COMP0084 Coursework 1

Anonymous ACL submission

1 D1

For this deliverable we are going to do text preprocessing and produce a plot which will show that the probability of occurence of the words in the text passage-collection follow Zipf's law.

The first step we did in the text pre-processing is removing all the symbols which are not English letters, because in further tasks where we were asked to match queries with passages, we identified that the queries do not have any non-english words and symbols and only a few have some punctuation symbols. Secondly, to break the raw text into tokens we first transformed all of the words so that they start with lower letters and then tokenized the words.

In the following plot you will see the log-log relationship between the frequency ranking (x-axis, in descending order) and the probability of occurence of the terms (y-axis). The blue line represents the data in the passage collection and the orange line represents the Zipf's law distribution which is represented by the following function:

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

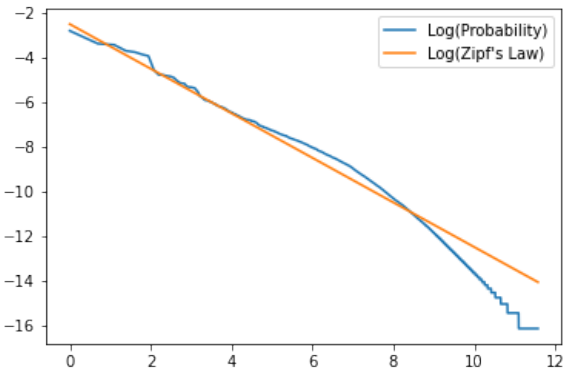

Figure 1: Log-log plot of the data frequency and rank

The vocabulary consists of 105851 terms i.e., $|V| = 105851$. From figure 1 can see that our empirical distribution follows closely Zipf's distribution except for lower frequency and higher frequency words where it makes the most errors. We have that Zipf's law suggests that rank*frequency $= C$. Hence, to empirically check whether our data follows Zipf's law closely (on average), then the proportion of terms with a frequency $a \leq x \leq b$ is given by $(k_a - k_b + 1)/N$ where N is the size of the vocabulary and $k_f = C/f$. We have that $C \approx 0.02131$, which we computed from the data by taking the mean of the rank*frequency of all terms. Let $a = 10^{-6}$ and $b = 10^{-4}$. So we have that $k_a = C/a = 0.02131/10^{-6} \approx 21307.46597$ and $k_b = C/b = 0.02131/10^{-4} = 213.07466$. Hence Zipf's law indicates that $(k_a - k_b + 1)/105851 = 19.93\%$ whereas empirically we have 21.05%, which is very close. Hence with high confidence we can say that the Zipf's law closely approximates our data.

2 D3

For this deliverable we are creating an inverted index for the candidate-passages-top1000.tsv dataset, which consists of the same passages as the passage-collection text however this time they are attached to queries. Before creating the inverted index we do the same text pre-processing as in D1 and we remove stop words as well.

In order to complete tasks 3 and 4, we needed an inverted index which would show how many passage a certain word appears in and how many times it appears in a certain passage (term frequency). The reason we needed the former values for these tasks is because in order to compute the TF-IDF representation of each passage and query, which is given by the following formula:

$$w_{i,j} = tf_{i,j} \times \log \frac{N}{df_i}$$

where $w_{i,j}$ is the TF-IDF of the word i in document j, $tf_{i,j}$ is the number of occurrences of the word i in j, N is the total number of documents and df_i is the number of documents containing i. Our inverted index looks like the following: Inverted Index $= \{\text{word}_1 : \{\text{passage}_1 : 2, \text{passage}_2 : 3, ..., \text{passage}_N : 5\}, \text{word}_2 : \text{passage}_1 : 4, ...\}, ..., \text{word}_{|V|} : \{...\}\}$, where $|V|$ is the number of unique words in the whole collection. In Python the inverted index is stored as a dictionary of dictionaries - each word has a dictionary attached to it. When calculating the $tf_{i,j} \iff tf_{\text{word}_i,\text{passage}_j}$ part of the TF-IDF formula, we can simply call `Inverted Index[word][passage]` and get the required term frequency. The same for $df_{i,j}$ - we can simply find the length of the dictionary of the specific word and since that dictionary doesn't contain passages which the word doesn't appear in we get the exact number of passages the word appears in.

The way we generated the inverted index is by looping through all of the passages and then looping through all of the words in the passages themselves. For each word i in the passage j we append 1 to the $(\text{word}_i : \text{passasge}_j)$ pair.

