# IRDM Course Project Part I: Ranked Passage Retrieval

Student Number: 16034489

## ABSTRACT

Information retrieval models play a vital part in many applications, from search and question answering to recommendation. In *Ad-hoc* passage retrieval a user generates a query and the system evaluates a collection of short texts (passages), returning them in an order of decreasing relevance. This work first critically explores some of the recent developments in passage-ranking models. Next, it verifies Zipf's law for a collection, builds an inverted index and lastly ranks candidate passages for each query using the Vector Space Model and BM25, both of which have been implemented manually.

## 1 LITERATURE REVIEW

One of the more successful traditional approaches for Information Retrieval (IR) utilises *probabilistic language modelling*, where the main goal is to define a probability distribution over the different linguistic units, such as words, sentences etc., thus constructing a language model $\theta_d$ associated with each document $d$. In this setting, documents are ranked according to the likelihood of a given query being generated under the probability distribution of estimated $\theta_d$. Formally, this is captured in the score assigned to a given document $d$ and query $q$ pairs: $\text{Score}(d,q) = P(q|\theta_d)$. Thus central to this method is choice of method for estimating $\theta_d$. However, this approach relies on finding an exact match between query and document terms but different words may be used in the query and document to express a similar meaning, or the same words may be used but in reference to different meanings. This lack of semantic comprehension leads to 'vocabulary gaps'. A recent approach by Ensan et al. proposes a Semantics-Enabled Language Model (SELM) - a probabilistic language model which represents documents and queries as a set of semantic concepts, and measures the relatedness of the query and a document based on the relatedness of their concepts. This is achieved through running the documents and queries through an entity linking system, and the concepts produced by the system then correspond to entities in the semantic network. Specifically, the document is modelled as an undirected graph, where nodes correspond to concepts and an edge between two nodes denotes relatedness between the concepts. To rank documents, the model computes the conditional probability of the query concepts given all the concepts present in the document and their relatedness relationships. The semantic insight offered by this model has its advantages as SELM has been able to find documents which keyword-based models failed to retrieve. Moreover, with the growing interest in semantic annotation and knowledge bases, there is a great scope for further developing these models further in the future. However, SELM struggled in cases where the *query term independence assumption* did not hold and when queries were annotated with a single, broad concept. Moreover, occasionally it failed to retrieve relevant documents which keyword-based models succeeded in finding. Hence, the authors advocate for combining SELM with various keyword-based models, such as SDM [10] or RM3 to exploit the synergy of these algorithms and thus improve the overall performance.

Passage-ranking is not only useful as a stand-alone task, but also in the context of document-ranking. One of the key challenges in document-ranking is computing a single relevance score for a document that is long and/or contains heterogeneous passages, especially as "often, documents are judged as relevant even if they contain only a short passage (segment) with pertaining information" [19]. To address this, ranking short text sections (passages), rather than entire documents, have been gaining traction. Sheetrit et al. proposed a novel idea of utilising inter-passage similarities in this setting by exploring the cluster hypothesis: *'closely associated documents tend to be relevant to the same requests'*. [17]. Recently, he formalised his approach [18] - firstly, clusters of similar passages are obtained by computing the geometric mean between passage vectors and using the kNN algorithm. Next, a learning-to-rank approach is used to rank the passage clusters by relevance, where relevance of a cluster is the fraction of relevant text in the passages, or simply the fraction of relevant passages. Lastly, the cluster ranks are converted to passage ranks. At the time this approach outperformed other feature-based passage-ranking models which do not utilise inter-passage similarities. Moreover, it allows for incorporating relevance priors into the model, which may be very useful in some settings. Nonetheless, being a learning-to-rank model it relies on hand-engineered features to compute the document-query similarities and on exact matches between document and query terms. Moreover, this model may suffer from lack of result diversification, since if top-ranked passages originate from the same cluster, they are likely to share a lot of similarities and not give a broader overview. This could be addressed through, for example, incorporating latent topic models as Liang et al. have done [6]. Also, Sheetrit et al. makes strong claims about neural-based models performing significantly worse than feature-based models, but he refers to papers written in 2015-2016, which does not make for a fair comparison.

