DM&S LAB 1 - ELIAS BASSANI

TEXT PREPROCESSING & REPRESENTATION

TOKENIZATION – IDENTIFYING WORDS

- A word in English is relatively simple to spot: words are separated by whitespace or (some) punctuation. Even in English, though, there can be controversy: is *you're* one word or two? What about *o'clock*, *cooperate*, or *eyewitness*?
- Languages like German or Dutch combine individual words to create longer compound words like *Weißkopfseeadler* (white-headed sea eagle), but in order to be able to return *Weißkopfseeadler* as a result for the query *Adler* (eagle), we need to understand how to break up compound words into their constituent parts.

TOKENIZATION – IDENTIFYING WORDS

- Asian languages are even more complex: some have no whitespace between words, sentences, or even paragraphs. Some words can be represented by a single character, but the same single character, when placed next to other characters, can form just one part of a longer word with a quite different meaning.
- It should be obvious that there is no silver-bullet tokenizer that will miraculously deal with all human languages. Therefore, it is necessary to choose the tokenizer on a case-by-case basis.

TOKENIZATION – IDENTIFYING WORDS

- However, not all languages have dedicated tokenizers, and sometimes you won't even be sure which language(s) you are dealing with. For these situations, we need good standard tools that do a reasonable job regardless of language.

NORMALIZATION – NORMALIZING TOKENS

- Breaking text into tokens is only half the job. To make those tokens more easily searchable, they need to go through a normalization process to remove insignificant differences between otherwise identical words, such as uppercase/lowercase. Perhaps we also need to remove significant differences, to make *esta*, *ésta*, and *está* all searchable as the same word. Would you search for *déjà vu*, or just for *deja vu*?
- This is the job of the token filters, which receive a stream of tokens from the tokenizer. You can have multiple token filters, each doing its particular job. Each receives the new token stream as output by the token filter before it.

STEMMING – REDUCING WORDS TO THEIR ROOT FORM

- Most languages of the world are inflected, meaning that words can change their form to express differences in the following:
  - Number: fox, foxes
  - Tense: pay, paid, paying
  - Gender: waiter, waitress
  - Person: hear, hears
  - Case: I, me, my
  - Aspect: ate, eaten
  - Mood: so be it, were it so

STEMMING – REDUCING WORDS TO THEIR ROOT FORM

- While inflection aids expressivity, it interferes with retrievability, as a single root *word* sense (or meaning) may be represented by many different sequences of letters. English is a weakly inflected language (you could ignore inflections and still get reasonable search results), but some other languages are highly inflected and need extra work in order to achieve high-quality search results.
- *Stemming* attempts to remove the differences between inflected forms of a word, in order to reduce each word to its root form.
- For instance *foxes* may be reduced to the root *fox*, to remove the difference between singular and plural in the same way that we removed the difference between lowercase and uppercase.

STEMMING – REDUCING WORDS TO THEIR ROOT FORM

- If stemming were easy, there would be only one implementation. Unfortunately, stemming is an inexact science that suffers from two issues: *understemming* and *overstemming*.
- *Understemming* is the failure to reduce words with the same meaning to the same root. For example, *jumped* and *jumps* may be reduced to *jump*, while *jumping* may be reduced to *jumpi*. *Understemming* reduces retrieval; relevant documents are not returned.
- *Overstemming* is the failure to keep two words with distinct meanings separate. For instance, *general* and *generate* may both be stemmed to *gener*. *Overstemming* reduces precision: irrelevant documents are returned when they shouldn't be.

STEMMING – ALGORITHMIC STEMMERS

- They apply a series of rules to a word in order to reduce it to its root form, such as stripping the final *s* or *es* from plurals. They don't have to know anything about individual words in order to stem them.
- These algorithmic stemmers have the advantage that they are available out of the box, are fast, use little memory, and work well for regular words.
- The downside is that they don't cope well with irregular words like *be*, *are*, and *am*, or *mice* and *mouse*.

STEMMING – DICTIONARY STEMMERS

- Dictionary stemmers work quite differently from *algorithmic stemmers*. Instead of applying a standard set of rules to each word, they simply look up the word in the dictionary. Theoretically, they could produce much better results than an algorithmic stemmer.
- A dictionary stemmer should be able to do the following:
  - Return the correct root word for irregular forms such as *feet* and *mice*
  - Recognize the distinction between words that are similar but have different word senses – for example, *organ* and *organization*.
- In practice, a good algorithmic stemmer usually outperforms a dictionary stemmer.

