BAN432
Applied Textual Data Analysis for Business and Finance

Preprocessing and cleaning, part II

Christian Langerfeld and Maximilian Rohrer

21 September, 2022

Files and packages for today's lecture

For today's lecture please make sure you have these packages installed:

- `udpipe`
- `readr`
- `tidytext`
- `dplyr`
- `wordcloud`
- `tm`

Files from Canvas:

- `data_for_lecture_06.Rdata`
- `text_file_in_iso-latin-1.txt`

Today's lecture

(1) encoding
(2) keyword/term extraction
(3) POS tagging
(4) tm-package
    - Document-Term-Matrix
    - cleaning tasks

1. Encoding

Encoding

- computers only understand numbers
- each (text) string is represented as a sequence of coded characters
- example for a string represented in hexadecimal notation (can you guess what it is?)

```
54 6F 64 61 79 20 69 73 20 57 65 64 6E 65 73 64 61 79
```

Encoding

- computers only understand numbers
- each (text) string is represented as a sequence of coded characters
- example for a string represented in hexadecimal notation (can you guess what it is?)

```
54 6F 64 61 79 20 69 73 20 57 65 64 6E 65 73 64 61 79
```

54	6F	64	61	79	20	69	73	20	57	65	64	6E	65	73	64	61	79
T	o	d	a	y		i	s		W	e	d	n	e	s	d	a	y

Encoding – The hexadecimal system

Space character in encodings that are based on the hexadecimal system:

ASCII	20
Unicode	U+0020
ISO 8859-1	20

- bits and bytes:
  - one byte = grouping of 8 bits
  - e.g. 0 0 1 0 0 0 0 0
  - 8 bits can make 256 different patterns ($2^8 = 256$)
  - one byte can store one character, e.g. 'A' or 'x' or '$'
  - every byte is mapped to a hexadecimal number: A=41, x=78 , n=6E

Encoding – plain text file in a hex-editor

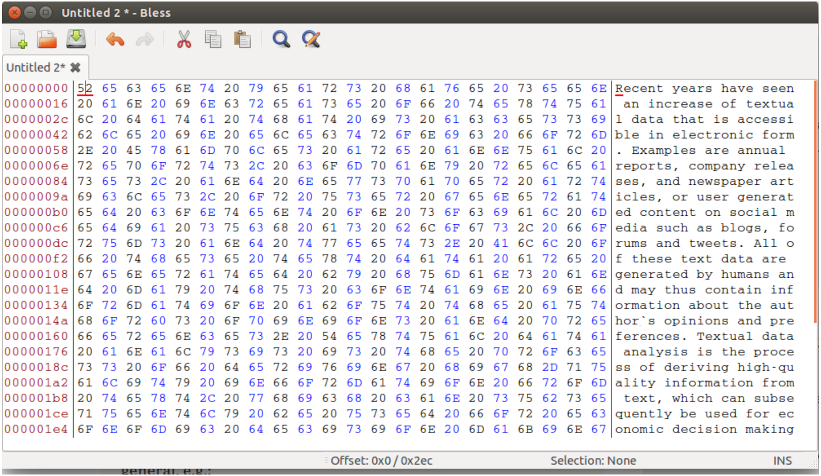

Figure 1: Screenshot of a hexadecimal editor

Encoding – Coding conventions (charsets)

- ASCII:
  - uses only 128 characters (out of 256 possible)
  - only for English
  - not possible to represent å, ø ,æ , ü, ä, ö, etc.
- ISO-LATIN-1:
  - uses the full range of 256 characters
  - with special characters from Western European languages
- UTF-8:
  - uses up to 4 bytes to represent one character
  - about 130,000 characters in the current version
  - UTF-8 is used by more than 97% of all websites

