IRDM Coursework 1

16040463

Abstract

In this report, we will be retrieving passages for queries using different types of retrieval models and language models.

1 Notation

For explicitness, the following notations will be used throughout this report.
N = number of passages in collection
$|V|$ = vocabulary length (number of unique tokens in collection)
$|D|$ = passage length n_i = number of passages $token_i$ appears in
$tf_i = \frac{f_i}{number\ of\ tokens\ in\ collection}$
f_i = occurrence of $token_i$ in collection
$fq_{i,t}$ = occurrence of $token_i$ in $query_t$
$tfq_{i,j} = \frac{fq_{i,j}}{number\ of\ tokens\ in\ query}$
$fp_{i,j}$ = occurrence of $token_i$ in $passage_j$
$tfp_{i,j} = \frac{fp_{i,j}}{number\ of\ tokens\ in\ passage}$

2 Task 1 - Text statistics

2.1 D1

Tokenisation and Normalisation are executed for the data 'passage-collection.txt'. The three necessary steps for text processing are Paring, Tokenisation and Normalisation. Here only Tokenisation and Normalisation are implemented since the data is already raw text. In tokenization step, the contents loaded from 'passage-collection.txt' are split to unigram tokens. In normalisation step, punctuations and numbers are removed which means only alphabets are kept and all alphabets are converted to lower case.
After preprocessing, vocabulary was built by taking all unique elements from preprossessed contents. Identified index of terms (vocabulary) is 117125.
The following plot shows the probability of occurrence against frequency ranking.

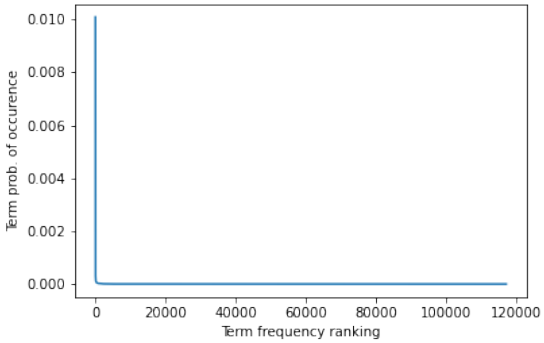

Figure 1: Probability of occurrence against frequency ranking

Zipf's law suggests that the probability of occurrence of a term is inversely related to the item's frequency ranking which we can see from Figure 1 that probability of occurrence decreases as frequency ranking increases.
In Figure 2, the empirical distribution has a similar

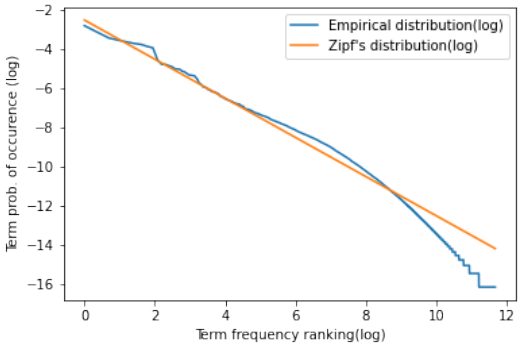

Figure 2: Compare Zipf's distribution and empirical distribution in log-log plot

trend as the Zipf's law but they are not identical. The reason is that Zipf's law suggests that the product of the probability of occurrence of a term and its frequency ranking is a constant. This also suggests that for two terms with adjacent frequency

ranking, the difference of their occurrence is the same. For empirical data, the occurrence fluctuates. Therefore, the empirical distribution is not a straight line that overlapping Zipf's law but they do have similar trend and values.

3 Task 2 - Inverted index

3.1 D3

According to Task 3, we need to know how many passages each token in the vocabulary from Task 1 occurs in and their occurrence in the collection to do Task 3 and 4. Therefore, the inverted index here would be a dictionary where the keys are tokens in vocabulary and the values are a list containing the number of passages the token appears in and their occurrence in the whole collection of tokens. Since the data in 'passage-collection.txt' is the same as passages in 'candidate-passages-top100.tsv', we only need to load 'candidate-passages-top100.tsv' to generate the inverted index. The preprocess function in Task 1 is reused here while lemmatization and stopwords removing are added to preprocess.

Build Inverted Index
1: Extract the unique passages from 'candidate-passages-top100.tsv'
2: inverted index = {'$token_i$':[n_i,f_i]}
3: **for** each unique $passage_j$:
4: Preprocess passage and store tokens in a list
5: **for** each unique $token_i$:
6: $n_i += 1$, $f_i += fp_{i,j}$

Table 1: Description of approach to generating inverted index

4 Task 3 - Retrieval models

4.1 D5

A basic vector space model with TF-IDF requires a representation of TF-IDF vector for each query and each passage, then the score is the cosine of the angle between the query and passage.
The entry of TF-IDF vector for each passage is,

$$TF - IDF^p_{i,j} = tfp_{i,j} \times log_{10}(\frac{N}{n_i})$$

. The entry of TF-IDF vector for each query is,

$$TF - IDF^q_{i,t} = tfq_{i,t} \times log_{10}(\frac{N}{n_i})$$

. Let $\mathbf{x}$ denotes TF-IDF vector of queries and $\mathbf{y}$ denotes TF-IDF vector of passages. The cosine similarity formula is,

$$\frac{\sum_{i=1}^{V} x_i \times y_i}{\sqrt{\sum_{i=1}^{V} x_i^2} \times \sqrt{\sum_{i=1}^{V} y_i^2}}$$

