Identification of efficient algorithms for web search through implementation of learning-to-rank algorithms

NIKHIL DHAKE*, SHITAL RAUT and ASHWINI RAHANGDALE*

Department of Computer Science and Engineering, Visvesvaraya National Institute of Technology,
Nagpur 440010, India
e-mail: dhakenikhil511@gmail.com; saraut@cse.vnit.ac.in; rahangdaleashwini@gmail.com

MS received 15 June 2017; revised 22 November 2018; accepted 22 December 2018; published online 27 March 2019

Abstract. Today, amount of information on the web such as number of publicly accessible web pages, hosts and web data is increasing rapidly and exhibiting an enormous growth at an exponential rate. Thus, information retrieval on web is becoming more difficult. Conventional methods of information retrieval are not very effective in ranking since they rank the results without automatically learning the model. Machine learning domain called learning-to-rank comes to the aid to rank the obtained results. Different state-of-the-art methodologies have been developed for learning-to-rank to date. This paper focuses on finding out the best algorithm for web search by implementation of different state-of-the-art algorithms for learning-to-rank. Our work in this paper marks the implementation of learning-to-rank algorithms and analyses effect of topmost performing algorithms on respective datasets. It presents an overall review on the approaches designed under learning-to-rank and their evaluation strategies.

Keywords. Information retrieval; machine learning; learning-to-rank.

1. Introduction

World Wide Web (WWW) is unfortunately unrestricted in spite of its very vital role in communication. There are no constraints for number of documents and links while designing web sites. Because of this, web is showing an unregulated growth at an exponential rate and becoming huge and more complex, leading to difficulty in an efficient information retrieval (IR) [1]. Search engines are used to find specific information available on WWW. For a given query, search engines try to retrieve the best matches for the query by searching on web and presents a list of results in the most relevant order. Search engines attempt to be comprehensive or selective. Comprehensive search engines are the ones that try to index everything they can, e.g., Google. Selective search engines are not aimed at indexing the complete web. They concentrate on searching the web within a particular subject area or resource type. These search engines are intended at covering a single subject area in a deep manner instead of spanning the information across different subject areas. Hence, it can easily index documents that are not covered even in mammoth collection of documents indexed by crawlers of large search engines. Hence, it can provide a helpful starting point for certain search queries, e.g., GameSpot, BioMedNet, etc. However, only static web pages can be indexed by search

engines such as publicly accessible documents present on servers. These search engines are differentiated based on frequency of indexing-specific web sites.

Ranking can be regarded as a standard issue in the area of IR. In IR, the process of ranking involves the arrangement of documents applicable as per their graded relevance associated with a given query. Nowadays, ranking is a very crucial component of web search, where commercially deployed search engines help the end users in finding the required and useful information among the very huge number of document corpus available on WWW. For a given query, conventional IR systems retrieve documents from their corpus and provide ranked list of results without using any prior information or learning. Different methodologies used in traditional IR can be widely described into two categories, viz., query-dependent models and query-independent models [2]. Query-dependent models try to fetch the documents based on whether the query terms appear in the respective documents. They include different models of ranking such as Boolean model [2], vector space model (VSM) [2] and probabilistic ranking models like BM25 [3] and Language Model for IR [4]. Boolean model figures out the relevance of a document to the query based on whether the document is to be included or not in the result, but it fails to determine the degree of relevance. Hence, VSM was proposed, which made use of TF-IDF weighting scheme. Term frequency (TF) is how many

*For correspondence

times a term appears in the document. The frequency of term t in the document is calculated as

$$TF(t) = \frac{\#\{\text{occurrence of term in document}\}}{\#\{\text{total number of terms in document}\}}. \quad (1)$$

Inverse document frequency (IDF) assesses the importance of a term treating all other terms in the documents with the same importance. IDF of a term can be calculated as

$$TF(t) = \log \frac{\#\{\text{documents}\}}{\#\{\text{documents that contain term } t\}}. \quad (2)$$

The terms in the document that get high score obtained from the product of TF and IDF appear most frequently in the documents. However, some terms might appear more frequently than other terms, e.g., of, to, is, etc. Hence, TF–IDF scheme can just be regarded as an improvement in normal counting of terms in the document. BM25 is a probabilistic retrieval framework, which is generally called as Okapi BM25 because of its initial implementation by Okapi IR System. It is not an individual model but entirely a group of ranking models. For a query q with N terms, viz. $t_1, t_2, \dots, t_n$ the score of BM25 model for a document d can be calculated as

$$BM25(d, q) = \sum_{m=1}^N \frac{IDF(t_m) \times TF(t_m)(c_1 + 1)}{TF(t_m) + c_1 \left(1 - c_1 + c_2 \frac{\#\{\text{terms in document } d\}}{L}\right)}, \quad (3)$$

where c_1 and c_2 are free parameters and L is the average length of document in the document corpus. Language model [4] for IR system calculates probability of the document to produce the terms in the query through the language model constructed for it and uses the method of maximum likelihood. The smoothing issue in language model is tackled with background language model that is computed for complete document corpus. Query-independent models rank the documents based on their own importance. The features depend only on document and not on the query, e.g., PageRank [5]. PageRank is applicable to web search since it uses the hyperlink structure of the web for the task of ranking. PageRank uses the probability that a person clicking the hyperlinks randomly visits a specific webpage to rank it. PageRank value for any page P is computed as

$$PageRank(P) = \sum_{j \in S_p} \frac{PageRank(j)}{Out(j)}, \quad (4)$$

where j is a page from set S_p , which includes all pages linking page P , and $O(j)$ is the number of out-links from page p . As an extension, a smoothing term is included based on the fact that the person clicking the hyper-links will not follow the present structure and will reach any other page. Its probability is denoted by $(1 - \alpha)$, where α is

the damping factor. Thus, PageRank for any page P is computed as

$$PageRank(P) = \sum_{j \in S_p} \frac{PageRank(j)}{Out(j)} + \frac{(1 - \alpha)}{\#\{\text{pages on web}\}}. \quad (5)$$

However, traditional methods used for IR are not trivial. It requires lot of efforts to tune the different parameters involved in the task of IR. Nowadays, the concept of the machine learning is also adopted for IR, which is known as learning-to-rank. In this paper, we focus on different state-of-the-art methodologies designed for learning-to-rank to date. It sets the strategy to search an efficient algorithm for the web search document retrieval using implementation of learning-to-rank and analyses the performance of existing standard learning-to-rank algorithms on different datasets.

