Information Retrieval and Data Mining (COMP0084)

22092172

Abstract

In this coursework, information retrieval models are built to solve the problem of passage retrieval i.e. given a query the model provides a list of documents starting with the most relevant. Task 1 starts with preprocessing the texts from passages and comparing the distribution of its terms with Zipf's law. In Task 2, an inverted index of the terms of the passages is created. Task 3 calculates the relevance score of each pair of queries and passages by analysing the TF-IDF values of the passages and the queries, then applying cosine similarity and BM25 methods. Three query likelihood language models are designed in Task 4 using Laplace smoothing, Lidstone correction, and Dirichlet smoothing and their performance is reported and compared.

1 Task 1

In task 1, `passage-collection.txt` file is imported and $182,469$ passages are extracted by reading the text file line by line. Before extracting 1-grams from the passages, some text preprocessing steps are applied to it. These preprocessing steps are as follows:

1. Change the text to lowercase: All the text in the passages is converted into lowercase to avoid confusion later on using the `regex` module in Python.

2. Remove punctuations: From the lowercase text, the punctuations characters like '!#$%&'()*+,-./:;<=>?{}' are removed using `string.translate` from `string` module in Python.

3. Remove numbers: The numbers are dropped using `regex` to get words only in each passage.

4. Tokenize words: The terms obtained after the preprocessing are tokenized using the `word_tokenize` function of nltk library. This returns a list of word tokens in the passages.

5. Remove stopwords (if selected): The stopwords, a set of commonly used words in English, are also removed from these word tokens using `nltk.stopwords` depending on if selected in the function argument.

6. Lemmatization and Stemming: Lemmatization is the process of converting a word to its base form and stemming removes the suffixes or prefixes from the word to obtain its stem/root. These processes are applied to the words using `nltk.stem` module to get their base or root form.

D1 Do not remove stop words in the first instance. Describe and justify any other text preprocessing choice(s), if any, and report the size of the identified index of terms (vocabulary). Then, implement a function that counts the number of occurrences of terms in the provided data set, plot their probability of occurrence (normalised frequency) against their frequency ranking, and qualitatively justify that these terms follow Zipf's law. How does your empirical distribution compare to the actual Zipf's law distribution? How will the difference between the two distributions be affected if we also remove stop words?

Ans. The vocabulary of words is created from the preprocessed text of $182,469$ passages which gives a dictionary of every term with its number of occurrences in all the passages texts. The vocabulary obtained without removing the stopwords is $182,003$ words.

Using this vocabulary dictionary, the probability of occurrence and frequency rank of each term is evaluated and plotted. For text sets $s = 1$, in the

Zipf's law distribution is given by,

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

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), N is the number of terms in our vocabulary and s is a distribution parameter.

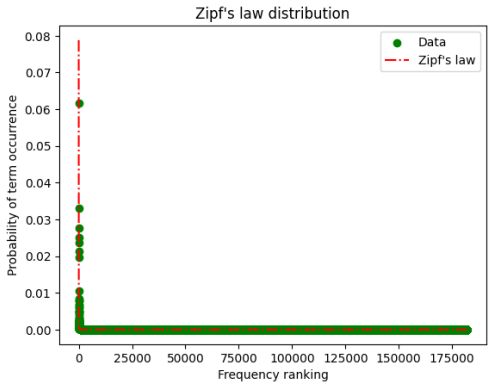

Figure 1: Comparison with Zipf's law

The above Figure 1 shows the normalized frequency of terms against their frequency ranking while giving a comparison with Zipf's law values.

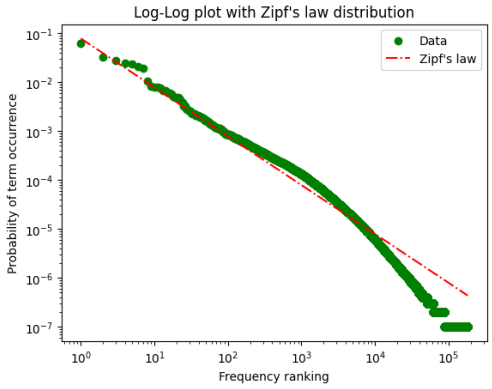

Figure 2: Log-log plot of distribution

The log-log plot in Figure 2 gives a better understanding of their relationship. It can be observed that the data follows zipf's law quite well. However, the data for lower-ranked words $> 10^4$ is found to deviate from the zipf's law significantly. This can be accounted for by some shortcomings of Zipf's law by assuming the most frequent term occurs twice as many times as the second most frequent term which is not always true for real data. Another reason might be the presence of words with misspellings in the data which is causing the deviation.

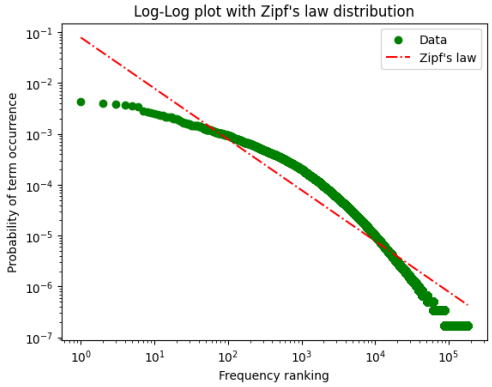

Figure 3: Log-log plot removing stopwords

Figure 3 shows the log-log plot of the probability of occurrence and frequency rank of each term along with theoretical values from zipf's law after removing the stopwords from the data. The data deviates more from zipf's law, especially for the high-ranked words. This can be due to the fact that the total number of words is reduced which included stopwords generally having high frequencies.

