Performance Comparison of Learning to Rank Algorithms for Information Retrieval

Ridho Reinanda  
Institute for Informatics  
University of Amsterdam  
Amsterdam, Netherland  
R.Reinanda@uva.nl

Abstract—Learning to rank is the problem of ranking objects by using machine learning techniques. One of the applications of learning to rank is for ranking document of search results. In this research, we compare the performance of three learning to rank algorithms: RankSVM, LambdaMART, and Additive Groves. RankSVM, which is ranking variant of the classical SVM algorithm, is commonly used as a baseline in learning to rank experiments. LambdaMART and Additive Groves is both tree ensembles algorithm. They belong to the class of algorithms that yield top results in the recent Yahoo! Learning to Rank Challenge. The comparison is performed by evaluating the results algorithms to a standard dataset. We also study the outcome of performing feature selection for final algorithm performance. Two feature selection techniques based on the filter and wrapper approaches are implemented. The experiment is conducted with the LETOR 4.0 dataset. This dataset consists of 46 features, such as: TF, BM25, LMIR, PageRank scores, etc. Parameter tuning is performed beforehand in order to find the best parameters for reliable comparison. The evaluation metric is Normalized Discounted Cumulative Gain (NDCG), a graded relevance measure for ranking results. The experimental results reveal that Learning to Rank algorithms outperform the conventional algorithms. In addition, applying feature selections improves the algorithm performances.

Keywords—AAAZ learning to rank algorithms; RankSVM; Lambda MART; Additive Group; Feature Selection.

I. INTRODUCTION

In information retrieval, the rank of a document is determined by the relevance of the document to a query. This rank can be obtained by computing the score of all documents with respect to the query, then sorting the documents based on the resulting score. Another possible approach is by applying machine learning. With machine learning approach, a ranking system ranks documents based on a ranking model that is derived from learning process. A search system has a learner component that processes training data to build a ranking model. This ranking model will be used by the main system to make predictions. The resulting prediction is in the form of relevance grade, which can be used to rank documents with respect to a given query.

The learning algorithm is trained against training data that represent query and document in the form of feature vectors, with labels that indicate the relevance between the query and the document. This label can be in the form of binary relevance (i.e. relevant, not relevant) or in graded relevance (i.e. not relevant, relevant, and extremely relevant). The goal of the learning phase is to derive a model that can perform this prediction to unseen data. This application of machine learning to perform ranking is called learning to rank. Learning to rank algorithms is classified into three main groups: the pointwise, pairwise, and listwise approach [1].

Few previous researches were focused on comparison of learning to rank algorithms. Beside the Yahoo! Learning to Rank Challenge, a comparison of several learning to rank algorithms has also been performed by Zhang et. al. [2]. In their work, the comparison of several learning to rank algorithms is conducted on LETOR 3.0 dataset and WebSE dataset, which is built exclusively for the research. The Yahoo! Learning to Rank Challenge introduced several best performing algorithms. Most of these algorithms were only tested on the Yahoo dataset. Our main motivation in this research is to compare the performance of the learning to rank algorithms in the Yahoo! Learning to Rank Challenge with other publicly available dataset. It is interesting to find out if the performance of these algorithms are consistent in different datasets.

The outcome of applying feature selection to the performance of learning to rank algorithms has not also been studied extensively. Because learning to rank data contains many features (some of them possibly irrelevant), it is suspected that feature selection can improve the final ranking performance. Therefore, we also try to study the effect of applying feature selection to the compared algorithms.

The rest of the paper is organized as follows. Section II describes previous works related to Learning to Rank algorithms. Sections III & IV describe Learning to Rank algorithms and Feature selection methods employed in this research, respectively. The experiment setup and results are
discussed in Sections IV & V, followed by conclusion in Section VI.

II. RELATED WORK

Yahoo held a competition in 2010 to test the performance of learning to rank algorithms using performance measures such as Expected Reciprocal Rank (ERR) and Normalized Discounted Cumulative Gain (NDCG) [1]. This competition is performed by running the algorithms to a dataset that is constructed from Yahoo query logs. Most of the best algorithms in this competition use ensemble learning techniques, which combines several learning models.

