IRDM Course Project Part-I: Developing Basic Information Retrieval Model for Short Passage Retrieval

1 INTRODUCTION

In this coursework, I have accomplished various *Subtasks* according to the coursework description. For instance, a literate survey is performed for identifying the state of the art research work in the field of Information Retrieval (IR) that are mainly focused on short passage retrieval and passage ranking. Next, for the provided dataset, I have designed an experiment to show how the words and their frequencies are correlated by following the Zipf's law pattern. Then, I have implemented an inverted index that can be built for any given passage collection. This index is also used to implement the basic IR models such as Vector Space (VS) model and BM25 to re-rank the passages.

The remainder of this report describes each of the *Subtask* in detail. Section 2 deals with the *Subtask-1* (Literature Review). Section and Section describe *Subtask-2* (Text Statistics) and *Subtask-3* (Inverted Index) respectively. And finally, Section 5 provides the implementation of *Subtask-4* (Basic Retrieval Models).

2 LITERATURE REVIEW

Over the last few decades, many research works had been done in the field of IR in various domains. Many works have discussed the improvement of classical IR models. Some of the papers mentioned the applications of IR models in other scientific fields. However, in this coursework, the primary purpose of this literature review is to discuss the research works that are relevant to short text/passage retrieval and ranking. In order to do this, initially I was trying to perform a systematic literate survey to identify the related research papers utilizing the keyword based searching in popular digital libraries, for instance, ACM DL, IEEE Xplore. I have used relevant keywords such as ***"passage ranking"***, ***"short passage retrieval"***, ***"short text retrieval"*** and ***"document retrieval"*** to filter the research papers. I also used the combination of exact keyword and partial keyword search in order to include all the papers that are relevant to the search keywords.

Therefore, a significant number of papers are found as a result of the primary search. Then, I identified the relevant research work from those papers based on the keyword and its implication in these papers. Next I categorized these papers into various groups. According to the coursework description, the following subsections provide an exhaustive description of each of the category that is particularly focused on short passage/text/document retrieval and passage ranking.

2.1 HMM-Based Passage Retrieval

Hidden Markov Model (HMM) provides a sound mathematical background that was used to developed efficient IR models for passage retrieval. Elke & Peter [24] adopted HMM approach to IR relying on two stochastic processes. In the first process, text fragments are generated that are relevant to the user query and then in the next process text fragments are produced with the independence of any certain query. Depending on the distribution of the query, the generated text fragments can have high probability or low probability. The ratio of the probability of these two processes correspondent to the score of the particular text fragment/passage. The experimental results of their approach was very promising that encouraged future researchers to apply HMM in IR. Later, David et al. [23] applied HMM for developing a basic framework to incorporate various word generation mechanisms within the same model. This simple framework outperformed compared to the classic *tf-idf* based ranking. This is because they presented various refinement methods to perform blind feedback in the HMM.

Ludovic et al. [7] also developed HMM based IR model on the assumption of a number of documents may have a sequential structure that can be exploited in IR. For example, the journal and conference papers have a similar structure containing particular sections such as abstract, introduction, conclusion etc. Prior information of these sequential structures can be used to develop a scholastic process by HMM. They evaluated their approach on *Reuter's* data and their approach showed significant improvement. Further, Jing et al. [11][12] applied the HMM for accurately identifying query specific coherent passages. They only put emphasise on how to extract a single relevant passage accurately from each document. Their approach is very similar to the existing HMM based approaches [23][7]. However, their approach varied from the architecture of HMM and estimation function for measuring the parameter values. Their experimental result provided a significant evidence that HMM based IR model can be utilized on top of any classic passage extraction approach.

2.2 Cluster-Based Passage Retrieval

One of the prime purposes of passage revival is to rank the passages according to the relevance of the user query. It is observed that relevant passages are more similar compared to the non-relevant passages. Therefore, clustering the query specific documents improves the retrieval effectiveness [30]. The clusters are developed by an initial ranking to make a group of similar retrieval documents. Many researches are conducted to adopt the clustering based approach to rank the document for user given query. For example, Oren & Carmel [15] developed a model of building query specific cluster that contains high percentage of relevant documents. They first proposed a palette of cluster properties. Then they identified the quantitative measure of these properties and organized the clusters using the aggregation of the ranking supported by the quantitative measure. Further an extension of this approach was developed for a language model by the Oren et al. [16]. Their assumption was that query specific cluster contains high percentage of the relevant documents. Thus, the documents associated with the cluster can be utilized as the proxy for ranking it.

