **V4** | **V5** | **V6** | **V7** | **V8** | **V9** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| council | lebanon | soviet | global | african | nuclear | european | trade | african |
| reform | arab | union | sustain | africa | weapon | europ | economi | situat |
| effect | resolut | relat | challeng | republ | treati | cooper | product | guinea |
| activ | problem | militari | chang | democrat | disarma | union | industri | africa |
| cooper | territori | forc | climat | millennium | arm | conflict | resourc | particular |
| strengthen | palestinian | republ | goal | commit | test | process | increas | hope |
| convent | withdraw | arm | commit | poverti | non | stabil | market | concern |
| role | principl | war | agenda | particular | prolifer | contribut | price | solut |
| confer | solut | socialist | respons | like | pakistan | bosnia | global | session |
| contribut | war | polici | address | summit | india | respect | financi | problem |

| **V10** | **V11** | **V12** | **V13** | **V14** | **V15** | **V16** | **V17** | **V18** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| council | problem | cooper | republ | power | africa | order | problem | israel |
| iraq | now | stabil | korea | great | south | principl | session | arab |
| resolut | oper | dialogu | democrat | viet | debt | social | oper | palestinian |
| law | negoti | terror | china | deleg | deleg | univers | confer | isra |
| aggress | hope | sudan | south | coloni | hope | cultur | solut | palestin |
| charter | agreement | call | korean | territori | process | today | resolut | east |
| violat | mani | process | asia | nam | environ | life | cyprus | middl |
| iraqi | last | promot | north | republ | confer | respect | concern | resolut |
| islam | way | commit | east | sea | welcom | societi | negoti | occupi |
| iran | possibl | issu | cooper | charter | conflict | becom | disarma | territori |

| **V19** | **V20** | **V21** | **V22** | **V23** | **V24** | **V25** |
| --- | --- | --- | --- | --- | --- | --- |
| island | today | per | terror | america | conflict | africa |
| small | war | cent | afghanistan | american | refuge | south |
| pacif | live | social | afghan | latin | war | independ |
| caribbean | want | educ | cooper | central | somalia | african |
| chang | terror | poverti | terrorist | respect | assist | struggl |
| ocean | mani | health | problem | social | humanitarian | regim |
| issu | let | drug | global | process | mani | namibia |
| sea | now | democraci | asia | solut | elect | apartheid |
| climat | everi | programm | central | possibl | situat | coloni |
| call | know | million | issu | express | forc | racist |

The following plot allows us to intuitively get information on the share of the different topics at the overall corpus.

plot(

model.stm,

type = "summary",

text.cex = 0.5,

main = "STM topic shares",

xlab = "Share estimation"

)

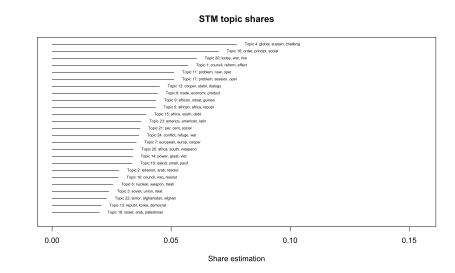


Figure 8: STM topic shares

Using the package stm, we can now visualize the different words of a topic with a wordcloud. Since topic 4 has the highest share, we use it for the next visualization. The location of the words is randomized and changes each time we plot the wordcloud while the size of the words is relative to their frequency and remains the same.

stm::cloud(model.stm,

topic = 4,

scale = c(2.25, .5))

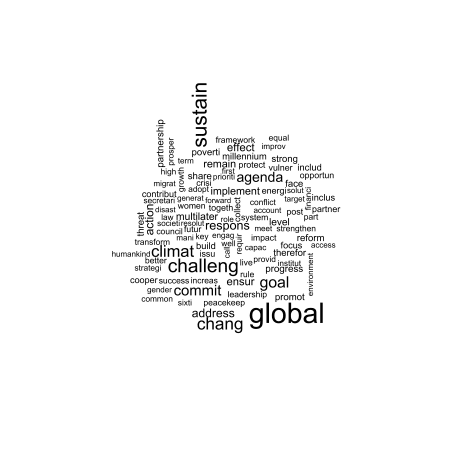


Figure 9: Wordcloud with stm

If we want, we can also put several different topics in visually perspective using the following lines of code:

plot(model.stm,

type = "perspectives",

topics = c(4, 5),

main = "Putting two different topics in perspective")



Figure 10: Wordcloud using stm – Perspective plots