2. Learning-to-rank

In this section, we discuss learning-to-rank framework in brief (figure 1). We overview the learning-to-rank approaches and the state-of-the-art designs in these approaches using different machine learning methods.

2.1 Learning-to-rank framework

Suppose a search engine gives a ranked list of results for a specific query. If top n documents are considered, then we can define learning-to-rank as the process in which a better ranking is generated for the top n documents by making use of machine learning methodologies. Formally, for each query $q_i \in Q$ where Q is the set of queries and a list of m associated candidate documents, $D_i = D_{(i,1)}, D_{(i,2)}, \dots, D_{(i,m)}$, document ranking D_i is generated as a permutation π_i of the document indices $1, 2, 3, \dots, m$. To generate the optimal permutation π_i , we require an efficient learning function that gives less error as much as possible.

2.2 Approaches of learning-to-rank

Learning-to-rank framework is mainly categorized in three approaches, viz., pointwise approach, pairwise approach and listwise approach.

2.2a Pointwise approach: Pointwise approaches look at a single document at a time in the loss function. In the pointwise approach, any existing regression or classification method can be used in learning process for ranking. Every query–document pair with its relevance judgment is regarded as a separate training instance and final real

Figure 1. Learning-to-rank framework [6].

valued score is calculated for every query–document pair. The scores computed for documents are not dependent upon other document scores in the final list of outcomes for a specific query. The final ranked list is obtained by just sorting result list of document scores. A regression method called Multiple Additive Regression Trees (MART) [7] and classification method called Random Forests [8] are the examples algorithms that use pointwise approach. However, ranking problem is about finding a relative order than finding the exact relevance degree and the main problem with pointwise approach is that it ignores the relative ordering of documents. Since the position of documents is not visible to loss function, the insignificant documents, which have lower ranks in the finally obtained list, may be more highlighted by loss function than the important ones [6].

2.2b Pairwise approach: Pairwise approaches look at a pair of documents at a time in the loss function. With the pairwise approach, the relative order among documents can be modelled better. It first separates documents for a query, then takes two documents at a time and comes up with optimal ordering by deciding which of them should be ranked higher. The aim is to reduce the wrongly ordered pairs. Pairwise approaches are found to be better in practice than pointwise approaches since finding relative order resembles relevance ranking more than computing the relevance score. RankSVM [9], RankBoost [10] and RankNet [11] are the examples of pairwise approach.

2.2c Listwise approach: Listwise approaches directly look at the complete list of documents in the loss function. The aim is to obtain the optimal ordering for it. We can achieve listwise learning-to-rank in two ways:

- Direct optimization of IR measures such as normalized discounted cumulative gain (NDCG), e.g., AdaRank [11].
- Minimizing a loss function. Loss function will be defined based on the properties of type of ranking to be obtained, e.g., ListNet [12] and ListMLE [13].

The first subcategory of listwise approach minimizes the measure-specific loss function, which is the most suitable choice for listwise approach. However, it is very difficult to optimize the position-specific evaluation measures because of non-differentiable and non-continuous characteristic of evaluating measures. Thus, recently the main focus is on handling the difficulty of optimizing such objective functions from first principles. To tackle these challenges, several attempts have been made. The first solution is to approximate the measure-based ranking error [11, 14–16]. Another solution is to alternatively optimize a continuous and differentiable (and sometimes even convex) bound of the measure-based ranking error [17]. The second subcategory of listwise approach is not measure specific. It reflects the inconsistency between the output of the ranking model and the ground truth permutation [12, 13].

Most of the listwise learning-to-rank algorithms yield a single linear ranking model for all ranking items, but this ranking model may be insufficient to capture the underlying nonlinear ranking mechanism among all items. Thus, Ou *et al* [18] propose a divide-and-train method to learn a nonlinear ranking model from listwise training data. It uses rank-preserving clustering approach to explore the structure of objects in their preference feature space and all the items in training data are divided into several clusters. A multi-task listwise ranking approach is then employed to train linear ranking functions for all the clusters and get the results using the entire preference feature space.

2.3 Learning-to-rank algorithms

We have implemented and analysed the state-of-the-art learning-to-rank algorithms, which fall into three major categories depending upon the optimized loss function and input taken. We briefly discuss these algorithms in this section. Generally, ranking models design with MART [7], Random Forrest [8] and Linear Regression [19] fall into the pointwise approach whereas SVMRank [9], RankNet [8], RankBoost [10] and LambdaMART [20] algorithms are designed under the pairwise approach. Listwise approach includes two categories. One of this directly optimizes the loss function. It includes ListNet [12], ListMLE [13] and BoltzRank. The second category optimizes the IR

evaluating measures. It includes a number of algorithms like SoftRank, AppRank, Es-Rank [21], AdaRank [11], SVM^{MAP} [22], SVM^{NDCG} [17], etc.

MART [7], a pointwise approach proposed by Friedman in 2000 is a boosting technique that combines weak rankers to form a strong ranker using gradient descent. This method is also called as Gradient Boosting Regression Trees. Random Forests [8], a method proposed by Breiman in 2001, is an ensemble of multiple decision trees. Every decision tree is created by a part of the data obtained through random sampling. The remaining data are used for testing on the created tree and simultaneously out-of-bag (OOB) error is computed. The final outcome model is constructed by selecting the tree with minimum error. Linear Regression approach uses least-square method proposed by Lawson and Hanson [19] in 1995. It is a pointwise approach that just simply seeks to differentiate and classify documents by finding a regression line. Based on these methods, McRank [23], PRank [24], Subnet-Ranking [25] and OC-SVM [26] algorithms have been proposed under the pointwise approach.

