# Information Retreival Using BM25, TF-IDF, and Query Likelihood Model


## Abstract

The main objective of this paper is to implement and investigate different information retrieval models, among which the tf-idf, BM25, and the query likelihood model. These models are utilized to generate a list of ranked passages for a given set of queries. By comparing the rankings of each model, it is possible to identify and analyze the distinctions between them.


## 1 Introduction

### 1.1 Data

Three data sets were used for this research project:

- test-queries.tsv - a tab separated file, where each row contains a test query identifier (qid) and the actual query text.
- passage-collection.txt - a collection of passages, one per row.
- candidate-passages-top1000.tsv - a tab separated file with an initial selection of at most 1000 passages for each of the queries in test-queries.tsv. The format of this file is <qid pid query passage>, where pid is the identifier of the passage retrieved, query is the query text, and passage is the passage text (all tab separated). The passages contained in this file are the same as the ones in passage-collection.txt. However, there might be some repetitions, i.e. the same passage could have been returned for more than 1 query.

### 1.2 Text Processing

To initiate the project, the text underwent preprocessing, which entailed loading the passage-collection.txt file as a unified string, then replacing all new lines and punctuation marks with empty spaces, and finally converting all letters to lowercase. This standardized form of the text was then separated into tokens based on empty spaces, and a function was developed to carry out all of these procedures. The function accepts text in the form of a string as input, and its final output is a list of tokens. This function was used throughout the project with the rest of the data files while working on building the various Information Retrieval models.

### 1.3 Vocabulary

Once the 1-grams were extract from the passage-collection.txt, and returned as a list of tokens, the next step involved building a vocabulary, or a list of unique terms in the corpus. A dictionary was initiated which would keep the vocabulary term as a key and the term frequency as a value. By looping through the list of tokens, every time a new word is encountered an element is added to the dictionary with a value of 1, and every time a repeated vocabulary term is encountered, its value in the dictionary is increased by one. The result is a dictionary with all unique vocabulary terms along with their frequency.

The total vocabulary count without the removal of stop words is 131058.

After this step, stop words are removed. This is done by importing the nltk.corpus library which contains stop words. Each of these stop words is removed form the vocabulary dictionary.

The total vocabulary count with the removal of stop words is 130907.

The vocabulary without stop words is saved under the intermediary file name "vocabulary_1.csv".

### 1.4 Zipf's Law

Zipf's law is a law that describes that the frequency $f$ of a variable, in this case a term, is inversely proportional to its frequency rank. More formally, the formula of Zipf's law is represented as follows:

$$f(k; s, N) = \frac{k^{-1}}{\sum_{n=1}^{N} i^{-s}}$$

where $f(.)$ is the normalised frequency of a variable or in this case of the term, $k$ is the variable's frequency rank or in this case the the term's frequency rank in the corpus, $s$ is a controlled parameter which is set to 1 when working with text sets, and $N$ is the set of variables or in this case the number of terms in the vocabulary.

Theoretically, what this law unravels is that in natural language the first term appears twice as likely as the second term, three times as likely as the third term and so on. Following this law we can see that a small number of terms appear more often, and a big amount of terms appear very infrequently.

Bellow is a plot of how the Zipf's law compares to the actual vocabulary in the corpus. There are two plots - 1 presents a plot in the absolute scale while the 2 is given in the log-log scale.

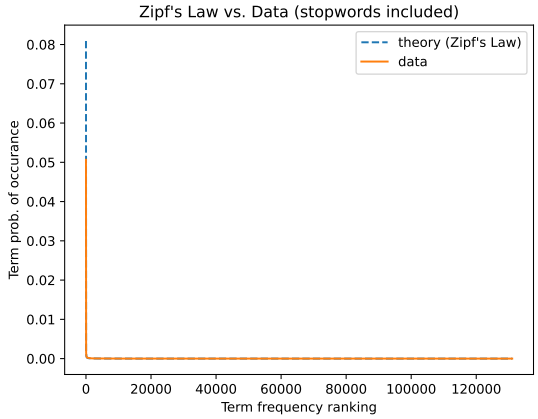


Figure 1: Zipf's Law vs. Data (stop words included)

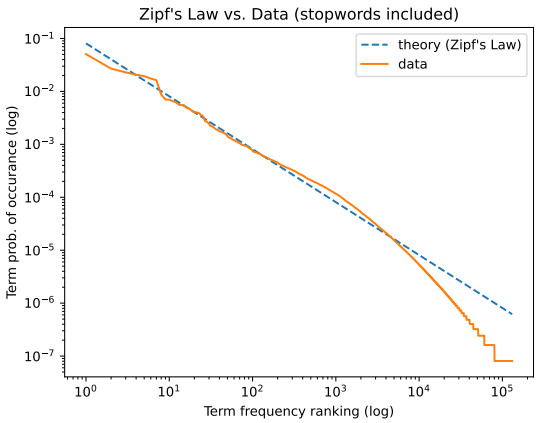


Figure 2: Zipf's Law vs. Data log-log (stop words included)

