Information Retrieval and Data Mining (COMP0084) Coursework 1

LiHao Liao

Abstract

Information retrieval models are crucial components of many applications, such as search, question answering and recommendation systems. In this experiment, I will establish a passage re-ranking system: given a candidate list of returned passages for a text query, re-ranking these passages based on an information retrieval model. The report is divided into three parts. The first part will explore different methods of selecting terms and the differences between empirical data and Zipf's law. The second part discusses how to build an inverted index. The final part examines the characteristics of language models, their empirical performance and the impact of smoothing parameters on different models.

1 Task 1 - Text statistics

1.1 Deliverable 1 & 2

Preprocessing In the data preprocessing function, I first replaced slashes with spaces as slashes are often used to separate different words that express the same meaning.

Then, I removed punctuation and numbers, which provides three benefits. First, my goal is to count the number of occurrences of terms, so punctuation doesn't help me. Removing punctuation makes my vocabulary index more accurate and improves counting efficiency. Second, this approach allows me to group tokens with minor differences, for example, changing "co-exist" to "coexist". Third, with punctuation removed, the meaning of numbers changes, such as "128.0.0.0" becoming "128000", making it unclear in the vocabulary whether it represents a number or network addresses.

Finally, I used the `split()` method for tokenisation, after the preprocessing the text only contained English letters. However, given the large volume of text and uncertainty about its original format, I did not pass a separator parameter, opting to use its underlying different splitting algorithm instead of just spaces(Python Software Foundation, 2020). During the normalisation process, I chose not to lower case everything because I would not further analyse the types of tokens in my preprocessing. Therefore, lower casing everything could easily change the meaning of some words, like the operating system "Windows" becoming "windows" in a house. In the end, I returned a dictionary using the collections.Counter function, which contains each term and the number of occurrences (counts the number of occurrences of terms). Hence, the size of the identified index of terms is the length of this dictionary, which is 215,257.

Probability of occurrences against frequency ranking After completing the preprocessing of text, I plotted their probability of occurrence against their frequency ranking and in order to qualitatively justify that these terms follow Zipf's law, I also compared it with Zipf's law. The formula for Zipf's law is shown in the Eq.1.

In this experiment, I set $s = 1$, where $f(\cdot)$ denotes the normalised frequency of a term, k denotes the term's frequency rank in our corpus (with $k = 1$ being the highest rank) and N is the number of terms in our vocabulary.

As show in the Figure 1, empirical data and the theoretical Zipf's law data essentially fall on top of each other. However, Zipf's law gives a higher probability of occurrence for terms with a higher frequency ranking. The next step I will convert the plot into a log-log plot for a better visual confirmation.

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

$$\log\left(f(k; s, N)\right) = -s\log(k) - \log\left(\sum_{i=1}^{N} i^{-s}\right) \qquad (2)$$

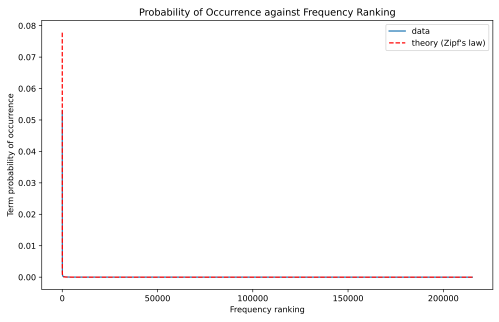

Figure 1: Probability of Occurrence against Frequency Ranking

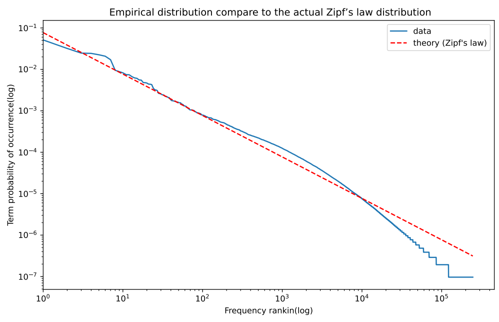

Figure 2: Empirical distribution compare to the actual Zipf's law distribution

Empirical distribution compare to the actual Zipf's law distribution in log-log plot Looking into the Figure 2, in my data the probability of occurrence for the highest frequency terms is less than the theoretical value predicted by Zipf's law. However, for terms ranked between $10^{0.5}$ and 10^{4}, their probability of occurrence is slightly higher than the theoretical values of Zipf's law. A significant difference appears in the last segment of the line. I assume that these differences stem from several factors. First, Eq.1 represents the general form of Zipf's law and since I am using a log-log plot, taking the logarithm of Eq.1 I can get Eq.2. It is at this point that I can observe that when s is constants, the slope is also constant to -1. This is why I obtain a straight line from Zipf's law in the Figure 2. But for the empirical distribution, the variation in the frequency of occurrences between terms of different ranks is not fixed. Therefore, when the points representing all the terms are connected, the resulting line does not form a straight line. Second, because the preprocessing did not remove very rare terms, these terms do not significantly help us and have led to larger differences in the lower ranks terms. Even if the empirical distribution does not completely fit the actual Zipf's law distribution visually, their trends are similar.

