

**CS 455 – Introduction to NLP**

**Assignment No. 01**

**Information Retrieval System for Judgments**

|  |  |
| --- | --- |
| Name | Abdul Rehman |
| Reg. No. | 04071913025 |
| Semester | BS 8th |
| Submitted to | Dr. Akmal Saeed Khattak |

Contents

[Introduction 3](#_Toc134134416)

[Data Collection 3](#_Toc134134417)

[Vocabulary Creation 3](#_Toc134134418)

[Queries Benchmark 4](#_Toc134134419)

[Cosine Similarity and Ranking 4](#_Toc134134420)

[Evaluation 5](#_Toc134134421)

[Flask Application 5](#_Toc134134422)

[Conclusion 7](#_Toc134134423)

# Introduction

In today’s fast-paced world, no one has time to search for things manually. Everyone needs their daily life tasks to be automated. Artificial Intelligence and Natural Language Processing have greatly helped the masses in this regard. This project aims to automate and fasten the search for relevant court judgments. An Information Retrieval System for Court Judgments has been developed to assist lawyers in accessing relevant judgments based on their queries, improving their efficiency and effectiveness. This system simplifies the process of finding relevant judgments, providing lawyers with quick access to up-to-date and relevant information, ultimately helping them make better-informed decisions and better serve their clients.

# Data Collection

For collecting data, I chose the website of Islamabad High Court. They have a proper [Case Management System](https://mis.ihc.gov.pk/frmSrchOrdr) on their website where we can search recent cases’ judgments by keywords. I selected eight categories for which I scraped all cases from the website. The categories are as follows:

* Administrative
* Civil
* Commercial
* Constitutional
* Criminal
* Environmental
* Family
* Tax

I created the scraper using Selenium which is an open-source automation testing framework used for web application testing, web scraping, and automating repetitive tasks, supports multiple programming languages and is known for its flexibility, versatility, and reliability.

The scraper downloads the pdfs for cases for each category and metadata about the cases and stores the cases in respective folders and relevant information in Excel files which can be later on used to display as an abstract of the document when appears in search results.

# Vocabulary Creation

Now I have a document collection placed in different categories. In this stage, I read each document and generated vocabulary from the entire document collection. I performed traditional preprocessing techniques such as:

* Tokenization
* Removing punctuation
* Stop words removal
* Ignored less than three-character words
* Stemming
* Lemmatization

Then I stored the vocabulary in a plain text file having index number and vocabulary term. Additionally, I assigned document ids to all the documents.

Inverted Index Construction

Now I have the vocabulary generated, for this part, I first computed the raw term frequency. Then computed the log frequency weighting using the log of base 10. Then for each term, I computed the inverse document frequency. Finally, I computed the tfidf and bm25 weightings using the relevant formulae.

Log Term Frequency:

Inverted Index Frequency:

TF-IDF Weighting:

BM25 Weighting:

# Queries Benchmark

Till now we have generated the tfidf and bm25 weightings. In this subtask, I created a Query Collection Benchmark containing ten queries out of which five are two-word queries whereas the rest of five are three-term queries. I added all the tfidf values for each term and found the top ten terms having highest tfidf values and similarly for bm25. Then out of these twenty terms, I generated ten queries randomly.

# Cosine Similarity and Ranking

In this subtask, I computed the similarity between each document and query and rank accordingly. To compute the similarity, I used the cosine similarity method. First, I normalized the tfidf and bm25 values of documents and then took the dot product with the normalized tfidf and bm25 values of queries. Then, I write the top ten documents for each query using both weighting schemes on plain text files.

# Evaluation

I evaluated this information retrieval system using precision, recall, f1-measure and average precision. Here are the results:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Query Terms** | **Weighting** | **P** | **R** | **F** | **AP** |
| **QueryTerm-1** | TFIDF | 0.900 | 0.900 | 0.900 | 1.000 |
| **QueryTerm-2** | TFIDF | 0.700 | 0.700 | 0.700 | 0.668 |
| **QueryTerm-3** | TFIDF | 0.800 | 0.800 | 0.800 | 0.766 |
| **QueryTerm-4** | TFIDF | 0.400 | 0.400 | 0.400 | 0.653 |
| **QueryTerm-5** | TFIDF | 0.400 | 0.400 | 0.400 | 0.510 |
| **QueryTerm-6** | TFIDF | 0.600 | 0.600 | 0.600 | 0.813 |
| **QueryTerm-7** | TFIDF | 0.600 | 0.600 | 0.600 | 0.744 |
| **QueryTerm-8** | TFIDF | 0.300 | 0.300 | 0.300 | 0.317 |
| **QueryTerm-9** | TFIDF | 0.300 | 0.300 | 0.300 | 0.476 |
| **QueryTerm-10** | TFIDF | 0.700 | 0.700 | 0.700 | 0.711 |

Mean Average Precision for 10 queries = 0.666

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Query Terms** | **Weighting** | **P** | **R** | **F** | **AP** |
| **QueryTerm-1** | BM25 | 1.000 | 1.000 | 1.000 | 1.000 |
| **QueryTerm-2** | BM25 | 0.400 | 0.400 | 0.400 | 0.567 |
| **QueryTerm-3** | BM25 | 0.800 | 0.800 | 0.800 | 0.876 |
| **QueryTerm-4** | BM25 | 0.400 | 0.400 | 0.400 | 0.830 |
| **QueryTerm-5** | BM25 | 0.400 | 0.400 | 0.400 | 0.608 |
| **QueryTerm-6** | BM25 | 0.600 | 0.600 | 0.600 | 0.830 |
| **QueryTerm-7** | BM25 | 0.500 | 0.500 | 0.500 | 0.810 |
| **QueryTerm-8** | BM25 | 0.400 | 0.400 | 0.400 | 0.388 |
| **QueryTerm-9** | BM25 | 0.400 | 0.400 | 0.400 | 0.385 |
| **QueryTerm-10** | BM25 | 0.400 | 0.400 | 0.400 | 0.875 |

Mean Average Precision for 10 queries = 0.717

In general, the results show that the BM25 weighting scheme outperforms the TFIDF scheme in terms of MAP. For TFIDF, the best performing query term was the first one with a perfect AP score, while the worst performing query term was the fourth one with an AP of 0.653. For BM25, the best performing query term was the first one with a perfect AP score, and the worst performing query term was the eighth one with an AP of 0.388. These results suggest that BM25 may be a more effective weighting scheme for information retrieval tasks compared to TFIDF. However, further analysis and experimentation may be necessary to confirm these findings.

# Flask Application

I also have created a flask application for this information retrieval system. The first page looks like a search engine that receives queries from the users.

When user enters a query and clicks on search, the search results page opens, on this page I have applied pagination which lets retrieved documents to be displayed in sets of 10 means that each page contains 10 documents. Each page contains a word cloud based on the documents displayed on that page, where each word is clickable and on click it becomes part of the previous query, letting user refine his or her query according to the result.

# Conclusion

In conclusion, this project aimed to develop an information retrieval system for court judgments to assist lawyers in accessing relevant judgments based on their queries. The system was developed by collecting data from the website of Islamabad High Court using a scraper built with Selenium, generating vocabulary from the document collection, constructing an inverted index using tfidf and bm25 weightings, creating a Query Collection Benchmark, computing cosine similarity between documents and queries, and evaluating the system using precision, recall, f1-measure, and average precision. The results show that the system achieved a mean average precision of 0.666 for tfidf and 0.783 for bm25, indicating that the system can effectively assist lawyers in finding relevant judgments based on their queries.