Information Retrieval and Data Mining (COMP0084) Coursework 1

Abstract

This coursework aims at developing information retrieval models that generate a ranked list of passages relevant to a given text query, ensuring that the most pertinent passages are shown at the top. The coursework is divided into four parts: Task 1 - Implements a function for processing raw text and conducts a comparative analysis between the empirical distribution of words and the distribution predicted by Zipf's law; Task 2 - Constructs an inverted index to allow more efficient search processes; Task 3 - Develops TF-IDF and BM-25 retrieval models to rank the list of passages by cosine similarity and BM-25 scores respectively; Task 4 - Built query likelihood language models with Laplace smoothing, Lidstone correction, and Dirichlet smoothing respectively, to rank the list of passages;

1 Introduction

In this coursework, unigram (1-gram) text representations will be used for all tasks. Python version 3.11.5 is used for developmenet. The libraries required to run the codes are listed in requirements.txt

2 Task 1 - Text Statistics

2.1 Text Preprocessing Choices

In this coursework, there are totoally 4 layers of text preprocessing, including normalisaion, tokenisation, stop word removal and stemming.

2.1.1 Tokenisation

All the text is split and turned to tokens using NLTK library, forming the foundation for constructing inverted index and retrieval models.

2.1.2 Normalisation

All text is converted to lowercase to ensure consistency in text format. Additionally, considering that all queries are in English, for simplicity, any characters not part of the English alphabet are replaced with a space. This includes punctuation, Greek letters, and non-printing characters such as "U+200B".

2.1.3 Stop word removal

Extremely common words, often lacking substantive meaning within sentences, are removed from the list of tokens to reduce computational demands. While removing stop words can improve processing efficiency, it may sometimes result in the loss of contextual meaning. In this coursework, stop words are removed during tasks 2, 3, 4 and part of task 1.

2.1.4 Stemming

Words are converted to their root form by using NLTK SnowballStemmer. Stemming increases recall and sometimes improves the accuracy of retrieval models. In this coursework, stemming is used in task 2, 3 and 4. It is not applied in task 1 as it does not affect the plots significantly and computing time can be significantly shorten without stemming.

2.2 Text Preprocessing Results

Without removing stop words, a total of 115,698 vocabularies and 10,284,445 tokens have been identified.

2.3 Zipf's Law Justification

Figure 1 illustrates the empirical and Zipf's law distributions of normalised frequency. The plots appear to be nearly identical, suggesting a good alignment of empirical results with Zipf's law. However, both lines converge towards values close to 0 across much of the graph, likely due to the extensive range on the x-axis. To have a more effective comparison of the two distributions, log-log graph is plotted in Figure 2.

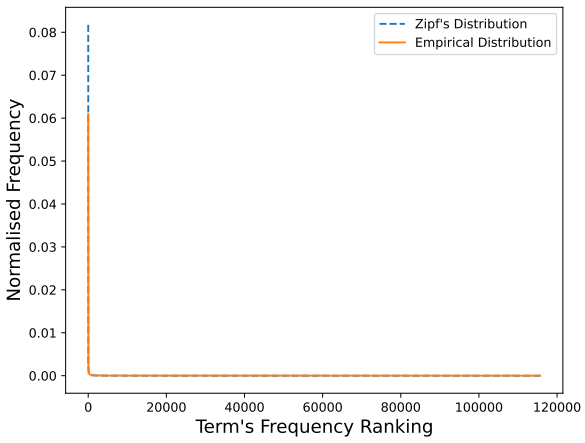

Figure 1: Comparison between empirical distribution and Zipf's law distribution

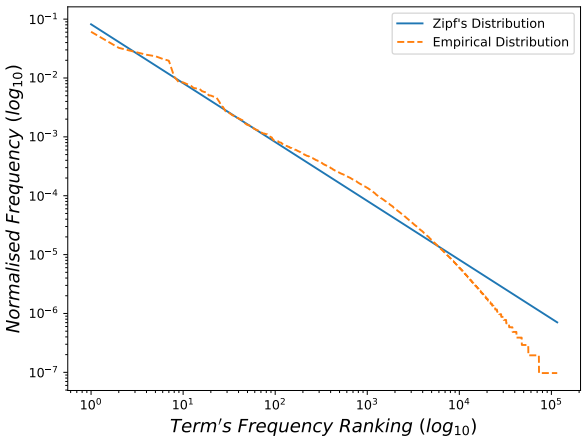

Figure 2: Comparison between empirical distribution and Zipf's law distribution (log-log scale)

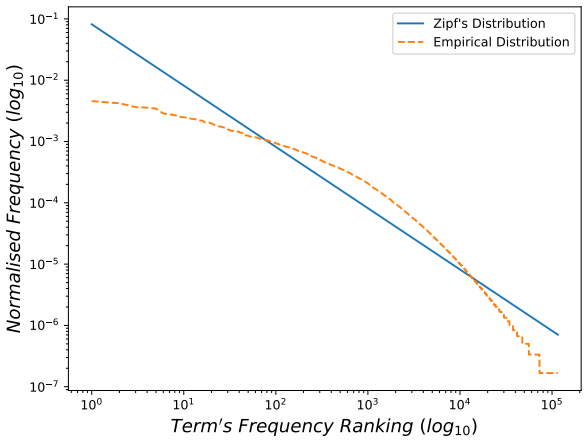

Figure 3: Comparison between empirical distribution and Zipf's law distribution (log-log scale)

With Figure 2, it is clear that Zipf's Law generally provides an accurate predictioin of normalised frequency. However, Zipf's law tends to be less precise for words with lower frequencies, as the empirical normalised frequency for these words is significantly lower than Zipf's prediction.

