COMP0084: Information Retrieval and Data Mining CW 1

21134487

Abstract

1 Introduction

1.1 Objective

Imagine a library with no dearth of books, since infinity is too large a number to imagine, let's assume there are a billion books, and the nature of readers who come to this library, do not strictly adhere to asking for books but generally tell the librarian about the topics they are looking for in the library, how will the librarian effectuate retrieving the relevant books from the library?

This is exactly what the scenario is today the imaginary library is but a microcosm of the internet today, where the number of web pages indexed on Google alone in the last one year, that is, February 2021 to February 2022 ranged from 10 billion (February 2022) to a peak of 55 billion pages (February, March, and April 2021)(Kunder).

Information retrieval concerns itself exactly with this problem, that is, finding relevant material from a large collection that caters to an information need.

The collection in the first example's case will be, books and in the internet's case will be a collection of documents. The 'information need 'in the case of the library would be a person inquiring about a relevant book, their words if imagined as a written text can be visualized as what a person would type on the internet to retrieve information from a search engine, a query.

In the tasks that follow, the overarching approach will be to develop different information retrieval models to retrieve passages for a given list of two-hundred queries each mapped to thousand different candidate passages.

However, the assignment is not restricted only to forming different passage retrieval models and will take into account other steps such as, pre-processing the data, extracting a vocabulary, creating an inverted index, etcetera. A comprehensive list of the objectives is as follows.

1. Extracting 1-grams from the provided passage-collection.txt and then performing pre-processing on the extracted tokens (1-grams) to build a vocabulary and verify if it follows the Zipf's Law.

2. Developing an inverted index for the passages in the provided candidate-passages-top1000.tsv using the vocabulary extracted in task 1 after another optional pre-processing step (removing stop-words).

3. Using the inverted index developed in task 2 above, develop TF-IDF vector representations for both the passage and queries, and then use the TF-IDF representations to calculate the Cosine similarity scores for each query and the corresponding 1000 candidate passages per query and then sorting and saving the top 100 per query into a file called tfidf.csv. Furthermore, the inverted index will be used to develop BM25, a probabilistic retrieval model to rank the top 100 passages from the 1000 candidate passages per the supplied 200 queries.

4. Implementing query likelihood language models to retrieve the top 100 passages from the provided 1000 passages per 200 queries. The models will be implemented with the provided configurations for each of the following-Laplace Smoothing, Lidstone Correction, and Dirichlet Smoothing.

1.2 The Dataset

Three files were provided for the task, test-queries.tsv, passage-collection.txt, and candidate-passages-top1000.tsv.

- test-queries.tsv is a file containing qid, that is, the query identifier, along with the query

text separated by tabs. Each row of the file corresponds to different pairs of qid and query text, comprising 200 rows in total. This file was used to extract the query save order, as instructed by the assignment description.

- passage-collection.txt is a text file containing the entire collection of passages where each row contains a different passage. In all this file contains 182,469 rows, corresponding to the same number of passages.

- candidate-passages-top1000.tsv is a tab-separated file containing values of <qid pid query passage> in each row. Each query has corresponding 1000 candidate passages (with some exceptions having 37 and lower). The file in all has 189,876 rows where some passages were found to also have duplicates implying that some queries fetched the same passages.

2 Tasks

2.1 Task 1 : Text Statistics

2.1.1 Task Overview

In this task, as instructed, the data in passage-collection.txt was used to extract 1-grams from the text, some pre-processing steps were performed before creating a vocabulary list of around 98,562 words. Then the number of occurrences for each word in the vocabulary were counted followed by sorting them in decreasing order to calculate ranks. The output was then stored into a data frame and more calculations were done per term, calculating the Zipf Frequency, Zipf Fraction, and Normalized frequencies followed by plots of Normalized frequency Vs Rank, and a log-log plot of Frequency Vs Rank. The vocabulary extracted from the task was saved as a dictionary using the np.save feature of the numpy module to be used in the subsequent tasks.

