Search engine using TF-IDF

1 Introduction

TF-IDF is a weighting system that assigns each term in a document a weight based on its term frequency (tf) and inverse document frequency (IDF). Terms with higher weight scores are considered to be more important.

2 Description

- Given a document d, the task of separating the text by words called tokens is called as tokenization, in addition special characters can be deleted, such as punctuation and convert the characters to lowercase. Example: Tipos: [ademas] [de] [inscri bir] [web] [mining] [inscribí] [data] [mining]
- To reduce the dimension of the vocabulary and eliminate terms that do not provide information, the terms that appear very frequently in most documents (stopwords) are eliminated. Like articles, pronouns, prepositions and conjunctions.
  Example: [el, la , ellos, ellas, nosotros, un, una , de, con , a , ademas, ya, y, muy, otro, cuando, cuanto]
- **Stemming**, process where the terms are transformed to their root to reduce the dimension of the vocabulary. It is done based on a set of word reduction rules. In our case we use the Porter Algorithm.
- Eliminating the stopwords and making use of stemming, the vocabulary of document d is as follows:

termId	value
t1	data
t2	inscrib
t3	mining
t4	web

- Let D be a collection of documents and V the vocabulary of all the terms extracted from the collection:
- The posting list of a term is the list of all the documents in which it appears at least once.
- An inverted index into a dictionary data structure that maps each term $t_i \in V$ to its postlist.

2.1 Vectorial Model

- In order to rank queries, or measure the similarity between two documents, we need a similarity metric.
- We represent documents as vectors of terms, where each term is a dimension
- These types of models are called Bag of Words. We lose the order of the words.
- The value of each dimension is a weight that represents the relevance of the term t_i in the document d

$$d_j \rightarrow \overrightarrow{d_j} = (w(t_1, d_j), \ldots, w(t_{|v|}, d_j))$$

2.2 Term Frequency - Inverted Document Frequency

- We define $Tf_{i,j}$, as te frequency of the term t_i in the document d_j.
- If for example a term appers ten times, then he must show more information.
- We can normalize the max frequency of the term in the document

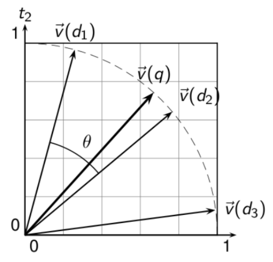
$$Tf_{i,j} = \frac{f_{i,j}}{max f_{i,j}}$$

- For example, the document 'El señor alcalde de Malloco'. The term' Malloco' appears in lesser documents than 'alcalde', for the reason the first word gives more information.

2.3 Similarity between vectors

- Dada la distancia euclideana entre dos documentos, lo más usado es usar el coseno del ángulo entre los vectores como medida de similitud.
- If the documents are equal, the angle is 0 and cosine 1. On the other hand, if they are orthogonal, the cosine is 0.
- The vectors must be normalized by their Euclidean norm $\|d\|_2$, the similarity is calculated as follows

$$\cos(d_1.d_2) = \frac{d_1.d_2}{|d_1| \times |d_2|} = \frac{\sum_{i=1}^{|V|}(w(t_i,d_1) \times w(t_i,d_1))}{\sqrt{\sum_{i=1}^{|V|} w(t_i,d_1)^2} \times \sqrt{\sum_{i=1}^{|V|} w(t_i,d_2)^2}}$$



2.4 Ejemplo

- Suppose we have 3 documents, which are formed from the following sequences of terms:

$$d_1 \rightarrow t_4 t_3 t_1 t_4$$

$$d_2 \rightarrow t_5 t_4 t_2 t_3 t_5$$

$$d_3 \rightarrow t_2 t_1 t_4 t_4$$

- We construct a term-document matrix of dimension 5×3 using the simple weights $Tf - idf$ (without normalization).
- We fill the cells with the given values.

	d1	d2	d3
t1	0.176	0.000	0.176
t2	0.000	0.176	0.176
t3	0.176	0.176	0.000
t4	0.000	0.000	0.000
t5	0.000	0.954	0.000

3 Implementation of the search engine

3.1 First troubles

- Large storage space, having close to half a million news, we could not store the entire matrix with all the points.
- Searches longer than 2 hours, calculating disk access, search filtering, and the large amount of data, our first queries came to light after just over 2 hours

3.2 Solutions

- We decided not to store unnecessary information, so topologically we reduce the size of the vectors in bytes, but not in dimension.
  Example: Given our matrix in $\mathbb{R}^n$, we decide to store the vector as follows:

  Indice_de_la_noticia : { id_palabra_1 : valor, ... , id_palabra_n: valor}

  In this way, the data storage space is considerably reduced.
- We pre-calculate the IDF, TF and the norm of each of our vectors, in this way we do not take time to perform calculations in full search.
- Even with the pre-calculation, the queries were delayed, so we created a daemon in GNU/Linux, in such a way that it allows us to keep our data in RAM memory, the purpose of this technique is to allow faster access to the data, since if we access by Hard Disk, our latency and response time take much longer.

3.3 Tools

- The web application was developed in the Ruby 2.1 programming language, under the Rails 4 framework
- The daemon was developed under the python 2.7 programming language, with the help of the Twisted library, this library helps us to communicate between the client and the server, in this way we obtain the search results and show it in the Web Application.

3.4 Results

- Tests were performed on a PC: AMD a6, 4-core, with 8 GB RAM, GNU/Linux Ubuntu x86_64 operating system
- After optimizations, pre-calculations and committing our data to memory, we got a gratifying delay between 1-3 seconds per search.