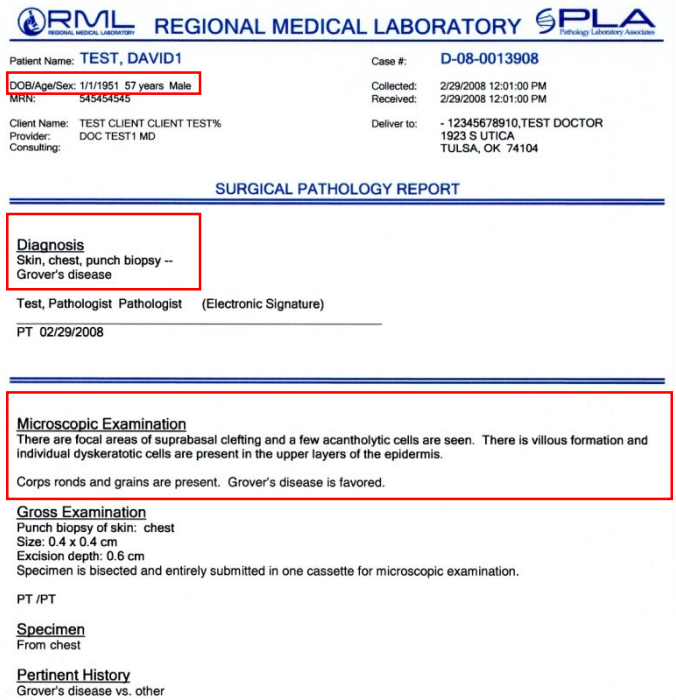
COMP20008
Elements of data processing

Semester 1 2020

Lecture 6: Unstructured data – preprocessing

Unstructured data – text

- No structure
- Tricky to organise
- Lacks regularity and decomposable internal structure
- How can we process and search for textual information?



RML REGIONAL MEDICAL LABORATORY SPLA

Patient Name: TEST, DAVID1
DOB/Age/Sex: 1/1/1951 57 years Male
MRN: 545454545

Client Name: TEST CLIENT CLIENT TEST%
Provider: DOC TEST1 MD
Consulting:

Case #: D-08-0013908
Collected: 2/29/2008 12:01:00 PM
Received: 2/29/2008 12:01:00 PM

Deliver to: - 12345678910,TEST DOCTOR
1923 S UTICA
TULSA, OK 74104

SURGICAL PATHOLOGY REPORT

Diagnosis
Skin, chest, punch biopsy --
Grover's disease

Test, Pathologist Pathologist (Electronic Signature)

PT 02/29/2008

Microscopic Examination
There are focal areas of suprabasal clefting and a few acantholytic cells are seen. There is villous formation and individual dyskeratotic cells are present in the upper layers of the epidermis.

Corps ronds and grains are present. Grover's disease is favored.

Gross Examination
Punch biopsy of skin: chest
Size: 0.4 x 0.4 cm
Excision depth: 0.6 cm
Specimen is bisected and entirely submitted in one cassette for microscopic examination.

PT /PT

Specimen
From chest

Pertinent History
Grover's disease vs. other

Patterns in text – scenario

- Scenario: we have a large collection of unstructured text data. You need to write wrangling code in order to
  - Check whether it contains any IP addresses (e.g. 128.250.65.5)
  - Find all of the IP addresses
- Requirements
  - Do it succinctly
  - Do it unambiguously
  - Have maintainable code
- Specify patterns in text – *regular expressions*
  - Good for calculating statistics
  - Checking for integrity, filtering, substitutions …

Pattern matching in text – regular expression patterns

Regular expressions

Regular expressions (RE)

Simple match – characters match themselves exactly

- The pattern **`hello`** will match the string **hello**
- **`Hello`** will match **Hello**

Metacharacters –special rules for matching

RE: metacharacters

.: matching any character

- For example, **a.c** matches **a/c**, **abc**, **a5c**
- To match **'.'** as a literal,

  escape with **'\.' a\.c** matches **a.c**

That is, **'\'** is also a metacharacter

RE: metacharacters

****: backslash character is used to

- escape metacharacters or other special characters, e.g.:
- match **'.'** as a literal: **a\.c** matches **a.c**
- match **'\'** as a literal: **a\\c** matches **a\c**

It also indicate special forms, e.g., special character set

\d for any decimal digit

RE: metacharacters

`[ ]`: matching a set (class) of characters; e.g., `[abc]`, `[a-zA-Z]`

- `[^]` : Complementing the set

  add `'^'` as the **first character** in the class (`[^z]` anything but `z`)

  What does the pattern `[z^]` match?