Like classification, Support Vector Machine (SVM) is also used for ranking. RankSVM [9] uses SVM for ranking model. It operates on pairs of examples, maps it into features and classifies using SVM. It mainly uses hinge loss function and optimizes it with less number or miss order pairs of document. The basic idea behind RankSVM is to maximize the margin for separating the documents so that it can generalize better on new examples. Moreover, model complexity is low when linear kernels are used. However, to manage complex non-linear tasks, non-linear kernels like polynomial kernel can be used [27].

RankNet is the first and most popular learning-to-rank algorithm proposed by Burges *et al* [28] in 2005, which uses neural network as the underlying model and logistic function as loss function defined on pair of documents. It is a probabilistic cost function, which helps in reducing the inversion while ranking the results, i.e., it tries to keep the correct order among the pairs of instances. Gradient descent techniques are used to optimize mean square error loss defined for RankNet. However, this underlying model is not limited to only neural network. Rather, whichever model that produces the outcome as a transforming function of parameters can be used.

RankBoost is a pairwise boosting approach proposed by Freund *et al* [10] in 2003. It is similar to AdaBoost [1] and largely acquires its properties. AdaBoost defines the distribution over a single document and RankBoost [10] over a pair of documents. RankBoost uses exponential function as a learning function defined over a pair of documents. It uses a linear combination of the weak rankers to produce the final list of ranked results. In every epoch, one weak learner is trained and finally all are combined to build the scoring function. Weights of each feature used as a weak learner are revised in every epoch.

LambdaMART is a pairwise boosting approach developed by Wu *et al* [20] in 2007. It is an ensemble method, which consists of boosted regression trees called MART [6] combined with LambdaRank [29]. LambdaRank is a method based on neural network algorithm. It works similar to RankNet [28] by utilizing back-propagation algorithm to minimize the loss function. The loss function of LambdaRank [29] is found to be better than the loss defined for RankNet in terms of execution time and performance on evaluation measures.

AdaRank is a boosting approach proposed by Xu and Li [11] in 2007. It works similar to RankBoost [10], except for the fact that it optimizes the performance measures like NDCG or MAP instead of optimizing the loss function. A loss function directly defined on the performance measures is optimized by employing a boosting technique in learning model. AdaRank takes the listwise approach to the learning-to-rank problem and tries to minimize the performance measures directly instead of indirect minimization of a loss function. With a defined number of iterations, AdaRank creates a weak ranker at every iteration and finally produces evaluating results at the end by linearly combining weak rankers. The features that have optimal weight performance among all features are chosen to be weak learners. The distribution of weights on training instances is calculated according to the performance of the ranking model created thus far.

Co-ordinate ascent (CA) is an optimization method proposed by Metzler and Croft [30] in 2007. It is a listwise approach and makes use of a linear model. Any performance ranking measure can be optimized by the CA method, which helps in learning the weights for the model. CA gets obstructed in local minimum while finding the global minimum of the MAP. This can be bypassed through numerous restarts.

ListNet is a listwise approach for learning-to-rank proposed by Cao *et al* [12] in 2007. It uses a neural network model whose parameters are learned by gradient descent method to optimize a listwise loss based on a probabilistic model of permutations. ListNet uses permutation probability and top k probability to transform ranking scores into probability distributions and utilizes cross entropy as the listwise loss function. The listwise loss function optimizes using a document list of objects at an instance [12].

ListMLE, a variation of ListNet, was developed Xia *et al* [13] in 2008. It employs the negative log likelihood of the permutation probability of human ranking as the loss function. It learns by employing the likelihood loss as the surrogate loss function. On training data, sum of the likelihood function is maximized with respect to all training queries. The stochastic gradient descent (SGD) algorithm is selected for conducting the minimization of loss function whereas linear neural network is chosen as the ranking model. The loss function of ListMLE has some strong statistical properties, including soundness and convexity.

3. Performance evaluation measures

The evaluation of a ranking model performance is measured by comparison between the ranking lists output given by the learning model and the ranking lists given as the ground truth. Performance evaluation measures used for learning-to-rank include Mean Reciprocal Rank (MRR), Mean Average Precision (MAP) [2], NDCG [31] and Expected Reciprocal Rank (ERR) [32], out of which only NDCG and MAP are popularly used.

3.1 MRR

For query q , if $R(q)$ denotes the position of the first relevant document found, then $1 / (R(q))$ is defined as MRR for query q . The documents whose ranking is lower than $R(q)$ are not entertained in MRR.

3.2 MAP

Firstly, Precision at position m ($P@m$) is defined as

$$P@m(q) = \frac{\#\{\text{relevant document in top } m \text{ positions}\}}{m}. \quad (6)$$

Average Precision (AP) is defined as

$$AP(q) = \frac{\sum_{m=1}^N P@m(q) \cdot I_m}{\#\{\text{relevant documents}\}}, \quad (7)$$

where N is the number of documents associated with query q and I_m is the relevance judgment for the document at the m^{th} position. MAP can be finally obtained after averaging AP for all the queries [2].

