Information Retrieval and Data Mining: Coursework 1

UCL Computer Science

1 Task 1 - Text statistics

1.1 Deliverable 1 & 2

Preprocessing During preprocessing stage of my data, I read the data by rows. Initially, I turn all upper case letter to lower case letter. Strings in each passage, i.e. document, will be removed except for lower case letter, space and number. Note that there are special symbols, which will not eventually contribute for retrieval process. As a result, I remove them. Then I tokenize each passage, i.e. document by space. Afterwards, I use a dictionary to record the occurrences for each token. The size of dictionary, i.e. vocabulary, is 207544.

Probability of occurrences against frequency ranking After complete the recording process of dictionary, i.e. vocabulary, I plot the tokens' probability of occurrences against their frequency ranking with comparison to Zipf's law, which is shown as in 1 and in 2:
where zipf's law is shown as equation 1:

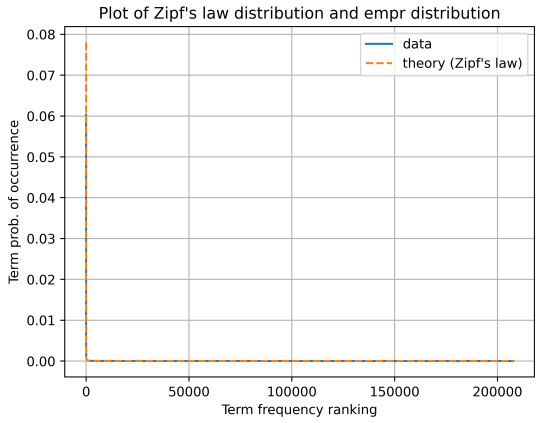

Figure 1: Distribution comparison

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

In our experiment, we choose $s = 1$, $f()$ 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.

It can be seen from 1 and 2 that, our vocabulary has less top rank frequency words compared to the prediction given by Zips'f law, which can be seen from the region $10^0 \sim 10^{0.5}$ in figure 2. After that, we can observe that we have more words in our vocabulary with less frequency ranking from the region $10^{0.5} \sim 10^4$. Finally, we have less words in our vocabulary with low frequency ranking from the region $10^4 \sim 10^5$. Then difference can also be obeserved by the equation 1. Note that the denominator is fixed whenever s and N is fixed. Then normalised frequency of a term given by Zipf's law is strictly proportional to k^{-s}, which will not be strictly obeyed by the true data. However, the trends of the two lines are almost the same.

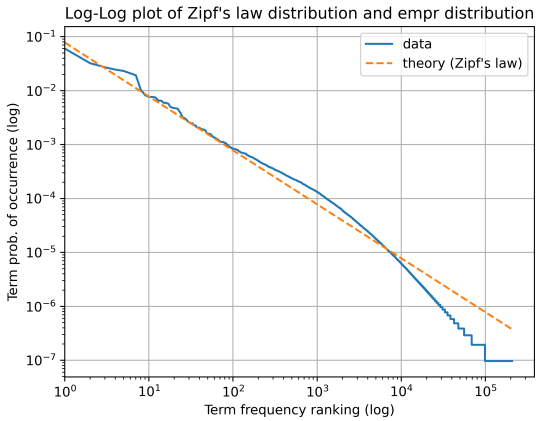

Figure 2: Distribution comparison(log)

2 Task2 - Inverted index

2.1 Deliverable 3 & 4

Remove stop words In my experiment, I remove the top 20 frequent word from my vocabulary in order not to be biased by these tokens. Note that 20 is selected by experience and this may not be optimal in this experiment.

Generating inverted index Note that I choose to store the occurrences of tokens in inverted index. Since afterwards we would use the information of occurrences of tokens by applying tf-idf and BM25 algorithms as well as query language models. It is effective to store the occurrences ahead.

