Implementation of an Information Retrieval System

Luka Č., Edoardo G., Alberto M.
Radboud University Nijmegen

ABSTRACT

Relevance represents the most important concept in Information Retrieval. However, while for the human beings the process to establish a document relevance with respect to a query is internalized, the machines need specific models, representations and measures to quantify and evaluate relevance. In this paper, we describe the implementation of an Information Retrieval System containing some "instantiations" of the Vector Space Model, able to assess and rank the relevance of the documents for a certain query, based on the concept of similarity. The goal of the project lies in the willingness of understanding the mechanisms, the relationships and the difficulties related to the implementation of such a system, with a focus aimed on the features of each instantation.

KEYWORDS

Information Retrieval System, relevance, Vector Space Model, similarity

ACM Reference Format:
Luka Č., Edoardo G., Alberto M.. 2019. Implementation of an Information Retrieval System. In *Radboud '19: Information Retrieval, June 23–12, 2019, Nijmegen, NL.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

The **Vector Space Model** belongs to the class of Similarity Based Models. In this context, as the name implies, the relevance of a document with respect to a certain query is defined by the concept of similarity. As a matter of fact, the Ranking Function, which is designed to evaluate the relevance of each document, is built up as the similarity measure between documents and query. As can be seen in Figure 1, in the Vector Space Model, the entire set of terms defines a high dimensional space (each term defines a dimension) in which both query and documents are projected as vectors with a direction and a verse. There are some assumptions and several implementations we can test.

- First, we need a word-representation method, in order to set up the way the model administrates terms within the documents.
- Second, we should think about the value we could assign to each term present in a document (binary values for only presence and absence, frequency of the term, et cetera).

Radboud '19, June 23–12, 2019, Nijmegen, NL

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

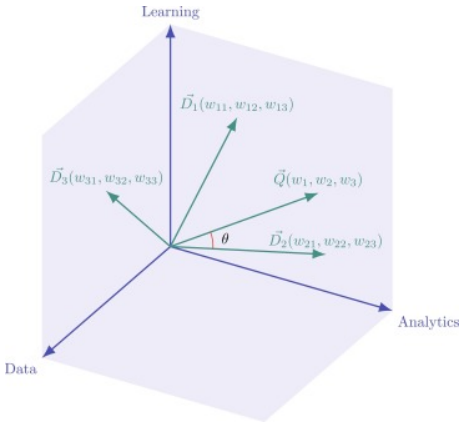

Figure 1: Example of Vector Space Model, each term defines a dimension.

- Last, we must provide a similarity function, in order to compute the relevance score of each document with respect to the query.

During the construction of our system, we have performed several trials in order to get and establish the most appropriate methods for our purpose.

In this way, we have tried to deeply understand the meaning of "improvement", recalling step by step the results and the discoveries made over the years by researchers in this field of interest.

Throughout the implementation of the system, we have also performed some pre-processing methods on the entire collections of texts. It concerned important steps, useful to reduce the noise and improve the quality of the data.

At the end, once having built each instance of the model, we needed a consistent system to evaluate the results. At that point, we would obtain some ranking of documents based on the input query so that for each instance, we would get the most relevant documents according to that particular query. To compare the results achieved at the end of the process and infer some helpful conclusions; it is necessary for us, however, to know *a priori* which are the best (most pertinent) documents for the query we provided.

2 SYSTEM OVERVIEW

The Information system implementation has been carried out by the use of the software Python and the collection of books from Project Gutenberg. In this paper we will treat five different instantiations of the Vector Space Model [4]. The development of the five steps is preceded by some pre-processing methods. In particular, once we fixed the documents and the corpus as a set of vectors, we adopted some common transformations:

- All the terms have been converted to lower case;

- Removal of all so-called **"Stop Words"**, i.e. most common terms in the language, such as articles, prepositions, etc;
- All the numbers have been transformed into their word representations;
- Removal of the words composed by a single letter;
- Implementation of the **Stemming method**, i.e. reduction of the (derived) words into their stem;
- Implementation of the **Lemmatization method**, i.e. conversion of a word into its base form.

2.1 First Instantiation

The first implementation of the Vector Space Model has been conducted considering the most basic assumptions. The model assumes as words depiction the **bag-of-words representation** of text. This method allows us to get the score of a query q with respect to a document d relying on the score achieved by each word in the query. In this way, a text is represented as a multi-set of its words, keeping the multiplicity but disregarding grammar rules.

