Information Retrieval & Data Mining, Coursework 1

Kyveli Christina Tsioli*
kyveli.tsioli.20@ucl.ac.uk
University College London

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00

1 DATA

The dataset that will be used for this task is available through this url. The dataset consists of 3 files:

- test-queries.tsv is a **tab** separated file, where each row contains a query ID (qid) and the query (i.e., query text).
- passage_collection.txt contains passages in our collection where each row is a passage.
- candidate_passages_top1000.tsv is a **tab** separated file, containing initial rankings that contain 1000 passages for each of the given queries in file test-queries.txv. The format of this file is <qid pid query passage>, where qid is the query ID, pid is the ID of the passage retrieved, query is the query text and passage is the passage text, all tab separated.

ADD SHORT DESCRIPTION OF THE DATA MAYBE?!

2 SUBTASKS

(1) Text Statistics.
Perform any type of required pre-processing on the collection of passages. Implement a function that counts the frequency of terms from the provided dataset, plot the distribution of term frequencies and verify if they follow Zipf's law. Report the parameters for Zipf's law for this collection. Use the full collection (passage_collection.txt). Generate a plot that shows how the results obtained from the model based on Zipf's law compare with the values gained from the actual collection.

(2) Inverted Index.
Build an inverted index for the collection so that passages can be retrieved from the initial set of candidate passages in an efficient way. When implementing an effective inverted index, consider storing additional information such as term frequency and term position. Report what type of information has been stored in the inverted index. Since the task in this project is to focus on **re-ranking** candidate passages that are given for each query, a separate index can be generated for each query with the use of the candidate list of passages for each query (candidate_passages_top1000.tsv).

(3) Retrieval Models, Language Modelling.
Implement the query likelihood language model with i) Dirichlet smoothing, where $\mu = 2000$, ii) Laplace smoothing and iii) Lindstone correction with $\epsilon = 0.5$ and retrieve 100 passages from within the 1000 candidate passages for each query. For all three smoothing variants, submit the 100 passages retrieved in sorted order (decreasing) for both models. Which smoothing version is expected to work better?

3 IMPLEMENTATION

3.1 Data pre-processing

Several steps have been followed in order to preprocess the data. More specifically:

- ***Tokenisation***
  First step towards text pre-processing is splitting sentences into individual terms, commonly referred to as "tokens". To perform this operation, the nltk library was used. Tokenisation procedure transforms each passage into a list of words("tokens") separated by a comma.
- ***Stemming***
  Refers to reducing each word to its root form (stem). Stemming is a widely used technique in information retrieval tasks. More specifically, stemming is a way to find the root word given many variations of that word that appear in the collection. For example, in the candidate passages collection we come across the words "studying", "studies", "study","studied", which according to the PorterStemmer we are using reduces down to "studi", which is the common root of all these words. In that sense, stemming could be perceived as a form of term normalisation. I used PorterStemmer as it is the most common algorithm for english stemming. It is worth mentioning that a word's stem itself may lack a semantic interpretation. At this stage lemmatisation could also be used instead of stemming.
- ***Stop words removal***
  This technique removes words that are common or often used and that usually do not convey any additional meaning to the context. Extremely common words which have little value in contributing to select documents matching a user's query need to be excluded from the vocabulary of the collection. Removing stopwords can help improve the performance of information retrieval models by making the retrieval faster because there are fewer meaningful tokens to be examined and processed. Typical such words are words like "the", "a", "will" etc. For this project the default NLTK's list of english stopwords was used.
- ***Lower-case***
  Every word of a passage has been converted to lower-case. Lowercasing is very useful for search in information retrieval, as usually users type queries in lowercase and we want from the retrieval system to match the same results for a query regardless of whether is being typed in upper-case or lowcase.
- ***Non-alphabetic words***
  Non-alphabetic characters are removed from the collection

https://www.nltk.org/
https://www.nltk.org/howto/stem.html
https://gist.github.com/sebleier/554280

as they do not provide any informative content for the retrieval.

REMOVE PUNCTUATION GOES WITH REGEXP FIRST BULLET?

3.2 Text Statistics extraction and Zipf's law verification

Retrieval models and ranking algorithms depend heavily on statistical properties of words that belong to the corpus (e.g. distribution of word counts). To extract the text statistics of the corpus, term frequencies have been computed for the passage collection. For this purpose, a dictionary is being used as a data structure to retain the information in pairs, with key being each unique token and value the corresponding frequency of that token in the entire collection. Note that the raw frequency is computed at this step, i.e. the number of times each specific token occurs in the collection. This step is necessary in order to plot the distribution of term frequencies and verify Zipf's law.

