Adaptive Training With K-Means and Hierarchical Clustering

Sho Cremers
s.a.cremers@student.tudelft.nl
(5052602)

Sukhleen Kaur
s.kaur@student.tudelft.nl
(5053307)

Kanya Paramita Koesomo
KanyaParamitaKoesoemo@student.tudelft.nl
(5035597)

ABSTRACT

Learning to rank (LTR) is a machine learning method of automatically sorting new documents based on their relevance, importance, or desirability. Most LTR methods tend to be trained on large datasets in hopes of good generalization to new data. But this does not always lead to optimal performance. This is where adaptive training comes into play. This is a method where there is multiple trained models. In this study, we make use of adaptive training using LambdaMART, and we compare the use of k-means and hierarchical clustering for adaptive training. The best-feature calibration (BFC) strategy was applied for comparison of performance of LTR models. The results indicate that adaptive training models shows more robustness compared to the baseline LTR model and we also find that there is no significant difference between the performance of k-means and hierarchical clustering for adaptive training.

KEYWORDS

Learning to Rank (LTR), LambdaMart, Adaptive training, K-means clustering, Hierarchical clustering, Best-feature calibration (BFC)

1 INTRODUCTION & BACKGROUND

Learning to Rank (LTR) is a machine learning task used to create a ranking model $f(q, d)$, where q is the query and d is the document. The ranking model predicts the relevance of the document and the query. The ranking model has traditionally been constructed without training but as web search engines became able to provide log data, these features made possible to apply machine learning techniques to automatically construct the ranking model [6]. The goal of LTR is to use the ranking model trained to assign scores to the documents, which can be used to provide a ranking list [6]. It has many applications, such as document retrieval, question-answering, and machine translation.

LTR has often been practiced by training one model on one dataset, and the test data is used to analyze the performance of the trained model. This means that every query-document pairs will be evaluated with the same weight for each of the features. However, it encounters issues where a specific feature is more informative to one kind of query than others. Adding query-specific features to the data may help to address the differences in queries better, but it will require a much larger training dataset. This is because the dataset would contain very large amounts of data on popular queries, but have a very small amount of data on uncommon queries.

To deal with this issue, previous researches have looked at query-dependent models in which multiple models are trained [1, 5, 7], and it has shown promising results.

Research done by Kuzi et al. [4] aimed to see the benefit of adaptive training over a commonly used global training method. In their work, they first computed the feature vector of each query by averaging the features of relevant documents. These queries were then clustered using k-means with $k = 5$. These clusters were then used to create a vector representation of query q by applying several metrics for each of the trained cluster models. The resulting queries were again clustered and new vector representation was created. This process of clustering was repeated twice. For choosing the cluster on evaluation, 3 techniques were used; selective cluster, cluster oracle, and cluster fusion. To compare these adaptive training methods with global training, they proposed a best-feature calibration (BFC) strategy, which they used a single feature that had the best performance. The results showed that adaptive training was especially effective for queries that performed worse on the LTR baseline model compared to the best single feature.

In this paper, we aimed to reproduce the work by Kuzi et al. [4] with several changes. We specifically looked at the performance of adaptive training using LambdaMART on the LETOR 4.0 dataset [8]. For the adaptive training, we compared the performances of two clustering techniques; k-means and hierarchical clustering. Furthermore, we made a slight change in the cluster model selection for evaluation. The specific steps can be seen in Section 3.

2 DATASET

For the experiment, we have used the dataset from LETOR 4.0 by Microsoft [8]. The dataset uses the Gov2 web page collection of approximately 25 million pages and a query sets from Million Query Track of TREC 2007 (MQ2007). It consists of 1700 queries with labeled documents. We used the dataset collection with the setting of Supervised Ranking.

As explained by Qin and Liu [8], each row in the dataset represents a query-document pair. For each query-document

pair, they provide a relevance label and 46 features. The first column of the dataset is the value of relevance label of the pair which can either take the value of 0 (least), 1 or 2 (most), the second column is the query ID, the following 46 columns is the 46 features and its values, and the last columns are comments about the pair, such as the document ID. Some of the features used in this dataset are query's term frequency (TF), inverse document frequency (IDF), document length (DL), BM25, and LMIR of different parts of the document. Table 1 shows sample rows from MQ2007 dataset. The dataset was divided into 5 folds, and so 5-fold cross-validation was performed.