3.3 NDCG

Discounted cumulative gain (DCG) considers the order of documents. It gives more importance to the documents that are at higher ranks and penalizes those that occur at lower ranks. For a given query $q_i \in Q$, the list of ranks y_i and a permutation π_i on d_i , DCG at position m is defined as

$$DCG@m = \sum_{k=1}^m \frac{2^{\pi_{i,k}} - 1}{\log(i + 1)}. \quad (8)$$

$NDCG@m$ is obtained by normalizing $DCG@m$, i.e., $DCG@m$ is calculated for ideal ranking and denoted as $IDCG@m$:

$$NDCG@m = \frac{DCG@m}{IDCG@m}. \quad (9)$$

3.4 ERR

ERR is the expectation of the reciprocal of the position of a result at which a user stops. Suppose for a query q , list of ranked results is ranked N for documents $d_1, d_2, \dots, d_N$. ERR at position m is defined as

$$ERR@m = \sum_{m=1}^N \frac{1}{m} p(\text{user terminates at position } m), \quad (10)$$

$$\begin{aligned} p(\text{user terminates at position } m) \\ = p(q, d_m) \prod_{i=1}^{m-1} (1 - p(q, d_i)) \end{aligned} \quad (11)$$

where $p(q, d_m)$ is the probability that the user gets satisfied with document m and stops at rank m , which is calculated in accordance with grade assigned to the query–document pair by some transformation technique. This implies that the user does not get satisfied with previous $m - 1$ documents ranked from $1, \dots, m - 1$, whose probability is $\prod_{i=1}^{m-1} (1 - p(q, d_i))$.

4. System design for analysing learning-to-rank

In previous sections, we have discussed the different learning algorithms designed under learning-to-rank and their performance evaluation measures. This section expresses the system design required to evaluate performance, efficiency and accuracy of ranking model. All algorithms experimented with the standard datasets are specifically designed for learning-to-rank. Figure 2 shows our design of system, which is mainly considered with analysing learning model for ranking. Selected datasets are transformed into training, testing and validation set. Datasets comprise instances having features for a query–

Figure 2. Design of the system model for learning-to-rank.

document pair along with relevance judgments. Training set containing data with ground truths is used to fit the parameters, e.g., weights. Testing set is used to assess the performance of a model that is already trained. Validation set can be anticipated as a part of data used for training the model since it is utilized to construct the model by selecting parameters and evading overfitting. The training set is given to the model to learn the ranking function. The final model will be built after interpreting the performance on validation set. These models are evaluated on the testing set for the given algorithm and the conclusions are noted. The best and efficient algorithm will be identified after comparison of performances exhibited by different algorithms.

Feature extraction [33], feature selection [34–36] and feature ranking are the major points while considering accuracy and efficiency of the learning model designed for learning models. Like classification, feature selection is one of the major concerns for learning the model. A number of ranking models have been proposed with different machine learning methods for learning-to-rank and work is still in progress to find the efficient method.

The learning models introduced for ranking are analysed for the number of features used for learning the ranking models. Thus, we have worked on feature or dimensional reduction of the publicly available learning-to-rank datasets.

4.1 Feature selection for learning-to-rank

Feature selection refers to the process of identifying the few most important variables or features by the value of some scoring function/selection function, which usually measures importance of selected features in terms of efficiency or accuracy of learning model. Generally, feature reduction has been implemented using forward stepwise selection or filtering of features [34]. Some methods select subsets of features based on their expected divergence over the relevance classes and the importance of features is estimated by the evaluation scores produced individually by the features [35]. The justification for the requirement of feature selection is listed in the following points:

- i It helps in data compressing and reducing the storage space required.
- ii It reduces the time required for performing some computations. Less dimensions leads to less computing. Large number of features are also not suitable for learning algorithms.
- iii It takes care of multi-collinearity, which improves the model performance and also helps remove redundant features.

In forward stepwise selection, we start with a single feature only, progressively adding only one single feature at a time, i.e., feature that produces the highest increase in performance will be added to the selected list. The

value of Akaike information criterion (AIC) [37] determines whether a feature can be added or not. AIC value for each model is calculated with the same dataset and the 'best' model is the one with minimum AIC value. AIC has been calculated using residual sums of squares from regression:

$$AIC = n \times \ln \left(\frac{RSS}{n} \right) + 2K \quad (12)$$

where n is the number of data points (observations), RSS is the residual sums of squares and K is the number of parameters in the model. The second-order information criterion often called $(AIC)_c$ takes into account sample size by essentially increasing the relative penalty for model complexity with small datasets:

$$AIC_c = AIC + 2K \frac{(k+1)}{(n-K-1)}. \quad (13)$$

The model with the lowest $(AIC)_c$ is the model that provides the best trade-off between the complexity of the model and its ability to explain the data. Further, RSS is calculated using the following equation:

$$RSS = \sum_{i=1}^n (y_i - (\alpha + \beta x_i))^2. \quad (14)$$

4.2 Normalized winning number strategy

Qin *et al* [38] in 2010 defined the Winning Number (WN), which represents number of other algorithms that an algorithm can beat over the set of datasets. Formally

$$WN_x(PM) = \sum_{y=1}^N \sum_{z=1}^N IF_{(PM_x(y) > PM_z(y))} \quad (15)$$

where j is the index of a dataset, n is the number of datasets in the comparison, i and k are indices of an algorithm, $PM_x(y)$ is the performance of x^{th} algorithm on the y^{th} dataset and PM is a ranking measure such as NDCG. $IF_{(PM_x(y) > PM_z(y))}$ is an indicator function such that

$$IF_{(PM_x(y) > PM_z(j))} = \begin{cases} 1 & \text{if } PM_x(y) > PM_z(y) \\ 0 & \text{otherwise.} \end{cases} \quad (16)$$

Tax, Bockting and Hiemstra proposed a normalized version of the WN called *Normalized Winning Number* [39], which is a metric to enable comparison of a sparse set of evaluation results. This Normalized Winning Number (NWN) takes only those datasets into account over which an algorithm evaluates and divides this by the theoretically highest WN of an algorithm that would have been the most accurate algorithm on all datasets. It considers only those datasets for which algorithms are evaluated with different evaluating measures. The best method over

all methods with m datasets is indicated by the indicator function IF :

$$IF_{(PM_x(y) > PM_z(j))} = \begin{cases} 1 & \text{if } PM_x(y) \text{ and } PM_z(j) \text{ are defined} \\ & \text{and } PM_x(y) > PM_z(j) \\ 0 & \text{otherwise.} \end{cases} \quad (17)$$