Before I construct inverted index in a dictionary form, I construct an inverted index matrix $\mathbf{I}$. If we set the number of passage in our whole unique document set to be N, the size of vocabulary is set to be m, then the size of inverted index matrix $\mathbf{I}$ is `(m,N)`, where $\mathbf{I}_{i,j} \in \mathbb{N}$ represents the i-th document has number of token j equal to $\mathbf{I}_{i,j}$. Note that passages and vocabulary are well-ordered. Passages are ordered by their `Pid`, while vocabulary is ordered by the frequency of each token. Note that inverted index matrix is in huge demand of memory because of its huge size. However, inverted index matrix is very sparse and easy to parallelized. Therefore, I store it in a form of sparse matrix.

Once inverted index matrix is constructed, we have all the statistical information of our documents. We can use inverted index matrix to construct inverted index dictionary.

3 Task3 - Retrieval models

3.1 Deliverable 5

TF-IDF TD-IDF (Ramos) is a general technique that used for information retrieval. It leverages the information of general importance of a specific word/token and the local frequency of the token together. In our experiment, I implement TF-IDF using a basic vector space model, i.e. treat every query and document to be an TF-IDF vector. I obtain the score the query-document pair by calculating the cosine-similarity between them.

Obtain TF-IDF representation Before we calculate the cosine similarity between query and document pairs, we should obtain the TF-IDF representations for queries and documents, respectively. Note that previously we store all statistical information of document set inside inverted index matrix. Then for i-th document, I take out the i-th row of $\mathbf{I}$ as I_i, then the tf of document d_i is:

$$tf_{d_i} = \frac{\mathbf{I}_i}{\sum_j \mathbf{I}_{ij}} \tag{2}$$

Here we assume the division is broadcasted. Similarly, we can obtain tf_{q_i} by the same process. tf for query and document is normalized.

Note that in this experiment, IDF is shared by queries as well as documents. IDF for token t is computed as:

$$idf_t = log_{10}(\frac{N}{n_t}) \tag{3}$$

where N is number of documents in document set, n_t is number of documents in which token t appears. In our case, this process can be parallelized as:

$$idf_{vocab} = log_{10}(\frac{N}{\sum_i \mathbf{I}_{ij}}) \tag{4}$$

where $idf_{vocab} \in \mathbb{R}^m$. Same as before, we assume that division here is broadcasted.

Then for each query/document, we can compute the tf-idf expression of query/document by simply multiplying tf and idf together, which is shown as:

$$tfidf_d = tf_d * idf \tag{5}$$

$$tfidf_q = tf_q * idf \tag{6}$$

Similarity Score Similarity score is obtained by computing the cosine similarity between tf-idf representations, which is shown as:

$$sim(q,d) = \frac{\langle tfidf_q, tfidf_d \rangle}{\|tfidf_d\| \times \|tfidf_q\|} \tag{7}$$

where d represents document, q represents query, t represents token, $tfidf$ represents the tf-idf representation for query/document. In our case, we can simply apply Hadamard product bewteen $tfidf_q$ and $tfidf_d$ to obtain the same results.

After I have all the score for all document-query pairs, I select at most 100 documents for each query and write them in `.csv` file.

3.2 Deliverable 6 & 7

Different from TF-IDF, BM25 (Robertson and Zaragoza, 2009) is the probabilistic model that used for information retrieval. However, the change on implementation between TD-IDF and BM25 is trivial. The score between query and document is expressed as:

$$BM25(q,d) = \sum_{i \in q} log \frac{N - n_i + 0.5}{n_i + 0.5} \times \frac{(k_1+1)f_i}{k1((1-b) + b \times \frac{d_l}{av_{d_l}} + f_i} \times \frac{(k_2+1)qf_i}{k_2 + qf_i} \quad (8)$$

where k_1, k_2 are parameters whose values are set empirically, d_l is document length, av_{d_l} is average document length, f_i and qf_i are the number of occurrences of the term in the document and the query, respectively.
In my experiment, I compute $log \frac{N-n_i+0.5}{n_i+0.5}$ ahead as "idf", then the right part in equation 8 can be viewed as "tf". Everything else is exactly the same as tf-idf.

4 Task 4 - Query likelihood language models

This task use language model to evaluate the relevance between queries and documents. It leverages the statistical information from document set to evaluate the probability of the occurrence of query tokens.