This competition introduces several top algorithms: Ensemble LambdaMART [3], BagBoo, and Additive Groves [4]. The winners all used algorithms based on decision tree. Ensemble LambdaMART, BagBoo, and Additive Groves all use tree in the model of the algorithms. Ensemble methods, i.e. boosting, bagging, and random forest, are dominant techniques that yield best performances. Additive Groves uses bagging, LambdaMART uses boosting, while BagBoo uses a combination of bagging and boosting. Based on the final score of the algorithms, there is little difference in accuracy between the winning approaches.

Some feature selection techniques have been applied for ranking problems. Gang et. al. [5] computes the importance of each feature based on MAP and NDCG score, then performs discounting to the importance score based on the similarity between features. The optimization problem between importance and similarity is solved by modeling the problem as a graph where the feature is represented by a node, weighted by its importance score, and the edge represents the similarity between two features. It prefers features with highest importance scores and minimal similarity. Pan et al. [6] employ boosted trees to assess the feature importance, which is calculated by averaging the importance weight from all trees involved. Similarly, Sorokina [7] also uses ensemble of trees for feature selection but employing bagging method instead for generating the trees. Features are re-ranked from among the best ensembles based on their ability to discriminate data. Dang & Croft [8] conduct feature selection based on Best First Search strategy for identifying the best feature subset, combined with Coordinate Ascent learning algorithm as black box to perform evaluation on the selected features. Hierarchical feature selection has been proposed by Hua et al. [9]. They first cluster the features using k-means clustering method, and then select representatives from each cluster through various delegation methods. Feature similarity is calculated using Kendall’s Tau.

Previous approaches mostly used advanced techniques, by applying discounting, clustering, bagging, and boosting techniques to the algorithm and dataset. On the other hand, the applications of a more straightforward method like simple filter and wrapper for the ranking problem has never been studied. Therefore, in this research we implement two feature selection methods, each based on the filter and wrapper approaches.

III. LEARNING TO RANK ALGORITHMS

Learning to rank algorithms can be categorized into three approaches: the pointwise, pairwise, and listwise approach. The pointwise approach simplifies the learning to rank problem into regression of document relevance. The pairwise approach is formulated as the problem of classifying preference between a pair of documents in the same query. The listwise approach performs learning directly to the document list that has been ranked as a result of a query.

A learning to rank pipeline is illustrated on Figure 1. The input data in learning to rank process is already in the form of feature vectors. All features are in continuous values. Basic preprocessing is later carried on the input data. The example of required preprocessing is the normalization of feature values into 0-1 range. Sometimes, additional preprocessing is required so that the input data suits the requested format of the algorithm implementation. This is necessary because some algorithm implementation expect a specifically formatted data for the optimality of computation.

![Figure 1. Learning to rank process.](image)

After the preprocessing, learning model can be directly built using all features. Feature selection can be performed beforehand with filter or wrapper methods. Feature selection with filter methods is only performed once without any optimization to underlying ranking algorithm. With wrapper method, several learning to rank models are built and evaluated to find the best model for the task at hand. The input data will be processed again later for final ranking according to the result of feature selection.

From the prepared data, the learning algorithms will process the data according to their specific training algorithm. The result of the learning phase is a model that
can be used to perform predictions. The structure of the model is determined by each learning algorithms. The model of RankSVM for example, is a linear combination of weight for each feature. For tree based ensemble algorithms, the resulting models are regression trees.

The resulting learning model can be used to generate predictions. The prediction can be in the form of direct estimation of relevance grade, or any continuous value that can be used to rank the documents, although the value may not directly represent absolute relevance between query and documents.

In the following we will describe three main learning to rank algorithms that we employed for performance comparison. These algorithms include RankSVM, LambdaMART, and Additive Groves.

A. RankSVM

RankSVM [10] is a variant of Support Vector Machine (SVM) algorithm designed specifically for learning to rank problem. RankSVM is a pairwise algorithm, meaning that the classifier is aimed to distinguish a more relevant document between two documents in the same query. The result of the classification between these two documents is called pairwise preference or pairwise constraint. Pairwise preference will later be used to infer a complete ranking of all the documents in the query.

The optimization that is performed during the learning phase of RankSVM is against the number of correctly classified pair of documents with respect to all preference pairs on the training set. The goal of the learning process is to build a model that can classify pair of instance as correctly or incorrectly classified (minimizing swapped pairs).

