CSCE604135 | Perolehan Informasi (Information Retrieval)

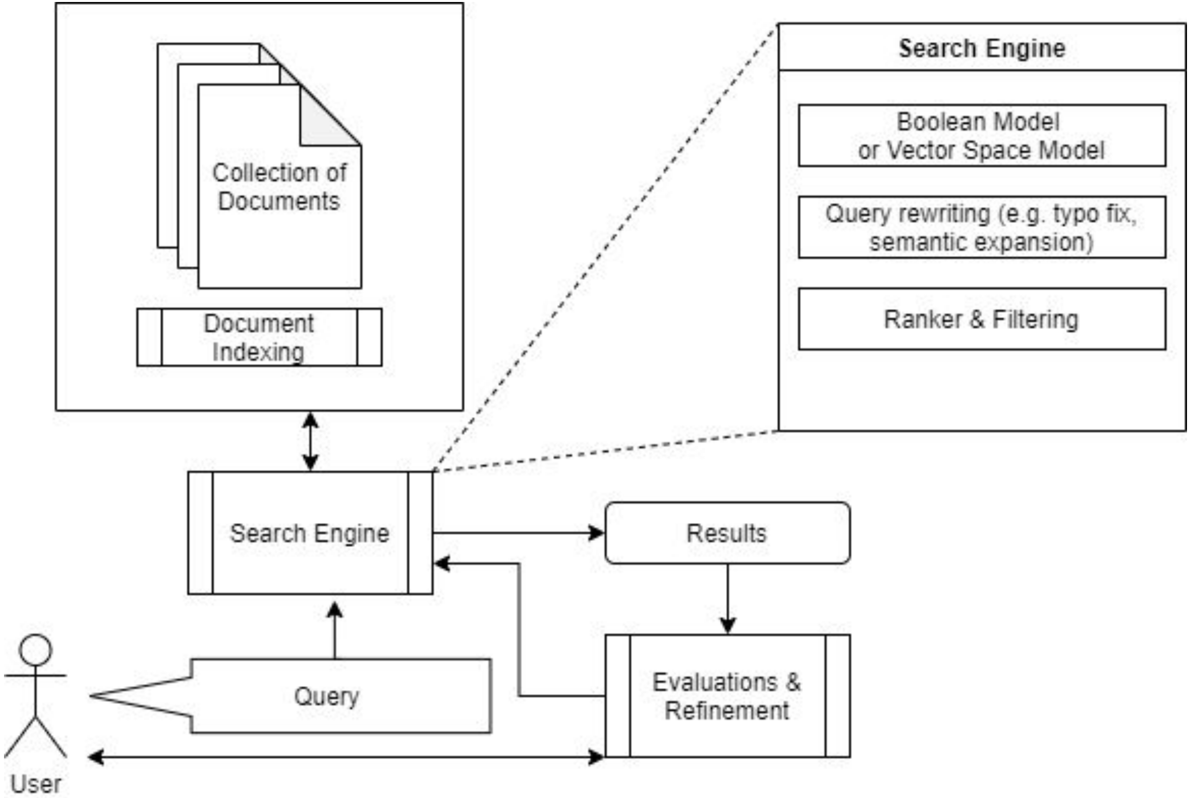
Text Processing pt. 1

Radityo Eko Prasojo, PhD

Contents

- Why and what is text (pre)processing
- Conceptual discussion of text processing pipeline: tokenization, stopwords, morphological analysis, (some) syntactic/semantic annotations

Overview of an IR System



Text Processing, why?

- To parse **queries** into keywords
- Specifically, **boolean queries** need to be parsed as a formula tree
- To parse **documents** in order to build the **inverted index**
- To annotate/fix textual/semantic information on queries or documents

Tokenization

- The process of splitting a long text into its **tokens**

  A book → chapters → sections → paragraphs → sentences → tokens

- Documents may have different file formats & structures

  For example, A book PDF vs A Wikipedia page

- However, paragraphs → sentences → tokens are quite universal

What are tokens?

