Week 4: Textual Data

LSE MY472: Data for Data Scientists
https://lse-my472.github.io/

Autumn Term 2024

Ryan Hübert

Slides last updated: 30 September 2024

Introduction

- This week we will focus on processing textual data
- Most file formats we work with in this course (.csv, .xml, .json, etc.) use text to store data
- The quantitative analysis of textual data is highly relevant in social science research and beyond
- We will discuss some basic analyses, but for a full course see MY459 in Winter Term

Plan for today

- ➜ Character encoding
- ➜ Text search: Globs and regular expressions
- ➜ Elementary text analysis
- ➜ Coding

Character encoding

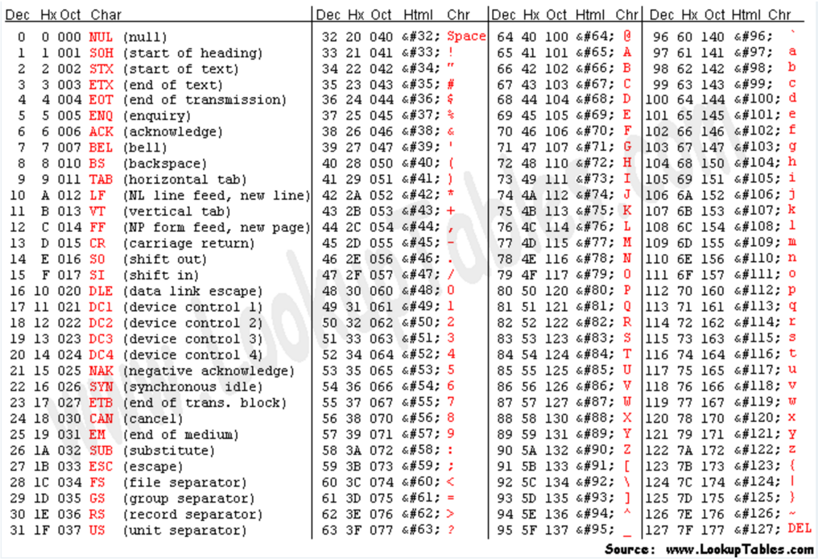
Revisited: Basic units of data

- Bits
  - Smallest unit of storage; a 0 or 1
  - With n bits, can store 2^n patterns
- Bytes
  - "eight bit encoding" represents characters through 8 bit, e.g. `A` represented as $65 = 01000001$
  - $8 \text{ bits} = 1 \text{ byte}$
  - Hence, 1 byte can store 256 patterns

Encoding

- A “character set” is a list of characters with associated numerical representations
- The unique numbers associated with characters are called “code points”
- ASCII: The original character set, uses just 7 bits (2^7), see https://en.wikipedia.org/wiki/ASCII
- ASCII was later extended, e.g. ISO-8859, using 8 bits (2^8)
- Unfortunately, encoding has became a mess of differing standards, see http://en.wikipedia.org/wiki/Character_encoding

ASCII



Dec	Hx	Oct	Char		Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	|
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

Potential encoding issues

(Wrongly) detected encoding:

- ➜ Encoding type/character set is not stored as metadata in plain text files
- ➜ So software guesses which encoding is used, which might go wrong
- ➜ Assuming the wrong encoding when reading in/parsing a text file leads to import errors and corrupted characters (Mojibake): Underlying bit sequences are translated into the wrong characters

Space:

- ➜ 8 bits are too few to store all known characters
- ➜ Encoding with 32 bits, however, would imply a lot of rarely used bits
- ➜ Those bits take up memory, implying unnecessarily large file sizes

Widely used character encoding today: Unicode

- Created by the Unicode Consortium
- Common Unicode encoding formats: **UTF-8** and **UTF-16** (Unicode transformation format)
- UTF-8 is a variable-width character encoding and by far the most frequent character encoding on the web today
- Variable amounts of bits are used for each character with the first byte/8 bits corresponding to ASCII
- Common characters therefore need less space, but system capable of storing vast amounts of character code points

UTF-8 details

Number of bytes	Byte 1	Byte 2	Byte 3	Byte 4
1	0xxxxxxx			
2	110xxxxx	10xxxxxx		
3	1110xxxx	10xxxxxx	10xxxxxx	
4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

https://en.wikipedia.org/wiki/UTF-8

Try it out: Create two .txt files, one containing a single line with the character *a*, the other one a single line with the character *ü*. Then check the sizes of both files in bytes which should be different if files are encoded in UTF-8.