The assigned value for each word is defined by **the bit-vector representation**, in which every vector has the same length and each query or document term will get the value 1 if the term is present or 0 if it isn't.

Eventually, the **Cosine Similarity** will be set as the similarity function, i.e. cosine of the angle between the two vectors (query and document).

$$cos(q, d) = \frac{q \cdot d}{\|q\| \, \|d\|} \tag{1}$$

Thus, in this instantiation we have:

- Bag-of-words representation
- Bit-Vector representation
- Cosine-Similarity

2.2 Second Instantiation

In the first step we have considered only the presence or absence of a certain term as its weight inside the similarity function. In this step the improvement is quite easy to implement. In fact, we would like to consider not only the presence, but also how many times the term appears in the document (or query). Hence, we can introduce the **Term Frequency** to establish a more concrete and useful criterion to calculate the similarity function:

$$TF(w, d) = count(w, d) \tag{2}$$

where w coincides with the word and d with the document. Thus, we have:

- Bag-of-words representation
- Term-Frequency representation
- Cosine-Similarity

2.3 Third Instantiation

The Term frequency itself can be misleading. It could often happen that documents contain very common words, which were not deleted by the stop words method. Matching these words between query and document is not very significant for our purpose. One approach to avoid this unpleasant result is to introduce the **Inverse Document Frequency**.

$$IDF(w) = \frac{M + 1}{df(w)} \tag{3}$$

Where M is the total number of document in the collection, while denominator shows the total number of documents containing the word w.

This adjustment leads to assigning an increasing weight (reward) to less common words, i.e. ones that appear in fewer documents.

2.4 Fourth Instantiation

At this point, the improvement implemented before assigns a reasonable weight to the vector elements (terms), that is the **TF-IDF weight**. According to the correction above, though, very rare terms reach disproportionate influence to the overall score of the document so we need to control the term frequency. A very common way, in these cases, is to apply a transformation to the interest function (here, the TF). Generally, when we need to restrict the range of some function, we have two basic options:

- Apply a Logarithm;
- Set an upper bound.

Based on the state-of-art methods, we could have decided to pick the second one. In fact, by introducing the parameter k and with some adjustments, is possible to control the upper bound of the function and build up the so-called **TF BM25**. However, we preferred to implement and observe the results of both of these methods:

- **Logarithmic Transformation**

$$y = \log(1 + x); \tag{4}$$

- **BM25 TF**

$$y = \frac{(k + 1)x}{x + k}. \tag{5}$$

In our case the term x is the TF. The fraction $\frac{x}{x+k}$ will never exceed 1, thus k+1 will be the upper bound of the function.

Moreover, one more advantage of this last transformation is that we can return to the instantiations above defined, just varying the parameter k value.

$k = 0 \implies$ bit-vector representation.
$k >> 0 \implies$ TF-IDF representation.

2.5 Fifth Instantiation

The last improvement concerns the idea of document length. It is easy to imagine that long documents share a higher probability to match terms in the query and thus to get higher scores. One immediate technique to control the document dimension is to apply a normalization. In particular, it sounds clever to penalize long documents, rewarding short ones in the meanwhile. One of the most used method is the **Pivot Length Normalization**. The idea behind it is to exploit the average documents length as a pivot.

$$Normalizer = 1 - b + b\frac{|d|}{avdl} \tag{6}$$

Through this step, we are going to penalize longer-than-average documents and reward shorter ones. This is possible by knowing

in advance the average length of all documents (*avdl*) as well as the length of the current document *d*, i.e. $|d|$.

The entire method is governed by the parameter *b*, which, varying from 0 and 1, controls the degree of normalization.
In fact, an increasing value of *b* implies a strong normalization. At the end, with this last improvement, we were able to define the model known as **BM25 Okapi**.

3 DATASET AND IMPLEMENTATION

The dataset itself consists of 148 books divided among 6 categories, namely: *Architecture, Astronomy, Biology, Chemistry, Computer Science*, and *Philosophy*. All books have been collected manually from the *Project Gutenberg* website and separated into thematically similar categories [1]. For the categories above mentioned, there are respectively 17, 25, 24, 19, 43, and 20 books for each.