According to Zipf's law, the distribution of word frequencies is very skewed, meaning that a few words occur very often, whereas other hardly ever occur. A typical example of this is that the two most common words ("the","of") of the collection make up for approximately 9.3% of all words occurrences in the passage collection. Zipf's law is a power law of the form $y = kx^c$ with c=-1 (theoretical value) which states that the product of a word's rank(r) times its frequency(f) is approximately a constant(k). In other words, this means that the probability of word occurrence is inversely proportional to its ranking, where ranking is considered to be the position of the order, when documents are ordered in decreasing frequency. Important thing to note is that in order to verify Zipf's law, stop words removal and stemming should **not** be performed, as they would change the text statistics (stopwords are expected to be the most frequent words in the corpus). First step to verify Zipf's law is producing a plot in which the x-axis represents the rank of word occurrence (in decreasing frequency) and the y-axis the probability of each occurrence.

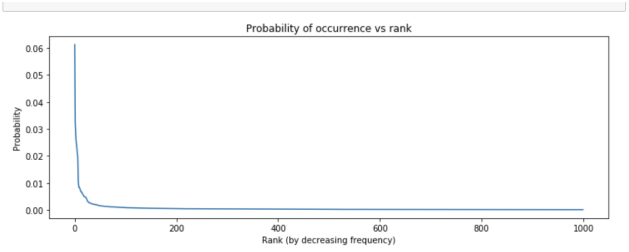

Figure 1: Probability of occurrence vs rank

Additionally, on a log-log plot power laws yield a straight line with slope c, i.e.:

$$\log(y) = log(kx^c) = \log(k) + c\log(x)$$

Plot 2 fits a linear regression line to capture the relationship of log frequencies of words against their corresponding log rankings. As it can be seen visually, the line has a very good fit to the data.

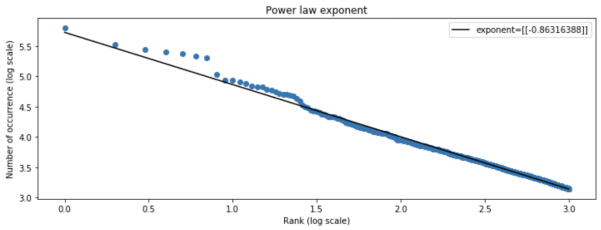

Figure 2: Rank (log-scale) vs Number of occurrence (log-scale)

More specifically,t he regression coefficient has been found to be equal to -0.86 which is very close to the theoretical value $c = 1$ of Zipf's law, and the intercept of the regression line was found to be equal to 5.72 for this collection. The black straight line refers to the theoretical line of Zipf's law while the blue dotted line corresponds to the line resulted from our data.

4 INVERTED INDEX

An inverted index is a commonly used data structure in text search which associates documents (passages) with words in an efficient way. It is named as "inverted" as it addresses the question: "Given this word, what are the documents that contain this word?". Since the task in this project is to re-rank the already retrieved passages for each query, a separate inverted index for each query is constructed by using the candidate list of passages for that query.

Before creating the inverted indices, three dictionaries are constructed which store the information needed in an efficient way. A dictionary which associates each query ID with its list of candidate passage IDs, a dictionary which associates each query ID with a list comprised of its corresponding list of pre-processed tokens, and a dictionary which associates each passage ID with its list of pre-processed tokens. At this point one can easily have access to the passage ids that are associated with each query, and to the tokens which make up each query and each passage respectively.

The inverted index for each query is constructed with the use of a dictionary whose keys are the passage IDs and whose values are the list of tokens of the corresponding passage. An inverted index is then constructed for each query by filtering out the passages that we are not provided with for this specific query.

For each query, an inverted index is created which contains the tokens of the passages associated with that query. Specifically, The information stored in the inverted index is the passage ID in which the pre-processed token of the passage occurred, and the position(s) at which it occurred in that passage. More specifically, for each token in a passage, the token is added only once as a key to the dictionary, inside this key there is another key of the passage ID and whose value is a set of positions that this token appeared in this passage ID.