Herbrich et. al. [11] is the one who first formulates this ordinal regression problem into the problem of pair preference optimization. The goal of the learning is to find function $h$, such that for each pairs of document in the same query, this following constraint is satisfied:

$$h(x_i) > h(x_j) \iff y_i > y_j$$  \hspace{1cm} (1)

Given a training set $(x_1, y_1), ..., (x_n, y_n)$, $P$ denotes the set of pairs $(i, j)$ where example $i$ has higher rank than example $j$, which is $P = \{(i, j) : y_i > y_j\}$, and define $m$ as $|P|$. The learning process will try to find a function $h(x)$ that will minimize this $m$, the number of swapped pairs $i$ and $j$.

B. LambdaMART

LambdaMART is a combination between MART and LambdaRank [3]. MART is a boosted tree model, which means each tree component of the ensemble estimates a residual value of the previous trees’ prediction. These tree components are trained in boosting iterations. The final output of the model is a linear combination of the output of several regression trees. Each tree has weight $w$ and function $f$, which is learned during the training process. The function $f$ maps a data point $x$ into a value by evaluating the features of $x$ along the tree model. The output of the values is directly associated with each leaf.

MART performs gradient descent to minimize the number of error in each stage. Each regression tree in the model represents the derivates of the cost with respect to the current model $F_n$ that has been built. This cost function can be defined according to requirements, for example NDCG. The algorithm will construct and optimize a representative of this cost function during the training process.

NDCG is a non-continuous function, since it depends on the resulting ranking instead of actual document similarity score. It is impossible to optimize NDCG directly, therefore in the training of LambdaMART, an approximation of this cost is derived by combining pairwise cross-entropy loss (RankNet cost) the delta of NDCG gained by swapping document pairs [3]. The delta NDCG optimizations contributes to the listwise nature of the algorithm.

C. AdditiveGroves

Additive Groves [4] is regression tree-based algorithm that combines bagging and boosting techniques. Basically, this algorithm builds a lot of weak learner in the form of regression trees. The trees are grouped in groves. Each grove is derived by bagging. This algorithm is a pointwise approach, because it performs value estimation directly to the relevance grade by performing Root Mean Square Error (RMSE) optimizations during the training process.

Additive groves consist of several additive models that are built using bagging procedures. Each element of the additive model (grove) is a tree. Each tree is trained based on the residual of the prediction from the other trees. The training is performed with the back fitting technique, building the trees repeatedly until there is no more improvement in the RMSE performance. Bagging is performed on top of each grove to reduce variance. Prediction of each grove is computed by the summations of the prediction from each tree in the groves:

$$F(x) = T_1(x) + T_2(x) + ... + T_n(x)$$  \hspace{1cm} (2)

IV. EVALUATION

This section evaluates the Learning to Rank Algorithms as described in preceding section. In the following will describe the experiment setup, and then discuss the results of the experiments.

A. Experimental Setup

For the experiment, we used the following algorithm implementations: RankSVM implementation from SVMLight package by Torsten Joachims [10],
AdditiveGroves implementation from the TreeExtra package by Daria Sorokina [4], and LambdaMART implementation from the jforests package by Yasser [12]. These algorithms will be tested with MQ2008 data, a subset of LETOR 4.0 dataset. We also perform the experiments without and with feature selection, as well as varying the method used for feature selection.

In this research, two feature selection methods are considered, each based on filter and wrapper approach. The filter-based approach uses top-k importance. In this method, each feature can be used to build a ranking model. For each query, the value of the feature is directly used to rank the documents. The final ranking performance achieved by sorting documents using the feature is then evaluated. Top features are later used in building the main ranking model. The wrapper-based method employs Greedy Backward Elimination [13], where an underlying algorithm is used to evaluate the selected subset. In every step, the method always chooses the subset that improves current performance. To perform greedy backward elimination, we start with full feature set, eliminating features one by one. A feature to be eliminated in each step is determined by building a model without the feature, then evaluating it against a separate data with a black box algorithm. Every time elimination results in increasing performance, we proceed to choose the next feature for elimination. The elimination process terminates when no feature elimination can improve the performance.

Preliminary experiment is conducted to find the best hyperparameters for each learning algorithm. This search for parameter is performed by evaluating the algorithm against a validation set. For RankSVM we performed search on the regularization parameter (C), and finally chose C=100. For LambdaMART, we varied two parameters: maximum number of leaves in a tree (m), and minimum number of observations per leaves (obs). Based on grid search, we finally use the value m = 14 of and obs = 0.3. For Additive Groves there are three parameters that must be tuned: the number of trees in a grove (N), the minimum number of observation per leaves (obs), and the number of bagging iterations (B). To determine these parameters, we use the automated training script from the Tree Extra package [4]. This training script guarantees that the algorithm will use the best parameters for the training data. After the parameters for the algorithms are determined, the main experiment is performed.