2.1.2 Pre-Processing

The NLTK(nlt) library in Python was used to tokenize the words while simultaneously converting all text into lower-case to be careful about not counting a term multiple just because of differences in capitalization, then only terms with alphabets were extracted as tokens and appended to a list of all the tokenized terms. This task is extremely pivotal from the perspective of improving the efficiency of the tasks which will sequentially follow

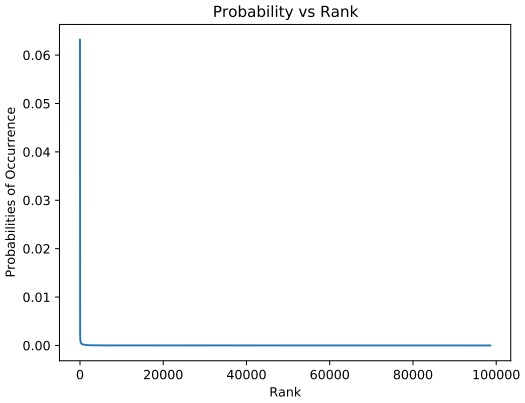

Figure 1: Probability of Occurence Vs Rank

as pointed out by (Morik and Scholz, 2004) as it essentially reduces the dimensions of the vector used to calculate the relevance in our case. If digits and punctuations were included, the dimensionality would have been significantly larger eventually resulting in computationally more expensive subsequent tasks of calculating the relevance(Srividhya and Anitha, 2010).

Additionally, post the basic pre-processing steps of extracting 1-grams of only terms with alphabets, lemmatization was performed on the extracted terms as opposed to stemming as it has the former has been found to improve the accuracy of document retrieval because of the underlying approach of morphological analysis to remove inflection endings reducing the word to a more dictionary-like form(Balakrishnan and Lloyd-Yemoh, 2014).Whereas, stemming removes derivational suffixes and inflections by reducing the word to stems (Balakrishnan and Lloyd-Yemoh, 2014).

In essence, lemmatization will be able to reduce and map synonyms to each other as juxtaposed to Stemming eventually resulting in more effective document retrieval. For both tokenization and stemming, the NLTK(nlt)library was used.

2.1.3 Zipf's Law: Text Statistics

Zipf's law is a power-law stating that the frequency distribution of words appearing in a human language can be explained through a simple mathematical law, where the r^{th} most frequent term has a frequency proportional to

$$f(r) \propto \frac{1}{r^{\alpha}} \tag{1}$$

Where,

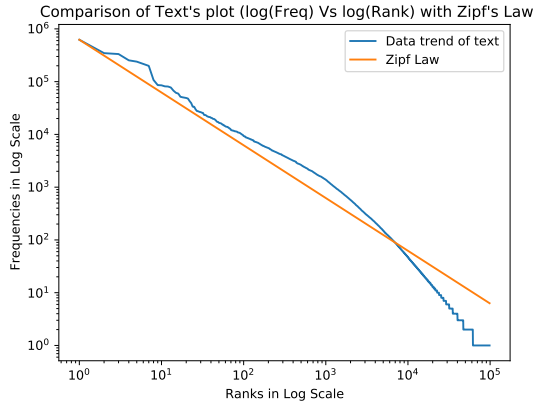

Figure 2: Log-Log Plot of Frequency vs Ranks

- $\alpha = 1$(Piantadosi, 2014)(Zipf, 2013)(Zipf, 2016)

- r corresponds to the frequency rank of a word and $f(r)$ denotes its frequency in the corpus.

That is, in the context of our example, where, $\alpha = 1$, the most frequent word with $r = 1$ will have a frequency $\propto 1$ and the second most frequent word will have a frequency $\propto \frac{1}{2}$ and so on.

As described above, a function was implemented to count the number of occurrences, to normalize them, the normalized frequencies were plotted against the ranks for all the 98,562 words.