STEMMING – DICTIONARY STEMMERS

- Dictionary quality:
  - A dictionary stemmer is only as good as its dictionary.
    (Most English dictionaries available for computers contain about 10% of English words.)
  - The meaning of words changes with time.
    (Dictionaries need to be kept current, which is a time-consuming task. Often, by the time a dictionary has been made available, some of its entries are already out-of-date.)
  - If a dictionary stemmer encounters a word not in its dictionary, it doesn't know how to deal with it.
    An algorithmic stemmer, on the other hand, will apply the same rules as before, correctly or incorrectly.

STEMMING – DICTIONARY STEMMERS

- Size and performance:
  - Finding the right stem for a word is often considerably more complex than the equivalent process with an algorithmic stemmer. (A dictionary stemmer needs to load all words, all prefixes, and all suffixes into memory. This can use a significant amount of RAM.)
  - Depending on the quality of the dictionary, the process of removing prefixes and suffixes may be more or less efficient. Less-efficient forms can slow the stemming process significantly.
  - Algorithmic stemmers, on the other hand, are usually simple, small, and fast.

LEMMATIZATION

- A *lemma* is the canonical, or dictionary, form of a set of related words–the lemma of *paying*, *paid*, and *pays* is *pay*. Usually the lemma resembles the words it is related to but sometimes it doesn't – the lemma of *is*, *was*, *am*, and *being* is be.
- Lemmatization, like stemming, tries to group related words, but it goes one step further than stemming in that it tries to group words by their *word sense*, or meaning. The same word may represent two meanings – for example, *wake* can mean to *wake up* or a *funeral*. While lemmatization would try to distinguish these two word senses, stemming would incorrectly conflate them.
- Lemmatization is a much more complicated and expensive process that needs to understand the context in which words appear in order to make decisions about what they mean. In practice, stemming appears to be just as effective as lemmatization, but with a much lower cost.

STOP WORDS REMOVAL – PERFORMANCE VERSUS PRECISION

- Back in the early days of information retrieval, disk space and memory were limited to a tiny fraction of what we are accustomed to today. It was essential to make your index as small as possible. Every kilobyte saved meant a significant improvement in performance. Stemming was important, not just for making searches broader and increasing retrieval in the same way that we use it today, but also as a tool for compressing index size.
- Another way to reduce index size is simply to index fewer words. For search purposes, some words are more important than others. A significant reduction in index size can be achieved by indexing only the more important terms.

STOP WORDS REMOVAL – PERFORMANCE VERSUS PRECISION

- So which terms can be left out? We can divide terms roughly into two groups:
  - Low-frequency terms: words that appear in relatively few documents in the collection. Because of their rarity, they have a high value, or weight.
  - High-frequency terms: common words that appear in many documents in the index, such as "the", "and", and “is". These words have a low weight and contribute little to the relevance score.

STOP WORDS REMOVAL – PERFORMANCE VERSUS PRECISION

- Which terms are low or high frequency depend on the documents themselves. The word "and" may be a low-frequency term if all the documents are in Chinese. In a collection of documents about databases, the word "database" may be an high-frequency term with little value as a search term for that particular collection.
- That said, for any language there are words that occur very commonly and that seldom add value to a search.
- Stopwords can usually be filtered out before indexing with little negative impact on retrieval.

SYNONYMS

- While stemming helps to broaden the scope of search by simplifying inflected words to their root form, synonyms broaden the scope by relating concepts and ideas. Perhaps no documents match a query for "*English queen*," but documents that contain "*British monarch*" would probably be considered a good match.
- A user might search for "*the US*" and expect to find documents that contain *United States*, *USA*, *U.S.A.*, *America*, or *the States*. However, they wouldn't expect to see results about *the states of matter* or *state machines*.

SYNONYMS

- This example provides a valuable lesson. It demonstrates how simple it is for a human to distinguish between separate concepts, and how tricky it can be for mere machines. The natural tendency is to try to provide synonyms for every word in the language, to ensure that any document is findable with even the most remotely related terms.
- This is a mistake. In the same way that we prefer light or minimal stemming to aggressive stemming, synonyms should be used only where necessary. Users understand why their results are limited to the words in their search query. They are less understanding when their results seems almost random.