4.1 Deliverable 8: Laplace smoothing

Laplace smoothing (Chen and Goodman, 1999) share weights through all vocabulary, i.e., it shares the weights of showed up tokens to those who don't show up but exist in vocabulary. The probability of every token in vocabulary for each document is shown as:

$$(\frac{m_1+1}{|D|+|V|}, \frac{m_2+1}{|D|+|V|}, ..., \frac{m_{|V|}+1}{|D|+|V|}) \quad (9)$$

where $|D|$ represents the document length, $|V|$ represents the whole vocabulary size, m_i represents the occurrence of the i-th token in the document. Then, the probability of a query given a document is given by:

$$p(q|d) = \prod_{t \in q \cap d} p_t \quad (10)$$

In my implementation, I initially use inverted index matrix to derive the probability vector of each document by taking one row of inverted index matrix out and implement equation 9. Then for every query, I compute the logarithm of $p(q|d)$ for each document in order to obtain a valid result.

4.2 Deliverable 9: Lidstone correction

Lidstone correction applies almost the same technique as Laplace smoothing, while Laplace smoothing gives too much weight to unseen tokens, Lidstone correction gives the weight to unseen tokens that can be controlled by ϵ, which is shown as:

$$p(q|d) = \frac{tf_{q,d} + \epsilon}{|D| + \epsilon|V|} \quad (11)$$

Note that in this experiment, we set $\epsilon = 0.1$. Lidstone correction enables us to find a local/global optimum solution based on discounting methods by tuning ϵ.

4.3 Deliverable 10: Dirichlet smoothing

While discounting methods such as Laplace smoothing and Lidstone correction is easy to implement and explainable, discounting treats unseen words equally. However, in common sense, some words are more frequent than others in whole vocabulary. Therefore, we can leverage the information of statistical information in whole vocabulary (interpolation method), i.e., we combine the information from local document and background probability to obtain the probability of words in document, which is shown as:

$$p_t = \lambda p(t|d) + (1-\lambda)p(t|C) \quad (12)$$

where $p(t|d)$ is the probability of a specific token measured by the maximum likelihood, i.e. counting, $p(t|C)$ is the background probability of a specific token measured on whole vocabulary, λ is a hyperparameter. In Dirichlet smoothing, λ is set to be $\frac{N}{N+\mu}$, where μ is set to be 50 in this experiment.

The implementation of Dirichlet smoothing is very similar to the previous discounting algorithm. Note that I compute the background probability ahead to facilitate the computation the scores.

4.4 Deliverable 11 & 12

What language model works better? The mechanism difference between discounting model and

interpolation method shows that, we have more information to reasonably share weights using interpolation method instead of equally shared using discounting model. The performance of Dirichlet smoothing should be better in this case.

Which ones are expected to be more similar? Laplace smoothing and Lidstone correction should be more similar. They use the same mechanism to compute the probability of a specific term, while they are different on sharing weight to unseen terms. They are lack of the statistical information from whole vocabulary set.

Comment on hyperparameter in Lidstone correction In this experiment, ϵ is set to be 0.1, which is a relative good setting. Given that in our data, document lengths are not huge, i.e. $\max |D| = 233$, which means that we should assign a relative small value to ϵ to make sure unseen words will not be shared with too much weights compared to existing words in document. In other words, we prefer to focus on existing words in document.

Comment on hyperparameter in Dirichlet smoothing If we set $\mu = 5000$, what would happen? Note that λ defines the weights between local information and global information. In other words, when λ is bigger, we tend to believe more on using local existence of words in a specific document to represents the probability of the words. Given that $\lambda = \frac{N}{N+\mu}$ and my longest document has length $N = 233$, when μ increases to 5000, λ is decreasing nearly to 0, i.e. we tend to believe almost all on using background information of words, while we nearly don't use the information of specific document. Thus, $\mu = 5000$ seems not to be a good choice in this scenario.

References

Stanley F. Chen and Joshua Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech Language*, 13(4):359–394.

Juan Ramos. Using tf-idf to determine word relevance in document queries.

Stephen Robertson and Hugo Zaragoza. 2009. The probabilistic relevance framework: Bm25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389.