From the trend in figure 1, it appears that the data follows Zipf's law, however, we will get to look at things more granularly through a log-log plot of frequencies Vs Ranks in figure 2. Here we can see that according to Zipf's law we should get a constant relationship between the log of the frequencies Vs the log of the ranks, however, we see a slight deviation, especially for the terms with very high ranks and fewer frequencies. The relationship largely doesn't deviate and appears to be constant for the first few terms "the", "a", "of". However, it is important to note, as pointed by(Clauset et al., 2009) that the qualitative adherence to power-law through visualization is not necessarily an accurate approach and needs closer scrutiny. The adherence is also because the corpus is very large(Moreno-Sánchez et al., 2016), if we were to remove stop words, we would see the corpus deviate from Zipf's law significantly.

2.2 Task 2: Inverted Index

2.2.1 Task Description

The task was to use the vocabulary extracted in task 1 and then to remove the stop-words (if required) before constructing an inverted index to be used in the subsequent tasks of creating tf-idf, and then calculating cosine similarity followed by a BM-25 model. To do the same, saved vocabulary from Task 1 was loaded and stop-words were removed from it using a union of stop-words contained by two python libraries, NLTK and Spacy. This was done because the corpus of stop-words contained by the two libraries slightly varies and by taking a union of the two and then using only the unique stop-words, the number of stop-words to be removed was expanded.

2.2.2 Constructing an Inverted Index

After removing the stop-words using NLTK and SPACY, the total vocabulary was reduced to 98,239 words. Then, all the passages and queries in the candidate-passages-top1000.tsv were tokenized using the same pre-processing stops as specified in task 1 along with the removal of stop-words and the occurrence for each word was counted and a nested dictionary was created like the structure shown below.

{ word 1 : {pid 1 : frequency , pid 2: frequency. . . .} word 2: {pid 1: frequency, pid 2: frequency. . . } ....}

A similar approach was taken for queries, where pid was replaced by qid in case of the queries.

Both the indexes were saved to be used later in Task3 and Task4. A list of vocabulary after removal of the stop-words was also saved as a list using the json library.

2.3 Task 3: Retrieval Models

2.3.1 Task Description

The task required to use the inverted index generated in task 2, to extract TF-IDF vector representations of the passage followed by the queries and then to compute cosine similarity and save the top 100 passages (according to similarity score) per the 200 queries into a csv file in an order specified in test-queries.tsv. The cosine similarity score was followed by a BM25 retrieval model, saved in the same format as cosine similarity, saving top 100 passages per query in the desired order in a csv file.

2.3.2 Calculating Term Frequency

A dictionary nested with lists was created with pid (passage id) as key values containing all the terms divided by the total passage length, that is, the number of total terms in the passage divided by the sum of the total terms in the passage. An example of the approach has been shown below

{7130104:
[['definition',0.21428571428571427],
['rna', 0.35714285714285715],
['example', 0.14285714285714285],
['type', 0.14285714285714285],
['molecule', 0.14285714285714285]]}

The formula can be seen as

$$tf_{(i)} = \frac{f_{(i)}}{N} \tag{2}$$

Where,

- $tf_{(i)}$= term frequency of the ith term in the passage
- $f_{(i)}$= frequency of ith term in the passage
- N=document length of the passage,that is, sum of total number of terms per passage.

2.3.3 Calculating IDF Weights

Followed by term-frequency, IDF weights were calculated for the entire vocabulary minus the stop-words and they were stored in a dictionary with the word as identifiers.However, smoothing by adding 1 in the formula was adopted to solve for division by zero or negative values.The formula used was

$$IDF_{(t)} = \log 10\left(1 + \frac{N}{n_t}\right) \tag{3}$$

Where,

- IDF(t) is the IDF weight of the tth term in the vocabulary
- N is the total number of documents in the collection, in our case total number of unique passages, 182,469 passages.
- n_t is the number of documents the tth appears in

An example has been shown below.

{'definition', 1.3297921351232929
'rna', 2.827267093105488
'example', 1.465158111651221
'type', 1.2367111934739756
'molecule', 2.138873692704368}

2.3.4 Calculating TF-IDF

Post the TF and IDF calculations, tf-idf vector representations were calculated per pid and qid, for the passages and queries respectively. The data structure used for this exercise was a dictionary with pid as identifiers in case of passages, and qid in case of queries, followed by a nested list of lists containing tf-idf vectors of the concerned passage or query.