Jangwon et al. [27] proposed an approach that measures the geometric mean for representing multiple documents in a cluster. This

Table 1: Summary of Passage Retrieving Algorithms

No	Algorithm	Summary
1	SiteQ	This approach computes the score of a passage by summing the score of *n-sentence* from the passage [17]. Each sentence is scored depending on the query term density in the passage. Interestingly, the approach calculates the weight of the query term based on its parts of speech and idf value [29].
2	MITRE	It is a word overlap based algorithm. This was first developed by the Light et al. [20] by simply counting common number of terms presented in the passage and query. This was evaluated on QA system and it is found that QA performance is correlated with the answer repetition [29].
3	MultiText	This algorithm focused on term density [5]. The approach is very effective for the short passages having many terms with large *idf* values. For a particular question, the approach identifies the hot-spots where the answers may be located. Then it analyze the hot-spots depending on the answer types, term redundancy to extract the relevant answers [29].
4	ISI	This passage retrieval technique ranks the passages by measuring score following the general *idf* score and also considering the similarity score of the proper names and stemmed words between the query and passages [9]. This techniques utilized the semantic and syntactic knowledge to enrich the result accuracy [29].
5	IBM	It calculates various distant measures for the passages in a QA system [1]. For example, in *"matching word measure"*, the idf values are summed. And then in *"thesaurus match measure"*, idf values of the query words those synonyms are found in the passage are added. Moreover, *"dispersion measure"*, *"mis-match words measure"* and *"cluster words measure"* are calculated and these measures are then combined to get the final score for ranking the answers [29].
6	Alicante	This techniques is similar to Vector Space (VS) model, however, here the score is computed by cosine similarity with non-length normalized vector between the query and the passage [21]. It considers the number of appearance term in the query and the passage and also includes the idf values [29].
7	Voting	It is a hybrid passage ranking algorithm [29] combining *SiteQ*, *IBM* and *ISI*. It calculates initial rank score by running each of the approaches. Then, it computes overall rank score by considering the answers the other algorithms returned for the same questions.

is because, many documents may contain a specific term, however, the geometric mean of this term grows exponentially where the arithmetic mean increases linearly. This properties provide the robustness of the geometric mean that helps to a better representation of multiple documents. Moreover, they theoretically and empirically showed that geometric mean can be closer to the centre of the statistical manifold. Fiana & Oren [26] also developed a clustering based document ranking approach namely *ClustMRF* based on the *Markov Random Fields* (MRF) because it helps to integrate the variety of cluster relevancies such as query similarity, intra-document similarity etc. *ClustMRF* aims to rank the cluster by re-ranking the initially retrieved documents list where those cluster are created by the documents highly ranked in the list. Later, Sheetrit et al. [28] designed a cluster based document retrieval approach that can rank the cluster of similar passages utilizing the learn-to-rank techniques. After performing the cluster ranking, then it is transformed to the passage ranking. Their approach utilized the approach if *ClustMRF* [26] for feature extraction and they also introduced some novel feature to rank the clusters.

2.3 Passage Retrieval for Question Answer (QA)

Nowadays Question Answering (QA) sites are very user friendly and capable of processing natural language. Users normally ask a question in natural language to the QA site and then the site provides a single or multiple answers that are expected to the users. In various aspects, retrieving the appropriate answers corresponding to a question is very different from traditional document retrieval approaches that only retrieve the full length document not the relevant portion or passage that actually contains the actual answers. These approaches only process the query terms ignoring the surrounding text that may be helpful for locating the answers.

Many IR based research works have been done to develop IR model for QA sites and to improve the performance of existing models for solving passage retrieval problem. For example, Jian et al. [10] developed a Hot-Spot passages retrieval approach based on the classic BM25 model focusing on retrieving answers from the QA systems. They used a variation of this model called blurred BM25. This approach also included height and coverage measurements that improve its effectiveness. Later Jose et al. [8] proposed a language independent based passage retrieval technique for finding pertinent answer from the QA systems. Initially they followed the traditional VS model with the variation of *n-gram* extraction from the question and the answer collections. They used the *n-gram* for measuring the similarity between the question and answer. The approach is applied on various language QA systems and promising results were found as it improved the performance for multilingual QA systems. Further, Davide et al. [4] also adopted the *n-gram* based approach to build a passage retrieval engine for QA. However, they split the whole approach in various phases where at first, pattern based classification and supervised classification were used to improve the effectiveness of answer retrievals. In next phase, they used *n-gram* based *Clustered Keyword Positional Distance* model to measure the similarity among the questions and answers. The approach is applied on the dataset of five languages. It performs better compared to contemporary simpler keywords matching model.

