# Enhancing BM25 with Sparse Retrieval-Based Ranking and Word2Vec Models: A Pointwise Learning-to-Rank Method

KANISH DWIVEDI, Delft University of Technology (TU Delft), Netherlands
DYLAN DURAND, Delft University of Technology (TU Delft), Netherlands
IRTAZA HASHMI, Delft University of Technology (TU Delft), Netherlands
PENGZHI YANG, Delft University of Technology (TU Delft), Netherlands

In information retrieval, classic methods such as TF-IDF, TF, and BM25 have exhibited promising results, while relying on textual matching to assess document relevance and rankings. Meanwhile, advancements in natural language processing, such as Word2Vec, provide views from semantic word embeddings. Our research investigates the potential of combining these lexical and semantic approaches to leverage their strengths. By adopting a pointwise Learn-to-Rank (LTR) strategy, the learned weighted scores are incorporated from both sparse retrieval-based rankings and Word2Vec embeddings to enhance the results from BM25, and an overall enhanced ranking performance is achieved, compared to the BM25 baseline.

CCS Concepts: • **Information systems** → **Learning to rank**; *Novelty in information retrieval*; *Rank aggregation*; *Presentation of retrieval results*.

Additional Key Words and Phrases: information retrieval, learning to rank, BM25, word2vec

**ACM Reference Format:**
Kanish Dwivedi, Dylan Durand, Irtaza Hashmi, and Pengzhi Yang. 2024. Enhancing BM25 with Sparse Retrieval-Based Ranking and Word2Vec Models: A Pointwise Learning-to-Rank Method. In *IN4325-Q3-24: Information Retrieval, TU Delft*. ACM, New York, NY, USA, 10 pages.

## 1 INTRODUCTION

Ranking is a central task in Information Retrieval (IR). Many IR problems are by nature ranking problems, such as document retrieval, collaborative filtering, key term extraction, definition finding, important email routing, and sentiment analysis [18]. For document retrieval, many ranking models have been researched to date, they can be roughly categorized as query-dependent models and query-independent models. Query-independent models can rank documents based on their own importance, independent of the query, a famous example of this is PageRank [25]. Query-independent models will be out of the scope of our research. Query-dependent models on the other hand retrieve documents based on the occurrences of the query terms in the documents. Basic examples of these include the boolean and vector space models. Term frequency-inverse document frequency (TF-IDF), is a statistical measure to evaluate the importance of a word in a document (relative to others in the corpus) [18]. It can be used to rank the relevance of a document for a particular query by computing the TF-IDF value for each term-document pair. However, for many years, and in many applications, models based on probabilistic ranking principles have been used. Okapi BM25, a bag-of-words retrieval function that ranks a set of documents based on the query terms appearing in each document, regardless of their proximity within the document, has been used extensively [17, 18].


Manuscript submitted to ACM

An important characteristic of these bag-of-word models of TF(-IDF) vector-based models, is that these representations are sparse [19]. With recent advancements in the field of machine learning (ML) and specifically natural language processing (NLP), dense retrieval methods have become an interest in the IR research community [19].

The interest of dense retrieval lies in understanding the semantics of the text to improve relevance scores [32]. Word embeddings can be generated using Word2Vec [1]. The average of a query and document text embedding can be used to calculate the similarity between each other. The results of these can be used to rank the documents based on their similarity to the query. This then results in a dense retrieval ranking method that is computationally less expensive than other well-known methods, like BERT [29].

Whilst these introduced ranking algorithms have made great progress, they often contain hyperparameters, and for optimal performance, tuning these is an important yet non-trivial process. This is due to IR evaluation measures being non-continuous and non-differentiable with respect to the parameters [18]. Furthermore, it is of interest to investigate how to combine multiple models to create a more efficient new model. Success in the field of ML, has shown means to automatically tune hyper-parameters, combine multiple features, and allow for generalization. Learning to rank (LTR) is a way to adopt ML technologies in the context of training a ranking model [17, 18]. LTR can be used in various ways, one can be to tune the parameters of existing IR models, another is to use discriminative training models to learn the optimal way of combining features extracted from query–document pairs, or another can be to directly combine "weak" ranking models in an ensemble [30]. Many LTR models make use of implicit feedback data as features, for example, click rates on documents, or dwelling times [15]. The data is easy to collect, and thus aids in training LTR, but is often biased, recent research is focusing on ways to combat this bias alongside achieving fairness [4, 12, 22, 31]. However, we noticed an interesting research gap in the context of combining rankers. LTR ensemble methods that combine rankers, like AdaRank, often use "weak" rankers that are decision stumps [17, 30], feature-based methods often use the score of BM25 as a feature, and try to enhance its performance by additional features like LMIR features, document meta data, query meta data, and various others [27]. We found the idea of combining "weak" rankers really interesting, and to the best of our knowledge, we don't know of a LTR model that combines sparse and dense rankers. Thus, to explore this, we made our research question:

Would combining sparse retrieval-based methods of TF-IDF, and TF, with Word2Vec (with IDF filter thresholds at 0.5 and 0.2) models providing semantic similarities through point-wise LTR linear regression improve the ranking performance of BM25?

We chose BM25, TF-IDF, TF for sparse rankers, these in particular, as they are simple to implement, and standalone are relatively "weak" rankers. We used Word2Vec as a dense ranker for its simplicity and "weak" standalone performance. Furthermore, as the used existing implementations of BM25, TF-IDF, and TF support basic query processing of removing stopwords, we extended this to Word2Vec by doing query preprocessing by removing words that had a IDF value of less than 0.5 and 0.2 respectively.

We make use of the Antique dataset [13] for purposes of training and testing our LTR pipeline. Our key finding was that our combined model comprising of BM25, Word2Vec, TF-IDF, and TF, surpasses the baseline BM25 model in performance. However, it falls short in comparison to the finely tuned BM25 model.

## 2 RELATED WORK

As our work primarily evolves around combining sparse and dense retrieval models using point-wise LTR, we look at two categories of related work: LTR and how its used in the context of combining rankers, and how sparse and dense rankers have been used together before.

### 2.1 Use of LTR to combine models

In the past, pointwise feature-based LTR has been explored in various forms. Many research papers have experimented with different types of models apart from linear regression. Probabilistic models have been tested, such as Gaussian [5] or logistic regression [7, 10], and even Support Vector Machines [24]. An interesting approach was to improve regression results by using Discounted Cumulative Gain (DCG) criterion but finding bounds on it to design appropriate convex learning formulations [8]. However, these approaches don't exactly use the scores of multiple rankers. They may use a single ranker, or no ranker scores at all, rather using the documents fetched by the ranker, and then re-ranking them using another set of features.

In [16] two sparse rankers (PRrank and RSVM) were combined, and it showed that this approach outperformed the performance of a single ranker. Similarly, in [27], a training dataset for various existing corpus' was created and used on various LTR algorithms. Their features mostly revolved around scores of BM25, results from a LMIR algorithm, IDF values, and various document/query/web metadata. In their experiments, they found that listwise LTR worked better in general, however, in some datasets it was seen that the simple pointwise regression approach outperformed.

However, these papers still don't consider using scores of dense rankers as features, this gives us the research inspiration to see the effect of using a combination of sparse and dense rankers.

There exist other approaches that are out of the scope of our research. Examples of this are AdaRank [30] and RankBoost [6], inspired from AdaBoost [9], where weak rankers (usually decision stumps) are trained iteratively, focusing on misclassifications of the previous ranker. LamdaRank [3] is another successful approach based on gradient boosting on an objective function that incorporates a pairwise or listwise loss function. These approaches are different to ours as they tend to the same base "weak" ranker and combine multiple of them.

### 2.2 Sparse and Dense Models

Incorporating Word2Vec's embedding has been shown to improve document ranking [23] compared to BM25. Along with this, experiments that combined Word2Vec with BM25 yielded positive results [11]. However, these approaches didn't make use of LTR for combination. Therefore, we think this is yet another reason for our research, and believe that combining BM25, Word2Vec, and other sparse ranking models will provide us with an overall improved performance in ranking documents.

## 3 METHODOLOGY

This section will describe the methodology that was followed to perform our experiments to answer our research question. Firstly, the data that was used for training and validation will be presented. Then we mention the setup of the five models we use. Next, we explain the LTR pipeline, and finally we explain the steps we took to evaluate our system.