Finally, NWN is defined as follows:

$$NWN_x(PM) = \frac{WN_x(PM)}{IWN_x(PM)}, \quad (18)$$

where IWN is the Ideal Winning Number defined as

$$IWN_x = \sum_{y=1}^N \sum_{z=1}^N EF_{(PM_x(y) > PM_z(j))} \quad (19)$$

where $EF_{(M_i(j), M_k(j))}$ is an evaluation definition function such that

$$EF_{(PM_x(y) > PM_z(j))} = \begin{cases} 1 & \text{if } PM_x(y) \text{ and } PM_z(j) \text{ are defined} \\ 0 & \text{otherwise.} \end{cases} \quad (20)$$

5. Experimental results and discussion

For performing the experiments, we have used public benchmark datasets released by Microsoft Research Asia [38] for learning-to-rank algorithms. Table 1 summarizes information for datasets used. We have used $NDCG$ as the performance measure in our work. $NDCG$ stands better when other performance measures are compared with it. In $NDCG$, we can have relevancy degrees for documents whereas other measures like MAP and MRR use binary relevance. A discount function is included in $NDCG$ over the ranks while many other measures uniformly weight all positions. This feature is particularly important for search

engines as users care about top-ranked documents much more than others.

Table 2 shows the performance of learning-to-rank algorithms on training data, testing data and validation data from LETOR dataset and figure 3 shows its graphical performance analysis. For pairwise classification approach, SVMRank has performed well on home page and named page finding datasets, viz., 2003 HP, 2003 NP, 2004 HP and 2004 NP. SVMRank performed fairly on the other datasets too as compared with other algorithms. On 2003 TD datasets, ListNet performed better than others whereas its variation algorithm ListMLE performed better on MQ2007 than other algorithms. On MQ2008, CA performed better. Random Forest outperforms all on 2004 TD dataset but its overall performance on other datasets is fair. MQ2007 and MQ2008 are the only datasets on which every algorithm has performed fairly. The pointwise approaches implemented, viz., MART and Linear Regression, show a very poor performance on LETOR dataset except Random Forest on 2004 TD dataset. Both pairwise and listwise approaches performed better than pointwise approach. Table 3 shows the performance of learning-to-rank algorithms on training data, testing data and validation data from OHSUMED dataset. OHSUMED dataset contains queries related to medical field. We can surely see that ListNet algorithm has performed better than others. SVMRank and CA have also performed fairly on OHSUMED dataset.

Table 4 shows the performance of learning-to-rank algorithms on training data, testing data and validation data from MSLR dataset. The MSLR datasets are very large in size and hence need longer time than other datasets. The pairwise approach LambdaMART has shown superior performance on both MSLR-WEB10K and MSLR-WEB30K datasets. ListNet has surprisingly performed the worst on both the datasets. Linear Regression also has performed poorly.

Table 5 shows the performance of RankNet algorithms on training data, testing data and validation data for selected datasets when different numbers of hidden layers

Table 1. Standard learning-to-rank dataset.

Dataset			Number of queries	Number of features	Number of features after reduction
LETOR	LETOR 3.0	2003 HP	150	64	42
		2003 NP	150	64	43
		2003 TD	50	64	44
		2004 HP	75	64	42
		2004 NP	75	64	50
		2004 TD	75	64	43
	OHSUMED		106	46	30
	LETOR 4.0	MQ2007	1700	46	29
		MQ2008	800	46	26
MSLR	MSLR-WEB10K		10000	136	118
	MSLR-WEB30K		30000	136	121

Table 2. Experimental results (*NDCG@10*) on LETOR datasets.

Algorithm/dataset	2003 HP	2003 NP	2003 TD	2004 HP	2004 NP	2004 TD	MQ2007	MQ2008
Training data								
SVMRank	0.8240	0.8099	0.3820	0.7930	0.7989	0.2978	0.4466	0.2312
MART	0.8402	0.8989	0.5201	0.7771	0.8141	0.5400	0.4711	0.5344
RankNet	0.5629	0.7044	0.2398	0.6898	0.7309	0.2586	0.4228	0.4834
RankBoost	0.8005	0.7725	0.3319	0.8296	0.7321	0.3126	0.4438	0.5165
AdaRank	0.7610	0.6158	0.3289	0.7348	0.6722	0.3018	0.4308	0.4978
Co-ordinate Ascent	0.8605	0.7320	0.4344	0.8231	0.7884	0.3741	0.4454	0.5178
LamdaMART	0.9214	0.7380	0.5183	0.8554	0.9351	0.5055	0.5038	0.5453
ListNet	0.7895	0.7201	0.3632	0.7217	0.7572	0.2863	0.4197	0.4871
Random Forests	0.9734	0.9800	0.8839	0.9734	0.9734	0.7134	0.4981	0.5891
ListMLE	0.7943	0.7705	0.3502	0.7599	0.7536	0.3101	0.4891	0.3088
Linear Regression	0.5682	0.6408	0.3766	0.5879	0.5789	0.3189	0.4327	0.4968
Testing data								
SVMRank	0.8180	0.7894	0.3571	0.7720	0.7950	0.2913	0.4436	0.2309
MART	0.6347	0.7121	0.2571	0.5343	0.5835	0.2616	0.4429	0.5036
RankNet	0.5266	0.7032	0.2354	0.6799	0.7261	0.2626	0.4213	0.4836
RankBoost	0.7033	0.6836	0.2808	0.6788	0.6305	0.3087	0.4333	0.5003
AdaRank	0.7151	0.5941	0.2652	0.7132	0.6342	0.2843	0.4300	0.4952
Co-ordinate Ascent	0.6939	0.6579	0.2846	0.7708	0.6670	0.3051	0.4371	0.5047
LamdaMART	0.6779	0.5523	0.2864	0.6133	0.5105	0.2682	0.4478	0.4933
ListNet	0.7895	0.7201	0.3632	0.7217	0.7572	0.2863	0.4197	0.4871
Random Forests	0.7182	0.7544	0.3588	0.6718	0.6611	0.3305	0.4411	0.4983
ListMLE	0.8004	0.7353	0.3269	0.7366	0.7302	0.2977	0.4894	0.3069
Linear Regression	0.5067	0.6218	0.3203	0.5536	0.5409	0.2750	0.4277	0.4871
Validation data								
SVMRank	0.8247	0.7952	0.3569	0.7908	0.8058	0.2939	0.4440	0.2287
MART	0.6758	0.7385	0.3071	0.5944	0.5834	0.3088	0.4455	0.5092
RankNet	0.5767	0.7157	0.2758	0.6981	0.7316	0.2860	0.4258	0.4881
RankBoost	0.7343	0.7344	0.3315	0.7402	0.6987	0.3141	0.4349	0.5077
AdaRank	0.7851	0.6254	0.3409	0.7615	0.6893	0.3171	0.4306	0.4996
Co-ordinate Ascent	0.6977	0.6589	0.3167	0.7927	0.7657	0.3394	0.4409	0.5083
LamdaMART	0.7416	0.6695	0.3259	0.6621	0.6600	0.3103	0.4516	0.5267
ListNet	0.8013	0.7500	0.3704	0.7486	0.7725	0.2896	0.4202	0.4902
Random Forests	0.7168	0.7483	0.3562	0.6517	0.6506	0.3568	0.4387	0.4973
ListMLE	0.8094	0.7705	0.3486	0.7973	0.7853	0.3165	0.4879	0.3117
Linear Regression	0.4887	0.6223	0.3163	0.5707	0.5287	0.2924	0.4272	0.4886