Sheetrit hasn't been alone sharing his skepticism for deep learning (DL) based models. Lin [7] criticised the 'neural hype' for claiming to achieve state-of-the art results while comparing with weak baselines, such as BM25. Recently, Yang et al. went on to empirically examine some of the claims made by Lin. Their rigorous investigation of the alleged results on the TREC Robust04 document test collection revealed that "the best reported results are from a decade ago and no recent neural approach comes close". They also performed an experiment were they used 5 recent neural models to re-rank lists generated by strong baselines and found that only one of them - DRMM [4] showed significant improvement.

However, there is a good reason for why development of state-of-the-art DL models in IR has been relatively slow in comparison with other fields. This is largely due to lack of sufficiently-large datasets, which are necessary for fine-tunning of millions of parameters in DL-models. However, the release of datasets such as MS Marco [1] in 2016 has changed the game for neural-based models. In the recent months the top of the leaderboard for both MS Marco and TREC Robust04 has been dominated with various adaptations of BERT. [3], [14] BERT (Bidirectional Encoder Representations from Transformers) utilises bidirectional self-attention to learn the interaction between the input terms. It is pre-trained using a masked modelling task and only requires fine-tuning a single additional output layer to adapt it for passage-ranking. [3] At the time, this simple implementation significantly outperformed both, existing traditional models and neural ranking-based models. One of the big advantages of neural-based IR models is that, unlike traditional models, they typically take raw query and text as input in the form of one-hot encodings and learn a representation, including latent representation, which is most effective for the given matching task. Yet, while BERT's variants continue achieving state-of-the-art results, it is not a faultless model. An in-depth analysis by Padigela et al. [15] comparing BERT with BM25 found that BERT struggled to capture query context for longer queries and performed relatively badly on numerical and entity-type questions. However, it excelled at retrieving passages with more new words and answering *abbreviation*-type questions.

Another interesting model which utilises attention is AQuPR [16]. It performs reading comprehension on passages to obtain a string span which is likely to hold the answer through training a character-level embedding for the query-passage pairs using a bi-LSTM. Then an RNN is trained, making use of a new, simplified attention mechanism to assign weights to query terms with respect to the passage and vice versa. The outputs of both attention layers are concatenated, passed through a cascade-forward NN and the scalar outputs provide a rank for each document. Heavily relying on embeddings, this model usually succeeds in capturing semantic similarities between entities, addressing the perennial vocabulary-gap problem that traditional learning-to-rank models struggle with. However, it struggles when queries make little reference to nouns, making it hard for the biLSTM to learn query syntax. Also, here it has been applied to a very niche domain (technical queries) - its success in a more general setting is yet to be tested.

Both learning-to-rank and neural-based models have their distinct strengths - can one combine them? This is precisely what Duet [12] attempts to do, bringing together representation learning and using exact query-document pattern matches for feature learning . Recently, it has received several updates [11], such as introducing word embeddings to reduce training time, thus allowing training on larger datasets under time constrains, weighting a previously-binary interaction matrix with IDF, non-linearly combining local and distributed and models, replacing Tanh activation functions with ReLU and incorporating *bagging* ie. training multiple models on different random seeds with different samples of training data. These enhancements allow Duet to achieve results comparable with other non-BERT-based top performing models. One of the big advantages of Duet over Bert and CKNRM [2] models is the smaller number of learnable paramteres, which for Duet is only around 33 million, while the latter two require training of a few hundred million parameters. This makes Duet much faster to train, which is important when under a time-constraint.

Interaction-based models have been gaining traction recently, and of particular interest is CKNRM [2], which not only extends the KNRM [20] model by incorporating interactions (computed as cosine similarity between vectors) between *n-grams*, as opposed to just unigrams, but it also allowing cross-matching of different length n-grams, hence relating concepts which may have a different number of terms. This is achieved through first computing *n-gram* embeddings using convolution filters and pre-trained unigram embeddings vectors. Next a set of interaction matrices $M^{r,t}$ is constructed, where $1 \leq r, t \leq h$, $h$ being the maximum *n-gram*. Each interaction matrix stores the cosine similarity between the query term vector, represented using r-gram embeddings and the document term vector, represented using t-gram embeddings. Furthermore, CKNRM allows for soft-matching by deploying a kernel pooling operation, and when trained on one domain, it was found to generalise to another domain much better than any of the strong learning-to-rank baselines.