- **Smallest components** in the text
- Most obviously tokens are:
  - Words: **“saya” “makan” “nasi”**
  - Punctuations: “saya” “makan” “nasi” **“.”**
  - Numbers: “saya” “makan” **“2”** “piring” “nasi” “.”
- But more than that, tokens can also be:
  - Abbreviations: “Mr.”, “M.D.”, “p2p”
  - Words with punctuation/symbols in between: “Jum’at”, “pro-aktif”, “kupu-kupu”, “micro$oft”, “ridho@kata.ai”
  - Phone numbers, which can be written in several ways
  - Foreign words: “saya lagi ada di ***call***”

So, paragraph → sentence → tokens

- Split paragraphs → sentences by **ending punctuations** [. ! ?]
- Split sentences into tokens by **whitespaces** [spaces, enters, tabs]

`“Saya makan nasi padang. Rasanya enak.” → [[“Saya”, “makan”, “nasi”, “padang”] [“Rasanya”, “enak”]]`

- [Q] is this good enough?

So, paragraph → sentence → tokens

- Split paragraphs → sentences by **ending punctuations** [. ! ?]
- Split sentences into tokens by **whitespaces** [spaces, enters, tabs]

`“Saya makan nasi padang. Rasanya enak.” → [[“Saya”, “makan”, “nasi”, “padang”] [“Rasanya”, “enak”]]`

- Though, some languages do not have spaces between words
  - なに これ？
- And that there are some “fake” ending punctuations
  - [“**Dr.** House”] not [[“**Dr.**”], [“House”]]
  - I bought **0.5** kg of rambutans
  - Ann sent a message to **bob@email.com**

Subword tokenization

- Split some words into subcomponents
- Derivative words (kata turunan)

Saya memakan nasi → Saya, me, makan, nasi

- Some languages, like German, combine words

Wirtschaftsinformatik (“Business informatics”)

- Subword tokenizations are used in some contemporary **vector space models** (though rarely in others)

Tokenization: ideal vs practical

- Sometimes a **smallest component** comprises of several words

Entity names: *"Joko Widodo"*, *"Garuda Indonesia"*, etc. and multi-word expressions (MWEs): *"piece of cake"*, *"sambil menyelam minum air"*

- Ideally, a tokenizer has to know when *to split or not to split*
- However this ideal is often not practical (too difficult, with little gains)
- Hence, often in practice, **everything is tokenized** into the words
- MWEs and entities are left to be dealt in later steps (can be in **indexing, ranking,** or in the **vector space model**)

Q: Joko Widodo

Doc23

Joko Widodo adalah presiden ....

Doc41

Joko diciduk polisi saat nongkrong di ... Rekannya, **Widodo**, sampai saat ini masih buron.

Doc3

Rendang is one of the most popular dishes in Indonesia. It has a savoury, sweet taste, with mild spiciness ...

Later steps

Despite the “Joko Widodo” being tokenized, we expect Doc23 to rank higher

Taking a step back: what were the tokens for?

- Documents are tokenized to build **inverse indexes**

```
“Nasi” → {doc5, doc6, doc13, …}
“Retrieval” → {doc1, doc2, doc9, …}
```

- Queries are tokenized, and then matched with the indexes to retrieve the relevant documents
- [Q] However, is this enough? What could be the problem?

Taking a step back: what were the tokens for?

- Documents are tokenized to build **inverse indexes**

```
“Nasi” → {doc5, doc6, doc13, …}
“Retrieval” → {doc1, doc2, doc9, …}
```

- Queries are tokenized, and then matched with the indexes to retrieve the relevant documents
- But what if…
    - query = “**n**asi”, but there is no “nasi” index, only “**N**asi”
    - query = “**me**makan, but there is no “memakan” index, only “makan”
    - query = ”lezat”, but there is no “lezat” index, only “enak”
    - etc.

“They should match, but they don’t”

- Ideally, all tokens that should **semantically** match in the Q and the index, should also **textually** match
- Some possible problems:

Casing problems [Nasi ↔ nasi], morphological problem [makan ↔ memakan], synonymy [enak ↔ lezat], typos

- We discuss some preprocessing technique to alleviate these problems

Casing

- Apply the same **casing** to all texts (queries and documents)
- **Lowercasing** → all to lowercase, the most common settings
  - Uppercasing achieves the same, but rarely used (who wants to read ALL CAPS TEXT?)
  - Also referred to as **uncasing**
- **Truecasing** → all to the original and correct casing
  - e.g. “joko widodo” becomes “Joko Widodo”
  - Ideal, but often impractical (too difficult with little gains). e.g. should “merpati” be capitalized?
  - truecasing programs are slower than lowercasing, IR systems want to be fast!
- Casing is typically done before tokenization

Morphological analysis

- **Morphology**: a study of formation of words
- In the context of IR: **morphological analysis** (MA) is used mainly to convert derivative words (kata turunan) into their roots (kata dasar)

