INFORMATION RETRIVEAL

ASSIGNEMNT – 1

JESLY JOHNSON U01093321

SANDHYA BODIGE U01100050

CONTENTS

INTRODUCTION

Introduction

This assignment offers a fascinating opportunity to learn more about information retrieval systems. Its main goal is to acquaint with basic ideas like term creation, indexing, and query processing methods. These ideas help to build a fundamental information retrieval system throughout this project. In this assignment we use Python and scikit-learn APIs to build and implement our own system. The assignment, is split into two main sections,

- Phase I: Building the indexing and search application.
- Phase II: Assessing the performance of your search engine

Datasets and Materials

To implement this assignment, we have used the Cranfield Dataset. The Cranfield Dataset contains 225 queries (search terms), 1400 documents, and 1836 evaluations derived from 225 selected queries. The Cranfield Dataset organizes documents and queries into different distinct parts or tags, including (.T) for title, (.A) for author, (.W) for body, (.I) for ID, among others.

Below are the descriptions of each file to implement the assignment:

- **cran.all:** The Cranfield collection corpus file containing various fields, with only the (.W) field/tag being used in this assignment.
- **query.txt:** A collection of 225 queries that can be utilized to assess the retrieval
- performance of the system, with the (.W) field/tag being the relevant one for this assignment.
- **qrels.txt:** Query relevance judgments for each query, with each query accompanied by a set of relevant documents listed in separate lines.
- **README.txt:** A detailed explanation of each column in the dataset files is provided in this file.

Phase I:

Indexer and Query Processor

Phase I: Indexer and Query Processor

In the first phase, we will create a competent Indexer and Query Processor using the Scikit-learn module, a powerful open-source machine learning tool for Python. The primary objective is to provide answers to common queries and examine the Cranfield Dataset.

Indexer Module:

The essential elements of the Indexer module are outlined in the subsequent steps:

1. **Data Preparation:**
   The initial task is to prepare two sets:
   - a corpus of documents (the training set) and
   - a set of queries (the test data).

2. **Text Preprocessing:**
   Tokenization:
   Divide a text document or character sequence into understandable chunks called tokens. Remove any unnecessary characters, like punctuation, throughout this operation.
   Case-Folding:
   Transform all letters in text documents to lowercase to create uniformity.
   This method guarantees that words with varying capitalization instances are treated consistently.
   Stop-Word Removal:
   Eliminate terms that appear frequently, also known as "stop words," from the text.

Vectorization Module

In this assignment we use 2 vectorization modules, Binary Vectorizer and TF-IDF.

1. **Binary Vectorizer**
   - **Usage:**
     We Employ the Binary Vectorizer, which assigns '1' if a term is present in a document/query and '0' otherwise. We use the default Binary Vectorizer.

   - **Configuration:**
     Ensure to enable stop_words and lowercase options. Tokenization is handled by default.

```
# Pipeline for Binary Vectorizer
binary_pipeline = Pipeline([
    ('vectorizer', CountVectorizer(binary=True, stop_words=stop_words, lowercase=True)),
])
```

2. Custom TF-IDF Vectorizer

Background:

One of the most important metrics in text mining is TF-IDF, which indicates how important a word is inside a corpus of documents. The IDF component has historically used a logarithmic scale. We will use a variation in this implementation.

Usage:

TF-IDF Vectorizer that is specifically tailored to work in unison with CountVectorizer. Without using the logarithmic scale, this vectorizer should calculate term frequencies in the conventional way and find the Inverse Document Frequency (IDF), which is the reciprocal of the document frequency.

Configuration:

Term Frequency (TF):

We use the standard CountVectorizer for the initial calculation of term frequencies. It counts the occurrences of each term in each document.

Custom IDF Calculation:

We calculate the IDF as 1 divided by the number of documents containing that term. If the document frequency of a term is zero, leading to an undefined number in the calculation of IDF, set the IDF value of that term to one to avoid division by zero.