### 3.1 Data

The primarily make use of the Antique dataset [13] from IR-Datasets [20]. The only reason for this is because it has available qrels: query relevance values that have been obtained from a set of relevance assessments. For that matter, to recreate our experiments, any dataset can be used that also has available qrels. On this dataset, there already exists a training and testing split, and we make use of these for their intended purposes.

### 3.2 Models

We make use of five models in our experiments, here we explain how they were implemented and setup for experiments.

*3.2.1 BM25, TF-IDF, and TF.* Our implementation primarily makes use of the Python framework PyTerrier [21]. The framework has implementations available for BM25, TF-IDF, TF and other retrieval algorithms. We make use of these implementations to retrieve the top 500 documents and the respective scores given. An important note is that when setting up a retrieval pipeline, you can specify controls and properties. In our experiments, in the control parameters, we do not run the divergence from randomness query expansion. As for properties, we do keep the standard term pipeline configuration in which words in Terrier's standard stopwords are removed and Porter's English stemmer is applied. For BM25 we additionally set its 'b' parameter to 0.5. We found this value by testing performance in the training dataset via grid search (in range of 0.45 - 0.7).

*3.2.2 Word2Vec.* Word2Vec is a well known vector representation that is able to group words together based on semantics. In order to train our models some preprocessing had to be done beforehand. This consisted of filtering out words above a chosen IDF threshold. The reason for doing this is to filter out any less meaningful words. Two models are trained with different IDF thresholds as it is believed that, if any 'useful' words are accidentally removed in the model trained with a higher IDF value then the second model trained with a lower IDF will incorporate this instead. To find optimal thresholds we plan to employ K-fold cross-validation.

Given the unsupervised nature of the Word2Vec model and its application in IR ranking tasks, our cross-validation approach is relatively different from the normal ones: which use the same losses for training and validation. We first divide the training dataset into K-folds. Then, for each threshold under consideration, we validate the model for $K$ times, each time designating a different fold as the validation set and using the remaining folds for training. This process ensures that each fold serves as a validation set once. Rather than employing the standard loss function for validation as in conventional cross-validation, we utilize the nDCG@10 (see Section 3.4) value that the trained Word2Vec model achieves on the queries and documents within the current validation fold. Note, we can do this as we have qrels available, thus the ideal document ranking for each query in the fold based on the labeled relevance scores can be made. Finally, we average the validation results across all K segments, and assign this value to this particular threshold value under validation. The threshold value that achieves the highest average, can then be selected to be used for all further experiments.

With regards to training this model, all the text from the document has to be tokenized. This is done using the NLTK library [2]. After this tokenization, all the words above the selected IDF thresholds are removed. This will result in a list of filtered words.

Using the gensim library [28], Word2Vec was trained using 25 epochs, dimensionality size of 100, the distance between the current word set to five.

Using the embeddings created by training the model we were able to rank the documents based on the text similarity to the query. The average embedding of the query and document text were compared using their cosine similarity. This process was carried out for each document text in the corpus. The cosine similarity score was used to rank the documents, with the highest score being ranked first.

### 3.3 LTR pipeline

In general, LTR algorithms can be categorised into three approaches: pointwise, pairwise approach, and listwise [18]. We focus on pointwise approaches, where for a specific query, the input space contains a feature vector for each document, the corresponding output space contains the relevance degree for that document. The aim is to learn a function $f$ that takes in the feature vector of a document, and predicts the relevance of it. This function called the scoring function [18], can be used to then sort all the documents and produce the final ranked list.

The following is our LTR training pipeline, which explains how we make use of LTR in order to learn to combine the five rankers:

(1) For each of the training query in the Antique dataset, get the ranked documents from each of the models
(2) For each training query, obtain the list of all the documents for which we have a relevancy score for. For each ranker, check what score they gave this document, if no score was given, we assign a value of 0: as we know that this document was not in this rankers top 500, the safest assumption we can make is of a score of 0. Because we want to learn to combine our rankers, we use the rankers' scores as the documents feature vector. Thus, a training sample for LTR is a tuple that follows this structure: [query_id, document_number, relevance_label, bm25_score, tf-idf_score, tf_score, word2vec0.5_score, word2vec20.2_score]
(3) Use this training data to learn the scoring function. We use a linear regression function from the scikit-learn package [26] that makes use of the residual sum of squares loss function.
(4) The output of the pipeline are ranked documents by sorting them in descending order based on merging the weighted scores of the four other models into the top 500 of the BM25.