Graph based models are also developed for retrieving passage in QA systems. For example, Matthew et al. [3] proposed a rank-learning technique where they integrated the linguistic and semantic constraint checking during passage retrieval. They adopted the decomposition approach to view the question as a graph into atomic constrain so that is can be matched with the candidate answers. Another graph based passage retrieval approach is developed by Xin Li et al. [19]. Firstly, they expanded the question utilizing a KNN based method. The candidate passage are identified based on this expanded question model. After that, various features are extracted based on the content of the passages. These features are used to build a *passage-to-passage* graph model considering the

initial passage retrieval results. And then based on the graph based ranking method the initial ranked passages are re-ranked.

Giving emphasis on the question and answer syntactic structure, Elif et al. [2] proposed a technique for passage re-ranking in QA. The syntactic structure of the answers are extracted and used with respect to the candidate answer terms where these terms are identified by the question answer types. Thus, it creates the dependency between the questions and candidate answer passages. In some cases it outperformed compared to the traditional approaches in the case where the syntactic structure information extraction is possible. Similarly, Mahboob et al. [14] performed comparative study for evaluating the performance of *tf-idf* and *Okapi* retrieval models for QA task. However, their main focus was to promote sliding window based passage retrieval techniques that works better compared to those classical models. They evaluate the sliding window based technique on two benchmark dataset and found that compared to the disjoint windows, utilizing sliding windows expose significant improved performance.

Apart from these, many well-performed algorithms are found in the literature that are considered to be effective in passage retrieval. Considering the page limitation, a brief summary of those algorithms are represented in **Table 1**.

3 TEST STATISTICS

This section reports the explanation of how I have completed the *Subtask-2* (Text-Statistics). Initially, I discuss an overview of various text prepossessing approaches and the explanation of why this text processing is required. Next, I briefly discuss the Zipf's law and provide the steps that are performed on the passage collection to generate Zip's Law pattern and parameters.

3.1 Text Prepossessing

Various types of text processing are needed before analyzing the text. A brief description of each of these processing is as follow:

Tokenization: The first step of text processing is tokenization, which is a process of splitting a longer string into small chunks or tokens. For example, large text can be separated into chunks of sentences. And then sentences can be split into the token of words. This process is also known as text segmentation. In this coursework, the provided dataset contains a collection of passages where each passage is represented by a single line string. Therefore, I split the each passage by space to transform it into a list of tokens.

Lowercasing: After performing tokenization, next step is lowercasing. It is the most simple but effective phase of text processing. In this phase, each token is converted into its lowercase form. It is performed to make the tokens consistent for the whole dataset during analyzing.

Stop-word Removal: Stop-words refers to a set of common words that are used in a language frequently. For instance, in English language articles (e.g; *"a","an", "the"*) and prepositions (e,g;*"of","in","at"*) are considered to be stop-words. The motivation behind stop-word removal is to put emphasize on important words instead of analyzing low information words. Now in this phase, from each token list the stops words are removed. For this purpose, I utilized a list of stop words from *Reuters-RCV1* [18].

Stemming: Because of grammatical usage, words may appear in a text in different forms, for example, *write, writes, writing, written*. Although the semantic meaning of this word would be similar, however, different forms represent distinguished contexts. Therefore, while analyzing the text it will be helpful to have all the words in its base form. To do so, stemming is used. Stemming is the process of transforming a word into its base form by reducing the inflected words from it. For instance, if we perform stemming for the words *eat, eating*, we will get only the base form *eat* for any form of this word. Stemming is performed based on various suffix rules. For this project, I used a well known publicly available *Porter Stemmer* [25] to transform each token into its base form. During stemming, *Porter Stemmer* performs five sequential phase for words reduction where each phase contains multiple suffix rules to covet the word into its base form. Thus after stemming, a list of word tokens are generated that contain only the base form of each word.