Integration:

TF-IDF logic is embedded within a transformer that can be used in conjunction with CountVectorizer. This transformer will redefine the fit_transform method to automatically apply the custom IDF calculation

Additional Features:

We leverage CountVectorizer's functionalities like stop_words & lower_case removal.

```
class CustomIDFTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        # doc freq
        self.document_frequencies_ = np.sum(X != 0, axis=0)
        return self

    def transform(self, X):

        term_frequencies = X


        inverse_document_frequencies = np.where(self.document_frequencies_ != 0, 1 / self.document_frequencies_, 1)


        inverse_document_frequencies = np.squeeze(inverse_document_frequencies)

        # custom IDF
        tfidf = term_frequencies.multiply(inverse_document_frequencies)

        return tfidf
```

Query Processor

After the above preprocessing stage, we will use two distinct similarity metrics within the two vector space models to assess the similarity between searches and documents. **Manhattan Distance** and **Cosine Similarity** are two of the selected similarity metrics.

Cosine Similarity

The Cosine similarity can be calculated using the below formula

$$\cos(d_i, d_j) = \frac{d_i \bullet q_j}{\|d_i\| \|q_j\|} = \frac{\sum_{k=1}^{n} d_{i_k} q_{j_k}}{\sqrt{\sum_{k=1}^{n} d_{i_k}^2} \sqrt{\sum_{k=1}^{n} q_{j_k}^2}}$$

Where, n is the total number of terms,

k represents the term being iterated over.

The cosine similarity is 0 for orthogonal vectors, and 1 for identical vectors

But in our project, it is handled by the built in function and imported to our project.

Manhattan distance

The Manhattan Distance can be calculated using the below formula

$$\text{Manhattan}(d_i, q_j) = \sum_{k=1}^{n} \left| d_{i_k} - q_{j_k} \right|$$

Here, n is the total number of terms, and kk represents the term being iterated over. In this formula:

- d_{i_k} is the TF-IDF weight of term k in document i.
- q_{j_k} is the TF-IDF weight of term k in query j.
- The symbol | . | denotes the absolute value.

The Manhattan distance effectively adds up the absolute differences between the corresponding elements of the two vectors. In the context of text analysis, this measure calculates how far apart two documents, or a document and a query are, considering each term's weight. The smaller the Manhattan distance, the more similar the documents are.

This function is used in the assignment by using the default Manhattan distance by importing the necessary libraries.

Phase II:

Evaluation

Evaluation

We fetch the indices of the top 10 relevant documents from each query (225) for both vector space models (TFIDF and Binary) and both distance measures (Manhattan distance and Cosine Similarity). We end up with four lists, consisting of 225 queries each, where each query item contains the 10 most relevant documents for each model-measure configuration. The final shape of each of the data structures will be (225,10).

Purpose of Evaluation:

- Evaluating the Accuracy of Retrieval:
  Assess the system's capacity to retrieve relevant documents in response to user inquiries.

- Analyzing Vector Space Models:
  Assess the efficiency of two vector space models: Binary Vectorizer and Custom TF-IDF Vectorizer.

- Analyzing Measures of Similarity:
  Examines the effects of using two different similarity metrics: Cosine Similarity and Manhattan Distance.

- Analyzing System Variability:
  Analyze the impact of different vector models and similarity measurements on system performance.

Evaluation Metrics Used

Precision:

It is the proportion of relevant documents recovered out of the total number of documents retrieved.The precision calculation is defined as the ratio of the number of relevant documents retrieved to the total number of documents retrieved.

Recall:

It is a measure that quantifies the proportion of relevant documents that are successfully retrieved out of the total number of relevant documents.The recall is calculated by dividing the number of relevant documents retrieved by the total number of relevant documents.

F-score:

The F-score, also known as the F1 score, is a measure that combines accuracy and recall by taking their harmonic mean. It aims to strike a balance between the two.