### 3.4 Evaluation

To evaluate the performance of our proposed LTR approach against baseline methods, we employ two widely recognized metrics: Mean Average Precision (MAP) and Normalized Discounted Cumulative Gain (nDCG). These metrics allow us to assess the effectiveness of our ranking model comprehensively. Specifically, after we train the model, we obtain the rankings for queries in the Antique test set and calculate the metric values.

*3.4.1 MAP.* MAP computes the mean of Average Precision (AP) while also considering the order of the documents. For a single query, AP can be defined as:

$$AP@K = \frac{\sum_{k=1}^{K}(Precision(k) \times rel(k))}{N}, \tag{1}$$

where $Precision(k)$ is the precision at cutoff $k$ in the list of retrieved documents, $rel(k)$ as a binary indicator represents whether the item at $k$ is relevant to the query or not, while $N$ denotes the number of relevant documents for the top-$K$ recommendations. Then, AP is averaged by the number of queries $Q$ to compute MAP:

$$MAP = \frac{\sum_{q=1}^{Q} AP_q@K}{Q}. \tag{2}$$

*3.4.2 nDCG.* However, MAP considers the relevance of each document as a binary value: 0 indicates the document is irrelevant to the specific query, while 1 indicates relevance. Comparatively, nDCG@$K$ accounts for the graded relevance scores of documents, providing a more nuanced evaluation by considering different levels of relevance rather than a simple binary distinction. This measure is particularly valuable when document relevance exhibits varying degrees.

Specifically, DCG is defined as:

$$DCG@K = \sum_{k=1}^{K} \frac{rel_k}{\log_2(k+1)}, \tag{3}$$

where $rel_k$ represents the relevance score of the item at position $k$, as determined by any ranking algorithm. To facilitate comparison across different queries, DCG is divided by ideal DCG (IDCG, the DCG calculated using the ground truth relevance labels):

$$nDCG@K = \frac{DCG@K}{IDCG@K}. \tag{4}$$

## 4 EXPERIMENTS

In this section we evaluated our model and compared it to different baselines using the metrics explained in Section 3.4 on the Antique dataset explained in Section 3.1. For the evaluation, we want to compare different combinations of models to see how well our proposed model performs. In Table 1 we refer to Sparse and Dense which entail the combination of TF + TF-IDF and the two Word2Vec models, respectively.

By evaluating these combinations of models we can deduce if our proposed model improves performance compared to different combinations. We first experimented with the baseline model, BM25, and evaluated its results to establish a baseline. Next, we tuned the model by playing around with its hyperparameters e.g. *bm25.b* and performed experiments. The chosen value for *bm25.b* was 0.5 and was found using grid search. After tuning the model on the training set, we noticed an improvement on the MAP and nDCG scores on the validation set. This result infers that tuning a model for a specific dataset usually improves the performance. Next, we combined our baseline model, BM25, with the TF-IDF, which is a weaker model than BM25. As expected, the experiments produced a decrease in performance. This infers that combining a strong model with a weak model often deteriorates the overall performance. Next, we combine the BM25 model with the TF model, which is the weaker than TF-IDF model. As a result, the performance decreased compared to the BM25 + TF-IDF model, as expected. Combining an even weaker model leads to worse performance. Our next combination was BM25 with the Sparse model. As mentioned before, the Sparse model refers to TF + TF-IDF. As a result, we improved our performance compared to the BM25 + TF-IDF and BM25 + TF. The results infer that by combining multiple weak models, the ensemble model can be more powerful. For the next ensemble model, we combined our BM25 with our first Word2Vec trained model. For this model, our IDF (inverse document frequency) filter threshold was 0.2, that is, we removed all the words from each document that had IDF of greater than 0.2. The ensemble model performed better than BM25 + TF but not better than BM25 + TF-IDF and BM25 + Sparse. The results inferred that our first Word2Vec model was weaker than the TF-IDF model but performed better than the TF model when combined with BM25. We trained another Word2Vec model with IDF filter threshold of 0.5, which was combined with BM25. As a result, the model had similar results compared to the first one. When combining both of the Word2Vec models with the BM25, we didn't see any major improvements in the results. Finally, we combined all the models to see how they perform. All the combined models performed better than all the combinations except the tuned BM25 model. This was expected, as by combining more weaker models, the larger ensemble can be more powerful than smaller combinations. A limitation of this experiment is that we only trained and validated our results on one dataset, the Antique dataset (see Section 3.1). The results may vary on different datasets. Moreover, the IDF filtering we do affects the performance of the Word2Vec models and hence the evaluation results. If other IDF filtering values were used the results may vary.

