University of Sheffield

COM3110 - Text Processing

IR Assignment Report

Lekha Mohta

November 29, 2024

1 Introduction

This report details the implementation and performance evaluation of an information retrieval system based on the Vector Space Model (VSM). The system integrates three term-weighting schemes—binary, term frequency (TF), and term frequency-inverse document frequency (TF-IDF)—along with preprocessing techniques such as stemming and stop-listing. Cosine similarity is used for document ranking. The primary objective of this project is to processes a document collection represented as an inverted index to efficiently retrieve documents relevant to a query. The system's performance is assessed using three evaluation metrics - Precision, Recall, and F-measure - against the CACM test collection.

2 Implementation Description

I started with gaining a thorough understanding of the provided code, from which I gathered the following information:
1) **IR_engine.py** : Serves as the retrieval engine's shell, parsing command line arguments to configure system parameters. Loads queries and inverted indexes from '**IR_data.pickle**' and computes top 10 ranked documents.
2) **my_retriever.py** : Implements core retrieval functionality, managing document ranking and retrieval based on queries.
3) **for_query methods** : Performs retrieval for individual queries returning a list of Doc ID's for relevant docs.
4) **eval_ir.py** : Evaluates performance metrics by comparing the output results file to the gold standard (cacm_gold_std.txt).

2.1 Step 1 : Creating Vector Representation for each Document

Inverted indexes, implemented in '**self.index**', store mappings of terms to document IDs and their frequencies, significantly reducing search operation time complexity and enhancing system performance. '**compute_number_of _documents** identifies all unique document IDs, enabling comprehensive indexing for efficient retrieval. '**compute _document_vectors**' generates document vectors by iterating through terms and their document mappings in the index, constructing a vector for each document that maps terms to their respective counts.

2.2 Step 2 : Implementing Weighting Schemes - Binary, TF, TF-IDF

'**compute_idf**' computes the inverse document frequency for each term in the index. '**compute_document _vectors**' function is improved to generate document vectors and their magnitudes based on the selected term weighting scheme. It processes the inverted index to create a numerical representation of each document, where each dimension corresponds to a term's weighted presence. '**compute_query_vectors**' iterates through the query and applies the specified weighting scheme, to output a dictionary consisting of keys as terms and values as corresponding weights '**compute_query _magnitude**' calculates magnitude of the query vector, which is used for normalization in cosine similarity calculations.

Term Weighting Schemes:
1) **Binary** - Each term is assigned a weight of 1, regardless of its frequency, to indicate presence / absence of the term.
2) **TF** - Each term is assigned a weight equal to its term frequency, which is the number of times the term appears in the query, emphasizing frequently occurring terms.
3) **TF-IDF** - Each term is assigned a weight equal to its TF in the query multiplied by its IDF, enhancing the significance of both frequent and rare terms.

2.3 Step 3 : Calculating Cosine Similarity and Implementing Ranking Algorithm

Cosine similarity calculates the cosine of the angle between query and document vectors. It normalizes vector lengths to focus on term distribution rather than magnitude, making it a robust metric for relevance. It is particularly effective in handling sparse and high-dimensional data. The '**for_query**' function begins by computing the query vector and its magnitude using specialized methods. It then calculates cosine similarity scores for all document vectors by utilizing the '**cosine_similarity**' function. Documents with scores below 0.1 are excluded, and the remaining documents are ranked in descending order of similarity. As Output, it returns the IDs of the top 10 highest-ranked documents.

The '**evaluate_metrics**' function retrieves documents and calculates the true positives - intersection of retrieved documents and the relevant documents. It then computes the Precision, Recall and F-measure and returns them.

3 Results and Analysis

NOTE

-s = Stoplisitng, excluding "non-content" words (a, am, about etc) to reduce size of inverted index

-p = Stemming, reducing words to their root or base form to group similar words together.