are used. Increase in the number of hidden layers does not guarantee an improvement in the performance. The performance of RankNet improved on 2003 HP and 2003 NP only when the number of hidden layers was increased from 1 to 2. All other datasets show good performance with a single hidden layer only.

Table 6 shows the performance of topmost algorithms on respective datasets before and after feature reduction. ListMLE shows considerable increase in performance on MQ2007 after feature reduction. The decrease shown by topmost learning-to-rank algorithms is not high. SVMRank on 2003 HP and 2003 NP and LambdaMART on MSLR-WEB10K and MSLR-WEB30K showed a gradual drop in their performances. The other algorithms exhibited a fair increase in their performance. Table 7 shows the calculations of *NWN* for learning-to-rank algorithms. According to

the results obtained for *NWN*, SVMRank having the highest *NWN* stands to be the best algorithm amongst other learning-to-rank algorithms. CA and Random Forests with the same *NWN* stand as fairly efficient algorithms followed by ListMLE and ListNet. Linear Regression has very less *NWN*, which has the least *NWN* amongst all. Thus, it can be inferred that Linear Regression should be avoided in learning-to-rank tasks.

Among all approaches of learning-to-rank, the listwise approach is widely used for most of the real-time applications. Listwise approach handles the problem of ranking in more natural way as compared with the pointwise and pairwise approaches. The pointwise approach only provides the score to the objects considered for ranking but it is not focused on the relative order among documents. Besides this, there is no transparency between the ranking position

Figure 3. Performance analysis of different learning-to-rank algorithms in terms of $NDCG$ on LETOR dataset.

Table 3. Experimental results ($NDCG@10$) on OSHUMED dataset.

Algorithm	Training data	Testing data	Validation data
SVMRank	0.4551	0.4504	0.4464
MART	0.5217	0.4279	0.4554
RankNet	0.4384	0.4451	0.4480
RankBoost	0.4785	0.4385	0.4581
AdaRank	0.4508	0.4478	0.4573
Co-ordinate Ascent	0.4671	0.4508	0.4577
LamdaMART	0.5671	0.4305	0.4619
ListNet	0.4697	0.4697	0.4648
Random Forests	0.6200	0.4383	0.4363
ListMLE	0.4403	0.4331	0.4513
Linear Regression	0.4362	0.4302	0.4209

Table 4. Experimental results ($NDCG@10$) on MSLR datasets.

Algorithm/dataset	Training data		Testing data		Validation Data	
	WEB10K	WEB30K	WEB10K	WEB30K	WEB10K	WEB30K
MART	0.4548	0.4598	0.4433	0.4551	0.4442	0.4552
RankNet	0.1948	0.2019	0.1964	0.2025	0.1967	0.2032
RankBoost	0.3417	0.3398	0.3353	0.3334	0.3353	0.3336
AdaRank	0.3419	0.3489	0.3431	0.3492	0.3435	0.3480
Co-ordinate Ascent	0.4251	0.4206	0.4215	0.4195	0.4214	0.4194
LamdaMART	0.4745	0.4709	0.4510	0.4615	0.4516	0.4616
ListNet	0.1803	0.1831	0.1803	0.1831	0.1805	0.1832
Random Forests	0.4136	0.4128	0.4095	0.4117	0.4104	0.4112
Linear Regression	0.2351	0.2384	0.2354	0.2384	0.2354	0.2383

Table 5. Comparison of RankNet results (*NDCG@10*) for different numbers of hidden layers.

Dataset	Training data		Testing data		Improvement (%)	Validation data	
	1 hidden layer	2 hidden layers	1 hidden layer	2 hidden layers		1 hidden layer	2 hidden layers
MQ2007	0.4229	0.3922	0.4236	0.3917	-7.53	0.4267	0.3934
2003 HP	0.5566	0.5567	0.5209	0.5347	2.66	0.5688	0.5939
2003 NP	0.6413	0.7018	0.6404	0.7145	11.57	0.6843	0.7157
2003 TD	0.2464	0.2444	0.2566	0.2221	-13.45	0.2728	0.2829
2004 HP	0.6814	0.3241	0.6911	0.3567	-48.39	0.7234	0.3574
2004 NP	0.7184	0.5013	0.7015	0.4912	-29.98	0.7424	0.5733
2004 TD	0.3257	0.2563	0.3181	0.2552	-19.78	0.3296	0.2580

Table 6. Effect of feature reduction on topmost algorithm for respective datasets.

