Information retrieval

Francesco Tomaselli

March 4, 2021

Contents

1 Vector space model

Text transformation The first step to process queries on text is to transform it into a model that can be computed by a machine.

Such a model should be suitable for any kind of text, should be able to capture the semantic of words and then should support the notion of *text similarity.*

The main idea of a vector space model is to map documents in a vector, so they appear as points in a certain space.

For instance, we can collect some words and index them, then build a vector for each document

$$[x_0, \ldots, x_n]$$

Where the value x_i stores the occurrences of the word indexed with i in that document.

Similarity This model somehow implements the notion of text similarity, represented by some sort of distance function between two documents in space, perhaps the *Euclidean distance* or the difference between the angle of the two vectors. There are a lot of distance measures, and they vary in applications. Some of them assume normalization in the document vectors.

Document-Term matrix If we arrange document vectors in a vector we obtain the *document-term matrix*, or the *term-document matrix* if we take the opposite. The number of dimensions in the space is equal to the number of words in the vocabulary.
That can be a problem without some sort of lemmatisation and stemming, as sparsity and dimension of the matrix increases.

Vector length problem By considering the length of the vector in a similarity measure, we are introducing some problems, let's just consider a text and its abstract, the Euclidean distance would be huge, as most of the words do not appear that often.
To solve the problem we can consider the *Cosine similarity* or normalize vector lengths.

The main phases in the generation of a vector space models are: tokenization, normalization, weight, and indexing.

1.1 Tokenization

The goal is to split the text in words, for instance we could split the document at the spaces.

In practice this can be obtained with:

- *Regex*
- *Corpus* based
- *Machine* Learning based

1.2 Normalization

The idea is to normalize each token, that could mean writing verbs in the normal form, or plurals at the singular.

Stemming One form of normalization is stemming, that simply truncate words to obtain singular or infinite forms. The problems here could be creation of meaningless words and also close words in meaning could be completely different. An example is the *Porter* stemming algorithm.

Lemmatisation Another way to normalize is to lemmatise, It's similar to stemming, but the words are replaced with the dictionary root, this also has a problem, indeed sometime mapping of same words to different lemmas can happen. A way to lemmatise is to use a *Dictionary based* lemmatisation, or something more complicated such as *Wordnet*.

Wordnet Wordnet is a large lexical database, verbs, nouns, adjectives and adverbs are grouped into cognitive synonyms, called synsets, each expressing a different concept. Synsets are connected to each other by conceptual relations and they are also connected to a lemma.

1.3 Term Weighting

Weighting the words is a crucial point in text processing, an easy way could be counting the frequency of the terms in the document.

Term frequency An example of weighing is the term frequency, which counts the frequency of a term given a document:

- *Boolean*: so a 0 or 1 if the word is present or not
- *Natural*: $tf_{t,d}$ = count of a specific word
- *Log*: $1 + \log(tf_{t,d})$
- *Augmented*: $0.5 + \frac{0.5 tf_{t,d}}{\max_{t'} tf_{t',d}}$
- *Log average*: $0.5 + \frac{0.5 \log(tf_{t,d})}{1+\log(avg_{t'} tf_{t',d})}$
- *Max tf norm*: $k + (1-k)\frac{tf_{t,d}}{tf_{\max}(d)}$

Stop words problem A problem in this model is the excessive presence of stop words and punctuations. To compensate, we can remove them in the normalization phase.
But sometimes the stop words can be important, for instance with phrasal verbs. A solution can be to count the number of documents that contains that given word.

Inverse document frequency The idea is to count the number of documents that contains a given words, and to prefer less frequent words:

$$idf = \log \frac{N}{df_t}$$

There are many variants:

- *Smooth*: $\log(\frac{N}{1+df_t}) + 1$
- *Max*: $\log(\frac{\max_{t' \in d} df_t}{1+df_t}) + 1$
- *Probabilistic*: $\log \frac{N-df_t}{df_t}$

TfIdf By multyplying term frequency and inverse term frequency we obtain this metric.

$$TfIdf(t,d) = tf_{t,d} \cdot idf_t$$

Regarding the value of terms:

- the terms with a high *TfIdf* appears in a small number of documents or a lot in a high number of documents
- the metric is low when a term appears a few times in a document or when it appears in many documents