The formula used to calculate is as follows.

$$tf_{(t)} \times idf_{(t)} \tag{4}$$

The t^{t} term's term-frequency is multipled with it's idf weight using the idf dictionary and then the tf-idf vector for each of the terms in the passage is stored in a dictionary with the structure specified above. An example of it has been show below.

{7130104:
[['definition', 0.2849554575264199],
['rna', 1.0097382475376744],
['example', 0.20930830166446013],
['type', 0.17667302763913936],
['molecule', 0.30555338467205256]]}

The tf-idf for queries was calculated using (4) with the difference being term frequency dictionary referred for it was obtained using equation (2) for each query after pre-processing the tokens along with the removal of stop-words.

2.3.5 Cosine Similarity

The calculate the cosine similarity a function was created which converted each passage and query into vectors, calculated the similarity score, and stored them into a list. Another additional dictionary was created to provide mapping of each query to the 1000 candidate passages. This way, cosine similarity was calculated for all the 1000 candidate passages per 200 queries and were stored in a dictionary nested with a list of lists.

The cosine similarity was calculated using the following formula.

$$Cos(X, Y) = \frac{X.Y}{\|\mathbf{X}\| \times \|\mathbf{Y}\|} \tag{5}$$

Where, X and Y represent two different vectors, in our case these were different qid,pid pairs mapped from the candidate-passages-top1000.tsv.The returned dictionary from the function was later sorted and top 100 passages were saved for each of the 200 queries into a file called tfidf.csv.

2.3.6 BM 25

The BM25 model was implemented using the following formula, BM25(Q,D)=

$$\sum_{i \epsilon Q} \log\left(\frac{N+0.5}{n(i)+0.5}\right) \times \frac{(k_1+1).f_i}{K+f_i} \times \frac{(k_2+1).qf_i}{k_2+qf_i} \tag{6}$$

Where,

- N is number of documents
- n_i is the number of times the i[th] appears in the document
- $k_1, k_2, K$are terms with values set empirically
- $K = k_1((1-b)+b.\frac{dl}{avdl}$
  Where,
  - dl is document length for which score is being calculated, and avdl is the average document length
  - b has a value varying from 0 to 1000
- f_i, qf_i are the frequencies of term in the document and query respectively

For the task the corresponding log component for each query term in equation (6) was separately calculated and stored in a dictionary called bmidf. Then the inverted index for both passages and query loaded from Task 2 was converted into a different data structure, that is, a dictionary with a nested dictionary for passages frequency per pid and a dictionary with a nested list of lists for term frequency per qid. Other than that a dictionary with document lengths was also created to make the BM25 calculations swifter. The values of k_1, k_2, b where chosen to be 1.2,100, and 0.75 as specified.

A similar approach to sorting and saving the returned dictionary was followed as task 2.3.5 and the file was saved as bm25.csv. In addition to the file, multiple other dictionaries and lists were created as sub-tasks to make the models efficient, all of these were saved using the numpy module to be used later in Task 4.

All the dictionaries generated in this task were saved using the numpy library to be used later in task 4.

2.4 Task 4: Query likelihood language models

2.4.1 Task Description

The task was to implement query likelihood language models with

1. Laplace Smoothing
2. Lidstone Correction with $\epsilon = 0.1$
3. Dirichlet smoothing with $\mu = 50$

To do these tasks we loaded dictionaries and lists saved from Task 3 which were in turn extracted from the specified files for these tasks, test-queries.tsv and candidate-passages-top1000.tsv.

2.4.2 Laplace Smoothing

Smoothing can colloquially be understood as a process to estimate the probability of missing words. Laplace smoothing is implemented by adding 1 to the occurrence of the term and then by re-normalizing by the sum of number of word occurrences in the document D and total unique vocabulary size V.

$$\frac{m_1+1}{D+V} + \frac{m_2+1}{D+V} ..... \frac{m_V+1}{D+V} \tag{7}$$

Where,$m_1, m_2 ... m_V$ are the number of word occurrences in the document.