However, one problem faced by all neural-based algorithms lies in their computational expense which is too great for retrieval problems with large collections, hence their use is usually limited to late-stage re-ranking. Several attempts have recently been made at addressing this.

Ji et al. [5]. propose a scheme for improving computational efficiency of 3 interaction neural-based models: DRMM [4], KNRM [20] and CKNRM [2]. In interaction-based models, each interaction matrix is computed by taking the dot product for every matrix element. Supposing a document with $n$ terms and a query with $m$ terms, the time complexity to compute cosine scores is $\Theta(nml)$ for DRMM and KNRM and $\Theta(nmF)$ for CKNRM, where F is the number of convolution filters used to compute the embeddings. Ji et al. [5] propose accelerating this computation through firstly approximating the kernel vectors using LSH and replacing cosine similarity with the hamming distance. LSH (Locality-sensitive hashing) works by hashing similar inputs into 'buckets' with high probability. Next, they implemented a fast histogram-based kernel computation by observing that the number of distinct similarity values is reduced when using the hamming distance. Also, while for DRMM and KNRM the size of embeddings of the term vectors is determined by the pre-trained word embeddings, for CKNRM these embeddings are generated using $F$ convolution filters on $n$-gram vectors and this computation can be very time consuming, hence Ji et al. advocate for pre-computing these. The downside of this scheme is the significant storage requirement for pre-computing the embeddings and the implementation as a whole is very model specific, thus difficult to apply to other models. However, these accelerations lead to no reduction in the performance of the models.

A method which can be generalised more easily was proposed by Mitra et al. [13], who note that one of the ways to drastically increase efficiency is via pre-computing term-document scores offline and storing them in a convenient data structure, such as inverted index. However, this is only possible for classical IR models which make the query term independence assumption, meanwhile state-of-the-art DL models usually utilise a complex matching function which uses the full query, and involve evaluation of millions of parameters. Mitra et al. counter this by introducing the query term independence assumption into 3 DL models: BERT [3], Duet [11] and CKNRM [2]. One would expect for the performance to decrease due to information loss yet that was only the case with BERT, while for other two models there was no significant performance drop. The performance of BERT can be restored at a later stage through iteratively re-ranking top scoring documents using the approach by Matveeva et al. [9]. While in general one may expect to observe some performance drop, the trade-off between efficiency and performance offered by this work opens new doors in neural IR tasks, providing for the first time a tractable scheme for deploying DL models for retrieval tasks and not just late-stage re-ranking.

## 2 TEXT STATISTICS

### 2.1 Text Pre-processing

In order to compute text statistics, it is necessary to pre-process the text. Firstly, all non-alphabetic characters are removed apart from the apostrophe using regex and replaced with a space. Next, the `word_tokenize` function from the `nltk` library is implemented, which splits a string into a list of individual tokens. All the tokens are made lowercase to allow comparison. Also, all of the stop words such as 'and', 'or', 'the' etc. have been removed to reduce computational cost and increase effectiveness. To achieve this, the stop words list from the `nltk` library was imported. The order of these operations is important because to remove the stop words the algorithm compares the passage tokens with the stop tokens, hence both must be lowercase, without punctuation etc. The apostrophe was retained since some of the stop words include it, such as 'weren't' etc. Next, all of the words are reduced to their stems. Stemming can be thought of as a normalisation technique - it reduces words such as 'clapping' and 'clapped' to their underlying stem, in this case to 'clap'. This is necessary since these two words point to the same meaning, but this will not be recognized otherwise. Here, *Porter Stemmer* was implemented - a popular stemming algorithm which comprises of a set of rules that identify long suffixes and reduce them to their stem. Lastly, only tokens of length 3 or more are retained.

As we wish to study the statistics for the entire collection, all of the passages are combined into a single list and the frequency of each unique term is computed.

### 2.2 Zipf's Law

To study the distribution of the terms in the passages we utilise *Zipf's Law*, which states that the collection frequency, $cf_r$ of a term is inversely proportional to its rank $r$ in the entire collection, as shown in equation 1, where $K$ is a proportionality constant.

$$cf_r \approx \frac{K}{r} \tag{1}$$

Another interpretation of Zipf's Law converts the collection term frequency into a probability of word occurrence - hence it can be stated that the product of term's occurrence probability and its rank is approximately constant, as seen in equation 2, where $C$ is the adjusted proportionality constant.

$$r \cdot P_r \approx C \tag{2}$$

For the rest of the question we will work with equation 2. First, the ideal Zipf's Law was computed for this collection, computing the inverse of each term's rank. To obtain the proportionality constant, the product of all terms and their normalised frequencies, which approximate the probability of word occurrence, $P_r$, was computed and the mean taken, as shown in equation 3.

$$C = \frac{1}{n}\sum_{r=1}^{n} P_r \cdot r \tag{3}$$

.