2 qid:10032 1:0.056537 2:0.000000 3:0.666667 4:1.000000 5:0.067138 ... 45:0.000000 46:0.076923 #docid = GX029-35-5894638 inc = 0.0119881192468859 prob = 0.139842
0 qid:10032 1:0.279152 2:0.000000 3:0.000000 4:0.000000 5:0.279152 ... 45:0.250000 46:1.000000 #docid = GX030-77-6315042 inc = 1 prob = 0.341364
0 qid:10032 1:0.130742 2:0.000000 3:0.333333 4:0.000000 5:0.134276 ... 45:0.750000 46:1.000000 #docid = GX140-98-13566007 inc = 1 prob = 0.0701303
1 qid:10032 1:0.593640 2:1.000000 3:0.000000 4:0.000000 5:0.600707 ... 45:0.500000 46:0.000000 #docid = GX256-43-0740276 inc = 0.0136292023050293 prob = 0.400738

Table 1: MQ2007 Rows Sample

3 METHODS

We follow the method used by by Kuzi et al. [4] that uses query-level adaptive training based on clustering (we'll refer to this method as cluster-based adaptive training) to construct the LTR ranking model. We also used their proposed BFC method to analyze the robustness of the constructed LTR model. We used Python 3 for our implementation of the code.

Clustering-Based Adaptive Training

Training a single model on a single training set is a common practice in LTR where a single ranking function is used to handle all queries [3, 4]. This approach might not fit well when they're tested in specific queries because different query types require different combinations of features. To address this problem we can use query-specific learning that enables dynamic weights on features [4].

We follow Kuzi et al. [4] on applying Clustering-Based Adaptive Training method that follows these steps:

1. For each query ID, retrieve all the relevant document (document IDs with relevance level 1 or 2), and average the features of the documents. These averaged features become the new representation of the query.
2. Cluster the queries according to the features.
3. For each cluster, create a dataset that contains all query-document pairs of queries in the cluster.
4. Train each cluster using the LTR algorithm.
5. For each query, evaluate using several metrics for each trained cluster model. The scores become the new query vector.
6. Repeat steps 2-6 twice more.

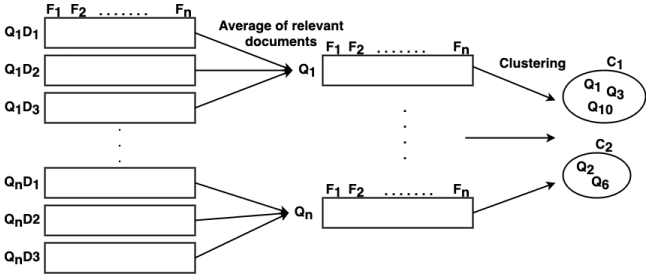


Figure 1: Averaging the features over the relevant documents and performing clustering using the result.

In the **first part** of the method, we apply clustering and training to get the LTR models for each cluster. The initial step is to partition training data into clusters. As shown in Figure 1, query q is represented by averaging the features of relevant documents. Using this query representation, clustering into k clusters is performed to the queries in the training dataset. In our work, these queries are clustered using **k-means** as well as **hierarchical clustering** algorithm.

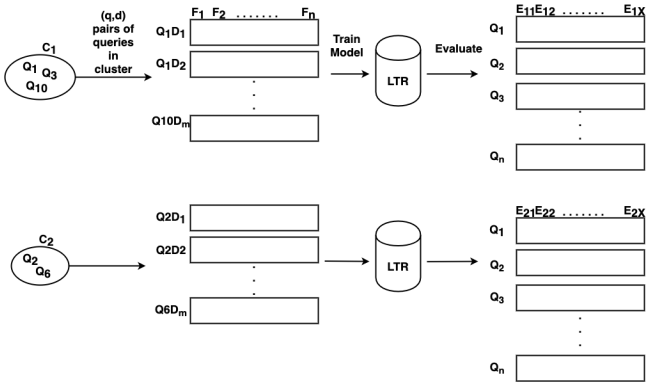


Figure 2: Getting evaluation metrics from LTR models.

