# INFORMATION RETRIEVAL SYSTEM

**June, 2020**

Shubham Sharma

# Contents

## 0.1 Introduction

An information retrieval process begins when a user enters a query into the system. In information retrieval, a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy.

Here, we have developed an information retrieval (IR) system by which we can obtain a set of documents based on the relevance to a certain query. We have also evaluated the performance of the IR system.

## 0.2 Dataset

A dataset is a collection of data. The dataset which we have used for the IR system consists of 200 documents belonging to five different categories. The categories are following:

1. **For Sale:** It consists of 44 documents. The documents of this category are related to the items or services that have been put on sale by the people.
2. **Politics:** It consists of 40 documents. The documents of this category are mostly related to middle-eastern countries like Israel, Palestine, Egypt etc.
3. **Religion:** It consists of 38 documents. The documents of this category are related to Christianity, Jesus Christ, the Bible etc.
4. **Space:** It consists of 42 documents. The documents of this category are related to space, Hubble Space Telescope, Jet Propulsion Laboratory, Pluto etc.
5. **Wanted:** It consists of 36 documents. The documents of this category are related to the items or services wanted by the people.

## 0.3 Benchmark Queries

We have manually created a set of 50 benchmark queries for the IR system, i.e. 10 benchmark queries for each category of the dataset.

A benchmark query for a certain category has been created keeping in mind the fact that it matches with the majority of the documents of that category.

## 0.4 Relevance Judgements

Relevance judgements are a vital part of an IR system that define what documents should and should not be considered relevant to a particular query.

This part of the project, which is the creation of relevance judgements for each query-document pair was done manually. All the documents of a particular category have been considered relevant to all the benchmark queries of that category, and the documents of other categories have been considered non-relevant.

## 0.5 Development of the IR System

### 0.5.1 Data Pre-processing

Whenever we have textual data we need to apply several pre-processing steps to data to transform this textual data into numerical features that can be easily analyzed and ideal to work with. Typically, whether we are given data or have to scrape it, the text will be in it's natural human understandable format of sentences, paragraph etc. So it is necessary to break down data into a format that computer can understand.

Steps involved are :-

1. Reading the data and removing unwanted spaces, punctuation's and characters like @, &, # etc which doesn't contribute to text retrieval. For this purpose we have used **Regular Expressions**. Regular expressions are patterns used to match character combinations in strings. We have used **'[a-zA-Z0-9]{2,15}'** to match numbers and characters, words with any kind of casing format of length ranging from 2 to 15.

2. **Tokenization** : This breaks up the string into a list of words on the basis of specified pattern using Regular Expressions (RegEx). We can also use methods available in **nltk** package.

```
from nltk import word_tokenize
sentence = "information retrieval systems are awesome"
result = word_tokenize(sentence)
print(result)
```

```
output: ['information', retrieval', 'systems', 'are', 'awesome']
```

3. **Remove stop words**: stop-words are the most frequently used words from the Natural Language Toolkit. There are 179 English words, including 'i', 'me', 'my',

‘myself’, ‘we’, ‘you’, ‘he’, ‘his’, for example. We usually want to remove these because they have low predictive power. But there can be cases when we need to keep them. If your corpus size is very small, removing stop-words would decrease the total number of words by a large percentage.

4. **Stemming**: shortens words back to their root form. It cuts off prefixes and/or endings of words based on common ones.

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize

ps = PorterStemmer()
words = ["program", "programs", "programer", "programing", "
programers"]
for w in words:
    print(w ,' : ', ps.stem(w))
```

```
program  :  program
programs  :  program
programer  :  program
programing  :  program
programers  :  program
```

Most of the time stemming doesn’t return a proper word that can be found in the dictionary.

```
Information : inform
retrieval : retriev
systems : system
are : are
awesome : awesom

```

5. **Word-Frequency Distribution Matrix**: After pre-processing of textual data we created a word frequency distribution matrix using FreqDist() method in nltk package. The Rows of the matrix represent all the unique words in the corpus and columns represent documents or document Id in the corpus. Each cell contains a value representing the frequency of that word in a particular document.

Below is a section of word frequency distribution dataframe -

| words | doc-101 | doc-102 | doc-103 | doc-104 | doc-105 |
|---|---|---|---|---|---|
| top | 1 | 1 | 0 | 3 | 0 |
| amazing | 1 | 0 | 2 | 1 | 4 |
| random | 0 | 2 | 4 | 1 | 1 |
| universe | 0 | 0 | 1 | 3 | 1 |
| tank | 2 | 1 | 0 | 0 | 3 |

### 0.5.2 TF-IDF Weighting

TF-IDF weighting is a measure that reflects on how important a word is to a document in a collection. TF stands for Term Frequency which gives us the number of times a word is occurring in a document, and IDF stands for Inverse Document Frequency which tells us about the rarity of a word in the collection. The rarer a word is, the higher its IDF weight will be.