- Use `'\'` to escape special characters `'['`,`']'` inside `[]`.

  `[\[]` matches the special character: `'['`

- Predefined special character set: e.g.,
  - `\d` any decimal digit == `[0-9]`
  - `\w` any alphanumeric character == `[a-zA-Z0-9_]`
  - `\W` any non-alphanumeric character == `[^a-zA-Z0-9_]`
- Special character sets are at https://docs.python.org/2/howto/regex.html

RE: metacharacters – cont.

`*` `+` `?` `{m,n}`: repeat a pattern

- `*` : zero or more repetitions
  - `go*d` matches all 4 patterns: `gd`, `god`, `good`, and `goooooood`
  - What does the pattern `a[0-9]*z` match?
- `+` : one or more repetitions
- `{m, n}` : at least `m` and at most `n` repetitions
- `?` : zero or one repetition
- Pattern search is greedy (will go as far as possible)
  - Given a string 'gogogoD', `(go)*` will match gogogo

RE: metacharacters – cont.

`|`: the "OR" operator (alternatives)

- Two patterns $\mathbf{P_1}$ and $\mathbf{P_2}$, $\mathbf{P_1}\,|\,\mathbf{P_2}$ will match either $\mathbf{P_1}$ or $\mathbf{P_2}$.
- Often used with parantheses `()`
  - `abc|xyz` will match `abc` or `xyz`
  - `xy|z` == `(xy)|z` ≠ `x(y|z)`
- To match `'|'` as a literal
  - Escape with `'\|'`
  - Put it in character class `[|]`

RE: metacharacters – cont.

`^` `$`: Anchoring

- `^` : start of string
  - `^from` will match `from` only at the start of the string, e.g. ‘from a to b’
  - `^from` will not match ‘I am from Melbourne’
- `$` : end of string
- To match `'^'` or `'$'` as a literal
  - Escape with `'\^'` or `'\$'`
  - Put it in character class `[$^]` (**note** special meaning if `'^'` is placed as the first character)

More complex regular expression

What do you think this pattern is for?

- `[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-]+`
- Could it be improved?

RE: metacharacters substitution & capturing groups

`(  )`: as in math notation, group patterns with the metacharacters **`( )`**

- Grouped patterns are captured and numbered
- You can specify the contents by back-references
- **`(.+) or not \1`** will match
  - '**X** or not **X**', '**play** or not **play**', '**2b** or not **2b**',
  - The pattern **`(.+)`**, one or more characters, is captured as a group and numbered as 1.
  - In the same regular expression, **`\1`** refers to the input captured by the pattern

Regular expressions and ELIZA

ELIZA: a computer psychotherapist

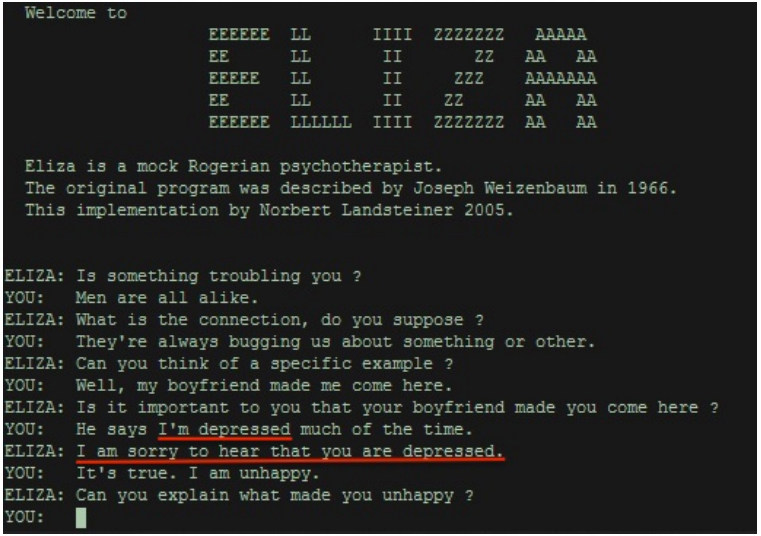
"works by having a series or cascade of regular expression substitutions each of which matches and changes some part of the input lines."