Based on the resulting clusters as shown in Figure 2, k separate LTR models are trained. Evaluation is performed on each ranking results using different n evaluation metrics. In this study, we use $ndcg@\{3, 5, 10\}$, $MAP@100$, $MRR@100$,

and $p@\{3, 5, 10\}$. Each training query has its own evaluation value for each of the metrics.

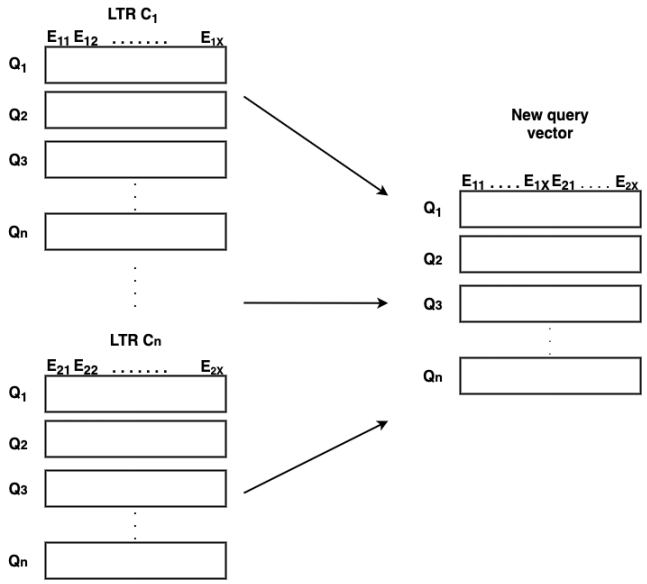


Figure 3: Constructing performance profile for each queris.

A performance profile is computed by calculating evaluation measures of n evaluation metrics for each of the k result lists that is ranked by the different clusters as explained before. This performance profile can be represented as

$$\overrightarrow{q} = (m_1^1, ..., m_n^1, ..., m_1^k, ..., m_n^k) \quad (1)$$

where m_j^i is the j-th evaluation measure of the query for when the model of cluster i is used for ranking. Figure 3 shows the illustration of this process. By repeating these processes several times, models are refined several times (twice in our work) using the performance profile of the queries in different cluster models. We now get k LTR models trained through the iterations.

Clustering Methods

K-Means Clustering. K-Means clustering is an unsupervised machine learning technique. The goal of this method is to group certain data that belong together and to determine patterns occurring in the data. It tries to do so by making sure the distance within a cluster is minimal and the distance between clusters is maximal. The working of this algorithm is defined as follows:

1. Define a value k which specifies the number of clusters to be found
2. Initialize by randomly picking k points as cluster centers
3. Assign data point to the closest cluster center. This distance is defined as the squared distance between the data point and the cluster center.
4. Change the position of the cluster center to the average of the points that have been assigned to it
5. Repeat steps 3 and 4 until the point assignments do not change anymore

K-Means always finds a solution for a finite number of iterations. It is computationally fast for large amounts of data and is scalable which important for the problem at hand. Hence the k-means clustering algorithm is used and it is implemented using `sklearn.cluster.KMeans` with $k = 5$.

Hierarchical Clustering. Hierarchical clustering is another unsupervised machine learning technique used to group similar data together. This method differs from k-means because instead of deploying cluster centers and assigning the data to them, it groups data in a bottom-up fashion. It does so as such:

1. Each data point is considered its own cluster
2. Determine two points with minimal distance
3. Combine the two closest points into a cluster
4. Stop when only one cluster remains or if a predefined k exists, stop when k clusters are found.

The minimal distance can be defined in many ways; single-linkage, complete-linkage, average-linkage and Ward's method. In this study, Ward's method is used where the variance between two points/clusters is minimized. One of the main assumptions of the k-means clustering algorithm is that the dataset has a spherical shape. It will underperform if the dataset does not meet that assumption. Hence, hierarchical clustering is used which does not take into account the spherical shape assumption. It is implemented using `sklearn.cluster.AgglomerativeClustering` with $k = 5$ and `linkage='ward'`.