The F-score may be calculated using the formula:

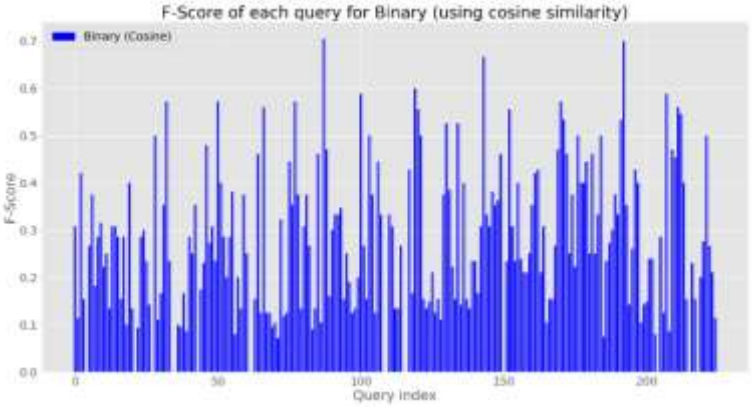
F-score = 2 * (Precision * Recall) / (Precision + Recall).

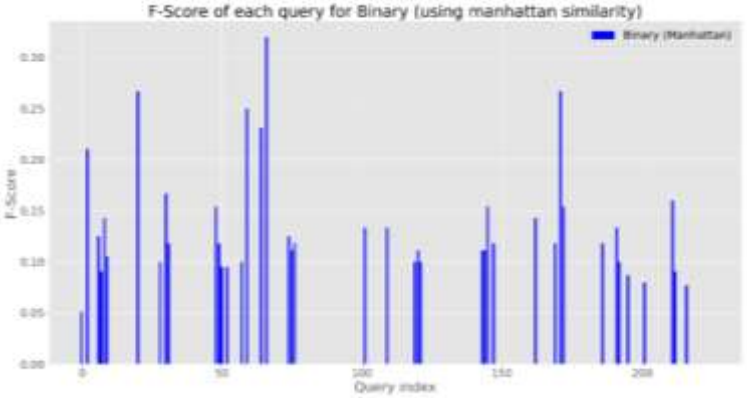
Mean and Maximum Metrics:

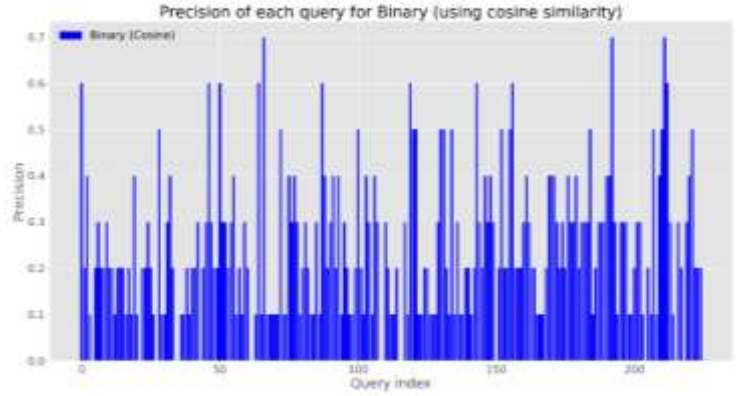
The mean and maximum Precision, Recall, and F-scores are computed to provide a concise summary of the overall performance of the system over several queries.