3.2 Zipf's Law

Zipf,s law is most commonly used distribution model that helps to analyze how words are distributed across the collections of documents. It states that the frequency of the rth most common word is inversely proposition to its rank (r_w) [6]. Therefore, for a word within the collection, the product of its rank (r_w) and its frequency (f_w) is approximately a constant (k).

$$r.f = k \tag{1}$$

We consider the probability of a word within a collection that refers the ratio of the number of times a word appears in the collection and the total number of words in the collection, then the product of (r_w) and the probability of the word (Pr_w) would be a constant c. For English language, the constant c is approximately 0.1 [6].

$$r.P_r = c \tag{2}$$

Verifying Zipf's Law Pattern After accomplishing the text processing [1], for the given passage list I generated a collection of word list. To analyze how the words are distributed in this collection, I identified the constant c deriving from the Zipf's law. The following steps were performed to identify this constant for the given passage collection.

(1) Counted the total number of words ($\#W$) in the collection.
(2) Created a list of all unique words and count frequency (f_w).
(3) Ranked (r_w) the unique words by its frequency (f_w).
(4) For each word,
   (a) Calculated the probability (Pr_w) for the word (f_w / $\#W$)
   (b) Identified the constant c by multiplying (r_w)* (Pr_w).
(5) Plot the Rank(r_w) vs (Pr_w) in a chart.

After performing the aforementioned steps in the provided passage collection, the total number of words, frequencies, ranks, probabilities and the Zipf's constant are found. For this passage collection, the value of the constant c is approximately between *0.062 to 0.135*. However, for top 100 words the average values of the constant c is approximately 0.092. In order to verify this result, whether it follows the actual Zipf's law, a line is drawn according to the Equation 2 with assuming *c=0.1*. In Figure 1, the *blue line* in the chart shows the pattern of actual Zipf's law for top 100 Words. It clearly represents how the frequency of word occurrence falls rapidly after the most common words. Next I plot the rank vs probability of

[1]During Zipf's Law pattern generation, *Stop-words Removal* is not performed as Stop-words contain high frequency

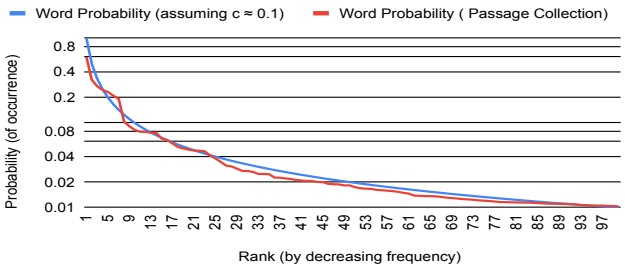

Figure 1: Zipf's Law Pattern for Passage Collection.

top 100 words derived from passage collection. The *red line* in the chart shows the pattern that derived from the passage collection. It is observed that the derived pattern follows the actual Zipf's law pattern.

4 CONSTRUCTING INVERTED INDEX

In this section, I discussed how I have implemented the inverted index for the given passage collection in *Subtask-3* (Inverted Index). It also includes the description of the additional information that I stored in the inverted index for future usage.

4.1 Inverted Index:

It is similar to Hash-Map data-structure where a particular entry directs to a value. However, an inverted index is a data structure that organizes and maps contents such as words/terms and numbers to a corresponding document sets or locations in a document. It can be record-level inverted index where each entry term is linked to a set of reference documents. Or it can be word-level inverted index that additionally stores the positions of each term in a document.

In this project, the inverted index that is developed will store term as a entry and it will directs a set of passages that are identified by a passage id (pid). The following information is stored for each indexed passage.

- **Passage Identifier (*pid*):** Unique identifier of a passage.
- **Frequency (*f*):** occurrence of the term in the passage.
- **Length (*l*):** Total number of terms in the passage.
- **Positions (*pos*):** Locations of the entry term.
- **Passage Frequency (p_f):** number of passage which contain the term (length of passage list related to that term).

I stored these additional information in the inverted index because it will help me for improving query time efficiency and implementing classical ranked retrieval models. For example, *Passage ID(pid)* helps for efficient query processing. The *Frequency* and *Length* can be utilized to measure the t_f score, and Passage Frequency can be useful for calculating idf score.