Choosing the cluster

Continuing to the **second part** where the goal is to select a cluster for a test query that best fit the queries in test dataset. A learning method is applied to predict a cluster for a test query from a set of clusters for each query. Following Kuzi et al. [4], we also used 3 different approaches on selecting cluster for queries. These approaches are explained as follows.

Selective Cluster. In this approach, we used a supervised learning method to predict the cluster for each query from the test dataset. The learning method we chose for the cluster prediction was the Logistic Regression model from `scikit-learn (V 0.20.3)` package. We first obtain the single strong feature which is a feature that obtained the highest performance on average in the training set. The single strong feature is acquired by applying the BFC approach that is explained in the upcoming section. We then select the

top-10 documents for each query using the single best feature. the features of these documents are averaged to obtain the query representation. The logistic regression model is trained using the query vectors and the cluster labels of the training queries, and the labels of test queries are predicted using the same vector representation.

Cluster Oracle. With this approach, we select the cluster that performed the best for each query. Since this method looks at the evaluation scores of all clusters before selecting, it cannot be applied for real-world use, but it serves as the upper bound of the adaptive training, where the system can select the best cluster of every query.

Cluster Fusion. For this method, we used a different definition from the one given in [4] In this paper, cluster fusion follows a similar approach to the selective cluster method. Here, we use the same logistic regression model and train with the same query vectors as the selective cluster method. With the trained regression model, the predicted probabilities for each of the clusters, $p(c|q)$ can be determined for testing queries. Given these probabilities, a new ranking model $f_{cf}(q,d)$ can be computed for each of the testing queries as the following:

$$f_{cf}(q,d) = \sum_{c \in C} f_c(q,d)p(c|q) \tag{2}$$

where $f_c(q,d)$ is the ranking model of the cluster.

Best-Feature Calibration

To analyze the benefit of adaptive learning on the query level, we followed the main paper on using the BFC strategy that is a method proposed on the paper [4]. This method is for analyzing the robustness of Learning to Rank (LTR) models. As explained by Kuzi et al., the way this method works is that the performance of the single best feature can be compared to that of baseline LTR models. Now, the difference in the evaluation of the models can be computed for each query. By doing so, the performance of adaptive training models can be compared to the baseline LTR models for the queries that performed differently between the single best feature and the baseline model.

In this experiment, the evaluation metric *ndcg*@5 was used on the training queries to determine the best feature.

Learning To Rank (LTR): LambdaMART

LambdaMART is based on the family of models known as MART (Multiple Additive Regression Trees) which make use of gradient boosting technique. The basis of LambdaMART lies in the observation that a ranking model can be trained without a cost function. The ranking model here is instead trained on gradients, which are the costs in terms of the ranking model scores [2]. Each of the trees contribute to minimizing loss and these groups of trees are then the final ranking model.

We used the Python learning to rank (LTR)[1] toolkit (`V 0.2.4`) to implement LambdaMART where the number of estimators were set to 500.

4 EVALUATION

Normalized Discounted Cumulative Gain (NDCG)

Discounted Cumulative Gain (DCG) is a method of measuring ranking quality. It evaluates the usefulness (gain) of a document based on its rank. Since the list of ranks of documents can vary in length, it is important to normalize their usefulness and hence we use the Normalized DCG. It is defined as follows:

$$\text{NDCG} = \frac{DCG}{IDCG} \tag{3}$$

where

$$\text{DCG} = \sum_{i=1}^{p} \frac{2_i^{rel} - 1}{\log_2(i+1)}$$

and

$$\text{DCG} = \sum_{i=1}^{|REL_p|} \frac{2_i^{rel} - 1}{\log_2(i+1)}$$

where p is the ranking position, rel_i is the graded relevance at position i and REL_P is the list containing the relevant document till position p. NDCG does not only tell us whether a document is relevant or not, it gives us information about how relevant the document is and hence it is used in this study. We use the NDCG function implemented in Pyltr.