Empirical distribution compare to the actual Zipf's law distribution without stop words In the previou preprocessing, I did not remove stop words. Since stop words are generally unaffected by lowercase changes and I did not convert the entire text to lowercase. However, the stop word set in `NLTK` Stopwords Corpus(NLT) only includes lowercase cases, so I added a function to remove stop words that start with a capital letter like "the" and "The" during the process. In order to ensure the normal operation of the program and not to change the directory file resources of the running machine, I have hard-coded these words.

Figure 3 shows the comparison after removing stop words. Obviously, removing stop words has a significant impact on the empirical distribution. However, the figures do not provide precise data, so Figure 4 shows the Kullback–Leibler divergence (Wikipedia, 2024) from the two experiments. Their values are approximately 0.34 and 0.1. Such results indicate that the empirical distribution of the text without removing stop words is closer to the expected distribution of Zipf's law. This may be because stop words are usually the most common words in the text (they occupy higher frequency rankings) and their removal changes the overall distribution of word frequencies which makes the terms ranked in the middle section become the highest ranked section, causing the distribution of the remaining vocabulary to deviate more significantly from the expected distribution of Zipf's law. In addition, due to the significant difference observed in Figure 3, I also compared the proportion of term frequencies between $a = 10^{-5}$ and $b = 10^{-2.5}$ according to the experimental data and Zipf's law to further demonstrate why there is such a difference. Figure 5 shows the results indicate that according to Zipf's law, there should be 3.61% of terms within this frequency range, but in my empirical distribution I have 5.26% of terms within this range. This further illustrates why the empirical distribution difference from the Zipf's law distribution. Therefore, by comparing Figure 2 and Figure 3, it can be observed that the occurrence probability of stop words in these paragraphs is very high. Removing stop words results in relatively low-frequency words obtaining higher rankings, leading to a situation where there are higher terms proportion within a certain frequency range.

