Information Retrieval and Data Mining (COMP0084)

Anonymous ACL submission

1 Task 1

The Task1.py script processes the passage-collection.txt file using several text processing steps such as lower casing, expanding contractions, removing punctuation numbers and also stop words(if required). Tokenization is also performed, to compute the frequency of each word using a counter. The normalised frequency for each word is then calculated to determine whether it follows Zipf's law. Zipf's Law states that the frequency of a word is inversely proportional to its rank. The empirical word frequency distribution is compared to the theoretical Zipf distribution using the Kullback-Leibler (KL) divergence using log-log plots. This helps to calculate how much the observed word frequencies deviate from Zipf's expectation. This analysis was performed twice once, with stop words and once with stop words removed. The figures are shown below.

Several pre processing techniques have been used. Firstly, the text is converted to lowercase to ensure the same words that are uppercase and lowercase are treated the same. Lower casing is a standard initial pre processing step as it removes case based distinctions which don't affect the meaning of the word. This helps to reduce the dimensionality of the text. The input text contains a lot of common contractions such as "don't", "it's" etc. These contractions have been expanded(E.g. "don't" is converted to do not) to prevent variations in contractions from affecting the text analysis, this leads to more accurate tokenization. Then, Apostrophes have been removed. For example words such as O'Connor have been converted to OConner to avoid creating different tokenization for words that are semantically the same. Then, hyphens are converted into spaces. For example, words "real-time" are treated as two separate words("real" and "time"), rather than one token. This is done so that the compound words are split appropriately. Punctuation is also removed since it does not add a value in word frequency analysis, it can also introduce noise and unnecessary distinctions between words that are semantically similar(E.g "hello!" and "hello"). Numbers have also been removed, numbers can hold significance for e.g. in financial documents or scientific text. These have been removed since our primary focus is on word frequency analysis and demonstrating Zipf's law. Also multiple spaces between words are replaced with a single space and leading spaces are stripped to avoid tokenization of these multiple spaces. Then tokenization is carried out to convert the raw text into a list of individual words. Furthermore, an option to remove stop words has been added. Removing stop words can help in focusing on the most informative words, but in some instances, depending on the text analysis that is carried out. Keeping stop words is beneficial.

The total number of unique terms in the identified vocabulary is 120,834.

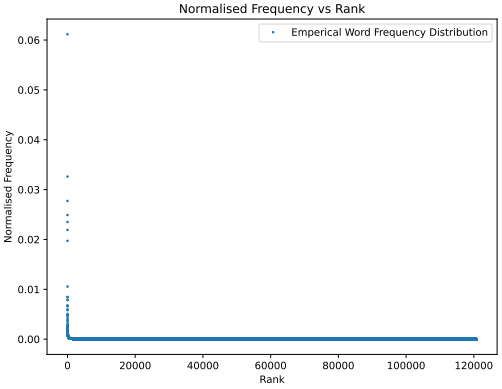

Figure 1: Normalised Frequency vs Rank.

Figure 1. Shows the probability of occurrences(normalised frequency) against their fre-

quency ranking. It shows that the terms follow Zipf's law. This states that the word frequency is inversely proportional to its rank. This is shown by the plot. Only A small number of words occur with very high frequency, while the majority of the terms occur infrequently, which causes the long tail.

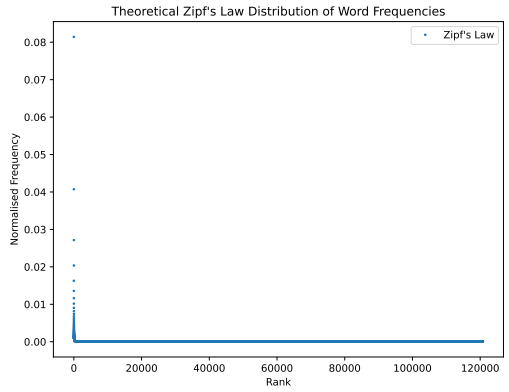

Figure 1.1: Theoretical Zipf's distribution.

Figure 1.1 shows the Theoretical normalised Zipf's frequency against rank, this shows that the normalised frequency for our data shown in Figure 1 closely follows the Zipfian trend.

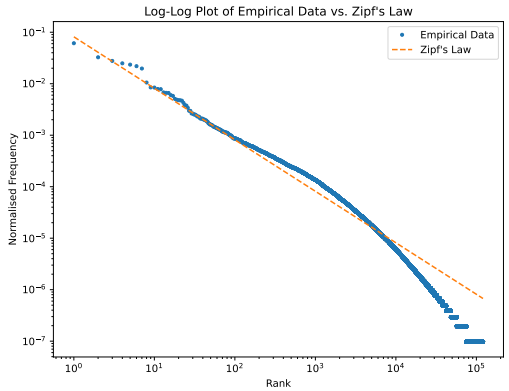

Figure 2: Log-Log plot comparing empirical word frequencies to Zipf's Law.