The term frequency is plotted on the x-axis and the probability of occurrence of the term is plotted on the y-axis in both of these figures. When constructing these figures, the stop words were not removed from the vocabulary.

| Term | Term Frequency | Norm. Term Frequency |
|---|---|---|
| the | 626892 | 0.051 |
| of | 334281 | 0.027 |
| a | 283558 | 0.023 |
| and | 255211 | 0.021 |
| to | 240943 | 0.019 |

Table 1: Top 5 most frequent words along with their term frequencies and normalised term frequencies

The graphs demonstrate that the empirical data closely adheres to Zipf's law, which could be expected given the large enough corpus size. It is sufficient to exhibit the phenomenon that Zipf's law describes in natural language.

To get a better overview of the law, 1 shows the term alongside their frequencies and normalized frequencies. The normalized frequency is calculated by dividing the term frequency over the total number tokens in the corpus.

Observing this small subset provides insight into the law. The term 'the', which is the most likely term, is almost twice as likely as the second term 'of' - a relationship Zipf's law determines. Furthermore, if these 5 most frequent words and their frequencies are observed, it could be noticed that at the start the difference in occurrence between the first 2 terms is larger than that between the second and third term, and so on. Although the term frequencies are not exactly as per Zipf's law in this case (the first term is not three times as likely as the second term), the general trend can still be observed.

Next, Figure 3 and Figure 4 below demonstrate the empirical data against Zipf's law theoretical data once stop words are removed.

It can be observed that now the the empirical data does not follow the theoretical Zipf's law data so well. This is something that could be expected given that stop words are the most frequent words and the most frequent words were removed from the vocabulary. The most frequent word in the vocabulary now appears with a normalizing frequency of about 0.01, whereas the most frequent words when stop words were included appeared with normalised frequency of 0.05.

This can be observed better in Figure 4 where previously, as seen in Figure 2, the empirical line

followed the theoretical line more closely, now there is a gap exactly at the place where the frequent words which were removed appeared (to the left in the area of low term frequency ranking and high term probability of occurrence).

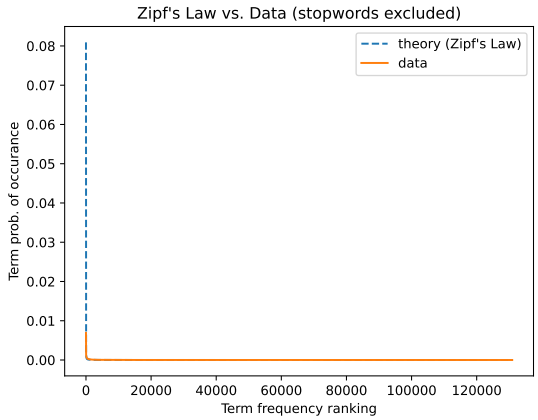


Figure 3: Zipf's Law vs. Data (stop words excluded)

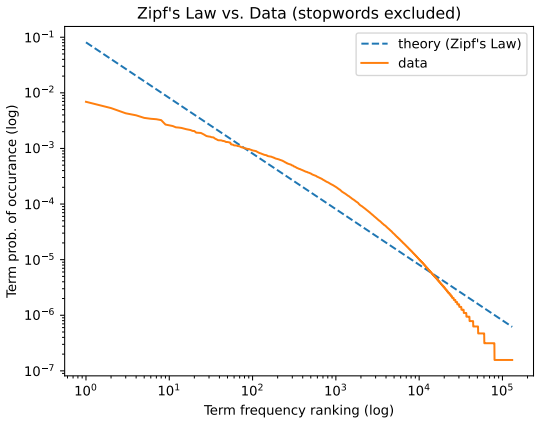


Figure 4: Zipf's Law vs. Data (stop words excluded)

## 2 Inverted Index

### 2.1 Inverted Index Approach

To begin with, the vocabulary generated in the initial task is loaded from an intermediary file called 'vocabulary_1.csv' and stored in memory. Additionally, the file 'candidate-passages-top1000.tsv' is loaded into memory. Subsequently, only the 'pid' and 'passage' columns are retained and the duplicate rows are eliminated in order to obtain a collection of unique 'pid' and 'passage' rows.

An inverted index dictionary is, then, defined such that the key is the unique vocabulary term and the value is a an empty list.

Next, for each of the passages, the preprocessing function is applied which returns a list of tokens. The token list is then passed on to another function which outputs a dictionary where the key is a term and the value is a frequency of the word. In this way, for each of the passages we obtain a dictionary with the unique words as keys and term frequency as values.