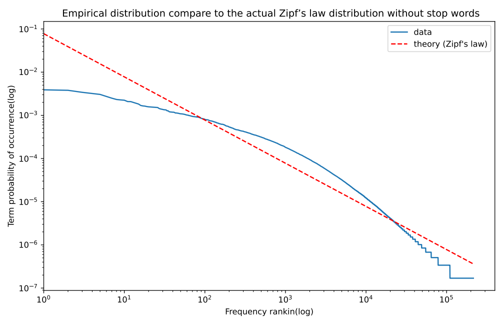

Figure 3: Empirical distribution compare to the actual Zipf's law distribution

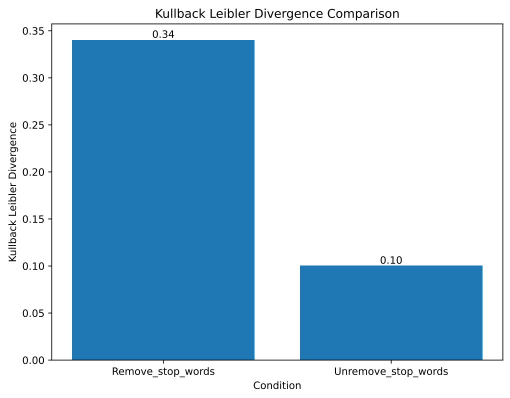

Figure 4: Empirical distribution compare to the actual Zipf's law distribution

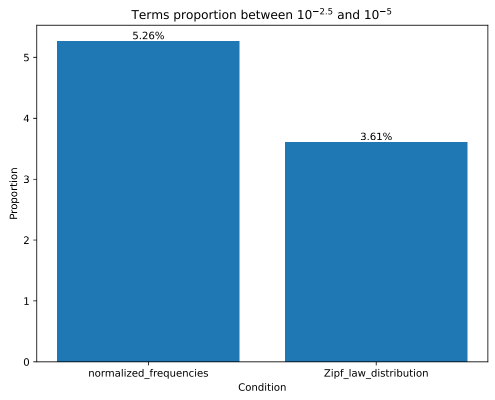

Figure 5: Empirical distribution compare to the actual Zipf's law distribution

2 Task 2 - Inverted index

2.1 Deliverable 3 & 4

Removing stop words In the process of building the inverted index, my preprocessing is consistent with the method mentioned in Task1, but I decided to remove the stop words. I made this decision for two reasons: firstly, to enhance search efficiency and save storage space. From the experiments in Task1, it is obvious that the occurrence frequency of stop words is extremely high. If stop words are retained as part of the inverted index, it would significantly increase the size of the index and the time it takes to traverse the index during a search. Moreover, the built inverted index will be stored as part of subsequent experiments, as each word needs to record which passages it appears in. Removing these words can not only reduce the size of the index but also save storage space. The second reason is to improve the relevance of the search results. Stop words appear almost in every passages and contribute little to the query, so in order to enhance the relevance and accuracy of the search results, I chose to remove them.

Information store in inverted index The inverted index can store a lot of information. In this experiment, I decided to store the frequency of each term appearing in the passage. I chose to store this information because it is necessary for subsequent experiments such as implementing tf-idf and BM25 algorithms, as well as building language models like Laplace smoothing. The frequency of each term appearing in the passage is essential for these experiments, so storing it in the inverted index can facilitate the implementation of subsequent experiments. Eq.3 and Eq.4 show the calculation methods of tf and idf(Wikipedia, 2023). Where $tf_{t,d}$ represents the number of times a term appears in a passage, while D represents the total number of terms in the passage (Total length of passage). N represents the number of passages in the collection and n_t represents the number of passages in which term t appears. As for language models like Lidstone correction, its formula is shown in Eq.5, where $tf_{w,D}$ still represents the number of times a term appears in a passage, but with the addition of a smoothing parameter . D also represents the total number of terms in the passage, while V represents the number of terms contained in the entire vocabulary. Inverted index allows me to directly access the passages where terms appear as well as their frequencies. The value of D can be obtained through the length of the value in the dictionary **passages_id_and_terms_info** that I established during the build of the inverted index, the structure of which will be mentioned in the next paragraph. These are the reasons why I store these information.

$$TF(t,d) = \left(\frac{tf_{t,d}}{D}\right) \tag{3}$$

$$IDF(t,d) = \log_{10}\left(\frac{N}{n_t}\right) \tag{4}$$

$$P_{Lidstone} = \frac{tf_{w,D} + \varepsilon}{|D| + \varepsilon|V|} \tag{5}$$

Approach to generating an inverted index Before building the inverted index, I performed some data preprocessing. In the Task1, I stored the identified index of terms as a txt file for reading. The file candidate-passages-top1000 contains 4 columns of data, but I only needed two of these columns to build the inverted index. Therefore, I built a dictionary **passages_id_and_terms_info** to store the terms appearing in each passage. Its key is pid while the value is all terms in the passage after preprocessing. These terms were selected based on the terms stored in the file from Task 1. After completing this preprocessing, I began building the inverted index. I chose the **defaultdict(dict)** data structure because I wanted the structure of the inverted index to be as shown in Eq.6. The first dictionary uses terms as keys and its value is the second dictionary, where the passage id acts as the key and the count of the term appearing in the passage acts as the value. By traversing the **passages_id_and_terms_info** that was just built, I could quickly build the inverted index. Finally, I stored both **passages_id_and_terms_info** and the **inverted index** as JSON files for future use.

$$\{\{terms : \{pid : count\}\} \tag{6}$$

3 Task 4 - Query likelihood language models

Based on the results from three language models, I will extract and analyze the top two passages returned by each model for the same query respectively and compare them. In order to present the experimental data more intuitively, both the queries and passages displayed are in the preprocessed form used for calculation.

Original Query: what slows down the flow of blood
Preprocessed Query: slows flow blood

Laplace Smoothing:

- Capable undergoing vasoconstriction vasodilation influence blood flow blood pressure
- condition occurs blood clot blocks artery disrupts flow blood brain

Lidstone Correction:

- Pulmonary valve stenosis condition deformity near pulmonary valve valve influences blood flow heart lungs slows blood flow
- Arterial insufficiency condition slows stops flow blood arteries Arteries blood vessels carry blood heart places body