In other words, the data structure of the inverted index is a dictionary with nested dictionaries, of the form:

$Token : PassageId : location$ i.e: $'studi' : 8406305 : [0], 7371584 : [11], ...$

By saving the positions at which the term appeared in the passage, it is easy to calculate the term's frequency in that passage. As the tf-idf calculation is needed to construct the vector space model, the

term frequencies need to be derived efficiently from the inverted index, and this is why the inverted index is designed in that way. Apparently, the term frequency is simply calculating by the length of the list where the positions are stored.
It is important to note that for the construction of the inverted index and the retrieval models, stemming and stopwords removal has been conducted together with the rest of the pre-processing techniques mentioned above. Stopwords would account for a large fraction of the whole corpus so omitting them greatly reduces costs associated with the inverted index data structure.

5 RETRIEVAL MODELS

5.1 Vector Space Model

To define how relevant a document is to a specific query, we need a notion of distance or "closeness". Queries and passages are thus represented as vectors in an |V|-dimensional vector space, where |V| stands for the vocabulary size of the corpus. The retrieval now becomes a task of ranking documents according to their proximity (i.e. similarity of vectors) to the specified query in this space.

5.1.1 Tf-idf representation.

To create the aforementioned vector representations of each query and its associated passages, a function that calculates the tf-idf scores of each token is created.
The tf-idf weighting score of term t is given by:

$$tf\_idf\_t = tf_t * idf_t \tag{1}$$

where tf_t is the number of times the token appears in the passage and idf_t is given by:

$$idf_t = \log(\frac{N}{n_t}) \tag{2}$$

where N is the number of documents in the collection for that query and n_t is the number of passages containing this term. Important thing to note is that in the tf-idf implementation filtering is performed on passages so that only the passages corresponding to the specific query are being considered.

5.1.2 Cosine similarity.

Cosine similarity measures the similarity between a query and its associated passages, both represented as vectors in the vector space. In this project, the query vector and each passage vector were constructed using the dictionary data structure. As such, each key of the dictionary is a token (of the query or the passages accordingly) and the associated value is its tf-idf score. The cosine similarity then is calculated based on the equation:

$$cos(\theta) = \frac{\mathbf{xy}}{||\mathbf{x}||||\mathbf{y}||} \tag{3}$$

It is worth mentioning that only the common keys (i.e. tokens) between the query vector and the passage vector will be multiplied in the nominator. In other words, by using dictionaries we can easily identify the tokens that belong to the intersection of the query and its passage each time we calculate the similarity score between them, and this technique eases the computational cost. However, it is important to note that when calculating the norms in the denominator, all the tokens that appear in each vector will be taken into account.

5.2 BM25

The BM25 score of a given query with its associated passages is calculated according to the following formula:

$$\sum_{i \epsilon Q} \log \frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5))/(N - n_i - R + r_i + 0.5} \frac{(k_1 + 1)f_i}{K + f_i} \frac{(k_2 + 1)f_i}{k_2 + qf_i} \tag{4}$$

where the parameters used in the formula are the following:

- $k_1 = 1.2$
- $k_2 = 100$
- constant K is given by :

$$K = k_1(1 - b + b * \frac{dl}{avdl}) \tag{5}$$

  where dl denotes the passage length of each passage of the corpus, and avdl denotes the average passage length
- $r_i = 0$, where r_i is the relevance of each word
- $R = 0$, where R is the relevance of the passage
- $b = 0.75$
- N is the number of documents in the collection relevant to each query
- n_i is the number of times the term occurs in the documents
- qf_i is the number of times the token occurs in the query

The parameters for the BM25 scoring function were chosen based on the referenced paper.. The scores were calculated for each query and its corresponding already retrieved passages given in the candidate_passages_top1000.tsv file and then ranked.

6 QUERY LIKELIHOOD LANGUAGE MODELS

In Information Retrieval, each document D is considered as a sample of language. It is a sample, because if the author is asked to reproduce the document, they would come up with a similar document, but not exactly the same. Considering all possible strings the author could have written while producing the document, some strings are more likely than others to appear in the document. The probability P(s|D) represents the probability that the author would write string s.
A standard language modelling approach in Information Retrieval is the "Query Likelihood Model". This technique is used to find the probability of generating a given query, given a document language model (i.e. a distribution over words), $P(Q|M_D)$.
This approach ranks documents by the probability that the query could be generated by the document model, and it works as follows: First, a language model M_D is estimated for every document D in the collection, and then documents are ranked by the probability of producing the specific query. In mathematical notation, the above translates to the following:

$$P(Q|M_D) = P(q_1, ..., q_k|M_D) = \prod_{i=1}^{k} P(q_i|M_D)$$

This approach uses the Maximum Likelihood Estimation (MLE) for each document in the collection, i.e:

$$M_D = P(w|D) = \frac{f_{w,D}}{|D|} \tag{6}$$

http://www.staff.city.ac.uk/ sbrp622/papers/foundations$_b m25_r eview.pdf$

where $f_{w,D}$ stands for the raw term frequency of the token in the document D and |D| states the length of the document.
For this project, Unigram Language Models are constructed, which make a fundamental assumption: each word has an associated probability and words are independent of each other. With that in mind, the probability of producing the query q, given the language model M_D of document D using the Maximum Likelihood Estimation and the Unigram assumption becomes:

$$P(Q|M_D) = P(q_1, ..., q_k|M_D) = \prod_{i=1}^{k} P(q_i|M_D) = \prod_{i=1}^{k} \frac{f_{q,D}}{|D|} \quad (7)$$

where M_D is the language model of document D, $f_{q,D}$ is the raw term frequency of term q in document D and |D| is the number of tokens in document D.
One weakness of this approach is that some words will not appear in the documents but could potentially appear in the user's query. Following the aforementioned approach for one word that does not appear in any of the documents, the estimated probability would be $P(w|M_D) = 0$ and that would mean that documents only assign to a query a non-zero probability if all of the query terms appear in the document. To prevent this from happening and improve our estimates for missing words, smoothing is used.
In this project, three smoothing variants were implemented, inside the same function where the argument specifies the type of smoothing chosen.
The objective is to rank documents (passages) based on probability that they are relevant to the query.
Smoothing is a technique that assigns non-zero values to terms in the query that are not present in the document.

6.1 Laplace Smoothing

The Laplace Smoothing technique estimates are given by the following equation:

$$P(Q|D) = \prod_{i=1}^{k} \frac{f_{q,D} + 1}{|D| + |V|} \quad (8)$$

This means that 1 is added in the raw frequency of each term q in the document D. By doing so, it is ensured that if a word does not appear in this document, it will still contribute to the ranking score. The score is normalised by dividing by the number of unique words in the entire collection (vocabulary size) denoted by |V| and the number of words in document D denoted by |D|. The normalisation takes place so as to obtain probabilities. Overall, Laplace smoothing could be considered as a technique that assigns uniform priors over words.

6.2 Lindstone Smoothing

Lindstone Smoothing is a modification of the Laplace Smoothing technique. As seen above, Laplace smoothing assigns too much weight to unseen terms. Lindstone method provides a correction on this with the following way: instead of adding 1 to all the counts of the words in the vocabulary, it adds a small number ϵ to every term. The formula for Lindstone correction is the following:

$$P(Q|D) = \frac{tf_{w,D} + \epsilon}{|D| + \epsilon|V|} \quad (9)$$

The project clearly states that ϵ should be set equal to 0.5.

6.3 Dirichlet Smoothing

The concern associated with the aforementioned smoothing techniques is that they treat unseen words equally, either by adding 1 or an ϵ term to their counts. However, some words are more frequent than others and this leads to the idea to combine or "interpolate" the MLE estimates with the estimates from the entire collection. For this purpose, the probability for word w in the collection language model (background probability) is introduced.
The Dirichlet Smoothing technique computes the weighted combination of the document language model and the collection language model for the entire collection. The weights associated with each term depend on a constant parameter μ which is specified by the user (in this project $\mu = 2000$) and the document length N. The combination of these two yields the parameter $\lambda = \frac{N}{N+\mu}$. The full model equation for Dirichlet Smoothing is given by the equation:

$$\log(P(Q|D)) = \sum_{i=1}^{n} \log(\lambda \frac{f_{q_i,D}}{|D|} + (1 - \lambda) \frac{f_{q_i,C}}{|C|}) \quad (10)$$

where D refers to the specific document, C refers to the entire collection, f_{q_i} is number of times the i-th token of the query has occurred in the entire collection.

7 COMPARISON OF THE MODELS

As mentioned above, smoothing provides an alternative way to deal with unseen words by assigning probability mass to them. A weakness of Laplace smoothing is that it assigns too much weight to unseen terns, by adding 1 to them. Lindstone, on the other hand, adds an ϵ term to them. Both these two techniques treat every unseen word equally, which is not always the case for the document collection at hand. Interpolation methods such as Dirichlet Smoothing, take into account the fact that some words have not the same probability of appearing in documents and build a "background" probability, which is calculated based on information from the entire collection.
To conclude, to my opinion Dirichlet will perform most efficiently in assigning retrieval scores to the documents as it considers the document collection.