SG1022 Seminar 4: Text analysis

Christopher Gandrud

2016

Objectives

- Selecting and Gathering a corpus of text data
- Importing the text data into R
- Preprocessing the data
- Creating word frequency plots and word clouds

Pop Quiz

- What is a **corpus**?
- What does it mean to **preprocess** texts and why is this important?
- What kinds of data can you generate from text analysis?
- What is a **stopword**?
- What is a **word stem**? Why would we use them?

Packages

You will need have the following packages installed and loaded:

- dplyr
- tm
- SnowballC
- wordcloud
- RColorBrewer

Selecting your data

Imagine that we want to compare the German Chancellor's discussion of refugees before and after the 2015-16 Cologne New Year's Eve Incident

To do this we might gather a corpus of press releases from the German Chancellor's website.

Gather the data

Note: The most efficient way to gather text data from the internet is with *automated web scraping*.

We mostly not cover this tool in this class. Instead we will hand download the texts. This is *laborious* so if you are interested in web scraping see me for more details.

That being said: at the bottom of this seminar sheet, there are instructions for how to scrape tweets. Take a look if you are interested.

Directory set up:

First create the folders that you will save the data into. We want to create one common directory called `press_releases`. Then create two sub-folders: `refugees_pre` and `refugees_post`. Your file tree should look like this:

```
press_releases
|_
    refugees_pre
    refugees_post
```

Copy-paste (1)

Now go to the following website, copy the speech text, and paste the text into a new **text file** called `chancellory_refugees_2015_09_09.txt`. Save this file into the `refugees_pre` folder.

- http://www.bundeskanzlerin.de/Content/EN/Artikel/2015/09_en/2015-09-09-generaldebatte-merkel_en.html

Note: To create a new text file in RStudio Click `New File > Text File`.

It is important to give the **files meaningful names**. Here we gave a title based on the subject and date. This will help you identify them later.

In this example we only have one text in the `refugees_pre` folder. But you can include many more.

Copy-paste (2)

Do the same for the transcript at the following URL. Save the text into a file called `chancellory_refugees_2016_01_12.txt` in the `refugees_post` folder.

- http://www.bundeskanzlerin.de/Content/EN/Artikel/2016/01_en/2016-01-11-konsequenzen-nach-koeln_en.html

Load the texts into R

To load the texts into R, first set your working directory to be the `press_releases` folder with the `setwd` function.

See Week 3's seminar for how to set the working directory.

Load the texts into R.

We will use the `tm` (text mining) package to load the data and pre-process it. To load the data use the `Corpus` function. We need to do this twice, once for the press releases before the Cologne Attacks and once for the press releases after:

```r
library(tm)

# Press releases before attacks
pre <- Corpus(DirSource('refugees_pre/'))

# Press releases after attacks
post <-  Corpus(DirSource('refugees_post/'))
```

You should see two new `list` class objects in your **Environment** pane, one called `pre` and the other called `post`.

Note on `Corpus(DirSource())`

`Corpus(DirSource())` will **load all** of the `.txt` text files into R that are in the specified file. So, if we had multiple press releases in the `refugees_pre` directory, then it would load all of them into R at the same time.

Preprocessing

Imagine that we want to preprocess the texts by:

- converting all of the letters to lower case
- removing extra spaces between words
- removing punctuation
- removing numbers
- removing stopwords
- reducing the number of unique words with stemming

Note: The order of these steps matters.

To do all of these things we will use the `tm_map` function.

Lower case

It is often a good idea to start your preprocessing by converting all of your letters to lower cases. This way, for example, the words `A` and `a` are treated as the same word.

To convert all letters to lower case use:

```r
pre <- tm_map(pre, content_transformer(tolower))
```

In the **Environment** pane, checkout the pre object to make sure that this worked as intended. Do this for all preprocessing steps.

Remove extra white space

Some times there are extra spaces between words (you only need one). To remove these use:

```
pre <- tm_map(pre, stripWhitespace)
```

Remove punctuation

In this text analysis, where we are just looking at word frequencies, we are assuming that punctuation is unimportant. To remove all punctuation use:

```
pre <- tm_map(pre, removePunctuation)
```

Remove numbers

We are also assuming that numbers are unimportant. To remove all numbers use:

```
pre <- tm_map(pre, removeNumbers)
```

Remove stopwords

In this analysis we are also going to assume that linguistic function words like `a` and `the` are unimportant. These are our stopwords. To remove stopwords use:

```
pre <- tm_map(pre, removeWords, stopwords(kind = 'english'))
```

Note that we specified the stopword `kind = 'english'`. If we were using texts in other languages, we should use a stopword list specific to that language.