Mean Reciprocal Rank (MRR)

The Mean Reciprocal Rank (MRR) is used to evaluate models that deliver a list of documents for a specific query based on the relevancy of the document. The Reciprocal Rank (RR) of a document is simply the inverse of the rank of the document. The MRR is then the mean of these RRs. Quantitatively it is described as follows:

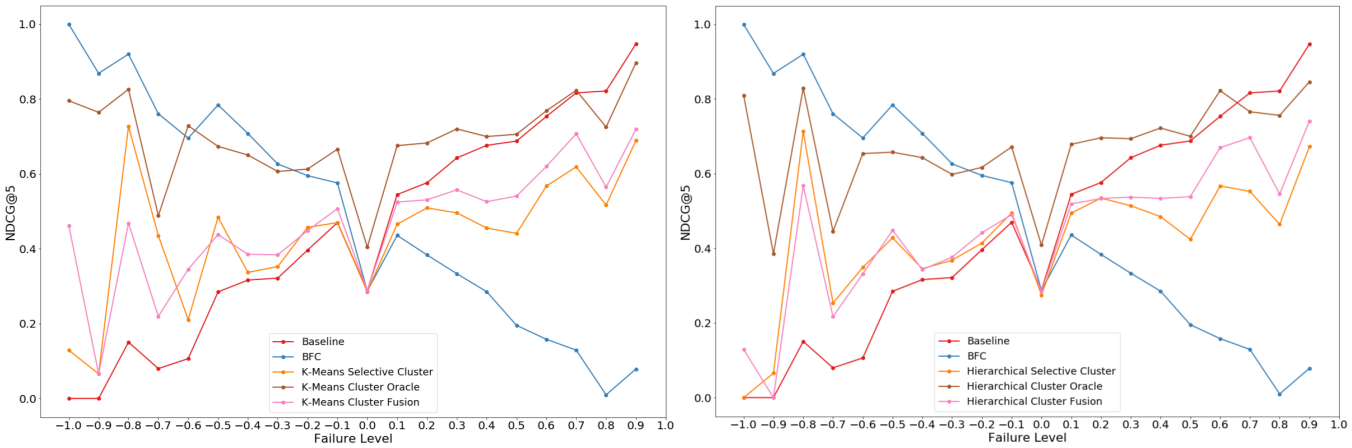
$$\text{MRR} = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{\text{rank}_i} \tag{4}$$

where rank_i is the position of the relevant document in the i-th query and Q is the total number of queries. MRR was chosen alongside NDCG as the evaluation metrics because some queries in the dataset contain many relevant documents and some almost none. It would not be fair evaluating with metrics such as MAP since queries that do not contain many relevant documents would not be able to perform well even if the relevant documents are ranked high. For this reason, MRR was chosen.

[1] https://github.com/jma127/pyltr

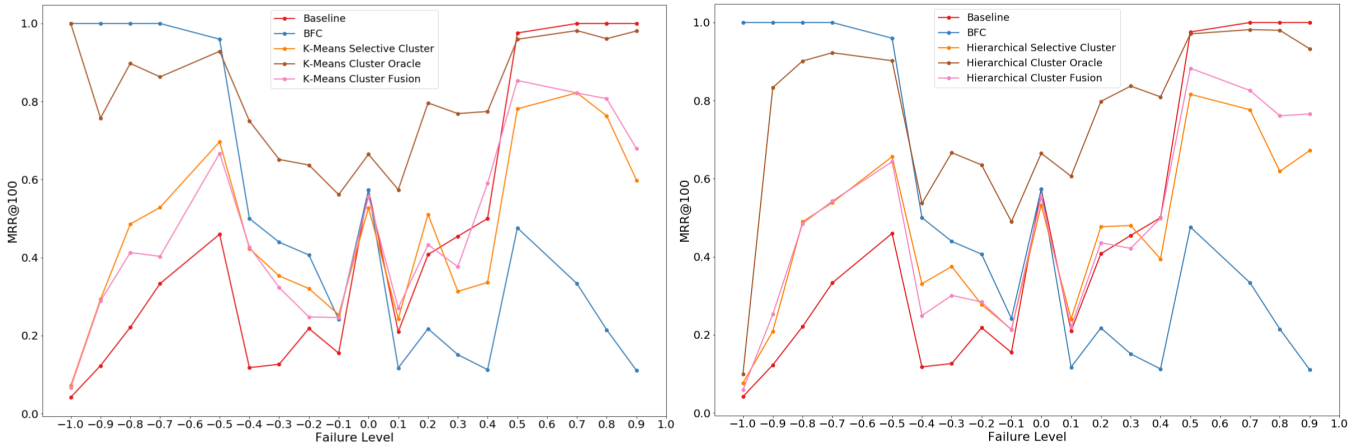
Table 2: Mean of the evaluation metrics for each model.

