Information Retrieval and Data Mining: Coursework 1

Anonymous ACL submission

1 Introduction

This coursework develops information retrieval models for passage retrieval. For a given list of passages from a text query, the passages are re-ranked based on the different retrieval models.

2 Task 1 - Text Statistics

The first preprocessing choice was case-folding. This ensures all the words in a string (the passage) are in lowercase, thus enabling a term such as 'Dictionary' at the start of a sentence to match with a query of 'dictionary'. Even though this may lead to unintended matches for proper nouns, it avoids the expectation that queries will be capitalised correctly.

The next step was to tokenise the passage into individual terms. The RegexpTokenizer from nltk was used, because it forms tokens out of alphabetic sequences, money expressions, and any other non-whitespace sequences. Alternatives that were considered included the whitespace tokenizer or the punctuation tokeniser. However, the RegexpTokenizer was chosen as it incorporates several rules of linguistics to split the sentence into the most optimal tokens.

One potential drawback is that it assumes the document unit is already in sentences, fortunately for the current use-case this is true. Finally, the option of removing English stopwords was provided using the nltk.corpus stopwords. Stopwords were not removed for Task 1. Stemming and lemmatisation were not used as the additional retrieval performance is relatively small for the English language (+4% for stemming and -10% for lemmatisation) (Hollink et al., 2004), although it would reduce the storage requirements. Following the preprocessing steps, one-gram terms have been extracted from the text. A vocabulary of 117,225 was created.

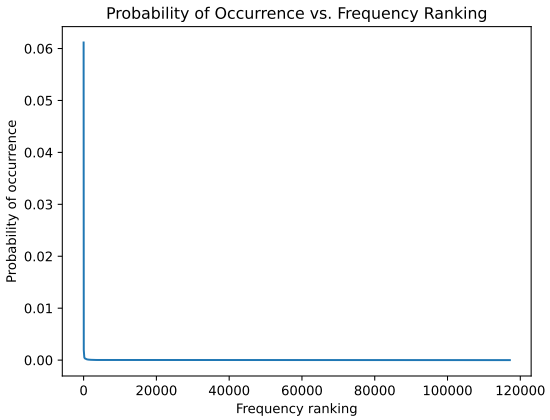

Figure 1: Normalised Frequency vs. Frequency Ranking

Figure 1, above, shows the normalised frequency of terms in the data versus their frequency ranking. Zipf's law states that the collection frequency of the ith most common term is proportional to $1/i$. The intuition behind Zipf's law is that frequency rapidly decreases with rank, such that the second most common term only appears half as often as the most common term. Figure 1 exemplifies Zipf's law, as one can observe the rapid decline in the probability of occurrence as the frequency ranking increases.

In Figure 2 the empirical distribution closely follows the actual Zipf's law distribution, until the low end of the frequency rankings (the most infrequent terms) where it begins to significantly deviate. As was previously explained, the equation for the Zipf's law distribution assumes that the decline in collection frequency falls proportionally to the rank. An explanation for the deviation is that there are a high number of terms in the data that are equally infrequent (for example spelling mistakes, which mean the term very rarely occurs). Therefore, the probability of occurrence is equally low for many of the terms at the low end of the frequency ranking.

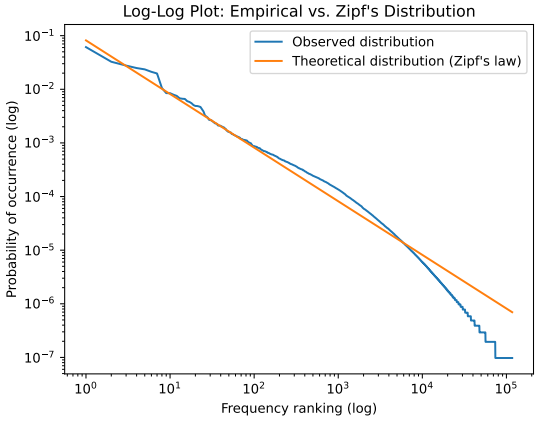

Figure 2: Log-log plot: Empirical and Zipf's Law Distribution

Furthermore, the power law no longer holds and the relationship between frequency ranking and probability of occurrence changes.