4.2 Inverted Index Implementation

The following steps provide a description of my inverted index implementation:

Step 1: Take the collection of passage as input
Step 2: For each passage,
  (a) Produce a token list by performing text prepossessing (e,g; Tokenization, Lowercasing, Stop-word Removal and Stemming).
  (b) Within the token list, count the number of unique terms, their frequencies, and identify the locations of each unique term.
  (c) Each unique term is stored as an entry key in a HashMap and an object named *CandidatePassage* that contains all additional information (e.g; pid, f, l, pos and p_f) is stored as the corresponding value of that entry.
  (d) Eventually, List<HashMap<*term*, *CandidatePassage*» is produced for all passages.
Step 3: A global *HashMap<term, List<CandidatePassage»* is declared as the actual inverted index
Step 4: For each entry of List<HashMap<*term*, *CandidatePassage*»
  (a) When a *term* is encountered for the first time, it is inserted to the global HashMap as the entry key and the *CandidatePassage* is added to the list $List < CandidatePassage >$ of the corresponding term in the global HashMap.
  (b) If it is found that the *term* is already included into the global HashMap, then only the *CandidatePassage* is added to the list $List < CandidatePassage >$ of the corresponding term in the global HashMap.
Step 5: Output the global map as the Inverted Index.

In the following section, this inverted index is used to implement basic information retrieval models.

5 BASIC RETRIEVAL MODELS

This section deals with the Subtask-4 (Basic Retrieval Models). It contains an exhaustive description of how I have implemented two basic IR models such as Vector Space and BM25 model.

5.1 Vector Space Model

Vector Space (VS) model is one of the most well-known IR models. In this model, documents and queries are considered as a *t-dimensional* vector where t is the number of terms. And each t_i dimension contains a weight of its $t_i th$ term. Typically, it contains the *tf-idf* weight [22] and the similarity between the documents and the queries is calculated by using consine-similarity function (e.g; vector dot multiplication).

Algorithm 1 represents the procedure how I have implemented the VS model. This algorithm performs the following steps:

Algorithm 1: Vector Space Model

Input : Query ID (*qid*), Query (*Q*), Collection of Passages (*P*)
Output: Top Ranked Passage

```
begin
    index ⟵ buildInvertedIndex(P);
    CP ⟵ ∅;
    foreach term ∈ Q do
        IP ⟵ CP ∪ getCandidatePassage(term, index);
    end
    PV ⟵ buildPassageVectors(CP, index);
    qv ⟵ buildQueryVector(Q, index);
    RP ⟵ ∅;
    foreach pv ∈ PV do
        score ⟵ calculateCosineSimilarity(pv, qv);
        RP ⟵ RP ∪ createRankedPassage(qv.qid, pv.pid, score);
    end
    sortPassageByScore(RP);
    return RP
end
```

Step 1: Take Query ID (*qid*), Query (Q) and Collection of Passages (P) corresponding to that Q as input.
Step 2: Construct Inverted Index (*index*) by invoking a function $buildInvertedIndex(P)$. This function takes P as input and produced inverted index followings the approach described in Section 4.
Step 3: Identify all terms from Q by performing text prepossessing. For each $term$ in Q, retrieve passages from the *index* and store them in a *Candidate Passages Set (CP)*. The function *getCandidatePassage(term,index)* helps to retrieve these passages by utilizing the *index*.
Step 4: Invoke the *buildPassageVectors(CP,index)* function that takes *index* and *CP* as input and produces a *Passage Vector List (PV)*. This function performs the following operations: for each candidate passage *cp* from *CP*.
(a) Get the passage *(p)* by *pid* from *P*.
(b) Tokenize the passage by performing text prepossessing.
(c) Now, for each unique term from the token, calculate the *term frequency* (tf), *Inverse Document Frequency* (idf) and ($tf * idf$) score. Many variant of *tf-idf* functions presented in the literature for calculating these scores. However, in this implementation, I used the *logarithm* based functions according to Equation 3, 4 ,5 as it helps to handle mathematical errors while calculating these scores [22]. In Equation 3, (tf_t, p) expresses term frequency of term t in passage p. And N represents the total number of passages in the collection, pf_t refers number of passages contain term t.

$$tf = 1 + log(tf_t, p) \tag{3}$$

$$idf = log\frac{N}{pf_t} \tag{4}$$

$$tf * idf = 1 + log(tf_t, p) \cdot log\frac{N}{pf_t} \tag{5}$$

(d) Now, store the unique word as an entry key of a map and the ($tf * idf$) score is stored as the value of that entry, for example, *Map<term,score>*
(e) This *Map<term,score>* represents a term-score vector corresponding to the *cp*.
(f) Finally for each candidate passage (cp), a Passage Vector (pv) is created.
Step 5: The Query Vector (qv) is created by invoking the function *buildQueryVector(Q,index)*. This function performs the similar operation as it is described how the passage vectors are built.
Step 6: After that with the The Query Vector (qv), for each Passage Vector (pv) from the Passage Vector List (PV)
(a) Calculate cosine-similarity score of pv and qv by performing vector dot multiplication based on Equation 6.