Metric	BFC	Baseline	K-Means Clustering			Hierarchical Clustering		
			Selective Cluster	Cluster Oracle	Cluster Fusion	Selective Cluster	Cluster Oracle	Cluster Fusion
NDCG@5	0.385	0.417	0.385	0.557	0.409	0.387	0.559	0.404
MRR@100	0.527	0.568	0.538	0.727	0.557	0.535	0.727	0.556



(a) K-Means **(b) Hierarchical**

Figure 4: Performance of the baseline model (red), Best Feature Calibration (blue), selective clustering (orange), cluster oracle (brown) and cluster fusion (pink) as given by NDCG@5 (y-axis) against the failure level (x-axis). This has been given for both the k-means (left) as well as hierarchical clustering (right).



(a) K-Means **(b) Hierarchical**

Figure 5: Performance of the baseline model (red), Best Feature Calibration (blue), selective clustering (orange), cluster oracle (brown) and cluster fusion (pink) as given by MRR@100 (y-axis) against the failure level (x-axis). This has been given for both the k-means (left) as well as hierarchical clustering (right).

5 RESULTS

Table 2 shows the mean results of the evaluation metric for each of the models. It can be seen that there is no difference between k-means and hierarchical clustering methods. Paired-sample t-tests were conducted to compare the

evaluation scores of the queries on cluster oracle between k-means ($MRR@100 : M = 0.73, SD = 0.40$. $ndcg@5 : M = 0.56, SD = 0.34$) and hierarchical clustering ($MRR@100 : M = 0.73, SD = 0.40$. $ndcg@5 : M = 0.56, SD = 0.35$). The results showed that for both evaluation metrics, no difference in scores was observed ($MRR@100 : t = 0.074, p = 0.94$. $ndcg@5 : t = 0.44, p = 0.66$). The cluster oracle model was used to compare the performance of the two clustering techniques as the performance of cluster oracle is not dependent on the cluster selection model.

Figures 4 and 5 show the mean evaluation scores of trained models on different failure levels. Figure 4 show the mean value of the evaluation metric NDCG@5 on the y-axis and the failure level on the x-axis for the baseline model, best single feature, selective cluster, cluster oracle and cluster fusion. In Figure 4a, the performance for k-means clustering can be seen and in Figure 4b the values for the performance for the hierarchical clustering can be seen. Similarly Figure 5 show the mean value of the evaluation metric MRR@100 on the y-axis and the failure level on the x-axis for the baseline model, best single feature, selective cluster, cluster oracle and cluster fusion. Figure 5a shows the performance of the k-means clustering method and Figure 5b shows the performance of the hierarchical clustering method. The failure level of a query is determined by subtracting the score of the best single feature model from the baseline LTR model. The query is then binned to the nearest 0.1. From the aforementioned figures, it can be observed that when the baseline model performs much worse than the best-single feature, adaptive training models seem to be able to perform better than the baseline model, showing robustness in comparison to the baseline model.

Both Figures 4 and 5 also show that cluster fusion may perform slightly better than the selective cluster, especially when the baseline model is performing better than the best single feature model. It can also be seen that the cluster oracle model performs better than the baseline and the best single feature, proving that with good cluster selection function, adaptive training models can outperform globally trained LTR models.

6 DISCUSSION

In this study, we aimed to reproduce the experiment done by Kuzi et al. in [4]. Specifically, we investigated the benefit of using adaptive trained LTR models over globally trained LTR models by comparing them to the best single feature model. Additionally, we looked at two methods of clustering for adaptive training.