The empirical value of the proportionality constant computed was $C = 0.03 \pm 0.04$. In the English language $C$ usually lies around 0.1, which, while it is outside of the standard deviation of our estimate, is relatively similar.

Next, to analyse the adherence of this dataset to Zipf's Law, the observed and model term frequencies were plotted against the observed ranks. A logarithmic scale was used to facilitate comparison, as shown in Figure 1.

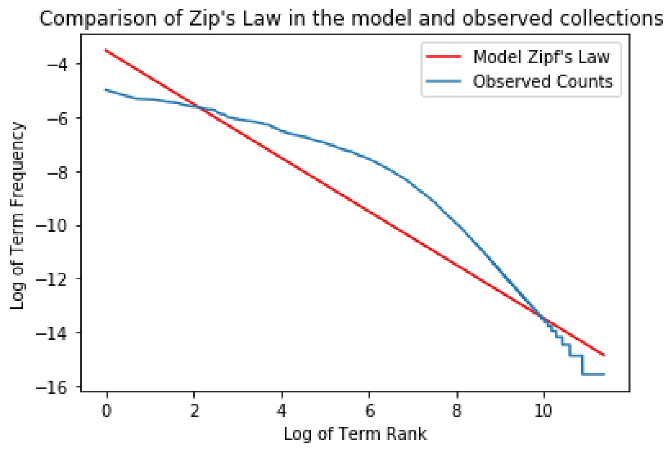

**Figure 1: Logarithmic plot comparing ideal and observed Zipf's Law for the test collection.**

## 3 INVERTED INDEX

Inverted Index addresses one the major issues in information retrieval, namely - the extreme matrix sparsity encountered when one attempts to build a matrix associating words with documents. The inverted index associates documents with words instead. Moreover, given the large collection size and relatively costly computations, it is crucial to pre-compute as many values as possible and thus accelerate the computation. The Inverted Index is implemented through first constructing a *dictionary* of unique terms from across all the passages, and then constructing a *postings list* for each term, which stores the unique IDs of document which the given term appears in. In this implementation, we compute a different inverse index for the candidate passages of each query, as this will significantly accelerate the latter computations. In a given inverse index, for each postings list will not only store the document IDs but also the term frequencies and the TF-IDF values (which shall be described in greater detail in the following section). In addition to computing the inverse index, we also pre-compute the list of all document IDs associated with a given query, `doc_IDs_full`, and a dictionary of unique terms across from across all the 1k passages considered for a given query, `unique_tokens_q`. The inverted indexes and other pre-computed quantities are stored in `q_precompute`.

## 4 BASIC RETRIEVAL MODELS

In this section we will implement two basic retrieval models, the Vector Space model and BM25 which are commonly used as baselines.

### 4.1 TF-IDF

To quantify the presence of a given term in a document one could simply use a binary vector, however this implementation carries no useful information about the frequency of the term in the document or measure of how rare a given term is. Hence, we weight the binary model using TF-IDF: Term Frequency - Inverse Term Frequency. Term frequency, $tf_i^d$ is the number of times the $ith$ term occurs in the given document. Potential issues may arise when lengths of passages vary, as an increased term frequency may be due to a longer passage. To account for this, all term frequencies can be divided by the lengths of their documents. However, all the candidate passages in this dataset are roughly similar size, hence this step was not necessary. The Inverse Term Frequency, $idf_i$, is the log of the number of documents in the collection, $N$, divided by the number of documents in which $i^{th}$ term appears, $n_i$.

$$tf_i = \sum_{t \in q \cap d} 1 \quad idf_i = log_{10}\left(\frac{N}{n_i}\right) \quad tf_i \cdot idf_i = tf_i \times idf_i \tag{4}$$

The value of tf-idf will thus be influenced by the frequency of the word in the document and it's rarity. In this work, tf-idf has been pre-computed for every term for every query and stored in the inverted index.

