Report of Coursework1 (COMP0084)

Anonymous ACL submission

Abstract

To retrieve passages for provided queries based on information retrieval models, an information retrieval system is developed in this coursework. To acquire data for constructing models, we extracted TF-IDF vector representations of passages and queries. Furthermore, we utilise Cosine Similarity, BM25, Query Likelihood Language Model to evaluate the relevance of queries and passages, finally retrieve top 100 relevant passages for each query with each model.

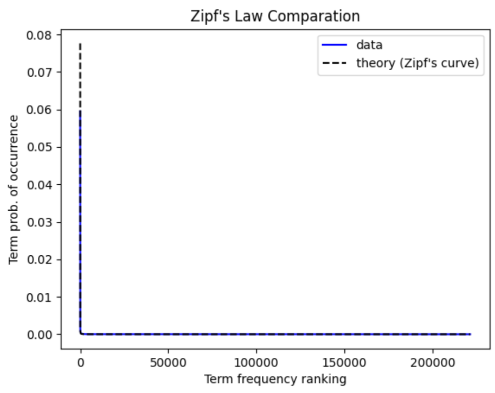
1 Introduction

Techniques of information retrieval are applied widely on searching engines, question answering, and recommendations. The task of this coursework is to construct an information retrieval system which solves the problem of retrieving passages for given queries based on information retrieval models including Cosine Similarity, BM25, and Query Likelihood Language Model (with Laplace smoothing, Lidstone smoothing, Dirichlet smoothing respectively).
The language of development in the coursework is Python 3.9.7. The total runtime of four .py files is around 10 minutes on the First Generation Macbook Pro M1 (8-core CPU, 16GB RAM).

2 Solutions

2.1 Deliverable 1

To acquire text statistics, I have done three steps of preprocessing. First, I lowered all raw texts as vocabulary are all in lowercase in query text. Second, replace punctuations by blank spaces and lever split method to get tokens (1-grams). Third, NLTK is utilised in stemming to normalise words. After preprocessing is taken, the total number of identified words is 107538. Plot their normalised frequency against frequency ranking, I get the Zipf's curve of the text collection.

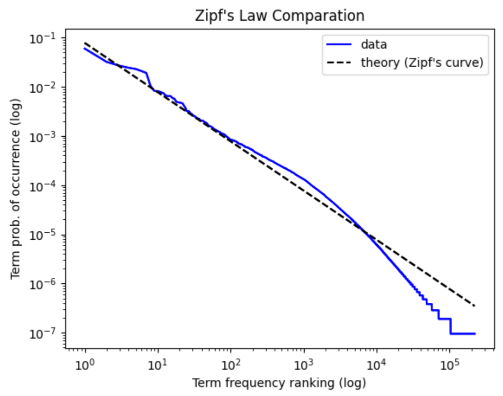


As for further analysis, here I state the Zipf's equation (s=1)

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

where f denotes the normalised frequency of a term, k denotes the term's frequency rank in the our corpus (with k = 1 being the highest rank), N is the number of terms in our vocabulary, and s is a distribution parameter which is set to be 1 for text.

From the graph above, I find the normalised frequency of a term is descending as its ranking increases with a shape similiar to the theoretical curve, which justifies that terms in the text set follow Zipf's law qualitatively. Though the tendency of the empirical distribution is similar to the theoretical curve, some noices are still obvious on the log-log plot below, especially at large ranking values. The reason for the differences might be that the denominator becomes increasingly large while the ranking k and the sum of i increments but the numerator is constant (equal to 1). That leads to increasingly lower f. Besides, the limited size of the data set is also an element leading to noises on the curve.



2.2 Deliverable 2

The code for task1 is stored in task1.py. This implementation will automatically save Zipf's curve plot and log-log plot under the same directory.

2.3 Deliverable 3

To get an inverted index, I first use the function constructed in task1 to extract terms in the passages. For every passage I get a token, and all tokens are stored in a data structure of list-sublist. Then, I use the structure of 'word: passage: word frequency in the passage' to store the inverted index, because in task3 I need to compute TF-IDF of passages which will use the frequency of a word in a particular passage.

2.4 Deliverable 4

The code for task2 is stored in task2.py. This implementation will automatically output inverted index data for later retrieval.

2.5 Deliverable 5

The cosine similarity scores between queries and passages are stored in tfidf.csv.

2.6 Deliverable 6

The BM25 scores between queries and passages are stored in bm25.csv.

2.7 Deliverable 7

The code for task3 is stored in task3.py. Also, some files will automatically stored for later analysis.

2.8 Deliverable 8-10

The values of query likelihood language models with Laplace smoothing, Lidstone correction , and Dirichlet smoothing are stored respectively in laplace.csv, lidstone.csv, dirichlet.csv.

2.9 Deliverable 11

1. Which language model do you expect to work better?
I expect the Dirichlet smoothing model work better because it releases the influence of document size on estimation making smoothing depend on document size. In this case, with $\mu = 2000$, it also treats the weight of unseen words at an appropriate level.
2. Which ones are expected to be more similar and why?
Laplace model and Lindstone model, because they are variants and have the same consideration for unseen terms as the equations show, only with difference in the weight of them. Therefore they tends to be more similar.
Laplace model:

$$P(w|D) = \frac{tf_{w,D} + 1}{|D| + |V|}$$

Lindstone model:

$$P(w|D) = \frac{tf_{w,D} + \epsilon}{|D| + \epsilon|V|}$$

where $tf_{w,D}$ denotes the number of occurrence of words in document D, |D| denotes the total number of word occurrences in the document D, |V| denotes the number of unique words in the entire collection, ϵ is a parameter for lowering weight of unseen terms.
3. Comment on the value of $\epsilon = 0.1$ in the Lidstone correction – is this a good choice, and why?
For Lidstone correction, this is a good choice, because that will treat unseen words less important when retrieving, which will contribute to a more user-satisfied result.
4. If we set $\mu = 5000$ in Dirichlet smoothing, would this be a more appropriate value and why?
It will not be a more appropriate value, because that will give lower weight to Maximum Likelihood Estimate and higher weight to Background Probability. The over importance counted for Background Probability I suppose might lead to a lower precision and a higher recall, or a less accurate query result.

2.10 Deliverable 12

The code for task4 is stored in task4.py.