Dirichlet Smoothing

- diastolic blood pressure low deliver blood capillaries high flow rate systemic show capillary walls thin enough allow oxygen exchange cells Capillaries far heart blood flow slows distance heart increasesc diastolic blood pressure low deliver blood capillaries high flow ratehe diastolic blood pressure low deliver blood capillaries high flow rate systemic capillaries supplied left ventricle lower cardiac output right ventricle
- Blood flow often slows bulging section aortic aneurysm causing clots form blood clot breaks aortic aneurysm chest area travel brain cause stroke Blood clots break aortic aneurysm belly area block blood flow belly legslood flow often slows bulging section aortic aneurysm causing clots form blood clot breaks aortic aneurysm chest area travel brain cause stroke

$$P_{Laplace}(w|D) = \frac{tf_{w,D} + 1}{|D| + |V|} \tag{7}$$

$$P_{Di}(w|D) = \left(\frac{D}{D+\mu} \cdot \frac{tf_{w,D}}{D}\right) + \left(\frac{\mu}{D+\mu} \cdot \frac{tf_{w,V}}{N}\right) \quad (8)$$

Which language model do you expect to work better? Based on the results, a notable feature I observed is that compare to Laplace and Lidstone the passages returned by Dirichlet tend to be longer. I believe this phenomenon arises because unlike the former two, Dirichlet also considers information from the entire corpus. As show in Eq.8, where $tf_{w,V}$ represents the frequency of word w in the entire corpus V, while N denotes the total number of words in the corpus. Through $\frac{tf_{w,V}}{N}$, even words with low frequencies in the current passage can obtain higher probabilities due to their high frequencies in the entire corpus. In this query, it's evident that "blood" carries significant weight globally.

Furthermore, upon a brief review of the results, all models answered the query "what slows down the flow of blood", but the results from Dirichlet Smoothing are comparatively more specific and detailed. Therefore, I consider Dirichlet Smoothing to be work better.

Which language models are expected to be more similar I believe Laplace and Lidstone are more similar. Firstly, it can be seen from Eq.5 and 7 that they have similar structures, with the only difference being whether the smoothing parameter ϵ is involved in the calculation. If ϵ is set to 1, then they are completely identical. Secondly, using the previous query as an example, the scores returned by Laplace fall in the range [-6.71,-8.57], while those returned by Lidstone fall in the range [-7.14,-9.65] and the score range returned by Dirichlet is [-10.74,-12.6]. From this perspective, the result intervals obtained by their calculation formulas are also more similar. Finally, as mentioned in the previous analysis they pay more attention to local information, so they tend to return shorter passages.

Choice of ϵ In this experiment, I believe setting ϵ = 0.1 is a good choice. Smoothing is performed to handle zero probability events, but at the same time we need to ensure that it does not significantly alter the distribution of terms or allocate too much weight. In this experiment, the maximum length D of the processed passages is only 152, which is not a large dataset. Choosing $\epsilon = 0.1$ appropriately removes the occurrence of zero probability events. Additionally, I also compared the results by setting ϵ to 0.4 and 0.7 (increasing by 0.3 each time). As shown in the Figure 6, as ϵ increases the gap between the results becomes larger. This is because in the case of a small dataset, as ϵ increases, more weight is allocated to other terms by ϵ, so setting $\epsilon = 0.1$ is more appropriate.

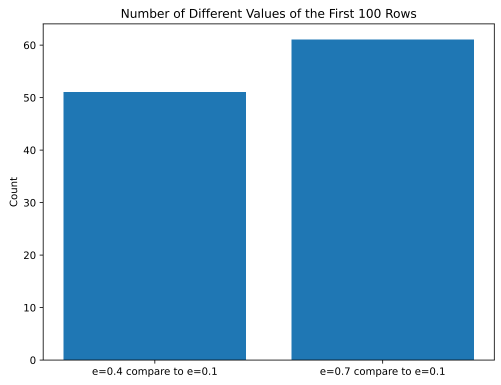

Figure 6: Differences in results under different value e

Choice of μ I believe setting $\mu = 5000$ is not a good choice for this experiment. The value of μ directly determines whether the search is more based on local information or global information. As I mentioned earlier, the maximum length D of the processed passages is only 152. Therefore, setting $\mu = 5000$ would result in almost zero weight on local information according to Eq. 8 - $\frac{D}{D+\mu}$. If we take the example of the longest 152-word passage, when $\mu = 5000$, $\frac{D}{D+\mu}$ is approximately 0.0029, and when $\mu = 50$ $\frac{D}{D+\mu}$ is approximately 0.75. The order of magnitude difference between these two numbers is approximately $10^{-2.4}$. Such a difference results in a significant change in the weighting of the query. In such a scenario, this is not a good choice to put almost all the weight on global information.

References

NLTK: Natural Language Toolkit. `https://www.nltk.org/`. Accessed: 2024-02-21.

Python Software Foundation. 2020. Python 3.9 documentation: str.split. Accessed: 2024-02-22.

Wikipedia. 2023. Tf-idf. Accessed 2024-02-24.

Wikipedia. 2024. Kullback–leibler divergence - wikipedia. Accessed: 2024-02-22.