As mentioned in the first section, we have provided support for several pre-processing techniques. After some experimentation, we have decided to use the removal of stop words, removal of letters, transformation of numbers into their word representations, and lemmatization for converting the words into their base forms [2]. For this, we used packages *nltk* and *num2words* available in Python, as well as some simple Regex patterns.

The complete implementation and the dataset used will be made available on our main project website [5].

4 EVALUATION

As said before, we need to assess the performance of our models. Evaluation is a very important part of implementing an IR system, since we should be aware about the improvement brought by any instance. Each instantiation of our system gives us a list of scored books within the entire collection, sorted by decreasing order of similarity, according to the input query. The next step consists in the verification of the consistency of such results.
We decided to carry out this step by relying on our *a priori* knowledge, applying the so-called **Offline Evaluation** technique. A very simple idea to measure the effectiveness of the models can be to pick one of the categories and use it as the query. We will have a list of retrieved documents and a certain number of relevant documents. For each query, we will fix the number of relevant documents to be retrieved as equal to the number of documents within the category specified in the query (e.g. for the query: "Astronomy", the number of documents to be retrieved will be equal to the number of documents in the *Astronomy* category). This method allows us to know which books have to be considered as relevant (if they belong to the same category as the query) or not relevant (the ones belonging to other categories). At this point, once all the models have been run, we can define as "true" the documents correctly classified by the system, and as "false" the rest of them. These definitions allow us to build the Confusion Matrix and to compute some important measures useful for determining the validity of our model, such as Accuracy and Precision.

Thus we have:

From which we can define:

		Action	
		Retrieved	Not retrieved
Doc	Relevant	*a*	*b*
	Not relevant	*c*	*d*

Figure 2: Confusion Matrix

- Accuracy:

$$A = \frac{a+d}{a+b+c+d} \tag{7}$$

- Precision:

$$P = \frac{a}{a+c} \tag{8}$$

In particular, Accuracy is defined as the total number of correct classifications (true positives plus true negatives) divided by the total number of documents, while Precision is the fraction of retrieved documents which are relevant to the query (true positives divided by true plus false positives). So, the first one is the percentage of the correct classifications made by our system, while the latter is focused on *a*, i.e. the total number of retrieved relevant documents.

5 RESULTS

In this section we are going to analyze the results returned by each model described in Section 2. It should be emphasized that each instance will show 6 different results, one for each category contained in the collection. Two of the used instances (namely BM25 and BM25 Okapi) have parameters, while others do not. For both of them, the parameter k has been fixed to 1.4, while additionally for Okapi, we used 0.75 as the value for b [3].

Originally, each table contained Recall and F1 scores in addition to Accuracy and Precision but after obtaining the results, we have decided to remove them from the results. This was done because in every experiment, the Precision and Recall we have obtained have had the same value – this is the result of our experimental setup: while retrieving documents for a certain query, the number of false positive documents retrieved will match the number of false negative documents which the model should have retrieved but didn't. Recall and F1 therefore have the same value as Precision, and have thus been removed.

Bit-Vector Identity		
Query	Accuracy	Precision
Architecture	0.7838	0.0588
Astronomy	0.7973	0.4000
Biology	0.8784	0.6250
Chemistry	0.8514	0.4211
Computer	0.8919	0.8140
Philosophy	0.8378	0.4000
Average	**0.8401**	**0.4532**

TF Identity		
Architecture	0.8919	0.5294
Astronomy	0.9324	0.8000
Biology	0.8919	0.6667
Chemistry	0.9324	0.7368
Computer	0.8378	0.7209
Philosophy	0.9054	0.6500
Average	**0.8986**	**0.6840**
TF-IDF Identity		
Architecture	0.9054	0.5882
Astronomy	0.9324	0.8000
Biology	0.8919	0.6667
Chemistry	0.9054	0.6316
Computer	0.7568	0.5814
Philosophy	0.9189	0.7000
Average	**0.8851**	**0.6613**
TF-IDF BM25		
Architecture	0.8378	0.2941
Astronomy	0.9054	0.7200
Biology	0.8784	0.6250
Chemistry	0.8784	0.5263
Computer	0.7838	0.6279
Philosophy	0.8784	0.5500
Average	**0.8604**	**0.5565**
TF-IDF BM25 Okapi		
Query	Accuracy	Precision
Architecture	0.8649	0.4118
Astronomy	0.9324	0.8000
Biology	0.8919	0.6667
Chemistry	0.9054	0.6316
Computer	0.7838	0.6279
Philosophy	0.8919	0.6000
Average	**0.8784**	**0.6230**
TF-IDF Log		
Architecture	0.8649	0.4118
Astronomy	0.9459	0.8400
Biology	0.8919	0.6667
Chemistry	0.9324	0.7368
Computer	0.7838	0.6279
Philosophy	0.9189	0.7000
Average	**0.8896**	**0.6639**