For each of the keys in this passage dictionary, an update is made to the inverted index dictionary. More specifically, for each key in the passage, the same key in the inverted index is accessed and the value, which is initially an empty list, is appended another list of length 2. This list contains the passage id as the first element and the term frequency as the second element.

The result is as inverted index defined as a dictionary such that the key of the dictionary is a vocabulary term and the value is a list of lists. Each list in the list of lists is of length 2. The first value in the list is the passage id of the passage that contains the vocabulary term and the second value of the list holds the frequency of the term within the passage.

This implementation allows for easy and fast access to the the passages that contain a specific term as well as the frequency of the term. Moreover, for each of the terms, it is also simple to extract the number of documents in which a certain term appears.

The choice to record both the passage id and term frequency for each vocabulary term was influenced by the forthcoming computations in the creation of IR models. The term frequency is a metric the IR models required to compute the rankings.

The inverted index is saved in the form of a json file in order to preserve the dictionary structure as an intermediary file under the name "inverted_index.json".

### 2.2 Stop-words Removal

Stop-word are removed from both the vocabulary and inverted index since they do not carry much meaning. This way the focus is placed on more important information that other words carry. Removing stop words leads to faster processing times and more accurate results. Lastly, given the nature of the retrieval models used in this project, i.e. retrieval models which are not context-dependent, keeping stop-words is not useful.

## 3 Query Likelihood Information Retrieval Models

### 3.1 Comparison between models

Query likelihood models work in a way such that a probability distribution of all the words in the document is created based on their frequency in the document. Once the document language model is created, documents are ranked based on the probability that the query is generated from the language model. Using the simplest form of this model, however, would lead to 0 probabilities for queries that contain words which are not present in the document. To mitigate this drawback smoothing methods are used. This project explores three types of smoothing methods: Laplace Smoothing, Lidstone correction, and Dirichlet smoothing.

The query likelihood model using Laplace smoothing counts the term frequencies in the documents and adds 1 to each of the frequencies. Then, the probabilities are re normalized by discounting the probabilities of the seen terms. In this way, Laplace smoothing assumes a uniform prior over the words. The disadvantage of the Laplace smoothing is that it assigns too much weight to unseen words.

The Lidstone correction is a generalization of the Laplace smoothing that addresses the disadvantage the Laplace smoothing faces. Instead of adding a fixed count of 1 to each of the term frequencies, this model adds value $\epsilon$. After this step, the probabilities are re normalized. The value of $\epsilon$ can range between 0 and 1. If set to 1, the model is equivalent to the Laplace smoothing model. If set to 0 it yields the maximum likelihood estimate.

Both Laplace smoothing and the Lidstone correction models suffer from the limitation of assigning equal probabilities to all unseen words. However, this approach is suboptimal since some words, like stopwords, are more frequent than others, and it would be better if the model assigned higher probabilities to more frequent words. Dirichlet smoothing improves on Laplace and Lidstone methods by addressing this issue. Specifically, it estimates the weight of unseen words using background probabilities. Unlike Laplace smoothing, which adds equal counts to each word, Dirichlet smoothing adds counts that are proportional to the estimated frequency of the word in the data. This approach allows Dirichlet smoothing to act as a Bayesian regularizer, where the counts represent prior knowledge about the distribution of the events. Therefore, Dirichlet smoothing can be especially valuable when prior knowledge about the distribution of events is available.

When considering the query with qid 929033 asking "what years were the crusades", Laplace smoothing suffers from a disadvantage which can be observed by examining its ranking. Specifically, Laplace ranks the passage with pid 5582496 as the top-ranked passage. This passage states "Canine Age Human Age 2 Months 14 Months 6 Months 5 Years 8 Months 9 Years 1 Year 15 Years 2 Years 24 Years 3 Years 28 Years 4 Years 32 Years 5 Years 37 Years 6 Years 42 Years 7 Years 47 Years 8 Years 52 Years 9 Years 57 Years 10 Years 62 Years 11 Years 67 Years 12 Years 72 Years 13 Years 77 Years 14 Years 82 Years.t 10, she's like a human of 65; at 12, a human of 75; and at 15, a human of 90. Dog Years. A dog's lifespan is only a fraction of the average human's lifespan, which means that a dog ages more quickly in the same amount of time. This chart shows how a dog's age might be adjusted to compare to a human's age.". In contrast, Dirichlet smoothing ranks the passage with id 2237411 as the top-ranked passage. This passage reads "if you really want to know when the crusades began and ended well here you go: the crusades bagan in 1096 and lasted for 9 years and ended in the year of 1272. if you really want to know when the crusades began and ended well here you go: the crusades bagan in 1096 and lasted for 9 years and ended in the year of 1272."