Table 1: Evaluation scores of different models using MAP and nDCG

| Model Name | MAP | nDCG |
|---|---|---|
| BM25 (untuned) | 0.146583 | 0.228598 |
| BM25 | 0.153428 | 0.244974 |
| BM25 + TF-IDF | 0.118265 | 0.211217 |
| BM25 + TF | 0.066238 | 0.113911 |
| BM25 + Sparse | 0.144464 | 0.228281 |
| BM25 + Word2Vec1 | 0.124607 | 0.201572 |
| BM25 + Word2Vec2 | 0.128277 | 0.208145 |
| BM25 + Dense | 0.12524 | 0.202981 |
| Combined | 0.150841 | 0.236082 |

## 5 DISCUSSION

In this section we discuss certain point of interests in regards to our research. First we attempt to explain our results. Secondly, cross validation is explained and how this could potentially improve result in the future. Lastly, the impact of different LTR models are examined.

### 5.1 Justification of the Results

From the experiments we can see that our combiner model slightly under performs compared to the ranking of a tuned BM25. There are quite a few factors at play that can cause this performance reduction.

After running our experiments, we analysed the ranking models of Word2Vec individually. The training result of Word2Vec proved to be a lot worse than we initially thought, as all of the documents labeled relevant were ranked very low.

We did notice, however, that in the ranking results, the document text did, somewhat, align with the test query. For example, using the query 'how do i get college money', the output ranking of the Word2Vec model does not return the required results, according to the given query labels. However, when we manually view the rankings we noticed that, in the case of the mentioned query, the provided documents do talk about money, economics, finding a job, etc. This suggests that the Word2Vec model encompasses the semantics of the words in the query but does a poor job of the semantics of the entire query sentence.

For future work, an approach could be to incorporate Sentence2Vec [14] instead. The method builds on Word2Vec but trains on sentence embedding instead of word embedding. This could improve the ranking of the documents as now the semantics of the entire query is captured. One drawback of using Sentence2Vec is that it is computationally more expensive, which would not fully align with our research goal of combining simple models to improve upon BM25.

### 5.2 Cross-validation

The validation phase to fine-tune Word2Vec's hyperparameters did not produce satisfactory results, leading us to exclude this component from our final LTR pipeline. Nonetheless, this experience provided us with valuable insights into deep learning-based model validation and hyperparameter tuning.

Besides, in order to find optimal IDF-threshold values for our Word2Vec models, we opted for a manual approach. The different values that were tested were [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6]. This resulted in us training two Word2Vec models with selected thresholds 0.2 and 0.5.

### 5.3 Use of other LTR models

To learn to rank the documents a Linear Regression pointwise approach was taken. This method provides us with weights for each model, in a timely manner.

The problem with using this pointwise approach is that the true relevance scores are needed to train the model. In our case only a handful of queries were labelled. This probably caused the regression to not provide very accurate weights. Also, it has been shown that linear regression performs worse than other list or pair-wise approaches [27]. Furthermore, in our experiments, we did not perform any regularisation, experiments in [27] showed that regularisation tended to aid in ranking performance.

For future works, one can attempt a more complex Learn-To-Rank model such as ListNet [27].

## 6 CONCLUSION

In this project, we proposed an approach that successfully combines different sparse retrieval models (TF-IDF and TF) and a deep learning-based dense retrieval model, namely Word2Vec, to improve the ranking results of BM25. Specifically, the Word2Vec model encoding dense representations provides semantic similarities.

Utilizing the generated document rankings, we successfully implemented a pointwise Learn-To-Rank algorithm with a linear regression model that learns the weights for each of the models. Our experiments showed that our proposed combined ranking model did not not improve performance on our test metrics, in comparison to the tuned BM25 baseline. We suspect this is due to the Word2Vec failing to encapsulate sentence semantics of the entire query.