Figure 2 shows the Log-Log plot comparing the log of the empirical Data to Zipf's Law. For a significant portion of the ranks, the empirical data follows the Zipfian distribution very closely. However, at the lower ranks(I.e high frequency words), the empirical data is slightly above the expected Zipfian distribution. This means that the most frequent words appear more often than expected by zipf's Law. One reason could be that the common words appear disproportionately more often than content words, hence could be the reason for the initial steeper decline at the lower ranks, causing the empirical curve to slightly exceed the theoretical Zipfian trend for lower-rank words.

Also, at higher ranks(I.e. low-frequency words), the empirical data is seen to be diverging below the Zipfian exception as the ranks increase, which means that the rare words occur less frequently than predicted. This can be due to the corpus containing specialized or uncommon words. The shape of the empirical frequency is strongly dependent on the size and content of the corpus. If the dataset is large and diverse, it will more closely follow Zipf's law because word frequencies naturally balance out across different contexts. But if the corpus is domain-specific, certain specialized terms appear more frequently than expected leading to common terms appearing less frequently than expected. This is seen in the corpus text file 'passage-collection.txt'. The corpus is mostly technical covering several domains such as biology, finance, firearms, banking, physics. For instance, blood is ranked 9 most common word with 15112 occurrences. This means that it contains a considerable amount of domain-specific terms. These terms would occur frequently within their respective fields but rarely occur outside of them, which explain the divergence in the higher ranks.

Furthermore, Zipf's law assumes that words are independently distributed based on their utility. However, in natural language, words are not independent. Many words can frequently appear together in phrases, named entities and collocations. For example "New York" may appear together far more often than what Zipf's Law assumes for these two separate words. This can affect the middle to lower ranks of the empirical distribution, and create a deviation from the predictions given by the Zipf's law equation.

Figure 3: Log-Log plot of Zipf's Law after removing stopwords.

Figure 3. Shows the Log-Log plot comparing the log of the empirical Data to Zipf's Law without the stop words. This shows that when the stop words are removed, the empirical word frequency distribution diverges more significantly from Zipf's law. This difference can also be analysed using the KL divergence which measures how one probability distribution differs from another. From the code output, it can be seen that the KL divergence increases from 0.112 to 0.362. This increase means that after removing the stop words, the empirical distribution is moving away from the Zipfian distribution. Figure 2 shows the empirical data following Zipf's law more closely since their presence leads to a smoother frequency decay and leads to the stabilisation of the distribution. Removing the stop words cuts the most frequent words which occupy the top ranks in the Zipfian distribution, this causes a significant drop in the head of the distribution. The remaining word often tends to be more topic-specific, which makes the frequency distribution noisier and less regular.

2 Task 2

In Task 2, an inverted index is generated. An Inverted index is a fundamental data structure used for fast retrieval and maps the terms observed in the corpus to the passages in which they occur. Firstly, the corpus was preprocessed in Task 1. This was used to construct the index, firstly the passages from 'candidate-passages-top1000.tsv' were read. Then, unique words from each passage were extracted. Passage ID(pid) was stored under each word in a dictionary. The inverted index is saved when task 2 is run with the name 'inverted_index.json'.

In this exercise, I employed an inverted index to store and retrieve passage information in an efficient way for ranking models such as TF-IDF, BM25, and Query Likelihood models (Laplace, Lidstone, and Dirichlet smoothing). The inverted index is built from candidate-passages-top1000.tsv, where each passage is preprocessed and stored in a dictionary data structure for fast lookup. Pre processing includes tokenization, lower casing, contraction expansion, punctuation stripping, and optional stop word removal to ensure term representation in documents is consistent. The passages are then looped through individually to identify unique words, and those words are then utilized as dictionary keys in the inverted index. The values of each term are (1) 'doc_freq', the count of the word as a number of passages, (2) 'total_term_freq', the total frequency of the word in all passages, and (3) 'postings', a nested dictionary from passage IDs (pid) to their term frequency (tf) and the length of the passage. This structure enables to efficiently compute TF-IDF scores, where 'doc_freq' is utilized to compute the Inverse Document Frequency (IDF) and 'tf' for computing term importance in a document. The 'postings' dictionary facilitates us to achieve meaningful statistics for BM25 normalization, which requires passage length for efficient scoring adjustments. The 'total_term_freq' statistic is significant to query likelihood models, as it helps smooth probability estimates, particularly in Dirichlet smoothing, in which term frequency is compared to collection-level term frequency. Passage lengths in the inverted index prevent repeated calculation and provide for efficient lookup when normalizing scores in ranking models. The motivation behind this design is scoring efficiency of Tasks 3 and 4 using ranking of passages from query-document similarities. The structured index facilitates quick retrieval of term statistics, which leads to computationally efficient scoring models. This approach ensures that all of the ranking models—TF-IDF, BM25, Laplace, Lidstone, and Dirichlet smoothing—have convenient access to term statistics without additional computation requirements, rendering the retrieval pipeline highly optimized. In most cases, the inverted index is designed to be compatible with several retrieval models with scalability and efficiency, ensuring that the system is capable of efficiently handling

large amounts of information.

