Text Mining and Search

Alberto Filosa

29/9/2020

Indice

1 Introduction

> Text mining is a burgeoning new field that attempts to glean meaningful information from natural language text. It may be loosely characterized as the process of analyzing text to extract information that is useful for particular purposes.

Text mining is generally used to denote any system that analyzes large quantities of text and detects lexical or linguistic patterns extracting useful information.

It is generally used for sentiment analysis, document summarization, news and other recommendation and text analytics.

Data mining can be more characterized as the extraction of implicit, unknown, and potentially useful information from data. With text mining, the information extracted is clearly and explicit in the text.

There are several problems about Text Mining: first, document in an unstructured form are not accessible to be used by computers, many data are not well organized and natural language text contains ambiguities on a lexical, syntactical and semantic levels.

The Information Retrieval make it easier to find things on the Web because it finds useful answers in a collection.

The main tasks affected by Text Mining are:

- *Text Summarization*, that produce a condensed representation of the input text;
- *Information Retrieval*, that identifies the most relevant documents to the query (used in Web Search Engine);
- *Content Based Recommender System*, that define the user profile and suggests the affinity of the research;
- *Text Classification*, that define categories according to their content (e.i. Sentiment Analysis);
- *Document Clustering*, an unsupervised learning in which there is no predefined classes, but only groups of document that share the similar topics.

1.1 Basic Text Processing

Basic Text Processing is often the first step in any text mining application. It consists of several layers of simple processing such as tokenization, lemmatization or normalization. The purpose of analyzing texts can be either predictive or exploratory.

Predictive Analysis develope computer programs that automatically detect a particular concept within a span of text. Main examples are:

- Opinion Mining automatically detect if a sentence is positive or negative;
- Sentiment Analysis automatically detect the emotional state of the author in a text;
- Bias detection automatically detect if an author favors a particular viewpoint;
- Information Extraction automatically detect that a short sequence of words belongs to a particularly entity type;
- Relation Learning automatically detect pairs of entities that share a particular relation;
- Text Driven Forecasting monitors incoming text and making prediction about external events or trends;
- Temporal Summarization monitors incoming text about a news event and predicting whether a sentence should be included in an on-going summary of the event.

Exploratory Analysis develop computer programs that automatically discover interesting and useful patterns in text collections.

A document is a list of text, structure, other media such as images and sounds, and metadata. *Metadata* associated with a document is data related to the document and it consists of descriptive, related to the creation of the document, and semantic metadata, relate to the subject dealt with in the document.

Metadata is information about information, cataloging and descriptive information structured to allow pages to be properly searched by computers.

2 Text Processing

The aim of *Text Processing* is extract and identifying the corpus of the text. When defining a collection, it is necessary read the document, so make it processable and understand the worlds in a semantic level. Terms are the basic features of text and they collected in a bag-of-world.

Tokenization is the process of demarcating and possibly classifying sections of a string of input characters. A token is an instance of a sequence of characters, so each of these is a candidate to be a meaningful term after further processing. There are many problems about this procedure: firstly, how split a city name like `San Francisco` or `Los Angeles`; secondly, there are some problem tokenizing hyphenated sequence; numbers, recognized in different terms, and language issues (like German or Arabic language). Basically, the tokenizations consists in breaking a stream of text into meaningful units that depends on language, corpus or even context.

`It's not straight-forward to perform so-called "tokenization"`

`It`, `'`, `s`, `not`, `straight`, `-`, `forward to`, `perform`, `so`, `-`, `called`, `"`, `tokenization`, `.`

To solve these problem is useful use Regular Expression or Statistical Method to detect the token in a specific sentence. They explore rich feature to decide where the boundary of a word is.

A *Stop Word* is a word, usually one of a series in a stop list, that is to be ignored by a Search Engine (e.g. `the`, `a`, `and`, `to`, `be`, etc.). However, Web SE do not use stop lists due to the good compression techniques and needed in most of search queries, such as song titles of relational queries.

The *Normalization* is the process of mapping variants of a term to a single, standardized form. It may depends on the language an so is intertwined with language detection. In Search Engines is necessary to normalizes indexed text as well as query terms identically.