## 7 SELF ASSESSMENT

(1) **Kanish** Code: Setting up LTR pipeline (data preprocessing, setting up LTR training data and training, LTR re-ranking). Report: Introduction, Related Work, Methodology

(2) **Dylan** Code: Helped with Word2Vec model, setting up of experiments, LTR. Report: Discussion, part of methodology and related work, conclusion, proofreading of the report

(3) **Irtaza** Code: Mainly trained the Word2Vec models and improved them. Worked on experiments by combining different models and evaluating their results using LTR on the evaluation metrics. Report: Worked on the Experiments and Conclusion section of the report. Moreover, contributed to the correctness of the report.

(4) **Pengzhi** Code: Helped with combining Word2Vec model with the sparse retrieval-based ranking methods, implementing cross-validation. Report: Abstract, Evaluation Metrics, Cross Validation

## 8 REPOSITORY

Our repository: https://github.com/irtazahashmi/ir_ranking_docs/tree/main

## REFERENCES

[1] V Kishore Ayyadevara and V Kishore Ayyadevara. 2018. Word2vec. *Pro Machine Learning Algorithms: A Hands-On Approach to Implementing Algorithms in Python and R* (2018), 167–178.

[2] Steven Bird. 2006. NLTK: the natural language toolkit. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*. 69–72.

[3] Christopher JC Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. *Learning* 11, 23-581 (2010), 81.

[4] Mouxiang Chen, Chenghao Liu, Jianling Sun, and Steven C.H. Hoi. 2021. Adapting Interactional Observation Embedding for Counterfactual Learning to Rank. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval* (<conf-loc>, <city>Virtual Event</city>, <country>Canada</country>, </conf-loc>) *(SIGIR '21)*. Association for Computing Machinery, New York, NY, USA, 285–294. https://doi.org/10.1145/3404835.3462901

[5] Wei Chu, Zoubin Ghahramani, and Christopher KI Williams. 2005. Gaussian processes for ordinal regression. *Journal of machine learning research* 6, 7 (2005).

[6] Harold Connamacher, Nikil Pancha, Rui Liu, and Soumya Ray. 2020. Rankboost+: an improvement to Rankboost. *Mach. Learn.* 109, 1 (jan 2020), 51–78. https://doi.org/10.1007/s10994-019-05826-x

[7] William S. Cooper, Fredric C. Gey, and Daniel P. Dabney. 1992. Probabilistic retrieval based on staged logistic regression. In *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Copenhagen, Denmark) *(SIGIR '92)*. Association for Computing Machinery, New York, NY, USA, 198–210. https://doi.org/10.1145/133160.133199

[8] David Cossock and Tong Zhang. 2006. Subset ranking using regression. In *Learning Theory: 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006. Proceedings 19*. Springer, 605–619.

[9] Yoav Freund and Robert E Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences* 55, 1 (1997), 119–139.

[10] Fredric C Gey. 1994. Inferring probability of relevance using the method of logistic regression. In *SIGIR'94: Proceedings of the Seventeenth Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval, organised by Dublin City University*. Springer, 222–231.

[11] Thiago Gomes and Marcelo Ladeira. 2020. A new conceptual framework for enhancing legal information retrieval at the Brazilian Superior Court of Justice. In *Proceedings of the 12th International Conference on Management of Digital EcoSystems* (Virtual Event, United Arab Emirates) *(MEDES '20)*. Association for Computing Machinery, New York, NY, USA, 26–29. https://doi.org/10.1145/3415958.3433087

[12] Shashank Gupta, Harrie Oosterhuis, and Maarten de Rijke. 2023. Safe Deployment for Counterfactual Learning to Rank with Exposure-Based Risk Minimization. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval* (<conf-loc>, <city>Taipei</city>, <country>Taiwan</country>, </conf-loc>) *(SIGIR '23)*. Association for Computing Machinery, New York, NY, USA, 249–258. https://doi.org/10.1145/3539618.3591760

[13] Helia Hashemi, Mohammad Aliannejadi, Hamed Zamani, and W. Bruce Croft. 2019. ANTIQUE: A Non-Factoid Question Answering Benchmark. arXiv:1905.08957 [cs.IR]

[14] Chaker Jebari, Manuel Jesús Cobo, and Enrique Herrera-Viedma. 2018. A new approach for implicit citation extraction. In *Intelligent Data Engineering and Automated Learning–IDEAL 2018: 19th International Conference, Madrid, Spain, November 21–23, 2018, Proceedings, Part II 19*. Springer, 121–129.