According to Zipf's law (Equation 1), with $s = 1$ and N being constant for a given collection (i.e. $\sum_{i=1}^{N} i^{-s}$ is a constant), it suggests that normalised frequency of a word is inversely proportional to its rank, as shown in Equation 2. However, in reality, the occurrence of words is very random and it leads to uncertainties. These uncertainties have a greater impact on low-frequency words, resulting in reduced accuracy of Zipf's Law in this region.

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

$$f * k = Constant \quad (2)$$

2.4 Effects of stop words removal

Figure 3 shows the two normalised frequency distributions after the removal of stop words. It is obvious that the stop words removal affects high-frequency words significantly while having minor effect on mid-to-low frequency words. This is because stop words are usually extermely frequent in any pieces of text.

3 Task 2 - Inverted Index

To construct an inverted index, the passages in "candidate-passages-top1000.tsv" are processed using the function developed in Task 1. This involves removing stop words and applying stemming to each token. The output from task 1 function is a two-dimensional list containing sublists of tokens extracted from each unique passages. A dictionary ('dict_pid_tokens') with the format '{pid : tokens extracted from the passage}' is built by mapping pids and the output.

Afterwards, another function is implemented to build the inverted index, and the output is a dictionary with format '{vocab: {pid: count} }'. The function iterates over every token in 'dict_pid_tokens', and counts the occurrence of each token within its associated passage. It then adds a new entry to the dictionary with the token as the key and pid: occurrence count as the value. This resulting dictionary serves as the inverted index. With a given token, the dictionary can inform in which passage it appears and how many times it occurs in that passage.

4 Task 4 - Query likelihood language models

I believe the language model with Dirichlet smoothing outperforms the others. Additive Smoothing including Laplace and Lidstone treat every unseen query tokens equally and potentially overemphasising their significance, despite the fact that these tokens have different importance. Dirichlet smoothing addresses this issue by adjusting the smoothing amount based on background probabilities and document length. Therefore, Dirichlet is expected to have a better performance.

The distinction between Dirichlet and additive smoothing can also be observed from the query "what contains specific instructions that make each living thing unique?". The query is tokenised into ['contain', 'specif', 'instruct', 'make', 'live',

'thing', 'uniqu']. When this query is input into the three language models, all three models rank the same passage as the top result. Thus, our comparison focuses on the second-ranked passage. Below shows the three second-ranked passages returned from different models.

Dirichlet: Chromosomes are thread-like structures located inside the nucleus of animal and plant cells. Each chromosome is made of protein and a single molecule of deoxyribonucleic acid (DNA). Passed from parents to offspring, DNA contains the specific instructions that make each type of living creature unique.

Laplace: It needs lots of bits to make it work, but most importantly it needs a computer programme. The computer programme can only work with code, which helps make the computer work. DNA (which stands for Deoxyribonucleic acid) is like the computer programme, except it is found in all living things. DNA contains the instructions for a living thing to grow and to work. These instructions are like the code in a computer programme, and in living things are called genes.

Lidstone: The nucleus is the part of a cell that contains the genetic information. The nucleus is surrounded by a nuclear membrane that, like the cell membrane, makes a boundary around the nucleus. Now let's explore chromosomes. Chromosomes are thread-like structures located inside the nucleus of animal and plant cells. Each chromosome is made of protein and a single molecule of deoxyribonucleic acid, or DNA. Passed from parents to offspring, DNA contains the specific instructions that make each type of living creature unique.

It can be observed that the passage returned from Dirichlet contains all query tokens while Laplace is missing the token 'specif' and 'uniqu' and Lidstone is missing the token 'thing'. This example prove that when some query tokens are absent from a passage, Laplace and Lidstone assign a relative large smoothing amount to the probabilities, leading to less relevant results. Since Laplace applies a larger smoothing amount ($\epsilon = 1$) than Lidstone ($\epsilon = 0.1$), the passage returned from Laplace is more irrelevant.

As previously discussed, both Laplace and Lidstone smoothing techniques are forms of additive smoothing, designed to ensure that the probabilities of unseens tokens are not zero. Laplace and Lidstone are therefore expected to be similar. This similarity is due to the fact that thery share the same generic formula as shown in Equation 3, where $\varepsilon = 1$ in Laplace.

$$P(W|D) = \frac{tf_{w,D} + \varepsilon}{|D| + \varepsilon|V|} \tag{3}$$

In the Lidstone Correction, I believe the value of $\varepsilon = 0.1$ is a reasonable choice for our models. Since the average document length in our corpus is relatively short and the occurrence of key words (excluding stop words) is generally less than 5 in a passage, and often equal to 1, setting $\varepsilon = 0.1$ does not give too much probabilities to unseen tokens. Lidstone model might be benefit from having smaller ε but not guaranteed to. This is also one of the reasons why Lidstone perform better than Laplace in the first example query.

In Dirichlet Smoothing, $\mu = 5000$ is not an appropriate choice for our corpus in my opinion. Since the Dirichelt smoothing follows the Equation 4, while average document length ($N_avg = 32.8$), $\frac{N}{N+\mu}$ will be nearly 0 and $\frac{\mu}{N+\mu}$ will be close to 1. This will results in the probability being predominantly dependent on background probability $P(W|C)$, which obviously is not the result we wanted. Hence, $\mu = 50$ should be a more appropriate value compared to $\mu = 5000$

$$P = \frac{N}{N+\mu} * P(W|D) + \frac{\mu}{N+\mu} * P(W|C) \tag{4}$$