Things to watch out for

- ➜ Many text production softwares (e.g. MS Office-based products) might still use proprietary character encoding formats, such as Windows-1252
- ➜ Windows tends to use UTF-16, while Unix-based platforms use UTF-8
- ➜ Text editors can be misleading: the client may display mojibake but the encoding might still be as intended
- ➜ Generally, no easy method of detecting encodings in basic text files

Some things to try with encoding issues

To determine the estimated character encoding of a file (note that this estimate might be incorrect)

- Linux, Unix, Mac: For example, `file -I filename.txt`, `file -I filename.json`, etc. in terminal
- Windows: For example, open with Notepad and check field in the lower right hand corner of the window

To change a file's encoding (see e.g. this Stack Overflow post)

- Linux, Unix, Mac: For example, `iconv -f ISO-8859-15 -t UTF-8 in.txt > out.txt` in terminal
- Windows: For example, open the text with Notepad, click "Save As", and choose a name and UTF-8 encoding. Alternatively, use PowerShell

Some things to try with encoding issues (in R)

In R, e.g. via `readr` (for more discussion, see R4DS)

- ➜ For a character vector `x`, obtain texts assuming a different encoding with `parse_character(x, locale = locale(encoding = "Latin1"))`
- ➜ Make guess about encoding with `guess_encoding(charToRaw(x))`

Globs and regular expressions

Globs

- Searching and counting specific words in texts is key for quantitative textual anaylsis
- Globs offer a simple and intuitive approach to search through text with wildcard characters
- Glob patterns originally used to search file and folder names

Globs: Exemplary syntax

Wildcard	Description	Examples	Exemplary matches
*	Any number (also zero) of characters	tax*, *tax*	taxation, overtaxed
?	Single character	??flation	inflation or deflation
[ab], [AB], [17], etc.	List of characters	module-[17].Rmd	module-1.Rmd or module-7.Rmd
[a-z], [A-Z], [0-9]	Range of characters	module-[A-Z].Rmd	module-A.Rmd or module-B.Rmd or module-C.Rmd . . .

Regular expressions

- Powerful and much more flexible tool to search (and replace) text
- Different syntax than globs
- Text editors (e.g. VS Code) can usually find and replace terms with regular expressions
- Can also be used in many programming languages, e.g. when counting or collecting certain keywords in text analysis
- In R, we can e.g. use `stringr` or `quanteda` to search for keywords with regular expressions
- Topic could fill lectures itself, we will cover some basics here

Sample text

```
Inflation in the Eurozone

2pm
2:30pm
2.15pm
2 15
11.30
22-30
5-15pm

Münster
Muenster
Munster

@
@JoeBiden
@KamalaHarris
```

Regular expressions: Syntax

- Regular expressions can consist of literal characters and metacharacters
- **Literal characters**: Usual text
- **Metacharacters**: ^ $ [] () {} * + . ? etc.
- When a meta character shall be treated as usual text in a search, escape it with (unless it is in a set []) \
- For example, searching **.** in regex notation will select any character, but searching \**.** will select the actual full stop character

Syntax: Specifying characters (1/2)

- `.`: Matches any character (also white spaces)
- `\d`: Matches any digit 0-9
- `\w`: Matches any character a-z, A-Z, 0-9, _
- `\s`: Matches white spaces
- Capitalised versions negate: `\S` matches everything that is not a white space etc.