- Match:

  I'm (depressed|sad|unhappy)

- Substitute by:

  I am sorry to hear that you are \1



https://web.stanford.edu/~jurafsky/slp3/2.pdf

By Unknown author - http://www.le-grenier-informatique.fr/medias/images/eliza-title.jpg, Public Domain, https://commons.wikimedia.org/w/index.php?curid=70571280

Re: metacharacters

The complete list of metacharacters

```
. ^ $ * + ? { } [ ] \ | ( )
```

Regular expression for text processing

Python re

```
import re

re.match()
re.search()
re.sub()
re.split()

p = re.compile(regular expression)
p.match()
```

Practice in the workshop

Text preprocessing – tokenisation

- Split continuous text into a list of individual tokens
- English words are often separated by white spaces but not always
- Tokens can be words, numbers, hashtags, etc.
- Can use regular expression

Text preprocessing – case folding

- Convert text to consistent cases
- Simple and effective for many tasks
- Reduce sparsity (many map to the same lower-case form)
- Good for search

```
I had an AMAZING trip to
Italy, Coffee is only 2
bucks, sometimes three!
```

```
i had an amazing trip to
italy, coffee is only 2
bucks, sometimes three!
```

Preprocessing – stemming

- Words in English are derived from a root or **stem**

  inexpensive → in+expense+ive
- Stemming attempts to undo the processes that lead to word formation
- Remove and replace word suffixes to arrive at a common **root form**
- Result does not necessarily look like a proper ‘word’
- Porter stemmer: one of the most widely used stemming algorithms
- **suffix stripping (**Porter stemmer)
  - sses → ss
  - ies → i
  - tional → tion
  - tion → t

Preprocessing – stemming

https://text-processing.com/demo/stem/

troubles → troubl

troubled → troubl

trouble → troubl

Preprocessing – lemmatization

- To remove inflections and map a word to its *proper* root form (lemma)
- It does not just strip suffixes, it transforms words to valid roots:
  - running → run
  - runs → run
  - ran → run
- Python NLTK provides WordNet Lemmatizer that uses the WordNet Database to lookup lemmas of words.

Stopword removal

- Stop words are ‘function’ words that structure sentences; they are low information words and some of them are very common
  - ‘The’, ‘a’, ‘is’,…
- Exclude them from being processed; helps to reduce the number of features/words
- Commonly applied for search, text classification, topic modeling, topic extraction, etc.
- A stopword list can be custom-made for a specific context/domain

Stopword removal

{‘ourselves’, ‘hers’, ‘between’, ‘yourself’, ‘but’, ‘again’, ‘there’, ‘about’, ‘once’, ‘during’, ‘out’, ‘very’, ‘having’, ‘with’, ‘they’, ‘own’, ‘an’, ‘be’, ‘some’, ‘for’, ‘do’, ‘its’, ‘yours’, ‘such’, ‘into’, ‘of’, ‘most’, ‘itself’, ‘other’, ‘off’, ‘is’, ‘s’, ‘am’, ‘or’, ‘who’, ‘as’, ‘from’, ‘him’, ‘each’, ‘the’, ‘themselves’, ‘until’, ‘below’, ‘are’, ‘we’, ‘these’, ‘your’, ‘his’, ‘through’, ‘don’, ‘nor’, ‘me’, ‘were’, ‘her’, ‘more’, ‘himself’, ‘this’, ‘down’, ‘should’, ‘our’, ‘their’, ‘while’, ‘above’, ‘both’, ‘up’, ‘to’, ‘ours’, ‘had’, ‘she’, ‘all’, ‘no’, ‘when’, ‘at’, ‘any’, ‘before’, ‘them’, ‘same’, ‘and’, ‘been’, ‘have’, ‘in’, ‘will’, ‘on’, ‘does’, ‘yourselves’, ‘then’, ‘that’, ‘because’, ‘what’, ‘over’, ‘why’, ‘so’, ‘can’, ‘did’, ‘not’, ‘now’, ‘under’, ‘he’, ‘you’, ‘herself’, ‘has’, ‘just’, ‘where’, ‘too’, ‘only’, ‘myself’, ‘which’, ‘those’, ‘i’, ‘after’, ‘few’, ‘whom’, ‘t’, ‘being’, ‘if’, ‘theirs’, ‘my’, ‘against’, ‘a’, ‘by’, ‘doing’, ‘it’, ‘how’, ‘further’, ‘was’, ‘here’, ‘than’}

Text normalisation

- Transforming a text into a canonical (standard) form
- Important for noisy text, e.g., social media comments, text messages
- Used when there are many abbreviations, misspellings and out-of-vocabulary words (oov)
- E.g.