SYNONYMS

- Synonyms can be used to conflate words that have pretty much the same meaning, such as *jump*, *leap*, and *hop*, or *pamphlet*, *leaflet*, and *brochure*. Alternatively, they can be used to make a word more generic.

  For instance, *bird* could be used as a more general synonym for *owl* or *pigeon*, and *adult* could be used for *man* or *woman*.

- Synonyms appear to be a simple concept but they are quite tricky to get right.

TYPOS AND MISSPELLINGS

- We expect a query on structured data like dates and prices to return only documents that match exactly. However, good full-text search shouldn't have the same restriction. Instead, we can widen the net to include words that may match, but use the relevance score to push the better matches to the top of the result set.
- In fact, full-text search that only matches exactly will probably frustrate your users. Wouldn't you expect a search for "*quick brown fox*" to match a document containing "*fast brown foxes*", "*Johnny Walker*" to match "*Johnnie Walker*", or "*Arnold Shcwarzenneger*" to match "*Arnold Schwarzenegger*"?

TYPOS AND MISSPELLINGS

- If documents exist that do contain exactly what the user has queried, they should appear at the top of the result set, but weaker matches can be included further down the list. If no documents match exactly, at least we can show the user potential matches; they may even be what the user originally intended!
- We have already looked at diacritic-free matching in Normalization, Stemming, and Synonyms, but all of those approaches presuppose that words are spelled correctly, or that there is only one way to spell each word.
- Fuzzy matching allows for query-time matching of misspelled words, while phonetic token filters at index time can be used for *sounds-like* matching.

TEXT REPRESENTATION – Vectors

- Binary term-document incidence matrix (also called "one-hot vector" - naïve representation):
  - Each document *i* is represented by a binary vector *Vi*, where |*Vi*| = |*dictionary*|
    If a document contains a term *t*, than the corresponding coordinate of *Vi* will be set to 1, 0 otherwise
- Term-document count matrices (naïve representation):
  - Similar to the previous representation - *Vi* will be set to *n* in position *t* if the document *i* contains *n* occurrences of the term *t*.

TEXT REPRESENTATION – Vectors

- Naïve vector representations do not consider, for example, the ordering of words in a document (they are often referred as Bag-of-Words)
- Bag-of-Words with n-gram:
  - Contiguous sequence of N tokens from a given piece of text
  - Pros: capture local dependency and order
  - Cons: a purely statistical view, increase the vocabulary size

TEXT REPRESENTATION – Weighted Representations

- IDEA: association of weights to the terms that represent a document
- Very frequent words assume a lower weight of significance
- Very frequent and infrequent words are eliminated from the indexes (upper and lower cut-off)
- How to assign weights to terms?
  - TF = Term Frequency
  - IDF = Inverse Document Frequency

TEXT REPRESENTATION – Weighted Representations

- The term frequency $tf_{t,d}$ of term *t* in document *d* is defined as the number of times that *t* occurs in *d*.
- However, pure term frequency is not what we want:
  - A document with 10 occurrences of the term is more relevant than a document with 1 occurrence of the term.
  - But not 10 times more relevant
- Possible solution: normalizing by max occurrency
  - $w_{t,d} = tf_{t,d} / max_{ti}\ tf_{ti,d}$

TEXT REPRESENTATION – Weighted Representations

- The document frequency $df_t \leq N$ of term *t* is defined as the number of documents that contains *t*.
  - df_t is an inverse measure of the informativeness of *t*.
- We define the idf (inverse document frequency) of *t* by
  - $idf_t = \log(N/df_t)$
  - We use log (N/df_t) instead of N/dft to "*dampen*" the effect of idf.

TEXT REPRESENTATION – Weighted Representations

- The tf-idf weight of a term is the product of its tf weight and its idf weight.
  - $w_{t,d} = (tf_{t,d} / max_{ti}\ tf_{ti,d}) \times \log_{10}(N / df_t)$
- tf-idf increases with the number of occurrences within a document.
- tf-idf increases with the rarity of the term in the collection.

TEXT REPRESENTATION – Word Embedding

- Word embedding (advanced representation):
  - Word embeddings are very powerful weighted vector representations that can represent ordering and other relationships among terms (such as semantic relationships).
- They can be resistant to misspelling and they can allow a machine to “understand" whether two words are synonyms (mapping those words in similar vectors).
- They are obtained through *Deep Learning* techniques.
- Beyond the scope of this tutorial. (Be curious!)