[15] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2016. Unbiased Learning-to-Rank with Biased Feedback. arXiv:1608.04468 [cs.IR]

[16] Dong Li, Maoqiang Xie, Yang Wang, Yalou Huang, and Weijian Ni. 2008. Multiple Ranker Method in Document Retrieval. In *International Conference on Intelligent Computing*. Springer, 407–414.

[17] Hang Li. 2022. *Learning to rank for information retrieval and natural language processing*. Springer Nature.

[18] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (mar 2009), 225–331. https://doi.org/10.1561/1500000016

[19] Yi Luan, Jacob Eisenstein, Kristina Toutanova, and Michael Collins. 2021. Sparse, Dense, and Attentional Representations for Text Retrieval. arXiv:2005.00181 [cs.CL]

[20] Sean MacAvaney, Andrew Yates, Sergey Feldman, Doug Downey, Arman Cohan, and Nazli Goharian. 2021. Simplified data wrangling with ir_datasets. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2429–2436.

[21] Craig Macdonald and Nicola Tonellotto. 2020. Declarative Experimentation in Information Retrieval using PyTerrier. In *Proceedings of the 2020 ACM SIGIR on International Conference on Theory of Information Retrieval (ICTIR '20)*. ACM. https://doi.org/10.1145/3409256.3409829

[22] Marco Morik, Ashudeep Singh, Jessica Hong, and Thorsten Joachims. 2020. Controlling Fairness and Bias in Dynamic Learning-to-Rank. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval* (Virtual Event, China) *(SIGIR '20)*. Association for Computing Machinery, New York, NY, USA, 429–438. https://doi.org/10.1145/3397271.3401100

[23] Eric Nalisnick, Bhaskar Mitra, Nick Craswell, and Rich Caruana. 2016. Improving document ranking with dual word embeddings. In *Proceedings of the 25th international conference companion on world wide web*. 83–84.

[24] Ramesh Nallapati. 2004. Discriminative models for information retrieval. In *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Sheffield, United Kingdom) *(SIGIR '04)*. Association for Computing Machinery, New York, NY, USA, 64–71. https://doi.org/10.1145/1008992.1009006

[25] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The PageRank Citation Ranking : Bringing Order to the Web. In *The Web Conference*. https://api.semanticscholar.org/CorpusID:1508503

[26] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Andreas Müller, Joel Nothman, Gilles Louppe, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2018. Scikit-learn: Machine Learning in Python. arXiv:1201.0490 [cs.LG]

[27] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. 2010. LETOR: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval* 13 (2010), 346–374.

[28] Radim Řehřek, Petr Sojka, et al. 2011. Gensim—statistical semantics in python. *Retrieved from genism. org* (2011).

[29] Jiajia Wang, Jimmy X Huang, Xinhui Tu, Junmei Wang, Angela J Huang, Md Tahmid Rahman Laskar, and Amran Bhuiyan. 2024. Utilizing BERT for Information Retrieval: Survey, Applications, Resources, and Challenges. *Comput. Surveys* (2024).

[30] Jun Xu and Hang Li. 2007. AdaRank: a boosting algorithm for information retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (Amsterdam, The Netherlands) *(SIGIR '07)*. Association for Computing Machinery, New York, NY, USA, 391–398. https://doi.org/10.1145/1277741.1277809

[31] Meike Zehlike, Ke Yang, and Julia Stoyanovich. 2022. Fairness in Ranking, Part II: Learning-to-Rank and Recommender Systems. *ACM Comput. Surv.* 55, 6, Article 117 (dec 2022), 41 pages. https://doi.org/10.1145/3533380

[32] Wayne Xin Zhao, Jing Liu, Ruiyang Ren, and Ji-Rong Wen. 2024. Dense text retrieval based on pretrained language models: A survey. *ACM Transactions on Information Systems* 42, 4 (2024), 1–60.