  2moro → tomorrow

  2mrw → tomorrow

  tomrw → tomorrow

  B4 → before

Noise removal

- Remove unnecessary spacing
- Remove punctuation and special characters (regular expressions)
- Unify numbers
- Highly domain dependent

Text/document representations

After preprocessing, the list of more 'regular' words (tokens) become the representation of the text (document).

For NLP and Machine Learning, we generate features

Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
warm-blooded	hair	yes	no	no	yes	no	mammal
cold-blooded	scales	no	no	no	no	yes	reptile
cold-blooded	scales	no	yes	no	no	no	fish
warm-blooded	hair	yes	yes	no	no	no	mammal
cold-blooded	none	no	semi	no	yes	yes	amphibian

We also generate features for unstructured text.

Text features – Bag of words

- The simplest vector space representational model for unstructured text.
- Disregarding word orders and grammar
- Each text document as a numeric vector
  - each dimension is a specific word from the corpus
  - the value could be its frequency in the document or occurrence (denoted by 1 or 0).

Text features – TF-IDF

- TF-IDF stands for Term Frequency-Inverse Document Frequency
- Each text document as a numeric vector
  - each dimension is a specific word from the corpus
- A combination of two metrics to weight a term (word)
  - term frequency (tf): how often a given word appears within a document
  - inverse document frequency (idf): down-weights words that appear in many documents.
- Main idea: reduce the weight of frequent terms and increase the weight of rare ones.

Text features – TF-IDF

- $\text{tf} = \dfrac{\text{frequency of the word}}{\text{total number of the words in the document}}$
  - Higher frequency, higher tf
  - Normalise by document length
- $\text{idf} = \log\left(\dfrac{\text{total number of documents}}{\text{number of documents containing the word}}\right)$
  - Logarithmic inverse of document frequency
  - Rare words, higher idf
- $\text{tf-idf} = tf \times idf$

TF-IDF example

- Two documents, A and B.

A: ‘the car is driven on the road’

B: ‘the truck is driven on the highway’

word	tf		idf	tf-idf	
	A	B		A	B
the	2/7	2/7	log(2/2) = 0	0	0
car	1/7	0	log(2/1) = 0.3	0.043	0
is	1/7	1/7	log(2/2) = 0	0	0
driven	1/7	1/7	log(2/2) = 0	0	0
on	1/7	1/7	log(2/2) = 0	0	0
road	1/7	0	log(2/1) = 0.3	0.043	0
truck	0	1/7	log(2/1) = 0.3	0	0.043
highway	0	1/7	log(2/1) = 0.3	0	0.043

Example TF-IDF features – cont.

- Two documents, A and B.
  A. ‘the car is driven on the road’
  B. ‘the truck is driven on the highway’

- Text features for machine learning

the	car	is	driven	on	road	truck	highway
0	0.043	0	0	0	0.043	0	0
0	0	0	0	0	0	0.043	0.043

Features from unstructured text

Features for structured data

Body Temperature	Skin Cover	Gives Birth	Aquatic Creature	Aerial Creature	Has Legs	Hiber-nates	Class Label
warm-blooded	hair	yes	no	no	yes	no	mammal
cold-blooded	scales	no	no	no	no	yes	reptile
cold-blooded	scales	no	yes	no	no	no	fish
warm-blooded	hair	yes	yes	no	no	no	mammal
cold-blooded	none	no	semi	no	yes	yes	amphibian
cold-blooded	scales	no	no	no	yes	no	reptile

Features for unstructured text

the	car	is	driven	on	road	truck	highway
0	0.043	0	0	0	0.043	0	0
0	0	0	0	0	0	0.043	0.043

Summary – unstructured text data

- Crawling & scraping
- Text search – approximate string matching
- Preprocessing
  - Regular expressions
  - Tokenisation
  - Case folding
  - Stemming
  - Lemmatization
  - Stopword removal
  - Text normalization
  - Noise removal
- Document representation and text features (BoW, TF-IDF)