Memakan → makan, went → go, beautiful → beauty

- Two different processes for MA: **stemming** and **lemmatization**

Stemming

- Stemming is **algorithmic**: a set of rules done to cut-off words until it reaches what it believes to be the root
    - Example of a simple stemmer: always cut prefix “me” and suffix “an”
    - Several stemming algorithms: Porter Stemmer, Lancaster Stemmer, Snowball Stemmer
- Lemmatization is **dictionary-based**: look into a dictionary for the root
    - Example: **WordNet**
- Interesting examples:
    - memakan → makan can be solved both by stemming and lemmatization
    - was → is can only be solved by lemmatization
    - a stemmer might decide that university → universe, lemmatization will not (the two words are semantically unrelated)

What if your stemmer reaches a wrong solution?

- For example: “memakan” → “mak”
- Would this cause any problem for your IR system?

What if your stemmer reaches a “wrong” solution?

- For example: “memakan” → “mak”, “marriage” → “marri”
- Would this cause any problem for your IR system?
- Surprisingly, often it is of a little problem, as long as it is consistent.
  - i.e. “dimakan” also goes to “mak”, “makam” does not go to “mak”
  - “marrying” goes to “marri, but “summary” does not go to “marri”
- Still, ambiguity does exist, errors are to be expected
  - “beruang” → “uang”?

So then, why not just use lemmatization?

- Language evolves; your dictionary might not keep up.
- Typos exist
- Stemmer can still process words that are not in the dictionary

Applying stemming/lemmatization on IR context

- On the documents, we can add over the indexes, e.g.

“Memakan” → [Doc16, Doc133, Doc297, …]
“Makan” → [Doc5, **Doc16**, Doc55, Doc66, **Doc133**, **Doc297**, …]

- On the query, we can rewrite the query such as

“memakan” into ”makan” or “memakan OR makan”

- **[Q]** If the stemming/lemmatization is already done on the documents, which query rewriting above is the best?

Synonyms, typos

- Synonyms and typos detection/generation is always **dictionary-based**
  - e.g. **WordNet** for synonyms, **Norvig Algorithm** with any dictionary for typos
- Similarly to stemming and lemmatization, they can be applied both in the documents (as index expansion) and queries (for rewriting)

“kpn” → [Doc16, Doc133, Doc297, …]
“kapan” → [Doc5, **Doc16**, Doc55, Doc66, **Doc133**, **Doc297**, …]

Stopwords

- Some words are not important, no need to process

English stopwords: `[“of”, “in”, “on”, …]`
Indonesian stopwords: `[“yang”, “di”, “pada”, …]`

- They can be removed from documents and also queries
  - imagine if we create indexes for these words, then they might include all documents!
  - if we have Q: “president of america”, then we can just remove the “of” because there is no index of “of” anyway!

Stopwords

- Stopword removal used to be important, especially on **boolean models**
- However, current **vector space models** often requires the stopwords to be kept, because removing stopwords might change the semantics of the text

“turn on the car” vs “turn the car”

- List of stopwords can be readily obtained
  - e.g. from a toolkit/library
- However, we can also prepare our own stopwords
  - We will discuss this when we go into more detail about **ranking**

Syntactic and Semantic annotation

- Polysemy, homonymy, and syntactic relationship

“beruang”, “milk”
“widodo” → is it Joko Widodo?
“Obama datang ke Indonesia dan makan nasi goreng”
“Obama datang ke Indonesia dan dia makan nasi goreng”

query “obama makan” or “barack obama” (with double quote) does not match

- Several text processing approaches that can help: part-of-speech (POS) tagging, entity linking, and dependency parsing
  - Typically, they are neither **algorithmic** nor **dictionary-based**, but rather a **statistical model** trained by **machine learning.** E.g.: ones from NLTK, SpaCy, and Stanza

Part-of-speech tagging

- Annotate text with its part-of-speech roles

```
[Doc55] I milk my cows → I/PRP milk/VB my/PRP$ cows/NNS
[Doc56] I drink milk → I/PRP drink/VB milk/NN
```

- The most widely used standard of POS tags is the [Penn Treebank]
- From here, we can extend the index e.g.

```
[milk] --> [Doc55, Doc56, …]
into [milk/VB] → [Doc55, …] & [milk/NN] → [Doc56, …]
```

Part-of-speech tagging

- Hence, for every query, we can also apply POS tagging

```
How to milk cows → How/WRB to/IN milk/VB cows/NNS
```