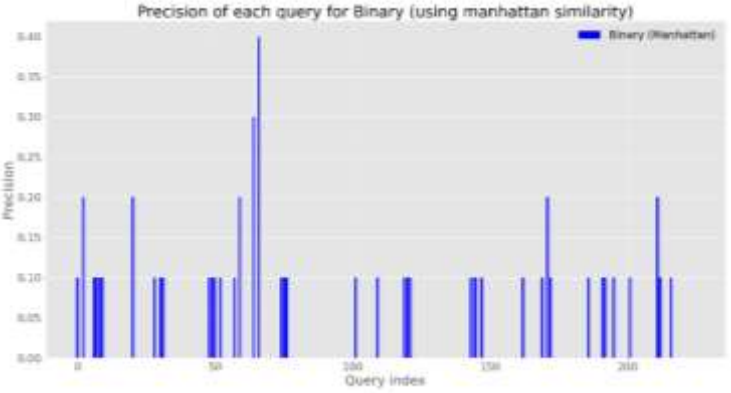
Screenshots

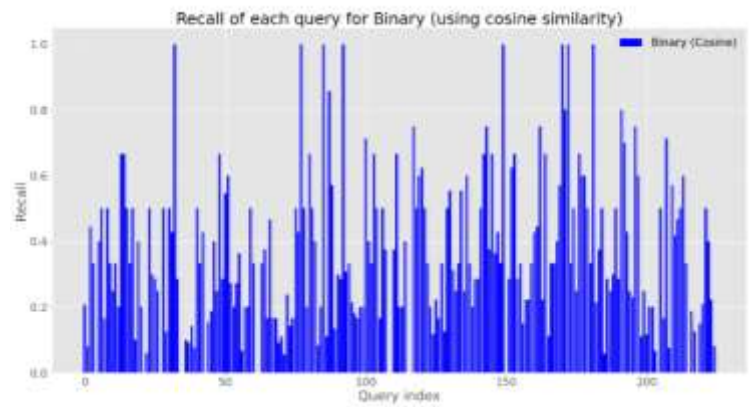
BINARY VECTORIZER

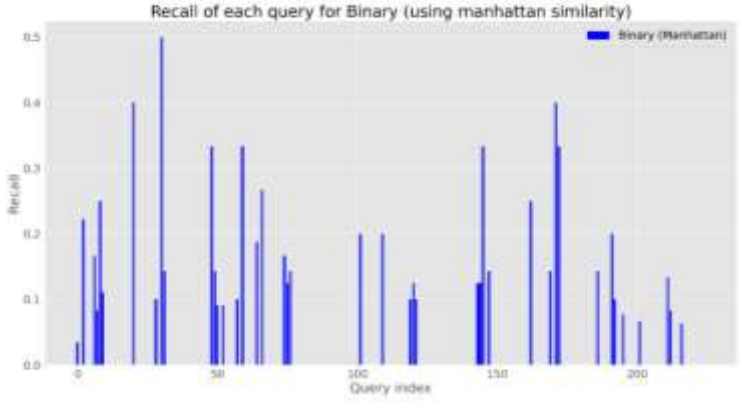




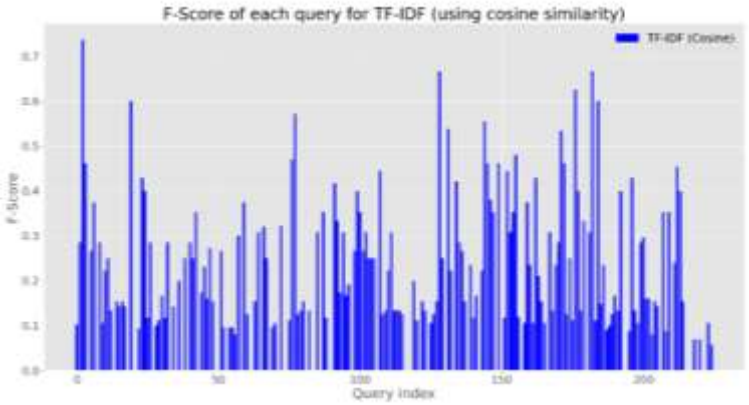


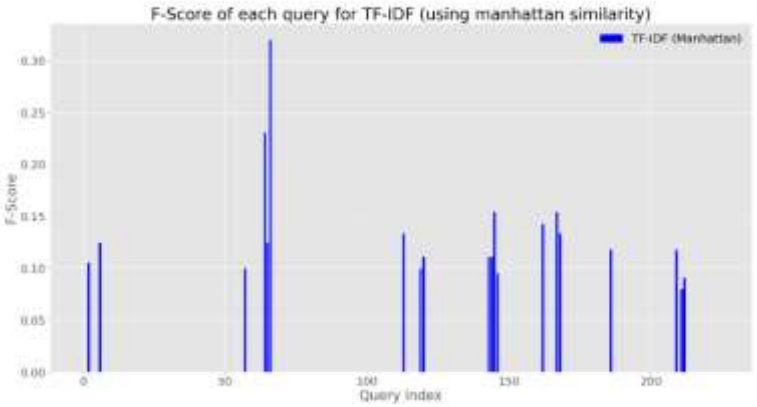


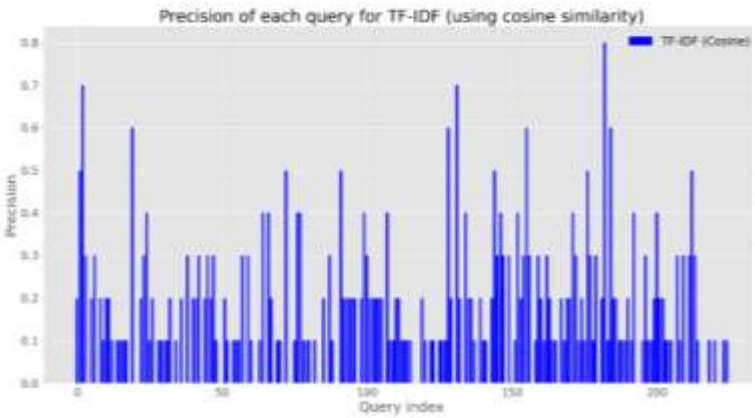