$$\cos(pv.qv) = \frac{pv * qv}{\|pv\| \cdot \|qv\|} = \frac{\sum_{i=1}^{n} pv_i \cdot qv_i}{\sqrt{\sum_{i=1}^{n} (pv_i)^2}\sqrt{\sum_{i=1}^{n} (qv_i)^2}} \tag{6}$$

(b) Create an object with this score including *pid*, *qid* and store the object in a *Raked Passage Set (RP)*.
Step 7: Sort the *RP* by the value of score in decreasing order.
Step 8: Finally, return the top 100 ranked passage from *RP*.

5.2 BM25 Model

BM25 is a probabilistic IR ranking model that extends the scoring function for the binary independence model [6]. Algorithm 2 represents the procedure of how I have implemented the BM25 model. The initial implementation of BM25 is almost similar to the VS model, however, the major difference is calculating the similarity score between the passages and the query. The following steps are performed during the implementation of this model.

Step 1: Accepting the input such as Query ID (qid), Query (Q) and the Collection of passage *(P)* related to that Q.
Step 2: Create the Inverted Index (index) by feeding P to the function *buildInvertedIndex(P)*. This function adopts the approach describe in Section 4.
Step 3: Perform text prepossessing on Q and identify all the unique terms. Then for each *term* in Q, retrieve the *Candidate Passage Set (CP)* from the *index* by invoking the function *getCandidatePassage(term,index)*.
Step 4: For each *Candidate Passage (cp)* from the set *CP*, the similarity score is calculated between the Q and *cp* according to the equation 7.
(a) Equation 7 is used as the scoring function of BM25. Many variations of this scoring function exist in the literature however, the form of Equation 7 is the most simplest one. It is developed based on probabilistic arguments and experimental validation.
(b) Here the summation represents that the score is individually calculated over all terms in the Q and then merge into a single score. Moreover, the equation contains various parameters and the description of these parameters is mentioned below:

$$\text{score}(D, Q) = \sum_{i \in Q} log\frac{(r_i + 0.5)/(R - r_i + 0.5)}{(n_i - r_i + 0.5)/(N - n_i - R + r_i + 0.5)} \cdot \frac{(k_1 + 1)f_i}{K + f_i} \cdot \frac{(k_2 + 1)qf_i}{k2 + qf_i} \tag{7}$$

(i) r_i= number of relevant passages contain term i.
(ii) R=number of relevant passages for this Q.
(iii) n_i= number of passages where the term i appear.
(iv) N=number of passages in the collection.
(v) qf_i= frequency of term i in Q
(vi) f_i= frequency of term i in the passage.
(vii) K, k_1 and k_2 are parameters whose values are set empirically.
(c) Since prior relevancy information is not available, for this implementation, therefore, I set the value of *R=0* and r_i=0.
(d) Here, constant k_1 determines how the term frequency weight changes as f_i increase. I used the value k_1*=1.2* as in text retrieval experiments this is considered to be the typical value. [13][22]. Similarly, constant k_2 determine how the query term weight fluctuates when qf_i increase. The value for k_2 typically ranges from 0 to 1000. Here we use the value k_1*=100* [13][22].
(e) Now K is more a complex parameter that will normalize the term frequency weight by the passage length. The value of K is calculated according to the Equation 8.

$$K = k_1((1 - b) + b \cdot \frac{|dl|}{\text{avgdl}}) \tag{8}$$

(i) dl= length of the passage.
(ii) $avgdl$=average length of a passage in the collection.
(iii) b= a parameter that control the impact of the length normalization. If b is set to 0, it refers no length normalization. And when b is set to 1, it means full length normalization. In this implementation, I set b=0.75 as it is considered to be effective in typical text retrieval experiment [13][22].

Step 5: After calculating the score, keep this score including *pid*, *qid* in an object. And store this object in a *Raked Passage Set (RP)*.

Step 6: Sort the *RP* by the value of score in decreasing order.

Step 7: And return the top 100 ranked passage from *RP*.