Dataset	Algorithm	Before feature reduction	After feature reduction	Improvement (2003 HP)
SVMRank		0.8180	0.7895	-3.48
2003 NP	SVMRank	0.7894	0.7734	-2.03
2003 TD	ListNet	0.3632	0.3659	0.74
2004 HP	SVMRank	0.7720	0.7821	1.31
2004 NP	SVMRank	0.7950	0.8163	2.68
2004 TD	Random Forests	0.3305	0.3565	7.87
OHSUMED	ListNet	0.4697	0.4763	1.41
MQ 2007	ListMLE	0.4894	0.5568	13.77
MQ2008	Co-ordinate Ascent	0.5047	0.5066	0.38
MSLR-WEB10K	LamdaMART	0.4510	0.4485	-0.55
MSLR-WEB30K	LamdaMART	0.4615	0.4571	-0.95

Table 7. Calculations of Normalized Winning Number for learning-to-rank algorithms.

Algorithm	WN	IWN	NWN
SVMRank	70	90	0.7778
MART	45	106	0.4245
RankNet	34	106	0.3208
RankBoost	55	106	0.5189
AdaRank	51	106	0.4811
Co-ordinate Ascent	74	106	0.6981
LamdaMART	48	106	0.4528
ListNet	63	106	0.5943
Random Forests	74	106	0.6981
ListMLE	61	90	0.6778
Linear Regression	27	106	0.2547

and loss function. On the other hand, pairwise approach provides the relative order among documents by considering the pair of documents at a given instance. However, the number of ordered pairs considered for ranking will be in an exponential order for large number of documents. There is also no transparency between the loss function and ranking position like pointwise approach. The listwise approach tackles this problem in a more natural way. Most of the

listwise algorithms are designed with direct optimization of evaluating measures. However, large computational cost for training is one of the major threatening problems for listwise approach. For large ranking lists of items, the number of permutations is also large and thus it requires large computation to provide the correct ranking.

As we have seen in the Introduction, different machine learning methods are used to propose learning-to-rank models. All these methods have distinct characteristics and their own importance. More or less, these methods have the same computational complexity for any learning function. For example, single hidden layer neural network uses exactly the same form as the SVM. However, complexity will increase for multiple hidden layers with a slight improvement in accuracy. For SVM, kernel function is used for the same effect but computational complexity does not rise with increasing number of dimensions [27]. However, for neural network, it obviously rises with increasing number of neurons.

The ensemble learning methods such as bagging and boosting can stably improve the performance of ranking model while improvement obtained by the RankSVM is not obvious. However, ensemble methods of learning largely increase the space complexity and time complexity. Comparatively, kernelized RankSVM trains the ranking model

using subsets obtained by partitioning the original training set and has better trade-off between the accuracy and efficiency.

Boosting is one of the general methods of machine learning mainly used to improve performance of ranking model. It is a method for finding a highly accurate relevance pair on the training set by combining weak hypothesis, each of which needs to be only moderately accurate on training set. All the afore-mentioned experiments have shown impressive improvements in generalization behaviour and suggest that boosting method for ranking is more robust to overfitting. On the other hand, overfitting is one of the major challenges for ranking algorithms designed with neural network model.

Learning-to-rank algorithms proposed with different machine learning methods typically minimize a loss function loosely related to the evaluating measures, since there is a need to provide a solution for it.

6. Conclusion

A supervised machine learning approach for ranking of documents called learning-to-rank has been assessed in this paper. It has been increasingly getting attention since ranking is a key problem in IR from huge web data. Focusing on the objective, different learning-to-rank algorithms were implemented and compared in this article. We have studied different learning-to-rank methodologies. NWN has been used in this paper to find the best algorithm for efficient web search. According to NWN strategy, SVMRank stands to be the most efficient learning-to-rank algorithm. SVMRank has performed fairly on all the datasets. It is inferred that SVM maximizes the margin on the constraint set of the training data. SVM will remain preferred because of its ability to learn arbitrary features automatically, to make fewer assumptions and to be more expressive. CA, Random Forest and ListMLE performed fairly well on public benchmark datasets. CA converges slowly on objective functions. Random Forest, though a pointwise approach, has surprisingly performed better than other pairwise approaches. Linear Regression exhibited very poor performance irrespective of the dataset. It achieved the least NWN amongst all. It looks only at linear relationships between dependent and independent variables, i.e., it assumes straight-line relationship between them, which is incorrect and leads to a performance fall-off. Linear Regression must be avoided in learning-to-rank because of its limitations. Moreover, feature reduction is also implemented using forward feature selection method of feature selection. Noticeable improvements are observed in the topmost performing algorithms using the feature selection method. ListMLE gained the maximum increase in performance amongst other learning-to-rank algorithms. The same topmost algorithm on different datasets showed varying

performance but no algorithm showed a considerable decrease in performance. Learning-to-rank is currently the most trending topic in machine learning. It is developing at a massive pace. This work has focused on providing the correct guidelines for future researchers in learning-to-rank. As a future work, more algorithms can be tested on benchmark datasets and compared to the current results. Other feature reduction techniques like principal component analysis (PCA), backward elimination and low variance can be implemented to check their effect on learning-to-rank algorithms. The research can be directed more towards pairwise and listwise approaches and developing new algorithms.