3 D5

For this deliverable we extract the TF-IDF vector representations of the passages and queries and calculate the cosine similarity score for each query-passage pair present in the candidate-passages-top1000.tsv dataset.

In order to calculate the cosine similarity we need to calculate a dot-product of two vectors of the TF-IDF representation of a query and passage. In order to do this, we looped through all of the unique passages and queries, and created a dictionary which had the TD-IDF score for each word in the passage or query. For example, if we had a passage or query "Yesterday Mark swam", the dictionary would look like $\{\text{yesterday} : 0.32, \text{mark} : 0.65, \text{swam} : 0.12\}$. We had to first create dictionaries of the TF-IDF vector representations for both the passage and queries. Note that some of the words in the queries are not present in the passages i.e., $df_{i,j} = 0$. For these cases we decided to assign IDF values of $\log(N)$, since some alternative formulas use $IDF(word) = \log fracN + 1df + 1$ in order to mitigate the division by 0. After attaching a TF-IDF vector to all unique query and passages, we looped through all of the query-passage pairs and calculated the dot products of the words which they have in common. For example let the passage be "Today was great" and the query be "Today was fantastic". The words in common would be "today" and "was".

After calculating the cosine similarity score, we looped through each query and ranked the best 100 query-passage pairs in descending order which was determined by the score.

4 D6

For this deliverable we calculate the BM25 score for each query-passage pair and paired each query with its best 100 candidate passages in descending order.

The formula for the BM25 score is $\sum_{i \in Q} IDFBM25_i \frac{k_1+1)f_i}{K+f_i} \frac{k_2+1)qf_i}{k_2+qf_i}$

Where $IDFBM25_i = \frac{N-df_i+0.5}{df_i+0.5}$ is a modified IDF value of the word i, $k_1 = 1.2$, $k_2 = 100$, f_i and qf_i are the frequencies of word i in the documents and the query respectively and $K = k_1((1-b) + b\frac{dl}{avgdl})$ where dl = passage length, $avgdl$ = average passage length.

In order to calculate the sum, we again use dictionaries and the inverted index to get the IDF and term frequency (inside the passage or query) values of the common terms between the query and passage, and then sum all of the values.

Finally, for each query we extract the best 100 candidate passages and rank them in descending order.

5 D8-10

For this deliverable we implement a query-likelihood language model with Laplace Smoothing, Lidstone correction and Dirchlet Smoothing.

Query-likelihood language models aim to rank documents by the probability that the query could be generated by the document model.

5.1 Laplace Smoothing and Lidstone correction

Laplace smoothing is used in order to remove instances where the probability of a word occuring is equal to 0. Hence, the probability of a word with laplace smoothing is $p(w|D) = \frac{tf_{i,D}}{|D|+|V|}$ where $|D|$ is the total number of words in the document D and $|V|$ is the number of unique words in the collection

(vocabulary size). Lidstone correction is similar to laplace smoothing however it mitigate the problem where laplace smoothing gives too much weight to unseen terms. The probability of a word occuring with Lidstone correction is $p(w|D) = \frac{tf_{i,D}+\epsilon}{|D|+|V|}$. We use $\epsilon = 0.1$.

For both Laplace smoothing and Lidstone correction we calculate the scores by using the inverted index to get the term frequencies (inside the passages) of the words for all the common words between the query and the passage and then sum all of these values ($(p(w|D))$) to get the score.

5.2 Dirichlet smoothing

With Dirichlet smoothing, the probability of a word is $\frac{N}{N+\mu}p(w|D) + \frac{\mu}{N+\mu}p(w|C)$ where N is the document length, in our case μ is equal to 50 and C is the collection of the passages.

To calculate the scores we use the dictionary we created in task 1 which contained the total number of occurrences of a word in the whole collection in order to calculate $p(w|C)$. We also use the inverted index to calculate $p(w|D)$. By summing all of the probabilities for the common words between the query and the passage we get the desired score.

6 D11

We think that the best language model would be with the Dirichlet smoothing since Dirichlet smoothing doesn't treat unseen words equally and hence it would give better results.

The Laplace smoothing and Lidstone correction models are expected to be more similar since they use very similar strategies for smoothing except that Lidstone gives less importance to unseen words.

We think that in order for Lidstone correction to be different than the Laplace smoothing (and hence give less importance to unseen words), ϵ should be small, since the larger it is, the more closer the scores are to the Laplace scores.

We do not think that $\mu = 5000$ would be an appropriate choice since μ is supposed to represent the estimate of the average document length, and in our case the average document length is 32.