From the results above, we can see a big improvement in TF model with respect to the Bit-Vector model. Simply by counting the words instead of keeping a "boolean flag" about their presence, precision is increased by more than 20%. Considering TF-IDF models, some unexpected results can be seen. Contrary to our intuition and expectations, the results are overall not better than the simpler TF model. TF-IDF builds upon TF by introducing Inverse Document Frequency whose purpose it is to reward words appearing in fewer documents and vice versa. The unexpected results indicate that the IDF component has not been particularly effective in doing so, at least for this particular collection of documents. However, among the last three TF-IDF methods (i.e. transformations), a general trend can be noted: Log transformation produces better results than the BM25 Okapi transformation which, in turn, produces better results than the BM25 transformation, showing a steady progress.

6 CONCLUSION

In the development of this project we wanted to retrace the path of improvement carried out by many researchers in order to build a retrieval system based on similarity. We have implemented several models to represent documents, starting from Bit-Vector (the simplest one), proceeding with TF (Term Frequency), then TF-IDF (including BM25 and logarithmic transformations), and ending with the BM25 Okapi, which is considered to be the most advanced among these instantiations.

The system gives as output a list of the documents ranked in descending order by their similarity with the query, computed through the *cosine similarity*. In order to have an evaluation about the performances of the system, we chose to select as queries the categories in which our collection of documents is divided, and then see if the model is able to retrieve the correct books, i.e. those which belong to that category. Finally we were able to construct the confusion matrix and to compute *Accuracy* and *Precision*.

These measures, defined in Section 4, helped us to estimate the weight of such enhancements. We have specified, in this regard, that Precision, Recall and Accuracy are some of the most common and useful indices, but we cannot use all of of them, given their nature and the way we set the number of documents to be retrieved.

As we expected, the simplest representation – Bit-Vector – gives the lowest performances; then, the more complex the model is, the better the results are, except for the fact that the logarithmic transformation of TF-IDF performs better than the BM25 Okapi.

Focusing our attention to the improvements, it is possible to inspect interesting effects. The highest gap in terms of Accuracy and especially Precision is obtained by the handing from the first to the second instance. In particular, the Term Frequency seems to remark its strong utility, boosting both the indices in almost all the categories available. The IDF weight, such as all the further improvements, on the contrary, seems to only fix the breakthrough carried out by the first adjustment.

In conclusion, we were able to construct our own retrieval system and to carry out experiments to evaluate it; we obtained positive and promising results that confirmed the effectiveness of our model.

7 FUTURE WORK

In terms of future improvements, we would like to dive deeper into the problem of TF-IDF. We think it is an interesting and unexpected twist that the more complex model with the IDF component produces poorer results than the simpler one, relying solely on the frequency component. We would like to test our models on different datasets to see if we can reproduce the results, or if the difference in quality, size, and content of the dataset would change the current trend of results.

REFERENCES

[1] [n.d.]. *Project Gutenberg.* Retrieved Dec 23, 2019 from http://www.gutenberg.org/

[2] 2019. *TF-IDF from scratch in python on real world dataset.* Retrieved Dec 23, 2019 from https://towardsdatascience.com/tf-idf-for-document-ranking-from-scratch-in-python-on-real-world-dataset-796d339a4089

[3] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2018. *Introduction to information retrieval.* Cambridge University Press.

[4] ChengXiang Zhai and Sean Massung. 2016. *Text data management and analysis: a practical introduction to information retrieval and text mining.* ACM Books.

[5] Luka Čupić, Edoardo Gervasoni, and Alberto Monaco. [n.d.]. *Implementation of an Information Retrieval System.* https://github.com/lukacupic/IR-Project/