Stemming words

We have reduced a lot of unwanted complexity in the corpus (e.g. all letters are now in lower case, there are no numbers). Now we can effectively combine different versions of the same word by stemming the corpus:

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

Remove corpus specific words

There may also be other words that are specific to the corpus that we also want to remove. Maybe because these are commonly used words in the Corpus that do not provide meaning for our research question. For example, we know that all of the texts are related to the German Chancellor (Angela Merkel) and they are in some way about Germany.

We can effectively create a custom stopword list and remove these words as well:

```
pre <- tm_map(pre, removeWords, words = c('angela', 'merkel'))
```

Note we do this **after stemming** to reduce the number of variations of the words to explicitly include in our custom stopword list (e.g. only `merkel`, rather than also `merkels`, etc.)

Preprocessing all together

Rather than running each of these preprocessing steps individually, we can combine them together with the `%>%` (pipe) function from the `dplyr` package:

```r
library(dplyr)

post <- Corpus(DirSource('refugees_post/')) %>%
    tm_map(content_transformer(tolower)) %>%
    tm_map(stripWhitespace) %>%
    tm_map(removePunctuation) %>%
    tm_map(removeNumbers) %>%
    tm_map(removeWords,
           stopwords(kind = 'english')) %>%
    tm_map(stemDocument) %>%
    tm_map(removeWords,
            words = c('angela', 'merkel'))
```

Wordclouds (1)

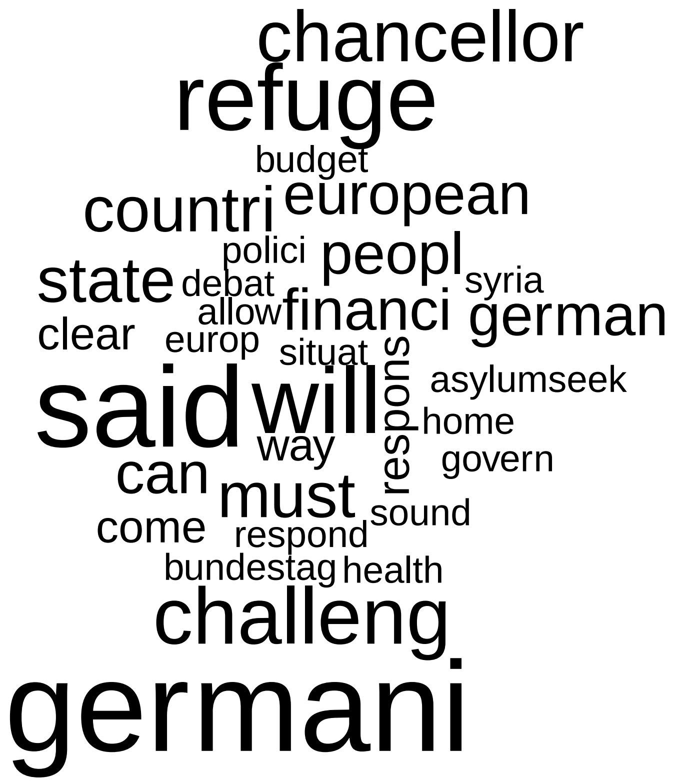
Now that you have your preprocessed word cloud, you can begin exploring it. First, lets make a **wordcloud** with the `wordcloud` function from the wordcloud package.

```r
library(wordcloud)
```

The size of the words gives us a sense of their **relative frequency** in the corpus. Note the wordclouds do not show all of the words, just more common ones.