Weighting scheme	Preprocessing	Precision	Recall	F-measure	Rel_Retrieved
Binary	None	0.07	0.06	0.06	44
	-s	0.13	0.10	0.11	81
	-p	0.09	0.08	0.08	60
	-s and -p	0.16	0.13	0.15	105
tf	None	0.08	0.06	0.07	49
	-s	0.16	0.13	0.14	103
	-p	0.11	0.09	0.10	73
	-s and -p	0.19	0.15	0.17	121
tfidf	None	0.21	0.16	0.18	130
	-s	0.23	0.17	0.20	138
	-p	0.27	0.21	0.23	166
	-s and -p	0.28	0.22	0.24	172

Table 1: Performance metrics for different weighting schemes and preprocessing methods

Binary Weighting - Adding '-s' and '-p' independently enhances performance metrics, but a combination of both '-s and -p' produces the highest improvement (F-measure increases from 0.06 to 0.15), indicating that preprocessing reduces redundant noise and variation in terms. However, Binary weighting still yields lower scores compared to TF and TF-IDF due to its simplistic representation of term occurrences without accounting for term frequency or informativeness.

TF Weighting - TF weighting outperforms Binary as it incorporates term frequency, making it sensitive to repeated occurrences of important terms in documents. Among application of single preprocessing techniques, '-s' is more effective than '-p', as it removes frequent and uninformative terms, reducing noise in term frequency calculations and improving focus on meaningful content. For instance, -s raises the F-measure from 0.07 to 0.14, while -p achieves 0.10. However, the best results are achieved when both '-s and -p' are applied together, with the F-measure increasing to 0.17 and relevant retrieved documents rising from 49 to 121. This combination ensures optimal term matching and retrieval performance.

TF-IDF Weighting - TF-IDF weighting achieves the best performance among the three schemes, with the highest F-measure (0.24) when both '-s and -p' are applied. This is because TF-IDF accounts for both term frequency and term rarity, giving higher weights to distinctive terms. Preprocessing further enhances results by prioritizing meaningful terms and reducing the influence of redundant ones. The significant increase in retrieved relevant documents (from 130 to 172) highlights TF-IDF's effectiveness in leveraging term informativeness.

Overall, preprocessing techniques, such as stemming and stoplisting, significantly enhance the performance of all three weighting schemes. While Binary weighting is simple but less effective, TF shows notable improvements by incorporating term frequency, making it more sensitive to the importance of repeated terms. However, TF-IDF outperforms both Binary and TF by accounting for term rarity, achieving the best balance of precision and recall. This makes TF-IDF particularly valuable for applications like document retrieval and text classification, where prioritizing distinctive, informative terms is crucial for optimal performance.

NOTE: The formulas for all the above methods are provided in the appendix.

A Mathematical Formulas

A.1 Term Frequency (TF)

The term frequency measures how frequently a term w appears in a document d:

$$\mathrm{TF}_{w,d} = \mathrm{freq}_{w,d}$$

A.2 Dot Product

The dot product of two vectors $\vec{A} = [a_1, a_2, \ldots, a_n]$ and $\vec{B} = [b_1, b_2, \ldots, b_n]$ is defined as:

$$\vec{A} \cdot \vec{B} = \sum_{i=1}^{n} a_i b_i$$

where a_i and b_i are the components of $\vec{A}$ and $\vec{B}$, respectively.

A.3 Inverse Document Frequency (IDF)

The inverse document frequency measures how rare a term w is across the document collection D:

$$\mathrm{IDF}_{w,D} = \log\left(\frac{|D|}{\mathrm{df}_w}\right)$$

where $|D|$ is the total number of documents, and df_w is the number of documents containing the term w.