### 4.2 Vector Space Model

The Vector Space Model is an example of a 'bag of words' (BOW) model, where the positions of words are not stored. Here we will explain the implementation for a single query, $q$, but this implementation is repeated in a loop for all of the 200 queries.

Firstly, it is necessarily to vectorize both, the document and the query. We will represent a single document as a vector of length $V$, where $V$ is the vocabulary size. We initialise all entries to be zero and then use the information stored in the inverted index to fill in the rows corresponding to terms present in the document with the associated TF-IDF value. For computational efficiency we here store all of the document vectors in a single matrix, `tf_idf_mat`, and we fill the matrix in row-wise, going through each term in the inverted index and filling out its TF-IDF values into the appropriate columns. As this requires a fair degree of indexing, we create dictionaries to facilitate this task: a dictionary called `term_dimensions` which maps tokens to rows, and a dictionary called `doc_to_col` which maps document IDs to columns.

Next, we build the vector representation of the query. Here, the query is also $n$-dimensional and we use a binary representation to mark presence of tokens. It is important to note that we first check whether the query terms are present in the vocabulary, and if that is not the case then we discard them.

To obtain a measure of similarity between the query and document, we compute the cosine similarity of the two vectors, using equation 5, where $\mathbf{q}$ is the query vector, $\mathbf{d}$ is the document vector and $V$ is the vocabulary size.

$$\text{cos-sim}\,(\mathbf{q}, \mathbf{d}) = \frac{\sum_{i=1}^{V} q_i \times d_i}{\sqrt{\sum_{i=1}^{V} q_i^2} \times \sqrt{\sum_{i=1}^{V} d_i^2}} \tag{5}$$

Lastly, we order the document IDs according to their similarity scores and save the first 100 entries to a dataframe. Thanks to the efficient implementation and pre-computations, the model takes less than a minute to rank all 200 queries.

### 4.3 BM25

Okapi BM25, usually shortened to BM25, is a probabilistic language model, which implements a probability distribution over strings of text. Intuitively, it assessed the conditional probability of how likely a given query is to be generated under the estimated language model. It is based on the binary independence model, building upon it by incorporating weights for the document and query terms. To implement this model we once again vectorize the documents and the query, storing all document vectors in a matrix, `BM25_mat` and we construct dictionaries to easily map the token dimensions onto rows and document IDs onto columns. It is also necessary here to discard the query terms which are not present in the vocabulary. Here, to compute the similarity between the document and query we compute the BM25 score, shown in equation 6. This equation has been obtained by simplifying the original BM25 score for the case where no relevance information is present, hence both $r$ and $R$ are zero. Also, the query term frequencies were all found to be 1 hence the $qf$ term has been omitted here. N corresponds to the total number of passages, $n_i$ is the number of passages in which the $i^{th}$ term occurs, $dl$ is the length of the document and $avdl$ is the average length of all candidate documents for given query. Meanwhile $k_1$, $k_2$ and $b$ are all parameters which must be set empirically.

$$BM25\,(Q, D) = \sum_{i \in Q} \log \frac{N - n_i + 0.5}{n_i + 0.5} \cdot \frac{(k_1 + 1)\, f_i}{K + f_i} \cdot \frac{(k_2 + 1)}{k_2} \tag{6}$$

where $K = k_1 \left( (1 - b) + b \cdot \frac{dl}{avdk} \right)$.

The values of $k_1$, $k_2$ and $b$ chosen here as follows: $k_1 = 1.2$, $k_2 = 100$, $b = 0.75$. The choice of parameters was made through first consulting relevant literature [8] to obtain a suitable range of values and then using the specified ranges to empirically experiment and find a set which performs well on this dataset.

The BM25 score is computed as follows: iterating through the document matrix, for every token (row), we extract its term frequency, associated document IDs and their length and we compute the BM25 score simultaneously for all of the document entries in that given row. Once the entire matrix has been computed, we use the binary query vector to sum over all the individual query terms present in each of the documents, thus obtaining a total BM25 score for each of the documents. Next, all the documents are ranked in order of decreasing BM25 scores and the top 100 are saved to the dataframe. This process is repeated until we iterate through all of the queries.