Algorithm 2: Okapi BM25

```
Input  : Query ID (qid), Query (Q), Collection of Passages (P)
Output : Top Ranked Passage
begin
    index ⟵ buildInvertedIndex(P);
    CP ⟵ ∅;
    foreach term ∈ Q do
        CP ⟵ CP ∪ getCandidatePassage(term, index);
    end
    RP ⟵ ∅;
    foreach cp ∈ CP do
        score ⟵ calculateScoreBM25(cp, Q);
        RP ⟵ RP ∪ createRankedPassage(qid, pid, score);
    end
    sortPassageByScore(RP);
    return RP
end
```

REFERENCES

[1] Martin Franz Abraham Ittycheriah and Salim Roukos. 2001. IBM's statistical question answering system-TREC-10. In *AUTHOR Voorhees, Ellen M., Ed.; Harman, Donna K., Ed. TITLE The Text REtrieval Conference (TREC-2001)(10th, Gaithersburg, Maryland, November 13-16, 2001). NIST Special*, Vol. 500. Citeseer, 316.

[2] Elif Aktolga, James Allan, and David A. Smith. 2011. Passage Reranking for Question Answering Using Syntactic Structures and Answer Types. In *Proceedings of the 33rd European Conference on Advances in Information Retrieval (ECIR'11)*. Springer-Verlag, Berlin, Heidelberg, 617–628.

[3] Matthew W. Bilotti, Jonathan Elsas, Jaime Carbonell, and Eric Nyberg. 2010. Rank Learning for Factoid Question Answering with Linguistic and Semantic Constraints. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Management (CIKM '10)*. Association for Computing Machinery, New York, NY, USA, 459–468.

[4] Davide Buscaldi, Paolo Rosso, José Manuel Gómez-Soriano, and Emilio Sanchis. 2010. Answering Questions with an N-Gram Based Passage Retrieval Engine. *J. Intell. Inf. Syst.* 34, 2 (April 2010), 113–134.

[5] Charles LA Clarke, Gordon V Cormack, Thomas R Lynam, and Egidio L Terra. 2008. Question answering by passage selection. In *Advances in Open Domain Question Answering*. Springer, 259–283.

[6] W Bruce Croft, Donald Metzler, and Trevor Strohman. 2010. *Search engines: Information retrieval in practice*. Vol. 520. Addison-Wesley Reading.

[7] Ludovic Denoyer, Hugo Zaragoza, and Patrick Gallinari. 2001. HMM-based passage models for document classification and ranking. In *Proceedings of ECIR-01, 23rd European Colloquium on Information Retrieval Research*. 126–135.

[8] José Manuel Gómez-Soriano, Manuel Montes-y Gómez, Emilio Sanchis-Arnal, Luis Villaseñor Pineda, and Paolo Rosso. 2005. Language Independent Passage Retrieval for Question Answering. In *Proceedings of the 4th Mexican International Conference on Advances in Artificial Intelligence (MICAI'05)*. Springer-Verlag, Berlin, Heidelberg, 816–823.

[9] Eduard H. Hovy, Ulf Hermjakob, and Chin-Yew Lin. 2001. The Use of External Knowledge of Factoid QA. In *Proceedings of The Tenth Text REtrieval Conference, TREC 2001, Gaithersburg, Maryland, USA, November 13-16, 2001*, Ellen M. Voorhees and Donna K. Harman (Eds.), Vol. Special Publication 500-250. National Institute of Standards and Technology (NIST).

[10] Jian Huang, Xuanjing Huang, and Lide Wu. 2004. Hot-Spot Passage Retrieval in Question Answering. In *Proceedings of the 7th International Conference on Digital Libraries: International Collaboration and Cross-Fertilization (ICADL'04)*. Springer-Verlag, Berlin, Heidelberg, 483–490.

[11] Jing Jiang and ChengXiang Zhai. 2005. Accurately Extracting Coherent Relevant Passages Using Hidden Markov Models. In *Proceedings of the 14th ACM International Conference on Information and Knowledge Management (CIKM '05)*. Association for Computing Machinery, New York, NY, USA, 289–290.

[12] Jing Jiang and Chengxiang Zhai. 2006. Extraction of Coherent Relevant Passages Using Hidden Markov Models. *ACM Trans. Inf. Syst.* 24, 3 (July 2006), 295–319.

