COMP0084: Information Retrieval and Data Mining Coursework 1

March 1, 2022

1 Task 1 - Text statistics

1.1 Text preprocessing

The corpus of raw text was preprocessed before extracting the terms for 1-gram models. First, basic text normalization steps were carried out, such as removal of special characters, converting to lowercase and grouping words with minor differences together. Then, Tokenization is done followed by Lemmatisation.

Text Normalization The following text normalisation steps were carried out:

1. Special characters add noise to the corpus, hence removal of such characters is an important step so that only meaningful words are taken into consideration. Following that, all the words were converted to lowercase. This was done to ensure that the words in the corpus are not considered differently due to their letter case.

2. Different abbreviations of the same country were mapped to the same word (e.g. 'u.s.a' and 'u.s' mapped to 'us') so that such words are not treated differently. Also, abbreviations with periods will be split into individual letters when tokenised, hence mapping them to a shortened form with only words will help us resolve this issue.

3. Common contraction forms are grouped together by removing the apostrophe in the word. This helps maintain the structure of the sentences, since due to the apostrophe the word will be split in two parts. The word "can't" will no longer be split into "can" and "t" but instead will result in the token 'cant'. This ensures that the meaning of the words are preserved.

4. Common hyphenated words with prefixes and suffixes are grouped together by removing the apostrophe in the word. These combining forms added at the beginning or ending of a word completely change the meaning of it.

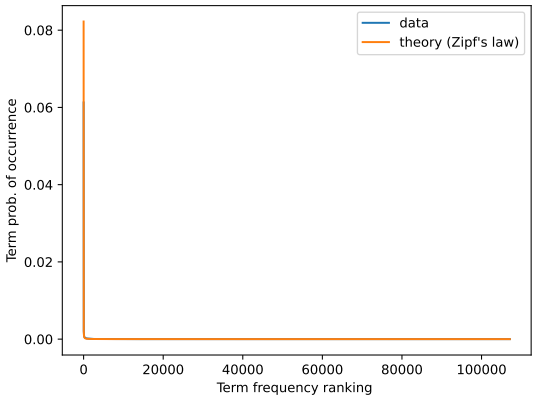

Figure 1: Ziph's law vs observed data

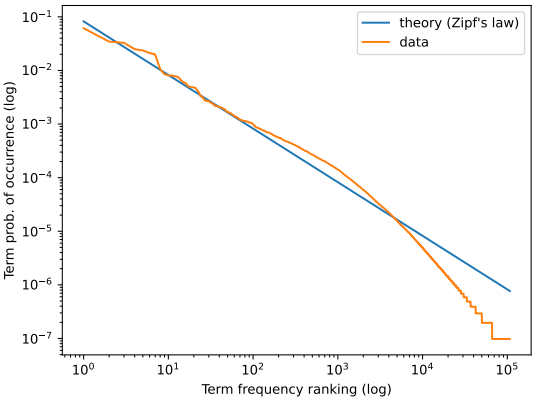

Figure 2: Log-log plot Ziph's law vs observed data

For example, "anti-inflammatory" means the opposite of "inflammatory". Hence, grouping them together will preserve the meaning of the compounded word. Otherwise these words would be split into two tokens, stripping the word of it's meaning.

Tokenization and Lemmatisation The normalized paragraphs were then processed into unigram tokens by breaking up the paragraphs into 1-word tokens by whitespace. This was done using nltk's tokenizer. Finally the tokens are lemmatized using nltk's lemmatizer. This helps in grouping inflected words to their root form (e.g. 'dogs' to 'dog') so that the inflected words are considered as the same.

Only nouns were lemmatised since no POS tagging were supplied for the tokens, since doing so would have tremendously increased the runtime.

Finally we get a vocabulary with 108,543 unique terms.

1.2 Comparison with Zipf's law

The Zipf's law states that rank of a word times it's frequency is a constant. When observing the frequency of different words and their ranks, we find that the most frequent word occurs approximately twice as often when compared to the second most frequent word. When the rank and frequency of each word are multiplied, it was found that we get around the same value for most cases. We can also verify visually with the help of Figure 1 that the distribution of the words follows Zipf's law.

When we take a closer look at the two distributions by switching to log-scale in Figure 2, we find that the distribution of observed frequencies of the words displays a downward concavity. We see significant deviation from Zipf's law for less frequent words as the relationship between rank and frequency is a constant. This is because such words tend to occur once or twice in the corpus and hence we see discrete steps for rare words. As per the Zipfian distribution, the frequencies of the word and the rank should follow a linear relationship in the log-scale which we see is not the case. However, the Zipf's law does give us an approximate for the frequency distribution of the words in the corpus.