3 Task 3

In Task 3, I used TF-IDF and BM25 retrieval models on an inverted index that was built from candidate-passages-top1000.tsv. I computed TF-IDF vectors for queries and passages, and ranked passages by cosine similarity, saving the top 100 results for each query in tfidf.csv. I also used BM25 with k1 = 1.2, k2 = 100, b = 0.75, ranking passages and saving results in bm25.csv. Both documents strictly adhere to the specified format and have precisely 19,290 rows.

4 Task 4

I expect Dirichlet smoothing to work better compared to Laplace and Lidstone smoothing for passage retrieve. One reason is because, Laplace smoothing applies a uniform additive smoothing(I.e. Adding 1 to every term count). This can distort the probability distributions, particularly for low-frequency words. This would cause overestimation of rare terms and under-estimation of frequent terms, this would affect the retrieval effectiveness. Lidstone smoothing reduces the issues caused by Laplace's smoothie by adding a smaller constant than 1. However, it still does not account for the term important based on corpus-wide distribution. In comparison, Dirichlet smoothing adjusts the probabilities of the terms dynamical baked on collection-level frequencies. Hence, terms that are common in the overall corpus will have probabilities that are more reasonable even when they are absent from a particular passage.

This improved performance can be seen in the 'Dirichlet.csv' file that is outputted from 'task4.py' file. It shows that the Dirichlet model produces higher-log likelihood scores for relevant passages compared to the Laplace's smoothing and Lidstone smoothing shown in 'laplace.csv' and 'lidstone.csv' respectively. In queries where important keywords are rare in the passage but frequent in the overall corpus, Dirichlet smoothing helps balance term importance leading to better ranking of relevant passages. Laplace smoothing tends to rank longer passages higher since it normalizes based on total words, this would disproportionately benefit passages with many words. Furthermore, Lidstone smoothing, even though it is an improvement over Laplace, still struggles in some cases. It may not perform well when dealing with rare but important terms in lengthy tests.

The Laplace and Lidstone models are expected to be more similar to each other compared to Dirichlet smoothing. This is because Laplace and Lidstone models perform fixed additive smoothing(I.e Laplace adds 1, Lidstone adds ϵ), hence, modifies the probabilities in a similar way. For instance, from my data and the .csv files mentioned above, the scores for the Dirichlet model for relevant passages are generally in the range of -28 to -32, whereas in the Lidstone model, the scores are significantly lower at around -60 or lower. Similarly, Laplace performs even worse with scores below -70. The higher scores in the Dirichlet model suggests that it assigns more probability mass to observed words in the passage while it smoothed unseen words, this leads to a more accurate ranking of passages.

The difference is due to Lidstone using a smaller smoothing factor(e = 0.1). This makes its probability closer to the raw term frequencies than Laplace. However, Dirichlet smoothing uses a global corpus frequency information, causing it to behave differently from both Laplace and Lidstone. Lidstone and Laplace smoothing models both have far worse log-likelihood scores than Dirichlet, while ranking the passages in a similar order. The Lidstone model (ϵ = 0.1) yields scores ranging from -61.66 to -64.05, and the Laplace model yields even worse scores, ranging from -71.67 to -73.95. This indicates that both techniques heavily penalize unseen terms due to their additive smoothing approach. Their proximity arises from the fact that they operate at the passage level, without considering collection-wide term frequencies. While Laplace smoothing presumes ϵ = 1, distributing probability mass more equally, Lidstone allows tuning with ϵ = 0.1.

The choice of ϵ = 0.1 in the Lidstone model is a bit conservative and is not necessarily an optimal value for smoothing- versus differentiation-bias trade offs among term frequencies. According to the lidstone.csv file, the log-likelihood scores in the Lidstone model are considerably lower than

for Dirichlet smoothing, indicating that passages are over-smoothed, reducing their ability to note substantial variation in term frequencies. A smaller value of ϵ (e.g., 0.01) would lessen the smoothing effect, thereby making high-frequency terms count more and also leaving non-zero probabilities for previously unseen words. Increasing ϵ to values such as 0.5 or 1.0 would result in an equivalent operation like Laplace smoothing, which has been shown to result in worse performance. An ideal range for ϵ would be between 0.01 and 0.05 because this range would cut down on over-smoothing while providing robustness to rare terms.

Setting $\mu = 5000$ in Dirichlet smoothing would drastically increase the reliance on corpus-wide statistics, reducing the impact of passage term frequencies. The current setting of $\mu = 50$ provides a good balance between passage-specific and corpus-wide term frequencies to give rise to effective rankings. But increasing μ to 5000 would make the model depend too much on global term probabilities, making it less sensitive to the specific aspects of particular passages. While this change might be useful when retrieving with queries that have very uncommon terms, it might lead to excessively general rankings in the usual retrieval case, degrading relevance. A more appropriate range would be $\mu = 100$ to 500, as this permits some effect from the set while enabling differentiation between passages, ultimately resulting in better retrieval performance.