2 Task 2

The `candidate-passages-top.100.tsv` file is imported and unique pairs of `pid, qid, passage` are extracted to create an inverted index. Stopwords are removed for this task and there is a total of 182469 unique pairs of `pid` and `passage`.

D3 Provide a description of your approach to generating an inverted index, report the information that you decided to store in it, and justify this choice. Hint: Implementing Tasks 3 and 4 should inform this choice.

Ans. The inverted index is the dictionary where the terms can be mapped to the passages in which they occur along with their frequency in that passage. Since the inverted index will be used in the tasks ahead to build tf-idf and query language models, storing the pids and the occurrences of the terms in the inverted index seems like a reasonable approach for this task. The process used to build the inverted index is as follows:

1. Create an empty dictionary to store the values of the inverted index
2. For a given pid and its corresponding passage, get every term in the passage and its frequency
3. Insert the pid and frequency of each term in the passage in the inverted index making the

term as its key i.e. {'term': { pid: frequency}}

4. Loop over all the pids and if the term occurs in them, insert the pid with the frequency of the term in that pid against the inverted index key i.e. the term.

In the inverted index, the keys are the terms in the passages corpus and their values are a dictionary of pid and its corresponding frequency in that passage. For example, `inverted_indx['rna']` returns a dictionary of pids and the frequency of the term `'rna'` for passages in which this term occurs.

3 Task 3

3.1 TF-IDF

The inverted index obtained in Task 2 is used to find TF-IDF vector representations for the passages. The IDF obtained from the passages is then used on the query data from `test-queries.tsv` to get the TF-IDF of queries. Using these TF-IDF of passages and queries and cosine similarity function, top-100 passages are obtained for every query in the `test-queries.tsv`. The `qid, pid, score` is stored in `tfidf.csv` where `qid` is the query's identifier, `pid` is the passage identifier, and `score` is the cosine similarity score.

3.2 BM25

BM25 i.e. Best Model 25 is a probabilistic model that estimates the relevance of a document for a query. BM25 is used to build a query likelihood language model for getting the top 100 most relevant passages given a query. The scores generated for each pair of queries and document are saved in the `bm25.csv` file. There are $19,290$ entries in this file.

4 Task 4

Query likelihood language models are created using three techniques:

1. Laplace smoothing: This simple technique adds a constant, 1, to each term's frequency. It assumes that all the terms have been seen at least once in the collection.

$$P_{Laplace}(w|D) = \frac{c(w)+1}{|D|+|V|}$$

where, $c(w)$ is frequency of term w, $|D|$ is the length of the document i.e. passage, $|V|$ is the size of vocabulary.

2. Lidstone correction: This is a modification of laplace smoothing where the constant term ϵ can be < 1, usually $\epsilon = 0.1$.

$$P_{Lidstone}(w|D) = \frac{c(w)+\epsilon}{|D|+\epsilon|V|}$$

3. Dirichlet smoothing: Rather than adding a constant value to the terms like in Laplace and Lidstone, Dirichlet estimates the probability of unseen terms based on the whole language model of the collection. This distribution is conjugate to multinomial distribution and works better at handling unseen terms in the data.

$$P_{Dirichlet}(w|D) = \frac{c(w)+\mu P(w|C)}{|D|+\mu}$$

where, $P(w|C)$ is the probability of the term in the whole corpus and μ is the smoothing parameter.

D11 Which language model do you expect to work better? Give a few empirical examples based on your data and experiments. Which language models are expected to be more similar and why? Give a few empirical examples based on your data and experiments. Comment on the value of $\epsilon = 0.1$ in the Lidstone correction. Is this a good choice and why? If we set $\mu = 5000$ in Dirichlet smoothing, would this be a more appropriate value and why?

Ans. In the Laplace and Lidstone correction, it is assumed that each word occurs equally frequently and they are not good at dealing with zero probabilities. These problems are handled in a more efficient way in Dirichlet smoothing where the probability of a term is evaluated based on the document length and the overall frequency of the term in the collection. Thus, Dirichlet smoothing is expected to work better. An example of its performance is how it was able to find pid 7651010 in the top 5 passages for a query with a qid of 1108939 which was observed by the BM25 approach as well.

Furthermore, Laplace smoothing is a commonly used technique for smoothing models where the length of documents is not too big. In this data, the maximum length of a passage is 153 and hence it works well for the language model. For example, it found the passage with the pid 5291621 in the top five relevant passages for the query with qid 1108939, which was also found using the cosine similarity and BM25 approaches.

As said before, Laplace smoothing and Lidstone correction are quite similar in their approaches. This is because both these methods add a constant term to the count of each term in the collection. These two language models can be said to be special cases of Dirichlet smoothing where they take specific parameter values. However, their approach is simpler compared to the complex structure of Dirichlet smoothing and thus are more similar to each other.

The value of $\epsilon = 0.1$ is appropriate for the data used to build the language model as the length of documents (passages) is not too big. If the value of ϵ is increased more weight will be given to unseen words. To ensure that unseen words are not shared with excessive weights compared to existing words in documents, the chosen value of epsilon is low and suitable.

If the value of μ is set to 5000 then it would not be a good choice. In most cases, the Dirichlet parameter μ is set based on the average document length in the data. Here, the average passage length is close to 32 so the choice of $\mu = 50$ works well. Setting it to 5000 may result in over-smoothing and, hence, underfitting of the model.