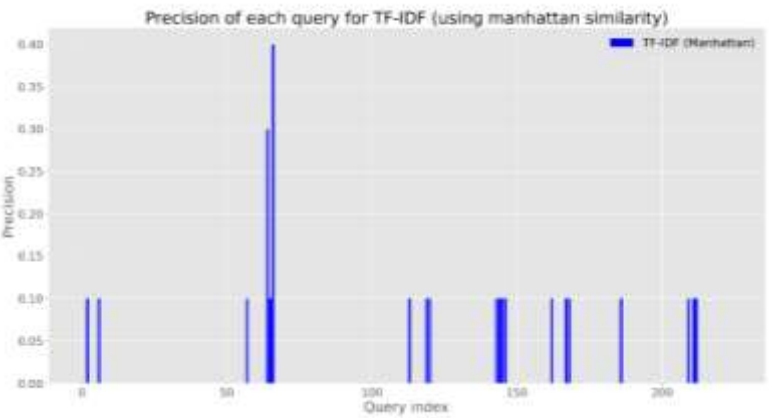


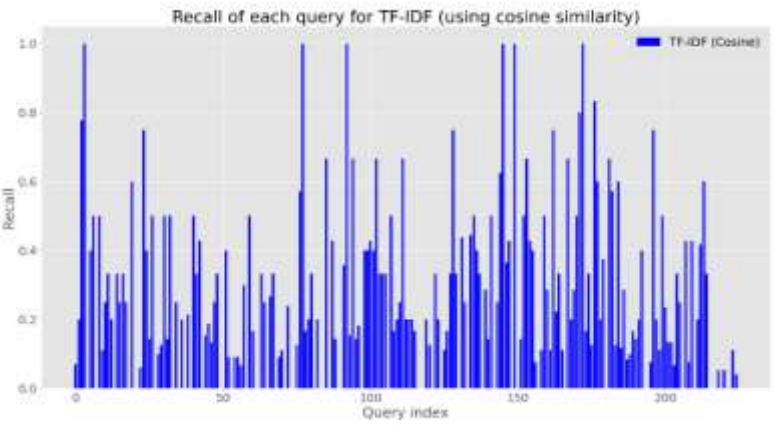
TF-IDF VECTORIZER

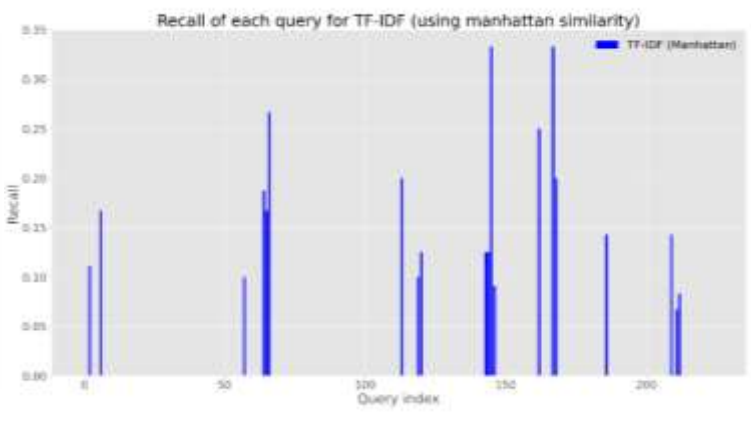












Results

Results

From the output generated and the graphical representations we can make certain analysis on the relevance of the retrieved data.