References

- [1] Yoav F and Schapire R E 1997 A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* 55(1): 119–139
- [2] Baeza-Yates R 1999 *Modern information retrieval*. Pearson Education India, India
- [3] Robertson S E 1997 Overview of the Okapi projects. *J. Doc.* 53: 3–7
- [4] Ponte J M and Croft W B 1998 A language modeling approach to information retrieval. In: *Proceedings of the Special Interest Group on Information Retrieval Conference*, pp. 275–281
- [5] Page L, Brin S, Motwani R and Winograd T 1998 The pagerank citation ranking: bringing order to the web. *Technical Report, Stanford Digital Library Technologies Project*
- [6] Liu T Y 2011 *Learning to Rank for Information Retrieval*. Springer Science and Business Media
- [7] Friedman J H 2001 Greedy function approximation: a gradient boosting machine. *Technical Report, IMS Reitz Lecture, Stanford*
- [8] Breiman L 2001 Random forests. *Mach. Learn.* 45: 5–32
- [9] Thorsten J 2002 Optimizing search engines using click-through data. In: *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 133–142
- [10] Freund Y, Iyer R, Schapire R and Singer Y 2003 An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.* 4: 933–969
- [11] Xu J and Li H 2007 AdaRank: a boosting algorithm for information retrieval. In: *Proceedings of the Special Interest Group on Information Retrieval Conference*, pp. 391–398
- [12] Cao Z, Qin T, Liu T Y, Tsai M and Li H 2007 Learning to rank: from pairwise approach to listwise approach. In: *Proceedings of the 24th International Conference on Machine Learning*, ACM, pp. 129–136
- [13] Fen X, Liu T Y, Wang J, Zhang W and Li H 2008 Listwise approach to learning to rank: theory and algorithm. In: *Proceedings of the 25th International Conference on Machine Learning*, ACM, pp. 1192–1199
- [14] Chapelle O and Wu M 2010 Gradient descent optimization of smoothed information retrieval metrics. *Inf. Retr.* 13(3): 216–235

- [15] Tao Q, Liu T Y and Li H 2010 A general approximation framework for direct optimization of information retrieval measures. *Inf. Retr.* 13(4): 375–397
- [16] Taylor M, Guiver J, Robertson S and Minka T 2008 Sof-trank: optimizing non-smooth rank metrics. In: *Proceedings of the 2008 International Conference on Web Search and Data Mining*, ACM, pp. 77–86
- [17] Xu J, Liu T Y, Lu M, Li H and Ma W Y 2008 Directly optimizing evaluation measures in learning to rank. In: *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, ACM, pp. 107–114
- [18] Ou W, You Q, Mao X, Xia F, Yuan F and Hu W 2016 Listwise learning to rank by exploring structure of objects. *IEEE Trans. Knowl. Data Eng.* 28(7): 1934–1939
- [19] Lawson C L and Hanson R J 1995 *Solving least squares problems*. Society for Industrial and Applied Mathematics
- [20] Wu Q, Burges C J C, Svore K and Gao J 2007 Adapting boosting for information retrieval measures. *J. Inf. Retr.* 13(3): 254–270
- [21] Ibrahim O A S and Landa-Silva D 2017 Es-rank: evolution strategy learning to rank approach. In: *Proceedings of the Symposium on Applied Computing*, ACM, pp. 944–950
- [22] Yue Y, Finley T, Radlinski F and Joachims T 2007 A support vector method for optimizing average precision. In: *Proceedings of the 30th Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, ACM, pp. 271–278
- [23] Li P, Wu Q and Burges C J 2008 Mcrank: learning to rank using multiple classification and gradient boosting. In: *Proceedings of Advances in Neural Information Processing Systems*, pp. 897–904
- [24] Crammer K and Singer Y 2002 Pranking with ranking. In: *Proceedings of Advances in Neural Information Processing Systems*, pp. 641–647
- [25] Cossock D and Zhang T 2006 Subset ranking using regression. In: *Proceedings of the International Conference on Computational Learning Theory*, Springer, pp. 605–619
- [26] Shashua A and Levin A 2003 Ranking with large margin principle: two approaches. In: *Proceedings of Advances in Neural Information Processing Systems*, pp. 961–968
- [27] Chen K, Li R, Dou Y, Liang Z and Lv Q 2017 Ranking support vector machine with kernel approximation. *Comput. Intell. Neurosci.* Article ID 4629534
- [28] Burges C J C, Shaked T, Renshaw E, Lazier A, Deeds M, Hamilton N and Hullender G 2005 Learning to rank using gradient descent. In: *Proceedings of the International Conference on Machine Learning*, pp. 89–96
- [29] Burges C J, Ragno R and Le Q V 2007 Learning to rank with nonsmooth cost functions. *Proc. Adv. Neural Inf. Process. Syst.* 19:193–200
- [30] Metzler D and Bruce Croft W 2007 Linear feature-based models for information retrieval. *Inf. Retr.* 10(3): 257–274
- [31] Wang Y, Wang L, Li Y, He D, Chen W and Liu T Y 2013 A theoretical analysis of NDCG ranking measures. In: *Proceedings of the 26th Annual Conference on Learning Theory (COLT 2013)*, vol. 8
- [32] Chapelle O, Metzler D, Zhang Y and Grinspan P 2009 Expected reciprocal rank for graded relevance. In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ACM, pp. 621–630
- [33] Pandey G, Ren Z, Wang S, Veijalainen J and de Rijke M 2018 Linear feature extraction for ranking. *Inf. Retr. J.* 21(6): 481–506
- [34] Gigli A, Lucchese C, Nardini F M and Perego R 2016 Fast feature selection for learning to rank. In: *Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval*, ACM, pp. 167–170
- [35] Gupta P and Rosso P 2012 Expected divergence based feature selection for learning to rank. In: *Proceedings of International Committee on Computational Linguistics 2012*, pp. 431–440
- [36] Lai H J, Pan Y, Tang Y and Yu R 2013 Fsmrank: feature selection algorithm for learning to rank. *IEEE Trans. Neural Netw. Learn. Syst.* 24(6): 940–952
- [37] Stone M 1977 An asymptotic equivalence of choice of model by cross-validation and Akaike's criterion. *Journal of the Royal Statistical Society, Series B (Methodological)* 39(1): 44–47
- [38] Tao Q, Liu T Y, Xu J and Li H 2010 LETOR: a benchmark collection for research on learning to rank for information retrieval. *Inf. Retr.* 13(4): 346–374
- [39] Tax N, Bockting S and Hiemstra D 2015 A cross-benchmark comparison of 87 learning to rank methods. *Inf. Process. Manag.* 51(6): 757–772