Case folding is the process reducing all letters to lower case. There are some exception, such as proper noun or acronymous, but often the best choice is to lower case everything.

Thesauri is the process to handle synonyms and homonyms in text sentences. It can be handled by hand-construct equivalence classes, rewrite the word to form equivalence-class terms or in Search Engines it can expand a query.

Soundex is an approach which forms equivalence classes of words based on phonetic heuristic, useful for spelling mistakes.

Lemmization is the process to reduce variant forms to base form:

`am`, `are`, `is` → `be`

`car`, `cars`, `car's`, `cars'` $\rightarrow$`car`

It implies doing proper reduction to dictionary headword form. A crude affix chopping is *Stemming*, that reduce inflected or derived words to their root form:

`automate`, `automatic`, `automation` → `automat`

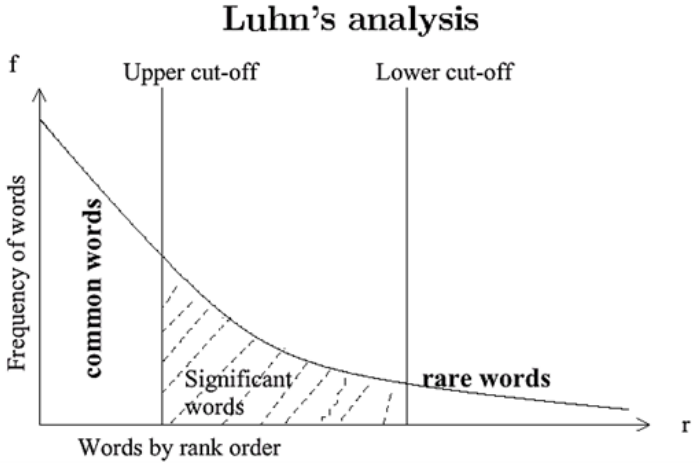
3 Text Representation

Text Representation is the process to representing the text with graphical method. The simplest way is the binary term-document weighting, in which each document can be represented by a set of term (or binary vector) $\in \{0, 1\}$. Another graphical method is the count matrix, that consider the number of occurrences of a term in a document, but this not consider the ordering of words in a document.

In natural language, there are few frequent terms and terms; the *Zipf's Law* describes the frequency, $f(w)$, of an event in a set according to its rank, approximately constant (usually in an increasing order).

$$f(w) \propto \frac{1}{r(w)} = \frac{K}{r(w)}$$

In particular, head words take large position of occurrences, but they are semantically meaningless (e.g. `the`, `a`, `we`), while tail words take major portion of vocabulary, but they rarely occur in documents (e.g. `dextrosinistral`).



According to this *Luhn's Analysis*, words do not describe document content with the same accuracy: word importance depends not only on frequency but also on position. So frequencies have to be weighted according to their position: two cut-offs are experimentally signed to discriminate very common or very rare words. Most significant words are those between the two cut-offs, those that are neither common nor rare. It automatically generates a document specific stop list, the most frequent words.

Weights are assigned according to corpus-wise, carrying information about the document content, and document-wise, so not all terms are equally important.

The *Term Frequency* $tf_{t,d}\}$ represent the frequency of the term t in the document d, often normalized by document length:

$$w_{t,d} = \frac{tf_{t,d}}{|d|}$$

To prevent bias towards longer documents, it doesn't consider the document length but the frequency of the most occurring word in the document:

$$w_{t,d} = \frac{tf_{t,d}}{\max\limits_{t_i \in d}\{tf_{t_i,d}\}}$$

The *Inverse Document Frequency* idf_t represents the inverse of the informativeness of the document for a term t:

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

df_t is the number of documents that contains t and N is the total number of documents.

The *Weights*, called $tf\text{-}idf$ weights, are the product of the two indices:

$$w_{t,d} = \frac{tf_{t,d}}{\max\limits_{t_i \in d}\{tf_{t_i,d}\}} \cdot \log\left(\frac{N}{df_t}\right)$$

The weight w increases with the number of occurrences within a document and with the rarity of the term in the collection.