The results showed that the adaptive training models were able to outperform for queries in which the best single feature had a better evaluation score than the baseline LTR model, which is in line with the original paper. The experiment was able to replicate the robustness of adaptive training models as they were in [4].

This study also showed that the cluster oracle model was able to perform better than both the best single feature model and the baseline model. This means that if the system can choose the best cluster to be evaluated for every (or most) query (queries), Adaptive training LTR models can be especially useful. In this study, a logistic regression model was fitted on the training queries to predict the cluster of evaluating queries. This predicting model was not very accurate, however, as the performance of the selective cluster model was low compared to the cluster oracle model. The cluster fusion model was also able to perform better than the selective cluster model, depicting that the predictive model needs to be improved for adaptive training models to be useful. Possible future studies include finding a better cluster predicting model for evaluating queries.

Comparing the adaptive training models of k-means and hierarchical clusters, no difference was found on the performance. The number of clusters may be a more important factor than the clustering method for the performance of the model.

A possible improvement in this study can be made to the implementation where the models are trained. In this study, we train the models on each partition sequentially using a `for` loop. This poses a problem when the dataset is large as it would take a really long time to compute the results. Hence, a parallel framework can be used to reduce the time taken to compute the results. One such example of a framework that can be integrated with our implementation in Python is Dask[2]. Given this framework, it would be interesting to see if our results would differ for larger datasets.

7 REFLECTION

We faced a lot of problems reproducing the paper. The paper lacked a lot of clarity around their implementation. They did not provide sufficient details with regards to parameters and did not provide their source code either. The length of the paper was also insufficient. It contained only 4 pages which are not enough to explain the complexity of their algorithms. It would have been nice had they provided at least some pseudocode.

8 RESOURCE REPOSITORY

The code and dataset used for this study can be found here: https://github.com/sukhleen-kaur/CoreIR2020

[2]https://dask.org/

REFERENCES

[1] Jiang Bian, Xin Li, Fan Li, Zhaohui Zheng, and Hongyuan Zha. 2010. Ranking Specialization for Web Search: A Divide-and-Conquer Approach by Using Topical RankSVM. In *Proceedings of the 19th International Conference on World Wide Web* (Raleigh, North Carolina, USA) *(WWW '10)*. Association for Computing Machinery, New York, NY, USA, 131–140. https://doi.org/10.1145/1772690.1772705

[2] Chris J.C. Burges. 2010. *From RankNet to LambdaRank to LambdaMART: An Overview*. Technical Report MSR-TR-2010-82. https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/

[3] Xiubo Geng, Tie-Yan Liu, Tao Qin, Andrew Arnold, Hang Li, and Heung-Yeung Shum. 2008. Query dependent ranking using K-nearest neighbor. 115–122. https://doi.org/10.1145/1390334.1390356

[4] Saar Kuzi, Sahiti Labhishetty, Shubhra Kanti Karmaker Santu, Prasad Pradip Joshi, and ChengXiang Zhai. 2019. Analysis of Adaptive Training for Learning to Rank in Information Retrieval. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (Beijing, China) *(CIKM '19)*. Association for Computing Machinery, New York, NY, USA, 2325–2328. https://doi.org/10.1145/3357384.3358159

[5] L. Lee, J. Jiang, C. Wu, and S. Lee. 2009. A Query-Dependent Ranking Approach for Search Engines. In *2009 Second International Workshop on Computer Science and Engineering*, Vol. 1. 259–263.

[6] Hang LI. 2011. A Short Introduction to Learning to Rank. *IEICE Transactions on Information and Systems* E94-D, 10 (2011), 1854–1862. https://doi.org/10.1587/transinf.e94.d.1854

[7] Hsuan-Yu Lin, Chi-Hsin Yu, and Hsin-Hsi Chen. 2011. Query-Dependent Rank Aggregation with Local Models. In *Information Retrieval Technology*, Mohamed Vall Mohamed Salem, Khaled Shaalan, Farhad Oroumchian, Azadeh Shakery, and Halim Khelalfa (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–12.

[8] Tao Qin and Tie-Yan Liu. 2013. Introducing LETOR 4.0 Datasets. arXiv:cs.IR/1306.2597