We can observe that, we only need to consider the intersection of tokens in query and passage. For the denominator, we only need to consider the tokens in the query or passage. Therefore, we do not have construct the sparse vector and execute vector multiplication. Instead, the following programme is executed,

Cosine Similarity
1: **Input** 'candidate-passages-top100.tsv' as data
Inverted index
2: **for** each row in data:
3: Preprocess the $query_t$ and $passage_j$
4: Compute TF-IDF for each token in query and passage respectively
5: Compute denominator of cosine similarity
6: **for** each unique $token_i$ in query:
7: nominator += $TF\text{-}IDF^p_{i,j} \times TF\text{-}IDF^q_{i,t}$
8: Compute cosine similarity score

Table 2: Description of approach to computing cosine similarity score

With the algorithm above, we generate an array where the first tow columns are qid and pid. The last column contains the cosine similarity score for this pair of query and passage. Then we rank the score for each query in descending order and retrieve the first 100 passages (less than 100 for some queries). In the end, we report the result as the order given in 'test-queries.tsv'. The result is stored in 'tfidf.csv' file.

4.2 D6

In this deliverable, BM25 retrieval model is implemented with assumption that no relevance information is given. Then the BM25 model follows the formula below,

$$\sum_{i \in Q} log(\frac{N - n_i + 0.5}{n_i + 0.5}) \times \frac{(k_i + 1)fp_{i,j}}{K + fp_{i,j}} \times \frac{(k_2 + 1)fq_{i,t}}{k_2 + fq_{i,t}}$$

With the algorithm above, we generate an array where the first tow columns are qid and pid. The

BM25
1: **Input** 'candidate-passages-top100.tsv' as data Inverted index
2: **for** each row in data:
3: Preprocess the $query_t$ and $passage_j$
4: **for** each unique $token_i$ in query:
5: $score+ = log(\frac{N-n_i+0.5}{n_i+0.5})\times \frac{(k_i+1)fp_{i,j}}{K+fp_{i,j}} \times \frac{(k_2+1)fq_{i,t}}{k_2+fq_{i,t}}$

Table 3: Description of approach to computing cosine similarity score

last column contains the BM25 score for this pair of query and passage. Then we rank the score for each query in descending order and retrieve the first 100 passages (less than 100 for some queries). In the end, we report the result as the order given in 'test-queries.tsv'. The result is stored in 'bm25.csv' file.

5 Task 4 - Query likelihood language models

5.1 D8-10

In this section, query likelihood language model is implemented with three smoothing methods.
Query likelihood language model uses the probability of tokens in query appear in the passage to estimate the probability of the query could be generated by the passage. Since the product of probability which is the likelihood is normally small, natural log is applied to avoid underflow. Therefore, the query likelihood here follows

$$\sum_{i \in Q} ln(P(w_i|D)),$$

where Q represents query, w represents token, D represents passage and C represents collection. In practical, not all tokens in queries are in passages too. When implementing query likelihood language models, the likelihood could easily go to 0. Therefore, smoothing methods are used for missing tokens where they will be assigned with different probabilities of occurrence depending on the smoothing method we use.
(a) Laplace smoothing:

$$P(w_i|D_j) = \begin{cases} \frac{fp_{i,j}+1}{|D|+|V|}, w_i \in D_j \\ \frac{1}{|D|+|V|}, w_i \notin D_j \end{cases}$$

(b) Lidstone correction:

$$P(w_i|D_j) = \begin{cases} \frac{fp_{i,j}+\epsilon}{|D|+\epsilon|V|}, w_i \in D_j \\ \frac{\epsilon}{|D|+\epsilon|V|}, w_i \notin D_j \end{cases}$$

(c) Dirichlet Smoothing:

$$P(w_i|D_j) = \begin{cases} \frac{N}{N+\mu} tfp_{i,j} + \frac{mu}{N+\mu} tf_i, w_i \in D_j \\ \frac{mu}{N+\mu} tf_i, w_i \notin D_j, w_i \in C \\ 0, w_i \notin C \end{cases}$$

5.2 D11

Dirichlet smoothing is expected to work better since it is not assigning random probability that we choose for the missing token but assigns probability according to the missing token's occurrence in the collection.
Laplace and Lindstone are expected to be more similar since the idea of how to assign probability to missing tokens are the same. The only difference is that Lindstone assigns a smaller probability to missing data.
ϵ in the Lidstone correction represents how many times do we believe the missing token appear in the vocabulary. The most used value for ϵ is 0.5. Our choice of seeting $\epsilon = 0.1$ is a a little small.
The larger the μ we chose, the more we believe the collection. Since in this task, we have an initial selection of at most 1000 candidate passages for each query which increased the reliability of our collection. Therefore, we could assign more weights to probability of tokens appears in collection. The most used value for μ is 2000 but our collection is quite small, setting $\mu = 5000$ would be acceptable.