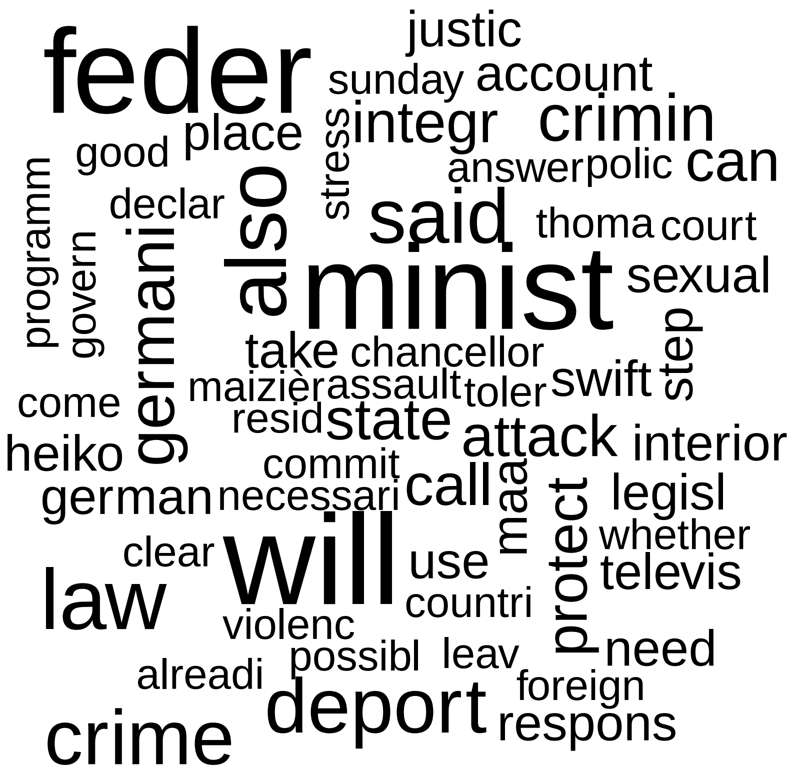
Wordclouds (2)

```r
wordcloud(pre)
```



Wordclouds (3)

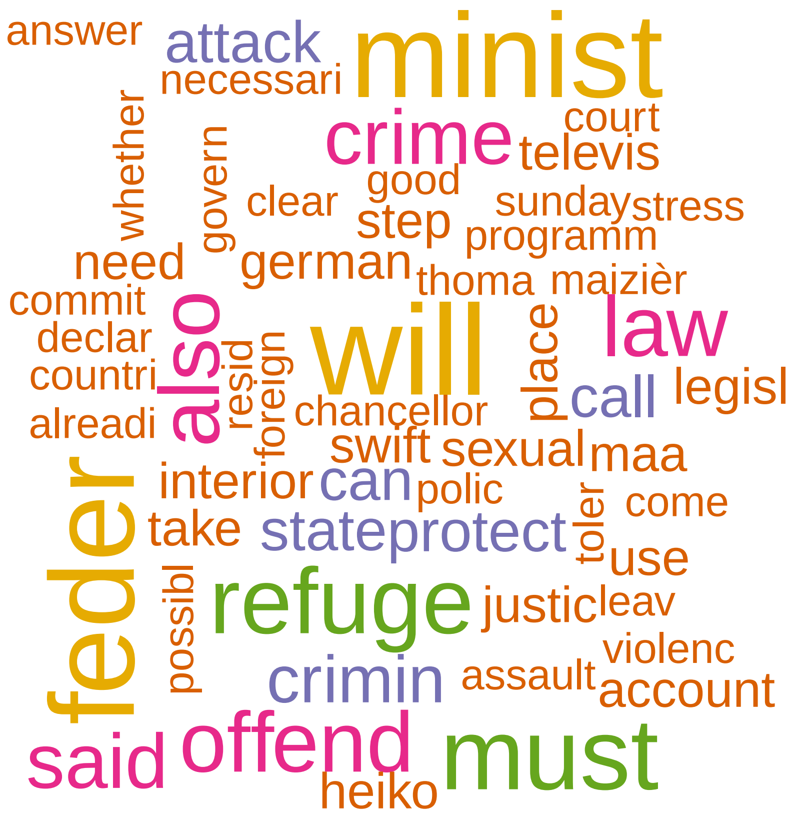
```
wordcloud(post)
```



Wordclouds (4)

Add colours with the `color` argument:

```r
wordcloud(post, color = brewer.pal(6, "Dark2"))
```



Full source

You can find a complete source code file for the tasks we just did at: https://raw.githubusercontent.com/christophergandrud/city_sg1022/master/seminars/week_4/refugee_word_cloud.R. **Use this as an example of how your source code files should look.**

Advanced

For more advanced analysis techniques, see: https://rstudio-pubs-static.s3.amazonaws.com/31867_8236987cf0a8444e962ccd2aec46d9c3.html

A lot of these tools would be great to use in your group project. Please see me if you have questions.

Your turn (1)

Re-preprocess our corpus in different ways (for example, don't convert all letters to lower case).

- How does the word cloud change?
- How do the conclusions you draw change?