When using **Binary Vectorizer** and **Cosine Similarity** we notice the following in the evaluation metrics

Fscore: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.268 and maximum variation is 0.705.

Precision: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.244 and maximum variation is 0.7.

Recall: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.353 and maximum variation is 1.0.

When using **Binary Vectorizer** and **Manhattan Distance** we notice the following in the evaluation metrics

Fscore: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.025 and maximum variation is 0.32.

Precision: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.023 and maximum variation is 0.4.

Recall: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.033 and maximum variation is 0.5.

When using **CustomTF-IDF Vectorizer** and **Cosine Similarity** we notice the following in the evaluation metrics

Fscore: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.185 and maximum variation is 0.736.

Precision: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.166 and maximum variation is 0.8.

Recall: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.253 and maximum variation is 1.0.

When using **CustomTF-IDF Vectorizer** and **Manhattan Distance** we notice the following in the evaluation metrics

Fscore: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.011 and maximum variation is 0.32.

Precision: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.011 and maximum variation is 0.4.

Recall: The mean variation from the lists of query relevant documents retrieved from the above calculations, against the list of relevant documents in qrels.text we get 0.014 and maximum variation is 0.333.

From the above metrics we notice the metric that has produced the best quality and quantitative results with the least variance from the qrel.s and retrieved documents is Custom TF-IDF vectorizer using Manhattan distance is Fscore , which has mean variance of 0.011 and maximum variation of only 0.32.

```
(.venv) D:\Jesly\MS\WSU\IR\IR_proj1>python assignment1.py
{'Binary': {'f': {'cos': (0.26861488575846976, 0.7058823529411764),
                  'man': (0.0252959745392464, 0.32)},
            'p': {'cos': (0.24400000000000002, 0.7),
                  'man': (0.023111111111111114, 0.4)},
            'r': {'cos': (0.3534356241268771, 1.0),
                  'man': (0.03303866759039173, 0.5)}},
 'TFIDF': {'f': {'cos': (0.1859044807293152, 0.7368421052631577),
                 'man': (0.011813436191145171, 0.32)},
           'p': {'cos': (0.16622222222222224, 0.8),
                 'man': (0.011111111111111112, 0.4)},
           'r': {'cos': (0.25356434093961583, 1.0),
                 'man': (0.014741782908449574, 0.3333333333333333)}}}
```

CONCLUSION

Conclusion

In this assignment we have used Cranfield dataset which has 1400 documents, 255queries and 1836 evaluations. The primary objective was to process the Cranfield Dataset and provide answers to standard queries. We used 2 vectorizers Binary and Custom TF-IDF and 2 distance measures Manhattan and Cosine Similarity. We fetched the top 10 relevant documents produced in each vectorizer and distance measures. Using the Evaluation metrics of Fscore, Precision and recall, we come to the conclusion of the mean and maximum variation of the retrieved document and qrels.txt(Relation giving relevance judgements).

The exact needs and objectives of your information retrieval system should guide the final choice between cosine similarity and Manhattan distance as well as the evaluation metrics we use (precision, recall, or F1 score) depends on the situation.

Precision is helpful, when ensuring that the retrieved papers are very relevant and correct,.
Recall is useful, even though it may require accepting some false positives, when trying to make sure that all relevant records are captured.
The F1 score comes in handy when a fair evaluation that takes into account both false positives and false negatives is required.

Hence, there is no fixed metric or vectorizer for any it depends on the requirement of which metric is useful.