Syntax: Specifying characters (2/2)

- `^`: Matches characters at the beginning of the line or string,
  - E.g. `^M` will select all capital m at the beginning of strings or lines
- `$`: Matches characters at the end of the line or string,
  - E.g. `m$` will select all lowercase m at the end of strings or lines
- `[]`: Character set, e.g. [a-zA-Z] selects single characters from the Latin alphabet in lower and upper case letters, [ai] selects characters that are “a” or “i”, [0-9] digits from 0 to 9
- `[\^ ]`: In brackets, `^` has a different meaning namely “not”, e.g. `[^a-z]` selects all characters that are not from the lower case alphabet

Syntax: Selecting sequences of characters

In order to select whole words, we need to add quantifiers to individual characters:

- `*`: Zero or more times, e.g. in[a-z]* will select *in* and also *inflation* in a search;
  - We could use `.*` to represent all characters and white spaces
- `+`: One or more times, e.g. in[a-z]+ will not select *in* but *inflation*
- `?`: Denotes optional characters, e.g. re?ally will select *really* and *rally*
- `{}`: Specifies lengths of sequences, e.g. `\d{3}` selects sequences of 3 digits, `\w{3,4}` selects sequences between 3 and 4 general characters, and `\d{3,}` selects sequences of at least 3 digits

Syntax: Boolean or and capturing groups

- `|`: Boolean or
- `()`: Capturing groups, e.g. `(ue?|ü)` selects u, ue, and ü.
    - This means that when searching text, the regular expression `M(ue?|ü)nster` will find *Münster*, *Muenster*, and *Munster*.
    - The captured groups can also be referenced with integer counts, which can be very helpful when replacing text
- https://en.wikipedia.org/wiki/Regular_expression

Regular expressions in R and beyond

- Regular expressions are used for flexible word searches in the `quanteda` package
- `stringr` is another good package for strings that uses regular expressions:
  - `str_view()` show results of searches with regular expressions
  - `str_extract()` allows you to extract keywords from strings through regular expressions
  - `str_replace()` finds and replaces regular expressions
- Detailed discussion of strings and regular expressions with `stringr` in R here
- R markdown with many examples here

More resources

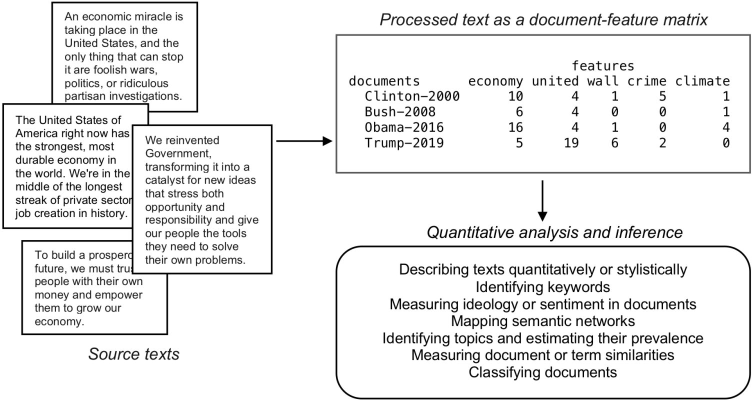
- Some good general discussions of the topic also on Youtube, e.g. here
- In depth treatment of regular expression (programming language independent): *Mastering Regular Expressions* by Jeffrey E. F. Fried
- A good place to test regular expressions and see the results visually is regxr.com
  - You can provide sample text, write a regex, and it will highlight matches

Elementary text analysis

Moving from texts to numbers

- To analyse text quantitatively, the key question is how to move from text to numbers
- We will look at very common approaches that count words in documents
- This abstracts from the sequential dependency of words (beyond n-grams) and is sometimes referred to as a bag-of-words approach

Common workflow



An economic miracle is taking place in the United States, and the only thing that can stop it are foolish wars, politics, or ridiculous partisan investigations.

The United States of America right now has the strongest, most durable economy in the world. We're in the middle of the longest streak of private sector job creation in history.

We reinvented Government, transforming it into a catalyst for new ideas that stress both opportunity and responsibility and give our people the tools they need to solve their own problems.

To build a prospero future, we must trus people with their own money and empower them to grow our economy.