```
    TF weight = 1 + log(term frequency)
    IDF weight = log(total number of documents/document frequency)
    TF-IDF weight = TF weight * IDF weight
```

### 0.5.3 Parallel Processing

The computation of TF-IDF weight for each and every word in the word document matrix takes a lot of time. So to lessen the computational time, we divided the process across all the cores of the processor of the machine which we working on. Pool object in multiprocessing package offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes (data parallelism).

### 0.5.4 Cosine Similarity

We have used cosine similarity technique to compute the similarity between the queries and the documents in the collection.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined to equal the cosine of the angle between them, which is also the same as the inner product of the same vectors normalized to both have length 1.

```
    cos theta = (a.b)/(|a|.|b|)
```

## 0.6 Evaluation of the IR System

To evaluate the performance of the IR system, we have computed precision, recall and f-measure for every benchmark query for its top 20 results.

- **Precision** = fraction of retrieved documents that are relevant

```
    Precision = (true positive)/(true positive + false positive)
```

- **Recall** = fraction of relevant documents that are retrieved

```
    Recall = (true positive)/(true positive + false negative)
```

- **F-measure** = (2 * Precision * Recall)/(Precision + Recall)

## 0.7 Results

The results (top 20 documents) corresponding to all the benchmark queries are saved in separate 'json' files.

### 0.7.1 Performance Evaluation

| Query ID | Precision | Recall | F-measure |
|---|---|---|---|
| q101 | 1.000 | 0.454 | 0.625 |
| q102 | 0.950 | 0.432 | 0.594 |
| q103 | 0.900 | 0.409 | 0.562 |
| q104 | 1.000 | 0.454 | 0.625 |
| q105 | 0.950 | 0.432 | 0.594 |
| q106 | 1.000 | 0.454 | 0.625 |
| q107 | 1.000 | 0.454 | 0.625 |
| q108 | 1.000 | 0.454 | 0.625 |
| q109 | 1.000 | 0.454 | 0.625 |
| q110 | 0.950 | 0.432 | 0.594 |

| Query ID | Precision | Recall | F-measure |
|---|---|---|---|
| q201 | 1.000 | 0.500 | 0.667 |
| q202 | 0.800 | 0.400 | 0.533 |
| q203 | 1.000 | 0.500 | 0.667 |
| q204 | 0.200 | 0.100 | 0.133 |
| q205 | 0.100 | 0.050 | 0.067 |
| q206 | 0.150 | 0.075 | 0.100 |
| q207 | 0.150 | 0.075 | 0.100 |
| q208 | 0.300 | 0.150 | 0.200 |
| q209 | 0.850 | 0.425 | 0.567 |
| q210 | 0.850 | 0.425 | 0.567 |

| Query ID | Precision | Recall | F-measure |
|---|---|---|---|
| q301 | 0.450 | 0.237 | 0.310 |
| q302 | 0.550 | 0.289 | 0.379 |
| q303 | 0.300 | 0.158 | 0.207 |
| q304 | 0.300 | 0.158 | 0.207 |
| q305 | 0.300 | 0.158 | 0.207 |
| q306 | 0.350 | 0.184 | 0.241 |
| q307 | 0.150 | 0.079 | 0.103 |
| q308 | 0.350 | 0.184 | 0.241 |
| q309 | 0.750 | 0.395 | 0.517 |
| q310 | 0.050 | 0.026 | 0.034 |

| Query ID | Precision | Recall | F-measure |
|---|---|---|---|
| q401 | 0.750 | 0.357 | 0.484 |
| q402 | 0.100 | 0.048 | 0.065 |
| q403 | 0.750 | 0.357 | 0.484 |
| q404 | 0.200 | 0.095 | 0.129 |
| q405 | 0.450 | 0.214 | 0.290 |
| q406 | 0.150 | 0.071 | 0.097 |
| q407 | 0.750 | 0.357 | 0.484 |
| q408 | 0.200 | 0.095 | 0.129 |
| q409 | 0.850 | 0.405 | 0.548 |
| q410 | 0.750 | 0.357 | 0.484 |

| Query ID | Precision | Recall | F-measure |
|---|---|---|---|
| q501 | 0.900 | 0.500 | 0.643 |
| q502 | 0.850 | 0.472 | 0.607 |
| q503 | 0.900 | 0.500 | 0.643 |
| q504 | 0.800 | 0.444 | 0.571 |
| q505 | 0.900 | 0.500 | 0.643 |
| q506 | 0.700 | 0.389 | 0.500 |
| q507 | 0.700 | 0.389 | 0.500 |
| q508 | 0.900 | 0.500 | 0.643 |
| q509 | 0.800 | 0.444 | 0.571 |
| q510 | 0.650 | 0.361 | 0.464 |