A.4 TF-IDF

The TF-IDF score combines term frequency and inverse document frequency to assign weights to terms:

$$\text{TF-IDF}_{w,d,D} = \mathrm{TF}_{w,d} \cdot \mathrm{IDF}_{w,D}$$

A.5 Cosine Similarity

Cosine similarity measures the similarity between a query vector $\vec{q}$ and a document vector $\vec{d}$:

$$\cos(\vec{q}, \vec{d}) = \frac{\sum_{i=1}^{n} q_i d_i}{\sqrt{\sum_{i=1}^{n} q_i^2} \cdot \sqrt{\sum_{i=1}^{n} d_i^2}}$$

A.6 Precision

Precision measures the proportion of retrieved documents that are relevant:

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} = \frac{\text{Relevant Documents Retrieved}}{\text{Total Retrieved Documents}}$$

A.7 Recall

Recall measures the proportion of relevant documents that are retrieved:

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} = \frac{\text{Relevant Documents Retrieved}}{\text{Total Relevant Documents}}$$

A.8 F-measure (F1 Score)

The F1 score is the harmonic mean of precision and recall:

$$F_1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

B Graphical Representations

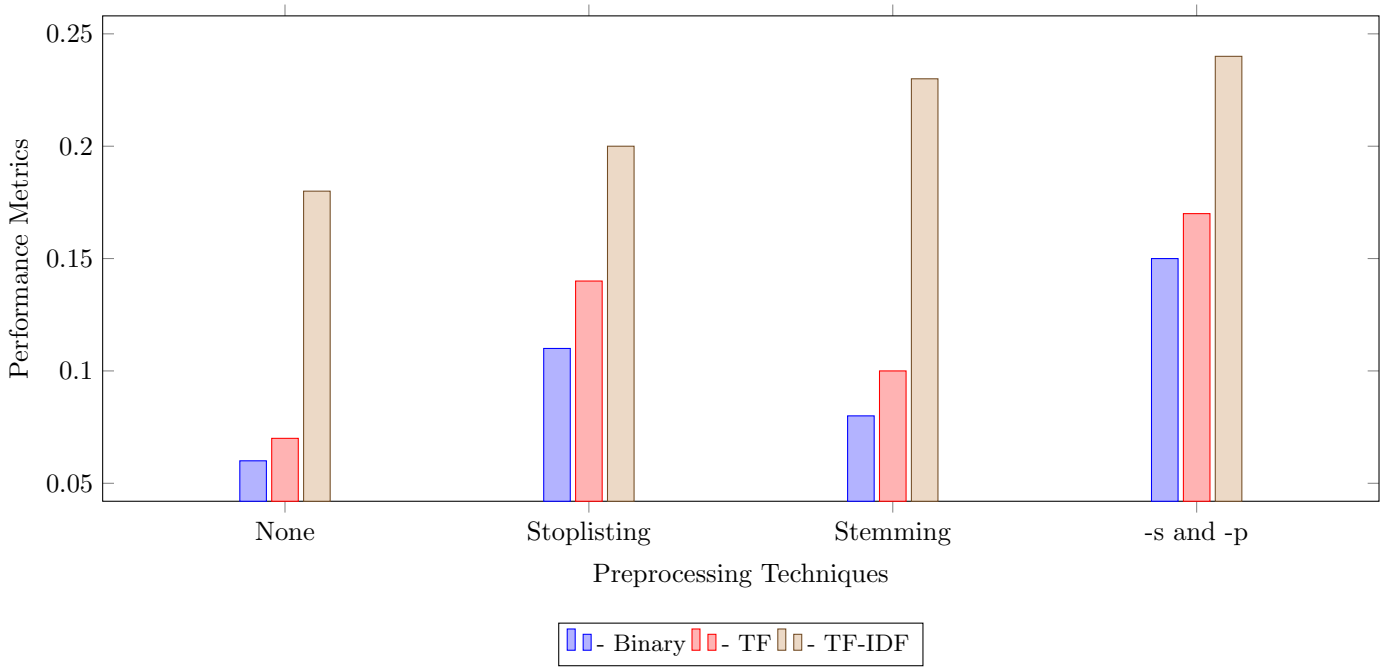

Figure 1: Comparison of F-measure for Preprocessing Techniques and Weighting Schemes