Source texts

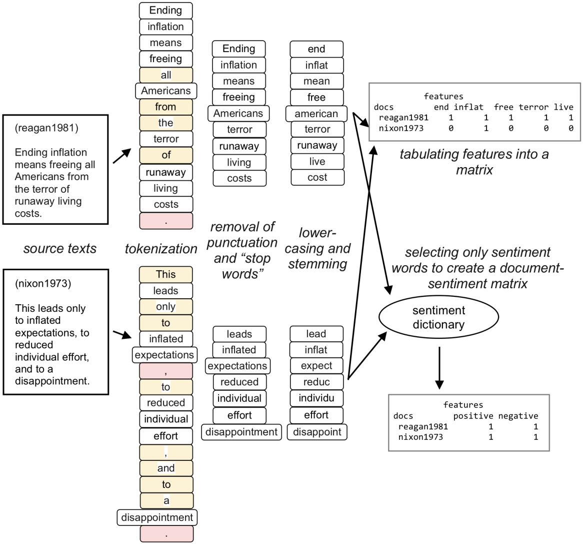
Processed text as a document-feature matrix

```
                               features
documents      economy united wall crime climate
  Clinton-2000      10      4    1     5       1
  Bush-2008          6      4    0     0       1
  Obama-2016        16      4    1     0       4
  Trump-2019         5     19    6     2       0
```

Quantitative analysis and inference

Describing texts quantitatively or stylistically
Identifying keywords
Measuring ideology or sentiment in documents
Mapping semantic networks
Identifying topics and estimating their prevalence
Measuring document or term similarities
Classifying documents

Common workflow: Tokenisation + dictionary method



(reagan1981)

Ending inflation means freeing all Americans from the terror of runaway living costs.

(nixon1973)

This leads only to inflated expectations, to reduced individual effort, and to a disappointment.

source texts

tokenization

removal of punctuation and "stop words"

lower-casing and stemming

```
          features
docs         end inflat  free terror live
 reagan1981    1      1     1      1    1
 nixon1973     0      1     0      0    0
```

tabulating features into a matrix

selecting only sentiment words to create a document-sentiment matrix

```
          features
docs         positive negative
 reagan1981         1        1
 nixon1973          1        1
```

Some key concepts

- Document-feature matrix (dfm): As many rows as documents, as many columns was words/features after cleaning
- Stopwords: Common words such as “the”, “to”, etc.
- Stemming: Heuristic process to obtain the stem of words which in essense groups terms, see the following link for a detailed discussion
- n-grams: Sequences of words, e.g. bigrams (2) or trigrams (3). For example allows to record “not good” as a feature

Dictionary approaches

- Map each word or phrase to a “dictionary” of words, e.g. associated with a known “sentiment” or psychological state or with certain topics
- Treats matches within each dictionary as equivalent
- Examples: Linguistic Inquiry and Word Count, or the General Inquirer

Dictionary example (from LIWC 2015)

```
Dictionary object with 1 key entry.
- [posemo]:
- like, like*, :), (:, accept, accepta*, accepted, acceptin
interests, invigor*, joke*, joking, jolly, joy*, keen*, kid
kind, kindly, kindn*, kiss*, laidback, laugh*, legit, liber
likeab*, liked, likes, liking, livel*, lmao*, lmfao*, lol,
```

Problems with dictionary approaches

- Polysemy – multiple meanings: The word “kind” has three!
- From State of the Union corpus: 318 matches
  - kind/NOUN – 95%
  - kind (of)/ADVERB – 1%
  - kind/ADJECTIVE – 4%
- These are known as false positives
- Other problem: False negatives (what we miss)
  - Missed: kindliness
  - Also missed: altruistic and magnanimous
- How to treat conflicting keywords in the same string? “Had a great day . . . not.”

Further topics

- Text classification: How do we use a feature matrix to predict document labels (e.g. spam/not spam)?
- Topic models: How do we find sets of words which tend to appear together?
- Word and document embeddings: How can we represent words or documents as vectors and analyse their distances/similarities?
- How do we take into account the sequential nature of text?
- Etc.

Coding

Markdown files

- 01-regular-expressions-in-r.Rmd
- 02-text-analysis.Rmd
- 03-parsing-pdfs.Rmd