[13] K. Sparck Jones, S. Walker, and S. E. Robertson. 2000. A Probabilistic Model of Information Retrieval: Development and Comparative Experiments Part 2. *Inf. Process. Manage.* 36, 6 (Nov. 2000), 809–840.

[14] Mahboob Alam Khalid and Suzan Verberne. 2008. Passage Retrieval for Question Answering Using Sliding Windows. In *Coling 2008: Proceedings of the 2nd Workshop on Information Retrieval for Question Answering (IRQA '08)*. Association for Computational Linguistics, USA, 26–33.

[15] Oren Kurland and Carmel Domshlak. 2008. A Rank-Aggregation Approach to Searching for Optimal Query-Specific Clusters. In *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '08)*. Association for Computing Machinery, New York, NY, USA, 547–554.

[16] Oren Kurland and Eyal Krikon. 2011. The Opposite of Smoothing: A Language Model Approach to Ranking Query-Specific Document Clusters. *J. Artif. Intell. Res.* 41 (2011), 367–395.

[17] Gary Geunbae Lee, Jungyun Seo, Seungwoo Lee, Hanmin Jung, Bong-Hyun Cho, Changki Lee, Byung-Kwan Kwak, Jeongwon Cha, Dongseok Kim, Joohui An, Harksoo Kim, and Kyungsun Kim. 2001. SiteQ: Engineering High Performance QA System Using Lexico-Semantic Pattern Matching and Shallow NLP. In *Proceedings of The Tenth Text REtrieval Conference, TREC 2001, Gaithersburg, Maryland, USA, November 13-16, 2001*, Ellen M. Voorhees and Donna K. Harman (Eds.), Vol. Special Publication 500-250. National Institute of Standards and Technology (NIST).

[18] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *J. Mach. Learn. Res.* 5 (Dec. 2004), 361–397.

[19] Xin Li and Enhong Chen. 2010. Graph-Based Answer Passage Ranking for Question Answering. In *Proceedings of the 2010 International Conference on Computational Intelligence and Security (CIS '10)*. IEEE Computer Society, USA, 634–638.

[20] Marc Light, Gideon S. Mann, Ellen Riloff, and Eric Breck. 2001. Analyses for Elucidating Current Question Answering Technology. *Nat. Lang. Eng.* 7, 4 (Dec. 2001), 325–342. https://doi.org/10.1017/S1351324901002819

[21] Fernando Llopis and José Luis Vicedo González. 2001. IR-n: A Passage Retrieval System at CLEF-2001. In *Revised Papers from the Second Workshop of the Cross-Language Evaluation Forum on Evaluation of Cross-Language Information Retrieval Systems (CLEF '01)*. Springer-Verlag, Berlin, Heidelberg, 244–252.

[22] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to Information Retrieval*. Cambridge University Press, USA.

[23] David R. H. Miller, Tim Leek, and Richard M. Schwartz. 1998. BBN at TREC7: Using Hidden Markov Models for Information Retrieval. In *Proceedings of The Seventh Text REtrieval Conference, TREC 1998, Gaithersburg, Maryland, USA, November 9-11, 1998*, Vol. Special Publication 500-242. National Institute of Standards and Technology (NIST), 80–89.

[24] Elke Mittendorf and Peter Schäuble. 1994. Document and Passage Retrieval Based on Hidden Markov Models. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '94)*. Springer-Verlag, Berlin, Heidelberg, 318–327.

[25] M. F. Porter. 1997. *An Algorithm for Suffix Stripping*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 313–316.

[26] Fiana Raiber and Oren Kurland. 2013. Ranking Document Clusters Using Markov Random Fields. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '13)*. Association for Computing Machinery, New York, NY, USA, 333–342.

[27] Jangwon Seo and W. Bruce Croft. 2010. Geometric Representations for Multiple Documents. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '10)*. Association for Computing Machinery, New York, NY, USA, 251–258.

[28] Eilon Sheetrit and Oren Kurland. 2019. Cluster-Based Focused Retrieval. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management (CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 2305–2308.

[29] Stefanie Tellex, Boris Katz, Jimmy Lin, Aaron Fernandes, and Gregory Marton. 2003. Quantitative Evaluation of Passage Retrieval Algorithms for Question Answering. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval (SIGIR '03)*. Association for Computing Machinery, New York, NY, USA, 41–47.

[30] Ellen M. Voorhees. 1985. The Cluster Hypothesis Revisited. In *Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '85)*. Association for Computing Machinery, New York, NY, USA, 188–196.