Your turn (2)

Now it is your turn. With a partner:

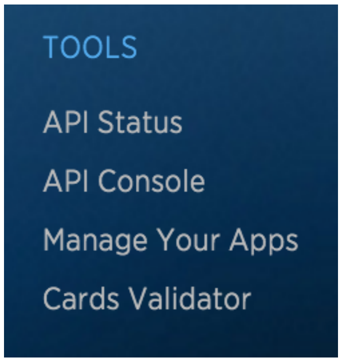
- **Think of a corpus** of texts that you want to analyse (e.g. newspaper articles, treaties, speeches, press releases). If you want to, see the instructions for scraping tweets below
- **Gather** a small sample of these texts. Save them as text files in a new directory.
- **Preprocess** the texts.
- **Create** a wordcloud from this corpus.
- What are the the **commonly discussed topics** in this corpus?
- **Validate** your finding, by reading a few of the texts. Do your conclusions about the main topics from the wordclouds match your conclusions when you read the whole document? If not, why not?

Advanced: Scraping Tweets (1)

We can use the twitteR package to scrape Twitter. **Note:** to do this you need a Twitter account.

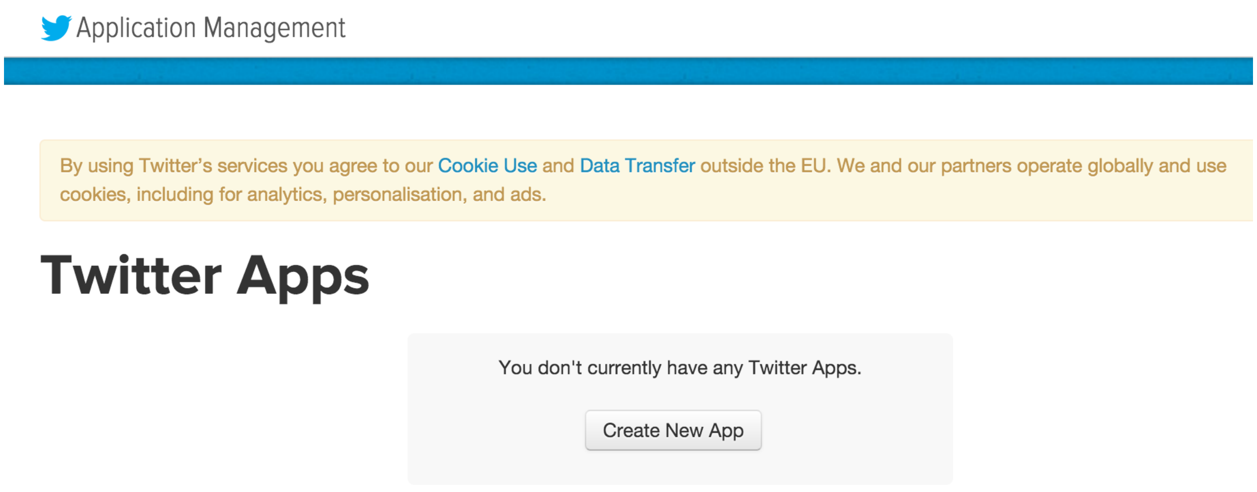
First you need to get authenticated. Go to: https://dev.twitter.com/. Make sure you are signed in with your Twitter account.

Scroll to the bottom of the page. Under `Tools`, click on `Manage Your Apps`.



Advanced: Scraping Tweets (2)

Click `Create New App`:


Application Management

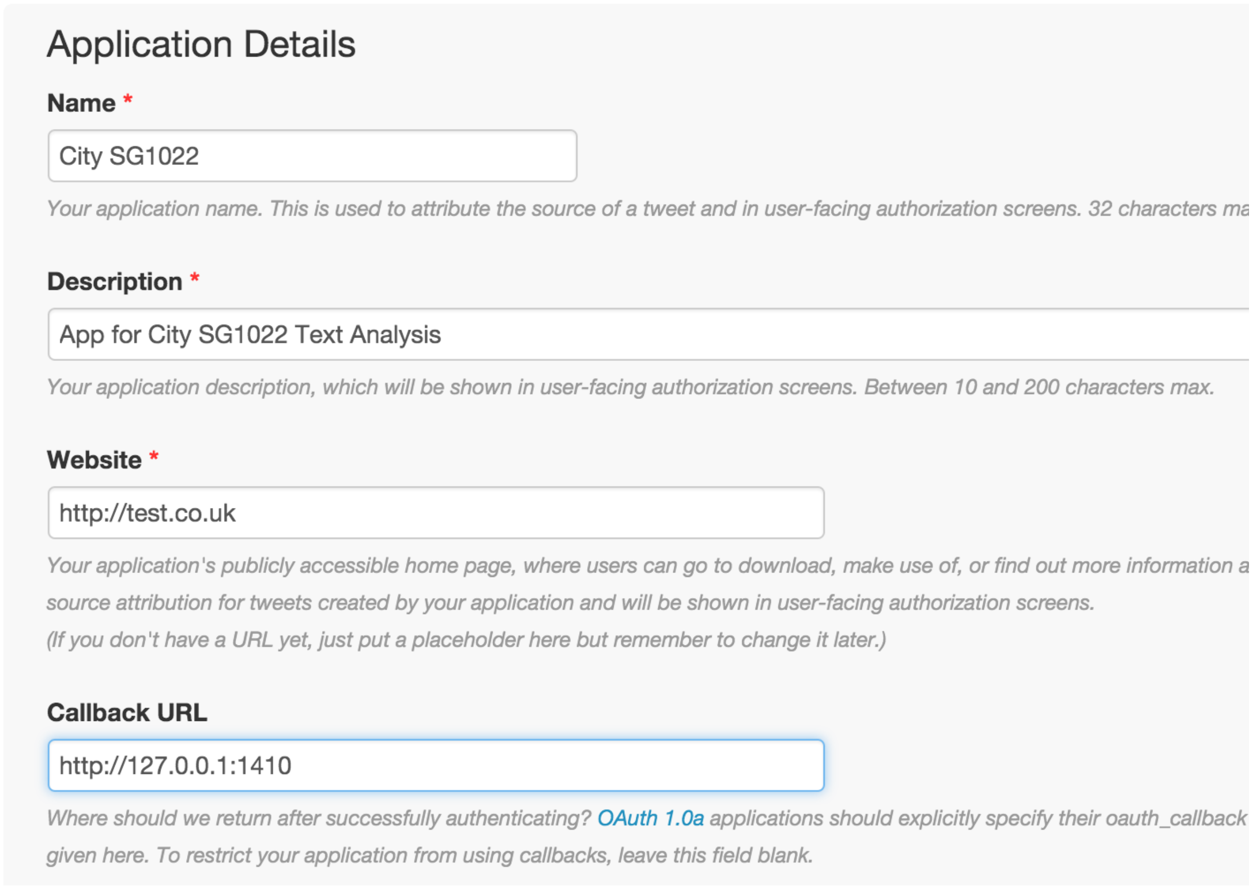
By using Twitter's services you agree to our Cookie Use and Data Transfer outside the EU. We and our partners operate globally and use cookies, including for analytics, personalisation, and ads.

Twitter Apps

You don't currently have any Twitter Apps.

Advanced: Scraping Tweets (3)

Fill in the App details similar to below. Then agree to the Developer Agreement.


Application Details

Name *

City SG1022

Your application name. This is used to attribute the source of a tweet and in user-facing authorization screens. 32 characters ma

Description *

App for City SG1022 Text Analysis

Your application description, which will be shown in user-facing authorization screens. Between 10 and 200 characters max.

Website *

http://test.co.uk

Your application's publicly accessible home page, where users can go to download, make use of, or find out more information a source attribution for tweets created by your application and will be shown in user-facing authorization screens. (If you don't have a URL yet, just put a placeholder here but remember to change it later.)

Callback URL

http://127.0.0.1:1410

Where should we return after successfully authenticating? OAuth 1.0a applications should explicitly specify their oauth_callback given here. To restrict your application from using callbacks, leave this field blank.

Advanced: Scraping Tweets (4)

Click on the `Keys and Access Tokens` tab. You should see two long strings of letters and numbers. One called 'Consumer Key (API Key)'. The other called 'Consumer Secret (API Secret)'.

Load `twitteR` and enter this information into the `setup_twitter_oauth` function. For example:

```r
library(twitteR)

setup_twitter_oauth("API key", "API secret")
```

Note: you should keep your Secret API Key secrete.

Advanced: Scraping Tweets (5)

Now that you have your App set up, you can begin downloading tweets. For example, to download about 100 tweets from `@realDonaldTrump` use:

```r
trump <- userTimeline('realDonaldTrump', n = 100)
```

Advanced: Scraping Tweets (6)

All of the text from these tweets is in the `trump` object. We need to do one small step to extract them into a character vector:

```r
trump_text = sapply(trump, function(x) x$getText())
```

Advanced: Scraping Tweets (7)

We now convert `trump_text` into a `Corpus` (using `VectorSource` rather than `DirSource`) and preprocess the data as before:

```r
trump_corpus <- Corpus(VectorSource(trump_text)) %>%
    tm_map(content_transformer(tolower)) %>%
    tm_map(stripWhitespace) %>%
    tm_map(removePunctuation) %>%
    tm_map(removeNumbers) %>%
    tm_map(removeWords,
           stopwords(kind = 'english')) %>%
    tm_map(stemDocument)
```