## REFERENCES

[1] Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, Mir Rosenberg, Xia Song, Alina Stoica, Saurabh Tiwary, and Tong Wang. 2016. MS MARCO: A Human Generated MAchine Reading COmprehension Dataset. arXiv:cs.CL/1611.09268

[2] Zhuyun Dai, Chenyan Xiong, Jamie Callan, and Zhiyuan Liu. 2018. Convolutional Neural Networks for Soft-Matching N-Grams in Ad-Hoc Search. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining (WSDM '18)*. Association for Computing Machinery, New York, NY, USA, 126–134. https://doi.org/10.1145/3159652.3159659

[3] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:cs.CL/1810.04805

[4] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W. Bruce Croft. 2016. A Deep Relevance Matching Model for Ad-hoc Retrieval. *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management - CIKM '16* (2016). https://doi.org/10.1145/2983323.2983769

[5] Shiyu Ji, Jinjin Shao, and Tao Yang. 2019. Efficient Interaction-based Neural Ranking with Locality Sensitive Hashing. 2858–2864. https://doi.org/10.1145/3308558.3313576

[6] Shangsong Liang, Emine Yilmaz, Hong Shen, Maarten De Rijke, and W. Bruce Croft. 2017. Search Result Diversification in Short Text Streams. *ACM Trans. Inf. Syst.* 36, 1, Article Article 8 (July 2017), 35 pages. https://doi.org/10.1145/3057282

[7] Jimmy Lin. 2019. The Neural Hype and Comparisons Against Weak Baselines. *SIGIR Forum* 52, 2 (Jan. 2019), 40–51. https://doi.org/10.1145/3308774.3308781

[8] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2018. *Introduction to information retrieval.* Cambridge University Press.

[9] Irina Matveeva, Chris J.C. Burges, Timo Burkard, Andy Laucius, and Leon Wong. 2006. High accuracy retrieval with multiple nested rankers. In *Proceedings of SIGIR* (proceedings of sigir ed.).

[10] Donald Metzler and W. Bruce Croft. 2005. A Markov Random Field Model for Term Dependencies. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '05)*. Association for Computing Machinery, New York, NY, USA, 472–479. https://doi.org/10.1145/1076034.1076115

[11] Bhaskar Mitra and Nick Craswell. 2019. An Updated Duet Model for Passage Re-ranking. arXiv:cs.IR/1903.07666

[12] Bhaskar Mitra, Fernando Diaz, and Nick Craswell. 2016. Learning to Match Using Local and Distributed Representations of Text for Web Search. arXiv:cs.IR/1610.08136

[13] Bhaskar Mitra, Corby Rosset, David Hawking, Nick Craswell, Fernando Diaz, and Emine Yilmaz. 2019. Incorporating Query Term Independence Assumption for Efficient Retrieval and Ranking using Deep Neural Networks. (July 2019).

[14] Rodrigo Nogueira and Kyunghyun Cho. 2019. Passage Re-ranking with BERT. arXiv:cs.IR/1901.04085

[15] Harshith Padigela, Hamed Zamani, and W. Bruce Croft. 2019. Investigating the Successes and Failures of BERT for Passage Re-Ranking. *CoRR* abs/1905.01758 (2019). arXiv:1905.01758 http://arxiv.org/abs/1905.01758

[16] Parth Pathak, Mithun Das Gupta, Niranjan Nayak, and Harsh Kohli. 2018. AQuPR: Attention Based Query Passage Retrieval. (2018), 1495–1498. https://doi.org/10.1145/3269206.3269323

[17] Eilon Sheetrit. 2018. Utilizing Inter-Passage Similarities for Focused Retrieval. (2018), 1453. https://doi.org/10.1145/3209978.3210222

[18] Eilon Sheetrit and Oren Kurland. 2019. Cluster-Based Focused Retrieval. (2019), 2305–2308. https://doi.org/10.1145/3357384.3358087

[19] Ellen M. Voorhees. 1985. The Cluster Hypothesis Revisited. (1985), 188–196. https://doi.org/10.1145/253495.253524

[20] Chenyan Xiong, Zhuyun Dai, Jamie Callan, Zhiyuan Liu, and Russell Power. 2017. End-to-End Neural Ad-hoc Ranking with Kernel Pooling. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval - SIGIR '17* (2017). https://doi.org/10.1145/3077136.3080809