As observed, Laplace smoothing assigns equal probabilities to all words, including words that are not relevant to the query. In this case, the words "canine", "age", "human", "months", "years", "chart", and "shows" are all present in the passage returned by Laplace, but they are not directly related to the question about the Crusades. Laplace smoothing treats all words equally and does not take into account their relevance to the query.

On the other hand, Dirichlet smoothing uses a prior distribution over the probabilities of the words in the vocabulary, which allows it to assign higher probabilities to words that are more relevant to the query. In this case, the passage returned by Dirichlet smoothing contains the exact answer to the query about the Crusades, and does not contain irrelevant words.

| qid | Lap Rank | Lap Score | Lid Rank | Lid Score |
|---|---|---|---|---|
| 2068541 | 7 | -30.92 | 1 | -31.67 |
| 6707713 | 9 | -30.95 | 2 | -31.77 |
| 8596285 | 10 | -30.97 | 4 | -31.88 |
| 3130232 | 13 | -31.07 | 3 | -31.84 |

Table 2: Comparison of query rank and score using Laplace smoothing and Lidstone Correction for select 4 queries

### 3.2 Model Similarity

The Laplace smoothing and Lidstone correction models are the most similar to each other. Both of these models assign uniform priors to unseen words. The only difference is in how the weight of the term frequencies is determined. In the case of Laplace smoothing, each term is added 1 to its frequency, whereas in the case of Lidstone correction a value of $\epsilon$ is added to each term frequency. In practice, if we set $\epsilon = 1$, the term frequencies, and renormalized term probabilities, using Lidstone correction would be the same as those determined by Laplace smoothing.

Table 2 contains the rank and score of 4 select queries for the passage with pid 1108939 as ranked using the Laplace (referred to as Lad in the table) and Lidstone (referred to as Lid in the table) correction (with $\epsilon = 0.1$). In this example it can be observed that even though the rank of the queries has changed, the rank is still rather similar when using Laplace smoothing and Lidstone correction.

### 3.3 Choice of $\epsilon = 0.1$ in the Lidstone Correction

The value of $\epsilon$ can take a value anywhere between 0 and 1. If the value is set to 1, then the model is equivalent to the Laplace smoothing model. If the value is set to 0 then the model reduces to a maximum likelihood estimate which does not involve smoothing. A small value of $\epsilon$ leads to a high variance, and high sensitivity to the training data. On the other hand, a large value of $\epsilon$ leads to a high bias, and estimates which are less sensitive to the training data but may be less accurate overall.

For the purpose of this specific project, a smaller value of $\epsilon$, or the suggested value $\epsilon = 0.1$ is appropriate. Setting the value of $\epsilon$ to be small means that there is weaker smoothing and that the model depends on the actual data in the training set. Since this project has a small training set and involves short texts, a smaller value of $\epsilon$ is more appropriate.

It is important to note that the best way to determine if this value of $\epsilon$ is indeed the best choice is by conducting cross-validation.

### 3.4 Setting $\mu = 5000$ in Dirichlet Smoothing

The hyper parameter $\mu$ acts as a regularizer in Dirichlet smoothing. Larger value of $\mu$ lead to larger regularisation, whereas smaller value lead to lower regularization. The choice of $\mu$ depends on the data available and the model complexity. In the case of this project, since the training data we have is small and the texts are short, a smaller value of $\mu$ may be more appropriate. This is because a smaller value of $\mu$ acts as a weak regulatizer and allows for fitting the data well. At the same time it would avoid over fitting by adding a penalty. Large values of $\mu$ act as a much stronger regularizer and provide a more general model. A very large value of $\mu$ such as 5000 may lead to a very general model and under fit the data.

To observe this we could look into the ranking formula using Dirichlect smoothing:

$$p(q|d, C) = \prod_{t \in q \cap d} (\lambda \cdot (t|M_d) + (1 - \lambda) \cdot (t|M_c))$$

where

$$\lambda = \frac{N}{N + \mu}$$

where $q$ is the query, $d$ is the document, $C$ is the corpus, $t$ is the term, $M_D$ is the document model, $M_c$ is the corpus model, $N$ is the length of the document and $\mu$ is a hyper parameter of choice.

If we set $\mu$ to be as high as 5000, and the average document length of this project is about 47, the value of $\lambda$ will become very small. On average very little weight would be put on the document model and a much larger weight would be put on the corpus model which would lead to under fitting. Effectively, what would happen is as the value of $\mu$ approaches larger numbers, the prior dominates over the observed data and the formula reduces to the MAP estimate.

It is worth to note that the best choice of the hyper parameter can be found empirically via cross-validation of $\mu$.