Encoding – UTF-8 and other encodings

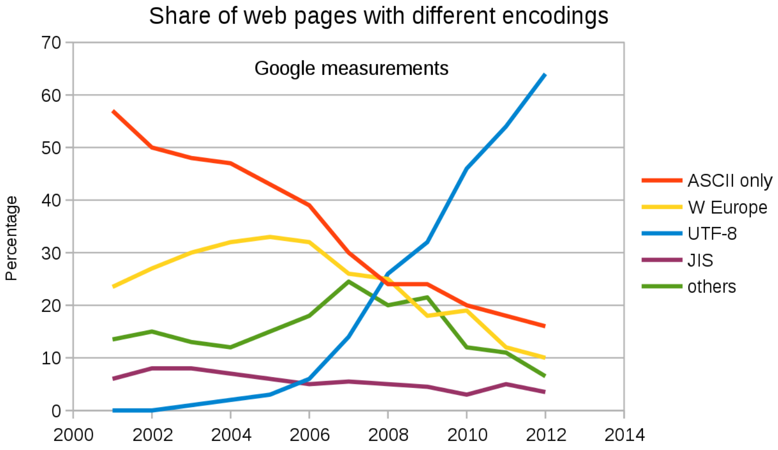

Figure 2: Usage of different encodings on the web as reported by Google

Encoding – UTF-8 and other encodings

- the first 128 characters of UTF-8 are encoded with 1 byte and are equal to ASCII
- in text mining, problems with character encoding are frequent
- a function often expects to get input in encoding A (usually UTF-8) and it gets text data in encoding B

Encoding – UTF-8 and other encodings

- UTF-8 covers characters from all writing systems in the world, but you will discover error messages
- "xy is not av valid Unicode character"
- often emoticons, e.g. in tweets
- if the input text data is not in UTF-8 encoding, it has to be converted
- for text mining we usually want to work with data in UTF-8 (exception: ASCII, which is valid UTF-8)
- *R* provides a function to change character encoding: `iconv`

Encoding – convert encoding with iconv

```
# load a sample txt file in a vector
txt <- readLines("data/text_file_in_iso-latin-1.txt")
txt
```

```
## [1] "This is a test file with strange characters: \xe5\xe6\xf8"
```

```
# convert to UTF-8 encoding
iconv(txt, f = "LATIN1", to = "UTF-8")
```

```
## [1] "This is a test file with strange characters: åæø"
```

Encoding

- to check the encoding of a file:
  - on Unix (Mac and Linux) `file -i filename` in a terminal window (uppercase -I on OSX)
  - on Windows: open file in Notepad and "Save as"
  - function `guess_encoding()` from package `readr`

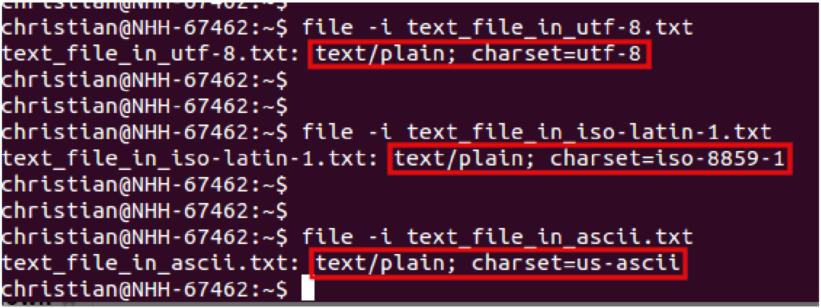

Figure 3: output of the file -i command in a Unix shell

2. Keyword extraction

Keyword/term extraction

- how to find words that are "typical" for a domain or text type
- e.g. the type "annual reports"

Task 1:

A simplified approach:

(1) compile a frequency list form a specialized **study corpus** (e.g. a corpus of 10-Ks)
(2) compile a frequency list from a large **reference corpus** of general language
(3) compare the frequencies of the words in the two lists
(4) term candidates for the (specialized) corpus could be:
    (a) words that occur only in the specialized corpus
    (b) words that occur more often in the specialized corpus than could be expected based on the frequencies in the reference corpus

We code an example for (4a) together in class

3. Part of speech tagging

POS tagging

- a part-of-speech tagger takes as input a text string
- the output is the same text, but each token is annotated with a part-of-speech (or word class) tag

Input:

```
Recent years have seen an increase of textual data
that is accessible in electronic form .
```

Output:

text	pos
Recent	JJ
years	NNS
have	VBP
seen	VBN
an	DT
increase	NN
of	IN
textual	JJ
data	NNS
that	WDT
is	VBZ

POS tagging

- today's' POS tagger report an accuracy of 93-95%
- the tagger looks up a word in a word list and finds possible tags:

```
I opened a can
```

- unambiguous case: there is just one possible tag:
  - `I|PRP opened|VBD a|DT`
- ambiguous case:
  - `can` `NN` `MD` (noun or modal)
  - use the context
  - if `can` is preceded by a determiner, it is much more likely to be a noun than a modal verb
- unknown words:
  - look at the ending or the surrounding tags and calculate the most likely tag