2 Task2 - Inverted index

Firstly, stop words were removed from the vocabulary since they do not add much information to a document and are present in abundance. Though this step may result in removal of some contextual meaning from the text. As we are considering bag of words models, such models do not take into account context of the words hence we can go ahead with this process. Stop word removal allows the retrieval model to focus on the important bits and generalise better. It also helps in reducing the number of tokens hence saving in memory space.

The inverted index is generated and stored in a dictionary data structure. A nested dictionary is used, in which the outer dictionary stores the vocabulary terms and the inner dictionary contains the passage id (pid) and the count of the key term in that passage.

This approach was taken so that fetching document frequency and the term frequency for a vocabulary term would take the least amount of time, since on average it takes constant time to lookup elements from a dictionary. These values would be used in calculating the similarity scores in the later tasks. The term frequency can be fetched directly from the inverted index dictionary, given a pair of pid and query id (qid). For document frequency, we count all the passage ids that are stored for a term in the inverted index.

3 Task3 - Retrieval models

TF-IDF For this task, tf-idf vector representations were created for both query and passages. For term frequency weights, the formula used was: $1 + log_{10}(t_{f,d})$. The $t_{f,d}$ weight is scaled with a log term rather than using the raw term frequency. This was done as we want the similarity score to go up with the count of a term, but not linearly. Adding log will dampen the importance of terms which have a very high count. If doc1 and doc2 have 20 and 30 occurrences of a term respectively, we do not have a clear cut idea which document should be considered since there isn't that big a difference in relevancy compared to a document with only 1 occurrence.

As the number of vocabulary terms is exceedingly large, to avoid memory issues, sparse matrices were used. We first iteratively find out the tf-idf vectors for both query and passage pairs. These vectors were stored in scipy's lil_mtarix, which is a row based list of list sparse matrix. It is more efficient to index and incrementally build this compared to other types of sparse matrices. The weight vectors are generated and stored in the matrices, one for the passages and one for the queries. Once the matrices were formed, the sparse matrix type is change to compressed sparse column, since arithmetic operations are more efficient to carry out in this format. A vectorized approach was used for finding cosine similarity scores between a query and passage pair wherein, pair wise dot product was taken of the tf-idf vectors stored in the rows of the two sparse matrices.

BM25 In this task, BM25 retrieval model was also implemented. Here we looped through query and passage pairs and calculated the scores for terms present in the query. Since no relevance information was provided, it was not taken into account when calculating the similarity scores.

4 Task4 - Query likelihood language models

In the final task, three unigram query likelihood models were implemented using different smoothing techniques namely, Laplace, Lidstone and Dirichlet. Smoothing techniques are used so that the likelihood model doesn't give zero probability when a query contains an unknown word, i.e word not present in the document. Since a document is only a sample from the corpus, it is not expected that it will contain all vocabulary terms. Therefore, unknown terms should not have zero probability of occurring. Smoothing techniques handle such cases by lowering the probability estimates for words that are present in the document and the 'left-over' probabilities are assigned to unseen words.

In Laplace smoothing we add 1 to every count of word occurrence to give uniform weightage to every word in the vocabulary. The probability is re-normalized by adding the total number of terms in the vocabulary to the denominator.

Though it solves the problem of zero probability for queries with unseen words, it gives too much weightage to such unseen words. We are adding 1 for each word even though we have not seen them in the document, which is not desirable.

In Lidstone smoothing rather than adding 1, we add a small value ϵ: Commonly, $\epsilon = 0.5$ is used, which makes no assumptions between number of relevance and non relevance documents. In this exercise, we know on average for each query we have 1000 candidate passages out of which we find 100 relevant passages. Hence using this prior $\epsilon = 0.1$ is a good choice. Lidstone and Laplace are expected to give similar results due to the uniform prior assumption made by the models. We have taken the negative log likelihood of the scores in both the cases.

In the above smoothing methods, the unseen words are all treated equally (either 1 or ϵ is added) but in reality some words are more likely than others. This is incorporated in Dirichlet smoothing by taking in consideration background probabilities of words in the corpus to which the documents belongs to, when setting the prior. Since the prior is more informed, Dirichlet is expected to perform better when compared to other smoothing techniques which have been discussed. We have looped through each query passage pairs and have found the negative log likelihood scores.

Generally μ in the prior is set approximately to the average number of words in the documents. In the passage collection, we find that the average number of words per passage is 32 hence $\mu = 50$ is appropriate. Setting $\mu = 5000$ will result in stronger priors for unseen words: $\mu/(N + \mu)$, compared to seen words $N/(N + \mu)$, where N is the number of words in a given document. Since $\mu = 5000$ will be on average much greater than N.