- So then when searching the index, only the milk with VB tag is returned

Entity linking

- Typically involves named-entity recognition (NER), entity disambiguation and coreference resolution

`Obama datang ke Indonesia → `<u>`Obama`</u>`[wiki:Barack Obama] datang ke `<u>`Indonesia`</u>`[wiki:Indonesia]`

- The underlining part is the NER, and the bracket addition is the disambiguation part

Entity linking

- Typically involves named-entity recognition (NER), entity disambiguation and coreference resolution

```
[Doc9] Obama datang ke Indonesia dan dia makan nasi goreng →
Obama[wiki:Barack Obama] datang ke Indonesia[wiki:Indonesia]
dan dia{0} makan nasi goreng
```

- The underlining part is the NER, the angle bracket addition is the disambiguation part, and the curly bracket is the coreference resolution part (meaning that it refers to the first word (index 0)) in the sentence.

Entity linking → Index

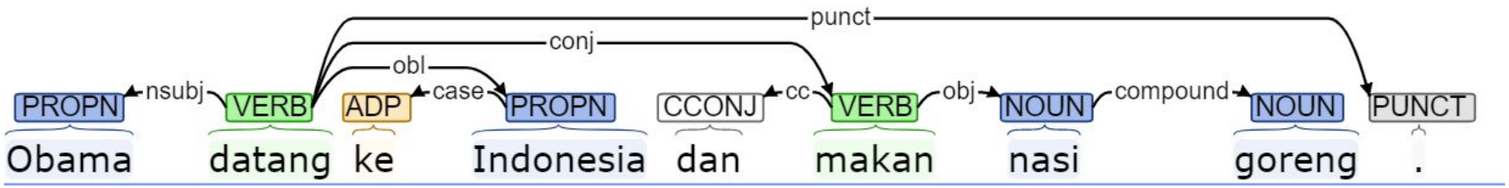
`[Doc9] Obama datang ke Indonesia dan dia makan nasi goreng →`
`Obama[wiki:Barack Obama] datang ke Indonesia[wiki:Indonesia]`
`dan dia{0} makan nasi goreng`

- Once we know that "Obama" here is really "Barack Obama", and "dia" is "Obama" we can then extend/create indexes

`"barack obama" → […, Doc9, …]`
`"obama makan" → […, Doc9, …]`

- Yes, we can have indexes with >1 word

Dependency parsing → Index



Result by stanza.run

- By analyzing the dependency edges `conj` and `nsubj`, we can see that the verb makan has subject Obama
- Similar to the previous example (entity linking), we can also add a new index for “obama makan”
- Complete list of dependency edges: [Universal Dependencies]

Syntactic & semantic tags: caveat

- Rarely used in industry-standard IR, which even nowadays rely highly on textual features only
    - such as **elasticsearch**
- This means that, none of the documents, their indexes, and the queries are processed in a way that involves syntactic and semantic annotation
- Nevertheless, some document collections are already **tagged**
- They are referred to as **corpus** (plural: **corpora**)
- For example, Brown Corpus is POS-tagged

Corpora

- In general, a corpus is a collection of text documents that has some structure or metadata (does not have to be syntactic tags)
- **Domain**: some corpus focuses on specific domains, which affect the tokens/contents
    - Twitter corpus vs Wikipedia corpus differ on language style
    - Customer service corpus might contain many phone number/email tokens
- **Language**: a corpus can be either monolingual or multilingual, which in turn can be parallel
    - A parallel multilingual corpus means that an IR system over it can receive a query in a language, then return relevant documents in another language

Parallel multilingual corpus: example

DocID	English	Indonesian
1	Obama visited Indonesia...	Obama datang ke Indonesia untuk ...
2	Rendang is regarded to be ...	Rendang banyak diakui sebagai salah satu...
...	...	...

Q: “Obama datang ke Indonesia”

The IR system can return the English documents because of the parallelism!

Corpora in the wild

- Brown Corpus (https://en.wikipedia.org/wiki/Brown_Corpus)
- Leipzig Corpus (https://corpora.uni-leipzig.de/)
- Oscar Corpus (https://oscar-corpus.com/)
- Wikimatrix (https://github.com/facebookresearch/LASER/tree/master/tasks/WikiMatrix)
- Kata-MT (https://github.com/gunnxx/indonesian-mt-data)

Supported by the Kampus Merdeka grant of
Ministry of Education, Culture, Research, and Technology
of Republic of Indonesia