POS tagging in R

- several packages that provide functions for POS-tagging, e.g. `koRpus`, `coreNLP` or `openNLP`, but they require that external taggers are installed on the computer

- we use the package `udpipe`:
  - before pos-tagging a document, a language model has to be downloaded
  - there are models available for many languages, also Norwegian

POS tagging with TreeTagger in R

```
require(udpipe)
text <- "Recent years have seen an increase of textual data."

# Load a language model (the file must be downloaded only once:
# udpipe_download_model("english")
tagger <- udpipe_load_model("english-ewt-ud-2.5-191206.udpipe")

# Tag the text
tagged <- udpipe_annotate(tagger, x=text) %>%
  as_tibble() %>%
  select(doc_id, token, lemma, xpos, upos)
head(tagged)
```

```
## # A tibble: 6 x 5
##   doc_id token    lemma    xpos  upos
##   <chr>  <chr>    <chr>    <chr> <chr>
## 1 doc1   Recent   recent   JJ    ADJ
## 2 doc1   years    year     NNS   NOUN
## 3 doc1   have     have     VBP   AUX
## 4 doc1   seen     see      VBN   VERB
## 5 doc1   an       a        DT    DET
## 6 doc1   increase increase NN    NOUN
```

4. Document-Term-Matrices

Document Term Matrix

- Document Term Matrices are frequently used for text mining tasks
- a way of representing word frequencies in a corpus
- rows correspond to documents
- columns correspond to terms

	a	an	art	belt	best	term i
doc01	12	34	0	1	0	
doc02	0	2	0	0	0	
doc03	2	0	0	0	2	
doc04	0	3	1	0	0	
doc05	6	1	0	0	0	
doc n	...	...	...	...	...	

Document Term Matrix as simple triplet matrix

- a DTM is usually a sparse matrix:
  - Zipf's law states that there are a couple of words that occur frequently in natural language
  - on the other hand the majority of words are very infrequent
  - consider a DTM of 100 documents:
    - there will be some few words that occur i all documents
    - the majority of words will just occur in some of the documents
    - the resulting DTM is "sparse", meaning there are many zeros
- in *R* sparse matrices are stored by their coordinates
- three values are used to store a DTM:
  - the row number of all non-zero entries
  - the column number of all non-zero entries
  - the number of fields that are $= 0$

Creating DTMs

Steps to construct a Document Term Matrix with the `tm` package:

(1) load the package `tm`
(2) read text data
(3) compile a corpus of texts
(4) create a DTM

Creating DTMs: steps (1) and (2)

(1) load the tm package:

```
require(tm)
```

(2) read the text data

- load data from a vector

```
text.source <- VectorSource(vector.object)
```

- load files from a directory

```
text.source <- DirSource("directory/")
```

Creating DTMs: step (3)

(3) compile a corpus of texts

```
corpus <- Corpus(text.source)
```

Creating DTMs: step (4)

(4) Create a DTM

```
dtm <- DocumentTermMatrix(
  corpus,
  control = list( removePunctuation = T/F,
                  stopwords = T/F,
                  removeNumbers = T/F,
                  stemming = T/F,
                  stripWhitespace = T/F,
                  tolower = T/F,
                  wordLengths = c(min.nchar, max.nchar),
                  bounds = list( global = c(occurance.minimum.docs,
                                            occurance.maximum.docs) )
                  )
  )
```

Convert DTMs to matrices

- often is is necassary to convert a Document-Term-Matrix (or a Term-Document-Matrix) into a matrix or a data frame

```
dtm.matrix <- as.matrix(dtm)
dtm.df <- as.data.frame(dtm.matrix)
```

Exercise

TASK 3:

We try to create a Document Term Matrix together in class.

(1) the file `data_for_lecture_06.Rdata` that you loaded into R contains a vector with 499 business descriptions taken from 10-K forms
(2) you find the vector `section.1.business` in your environment

Summary

- encoding of text
  - we usually want UTF-8 encoding for the texts we work with
  - we can use `iconv()` to convert the encoding
- keyword extraction
  - compare (relative) frequencies of words in a specialized corpus with those in a general language corpus
  - terms in specialized language tend to have a higher frequency in the specialized corpus
- POS tagging
  - words are tagged with their word class (part of speech)
- tm-package and Document Term Matrix
  - read text data from a file into a corpus
  - construct a DTM, based on the corpus