# Document Retrieval System Report

**Author:** Nirmal Eswar Vee

November 2024

## Introduction and Implementation

With this assignment, we were tasked with creating a document retrieval system that could implement the vector space model to rank documents based on how relevant they are to the query. My solution supports different term weightings, and the objective of this assignment was to build this system from the ground up without the use of high-level libraries to further deepen our understanding of the topic.

### Implementation

The implementation followed these 5 steps:

1. Look up the terms of the query in the inverted index:
   - For each query term, check its presence in the inverted index and collect relevant document IDs to form the set of candidate documents.
2. Convert queries into a sparse vector representation:
   - Sparse query vectors are memory-efficient and scalable as they only consider relevant terms.
   - **Binary:** Set the vector element to 1 if the term exists in the query.
   - **TF:** Set the vector element to the term frequency in the query.
   - **TF-IDF:** Compute the product of term frequency and inverse document frequency.
   - Handles edge cases where none of the terms in the query are in the collection.
3. Create sparse document vectors:
   - Sparse document vectors are created with non-zero weights only, with similar term weighting logic.
   - Precompute magnitudes for cosine similarity.
4. Compute cosine similarity between the query vector and each candidate document vector:
   - Compute the dot product and magnitudes of vectors.
   - Handle edge cases where magnitudes are zero.
5. Return the top 10 ranked documents based on similarity scores.

## Performance Results

Table 1 presents the performance results for different configurations:

## Conclusion

From my results, I can conclude that TF-IDF is the most effective weighting scheme as it takes both the document specificity and term frequency into account. Additionally, weightings with stoplists and stemming performed best, which can be seen in how they increased recall by reducing noise and grouping word variations. My results also showed a trend where stoplists seemed to be more effective than stemming alone until TF-IDF, where this was reversed. This suggests that stemming's ability to group word variations becomes increasingly important in the TF-IDF scheme, where there is a heavy focus on document specificity. Binary weighting schemes are inadequate for larger datasets due to their simplicity; this is seen from their poor results with their precision since they treat all terms equally without any regard for their rarity in the collection. Overall, my system achieves results on par with the example file and scales well to large datasets through sparse vector representation, ensuring memory efficiency and computational effectiveness. However, the current overall scores do suggest that this cosine similarity implementation would not work well in the real world due to the scores being low on average for a document retrieval system. This highlights the limitations of the project and that there is more to be done.

| **Configuration** | **Precision** | **Recall** | **F-Measure** |
|---|---|---|---|
| Example results file | 0.27 | 0.21 | 0.24 |
| Binary (no stoplist/stemming) | 0.07 | 0.06 | 0.06 |
| Binary (Stemming/No Stoplist) | 0.09 | 0.08 | 0.08 |
| Binary (Stoplist/No Stemming) | 0.13 | 0.10 | 0.11 |
| Binary (Stoplist + Stemming) | 0.16 | 0.13 | 0.15 |
| TF (No Preprocessing) | 0.08 | 0.06 | 0.07 |
| TF (Stemming/No Stoplist) | 0.11 | 0.09 | 0.10 |
| TF (Stoplist/No Stemming) | 0.17 | 0.13 | 0.15 |
| TF (Stoplist + Stemming) | 0.19 | 0.15 | 0.17 |
| TF-IDF (No Preprocessing) | 0.20 | 0.16 | 0.18 |
| TF-IDF (Stemming/No Stoplist) | 0.26 | 0.21 | 0.23 |
| TF-IDF (Stoplist/No Stemming) | 0.21 | 0.17 | 0.19 |
| TF-IDF (Stoplist + Stemming) | 0.27 | 0.22 | 0.24 |

Table 1: Performance Results for Different Configurations

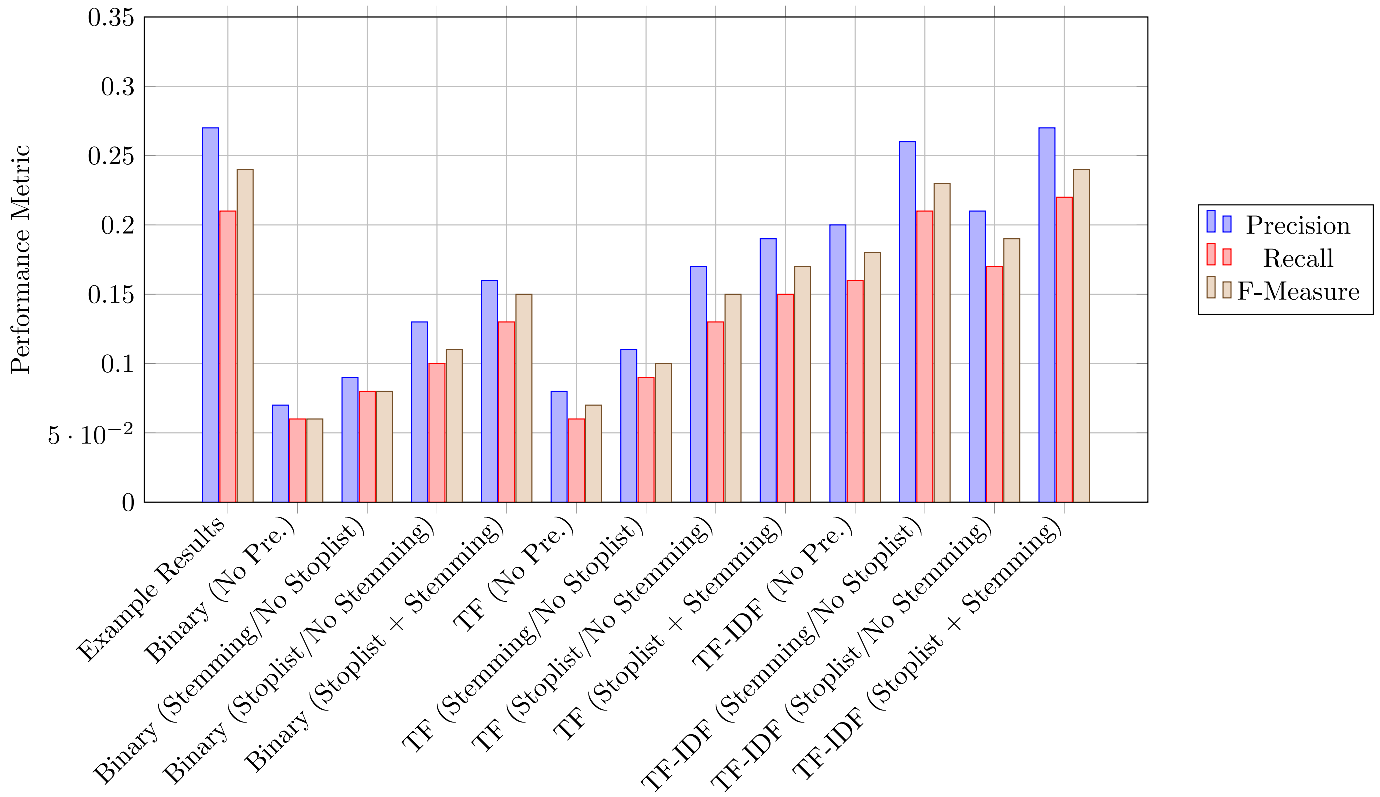

Figure 1: Performance Metrics for All Configurations