Removing stop words will cause more significant deviations between the two distributions as it will remove many of the terms at top of the frequency ranking, which also have a high probability of occurrence. Consequently, the terms which now represent the upper end of the frequency ranking will not have the proportional decline in the probability of occurrence as the terms are not used as frequently in the English language to form grammatically correct sentences. This can be observed visually in Table 1.

Word	Count	Multiplier	Zipf's
the	626,892	-	-
of	334,283	1.9x	2.0x
a	283,558	2.2x	3.0x
and	255,211	2.5x	4.0x
to	240,943	2.6x	5.0x

Table 1: Five most common words by frequency ranking (including stop words)

Word	Count	Multiplier	Zipf's
one	27,299	-	-
name	25,102	1.1x	2.0x
also	21,757	1.3x	3.0x
number	21,351	1.3x	4.0x
may	20,556	1.3x	5.0x

Table 2: Five most common words by frequency ranking (excluding stop words)

The change in the distribution is quantified in Table 1 & 2, which show the five most common words by count, the multiplier of their count to the most common word, and the corresponding theoretical multiplier according to Zipf's law. Table 1 shows the top five words by frequency rank when including stop words, while the data in Table 2 excludes stop words. It can be observed that in Table 1 the actual multiplier from the data follows the Zipf's law multiplier much closer than in Table 2, thus proving the aforementioned statement that when stop words are removed the probability of occurrence no longer declines proportionally to the frequency ranking.

3 Task 2 - Inverted Index

From Task 2 onwards, stop words were removed during the proprocessing step. The justification behind this was to improve the processing time due to having a smaller postings list. This will improve the efficiency of the inverted index, as the terms on the stop words list are likely to have the highest frequency of occurrence in the documents. Given the way the inverted index was constructed (see below), it will result in a significant reduction in the number of iterations the file will run through. It is acknowledged that this drawback could be resolved by using good compression techniques to reduce the cost of storing the postings. The result of doing so would lead to a minimal additional cost of storage and query processing time for the benefit of a better retrieval performance.

To build the inverted index, first the 'candidate-passages-top1000.tsv' file was loaded using pandas 'read.csv' function. The tab delimiter was used as the file is a tab-separated values file. In addition, the columns were named so later it is easier to isolate the query id, passage id, query or passage. This was particularly useful in Tasks 3 and 4. The passage id was used to archive the TF-IDF for each token within each passage, while the query id was used to isolate all the associated queries for a given query id when iterating through the test queries file in task 4.

After the data was loaded, a blank dictionary was created for the inverted index, and a separate DataFrame was created for the unique passages. The 'drop duplicates' method was used on the passage id column for this task.

After multiples iterations and changes to the inverted index after attempting Task 3 and Task 4, the most suitable inverted index format chosen was

to have a dictionary where each token represents a key, and the values are a list of tuples containing the passage id, frequency and length of the passage associated with the passage id. This allowed a straightforward method to build the TF-IDF representation of the passages. Likewise, for the BM25 it was undemanding to find the number of times a token/term occurred in a the corpus of passages, as it was the length of the values from the inverted index.

To create the index, each row of the unique passages DataFrame was iterated through, during which: 1) The passage was preprocessed into tokens, 2) the frequency of each token was counted using the Counter class from collections and stored in 'token frequency', 3) the length of the passage was counted using the len function on the output of the Counter function. Next, a separate for loop iterating through each token and its associated frequency meant that tokens with a frequency of one could be removed as they are very rare. The assumption was taken that these were more likely to be spelling mistakes than useful terms, so it is expected to improve precision, even if this means minimally harming recall. Finally, the setdefault method was used to ensure the value was set to the correct token before appending the list of values for the token with a tuple containing the passage id, passage frequency and the passage length.

As was mentioned in Task 1, stemming and lemmatisation were not used for the same reasons. Additionally, given stop words and infrequent terms were removed, performing stemming and/or lemmatisation would further simplify the vocabulary, thus further harming recall.