2.4.3 Lidstone Correction

Laplace smoothing can attribute too much weight to unseen terms and Lidstone smoothing is a method to discount the weights. It was implemented with the specified value of $\epsilon = 0.1$.

$$P(w|D) = \frac{tf_{w,d} + \epsilon}{D + \epsilon V} \tag{8}$$

Where,

- $tf_{w,D}$is the term frequency of the word W in document D

2.4.4 Dirichlet Smoothing

Dirichlet smoothing is a smoothing technique which relies on interpolation, that is, uses background probabilites to avoid giving too much weight to unseen terms like Laplace or equal weights to unseen terms like discounting methods such as Lidstone correction. In Dirichlet smoothing, the smoothing depends on the sample size. For our case, as specified, we chose the value of constant μ to be 50.

The estimates for words that occur is

$$\lambda P(w|D) + (1-\lambda)P(w|C) \tag{9}$$

Where,

- $P(w|C)$is the probability for word w in collection C

- $P(w|D)$is the probability for word w in Document D

And λ is a parameter defined by,

$$\frac{N}{N+\mu} \tag{10}$$

Where, N is the length of sample(document) and μ is a constant.

2.4.5 Comments

The language model expected to work better in this task was the one with Dirichlet smoothing since both Laplace and Lidstone smooth by addition, which makes them additive in nature. While Lidstone discounts the weights given to unseen terms, it's still not as effective as Dirichlet because it doesn't adapt to the sample size, or say the document length in question and resulting in an equal discounted weight.

With higher values of ϵ the Lidstone correction would head closer into the direction of Laplace. A value of 0.1 seems doesn't give too much weight to the unseen words and discounts effectively, albeit it can be argued that this is a hyper-parameter and it needs to optimized for the given task.

While (Bennett et al., 2008) do a comparative analysis of comparing various retrieval models, concluding Dirichlet smoothing to be better in most scenarios especially outperforming all other models like BM25 in the case of short key queries. As pointed by (Bennett et al., 2008) it is also important to tune the parameters in respective models according to the task at hand and no model can be generalized as the best retrieval model for all tasks.

In our case, as shown by (Seo and Croft, 2010) a high value of $\mu = 5000$ will work well with larger length queries, but as pointed out by (Zhai and Lafferty, 2017) the optimal value of μ lies between the range 500-10000, so ideally as corroborated by evidence the model performance should further improve with a value of 5000.

.

References

[link].

Vimala Balakrishnan and Ethel Lloyd-Yemoh. 2014. Stemming and lemmatization: a comparison of retrieval performances.

Graham Bennett, Falk Scholer, and Alexandra Uitdenbogerd. 2008. A comparative study of probabilistic and language models for information retrieval. In *Database Technologies 2008: Proceedings of the Nineteenth Australasian Database Conference (ADC 2008)*, pages 65–74. RMIT University.

Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703.

Maurice De Kunder. The size of the world wide web (the internet).

Isabel Moreno-Sánchez, Francesc Font-Clos, and Álvaro Corral. 2016. Large-scale analysis of zipf's law in english texts. *PloS one*, 11(1):e0147073.

Katharina Morik and Martin Scholz. 2004. The miningmart approach to knowledge discovery in databases. *Intelligent Technologies for Information Analysis*, page 47–65.

Steven T Piantadosi. 2014. Zipf's word frequency law in natural language: A critical review and future directions. *Psychonomic bulletin & review*, 21(5):1112–1130.

Jangwon Seo and W Bruce Croft. 2010. Unsupervised estimation of dirichlet smoothing parameters. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, pages 759–760.

V Srividhya and R Anitha. 2010. Evaluating preprocessing techniques in text categorization. *International journal of computer science and application*, 47(11):49–51.

Chengxiang Zhai and John Lafferty. 2017. A study of smoothing methods for language models applied to ad hoc information retrieval. In *ACM SIGIR Forum*, volume 51, pages 268–276. ACM New York, NY, USA.

George Kingsley Zipf. 2013. *The psycho-biology of language: An introduction to dynamic philology*. Routledge.

George Kingsley Zipf. 2016. *Human behavior and the principle of least effort: An introduction to human ecology*. Ravenio Books.