4 Task 3

Task 3 does not require any additional explanation.

5 Task 4

5.1 Best performing model

Since the queries in the test-queries.tsv file are relatively short on average, then the Dirichlet smoothing model is expected to work better than Laplace and Lidstone. By taking into account the document length for the degree of smoothing, it mitigates the likelihood that probability of unseen terms is either too high or too low. Dirichlet smoothing provides a weighted linear connection between the maximum likelihood estimate and the probability of the term occurring in the entire document collection. It accounts for the sample size when calculating the degree of smoothing. If the document length is longer more smoothing is needed as it is more prone to over-fitting, so the model becomes less sensitive to the observed data. This dynamic smoothing should result in a better performance. This is reflected in the experiments as the Dirichlet.csv file shows that the natural log probabilities of each query id retrieved are typically less negative (better) than both the Laplace and Lidstone log probabilities. This is due to the model retrieving passages that better match the query. A clear example of this is the first query id retrieved 1108939, where the best log probability for the passage retrieved is -9.73 (to 2 decimal places), versus -27.78 and -28.80 for Laplace and Lidstone, respectively.

A common drawback of Laplace smoothing is that unseen terms are given too much weight. Lidstone smoothing accounts for this by giving a smaller weight to unseen words. However, it treats each unseen word equally.

If the time was taken to properly tune the μ parameter, then it would give even greater confidence that Dirichlet smoothing would be the best performing model.

5.2 Most similar performance

The Laplace and Lidstone models are expected to be more similar as they are both simple smoothing methods, known as additive smoothing. Neither method uses interpolation of background probabilities (the probability of terms occurring in the entire document collection) for smoothing. Both methods add a constant to the term count and the Lidstone correction also adds a (small) constant to the document length, so the formulas of both methods are very similar. As a result, it can be expected that both models will produce similar retrieval results. This can be observed in the Laplace.csv and Lidstone.csv files, as the top five scoring passage id's retrieved for both the Lidstone model can be found in the top thirty for the Laplace model. The same can be observed for the top five passage id's retrieved from the Laplace model.

5.3 Choice of ε for Lidstone correction

The Lidstone correction was formulated as an improvement to Laplace smoothing, which adds one to every count (corresponding to having a uniform prior over words). The drawback of Laplace smoothing is that it gives too much weight to un-

seen terms, even though it succeeds in avoiding having zero counts. Therefore, by adding a small positive value less weight is given to unseen terms. With $\varepsilon = 0.1$ the value for epsilon is relatively small which will result in less smoothing. Typically, small values of ε are appropriate for large training data sets with large vocabularies as the likelihood of unseen terms are relatively low. The vocabulary size specified in Task 1 was 117,225 before removing stop words, compared to the second edition of the Oxford English Dictionary (OED) which defined more than 600,000 words (Manning et al., 2008). The vocabulary size of many large collections is even larger than the OED as it contains names of people, locations and more, so the vocabulary size from Task 1 is relatively sparse in comparison. As a result, $\varepsilon = 0.1$ is low and a better choice would be something closer to 0.5. However, a better way to find the optimal value would be through evaluating model performance on a validation set.

5.4 Choice of μ for Dirichlet smoothing

The value of μ in Dirichlet smoothing is sometimes chosen to be an estimate of the average document length in the collection. Therefore, by setting μ to 5,000, this would be setting μ even further away from the observed average document length of twenty-three. There is empirical evidence that increasing the value to 5,000 would be less appropriate as the scores for the top scoring passages for each query id worsen when this change is made to the results of Task 4.

The impact of increasing μ to 5,000 would make the query likelihood language model more similar to the background model and less sensitive to the observed data as it would decrease λ.

References

D Manning Christopher, Raghavan Prabhakar, Schütze Hinrich, et al. 2008. Introduction to information retrieval. *An Introduction To Information Retrieval*, 151(177):5.

Vera Hollink, Jaap Kamps, Christof Monz, and Maarten De Rijke. 2004. Monolingual document retrieval for european languages. *Information retrieval*, 7:33–52.