In our experiments, we aim to answer these following research questions:

- **RQ1**: How does the learning to rank algorithms performs against conventional approach?
- **RQ2**: How does feature selection affect the performance?
- **RQ3**: How does our result compare against the result of Yahoo! Learning to rank challenge?

As already mentioned earlier, for performance measure we use Normalized Discounted Cumulative Gain (NDCG). Given a list of ranked documents, NDCG assumes that a higher ranked document has higher gain than the other lower ranked documents. The score is then normalized by ideal DGC. More specifically, this performance measure is calculated as follows:

$$NDCG = \frac{DCG}{Ideal\ DCG} \quad DCG = \sum_{i=1}^{\min(10,m)} \frac{2^{r_i} - 1}{\log_2(i+1)}$$

where the score is calculated from the top ten documents (i.e., NDCG@10).

**B. Results of Experiments**

The main results of the experiments are listed on Tables I-III. At the beginning, to answer RQ1, an experiment is conducted to compare the performance of learning to rank algorithms against conventional approaches like BM25 or TF-IDF. Table I shows that all three learning to rank algorithms successfully outperform TF-IDF and BM25, achieving as high as 0.11 difference (15% improvement) in NDCG@10 performance (Additive Groves vs. TF-IDF). Base algorithm performance before applying feature selection showed that Additive Groves is the best method, followed by LambdaMART and RankSVM.

**TABLE I. PERFORMANCE COMPARISON BETWEEN BASE ALGORITHMS AND LEARNING TO RANK ALGORITHMS**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-IDF</td>
<td>0.7051</td>
</tr>
<tr>
<td>BM25</td>
<td>0.7800</td>
</tr>
<tr>
<td>RankSVM</td>
<td>0.8087</td>
</tr>
<tr>
<td>LambdaMART</td>
<td>0.8092</td>
</tr>
<tr>
<td>AdditiveGroves</td>
<td><strong>0.8165</strong></td>
</tr>
</tbody>
</table>

**TABLE II. ALGORITHM PERFORMANCE WITH FILTER FEATURE SELECTION**

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RankSVM + Top-10</td>
<td>0.7972</td>
</tr>
<tr>
<td>RankSVM + Top-20</td>
<td><strong>0.8182</strong></td>
</tr>
<tr>
<td>RankSVM + Top-30</td>
<td>0.8145</td>
</tr>
<tr>
<td>LambdaMART + Top-10</td>
<td>0.8115</td>
</tr>
<tr>
<td>LambdaMART + Top-20</td>
<td><strong>0.8150</strong></td>
</tr>
<tr>
<td>LambdaMART + Top-30</td>
<td>0.8088</td>
</tr>
<tr>
<td>AdditiveGroves + Top-10</td>
<td>0.6855</td>
</tr>
</tbody>
</table>
TABLE III. ALGORITHM PERFORMANCE WITH WRAPPER FEATURE SELECTION WITH GREEDY BACKWARD ELIMINATION (GBE)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NDCG@10</th>
</tr>
</thead>
<tbody>
<tr>
<td>RankSVM + GBE</td>
<td>0.8155</td>
</tr>
<tr>
<td>LambdaMART + GBE</td>
<td>0.8128</td>
</tr>
</tbody>
</table>

Next, we proceed on to answer RQ2. The results in Tables II & III show that feature selection, both for the filter and wrapper methods, successfully improves the performance of the algorithms. Around twenty features are enough to achieve a better or comparable performance against a full feature set. More or less than twenty features tend to degrade the algorithm performance. Simple filter method performed well, improving the performance of RankSVM and LambdaMART algorithms. For Additive Groves, using just the top ten features caused a significant drop to the performance. This probably happens because Additive Groves is a non-linear model that uses a lot of low-ranked features in its model. As shown in Tables II & III, the performances between optimal filter feature selection and wrapper feature selection with greedy backward elimination are comparable.

TABLE IV. ABSOLUTE SCORES FROM YAHOO! LEARNING TO RANK CHALLENGE[1]

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>NDCG@10 validation set</th>
<th>NDCG@10 test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>RankSVM (baseline)</td>
<td>0.7323</td>
<td>0.7592</td>
</tr>
<tr>
<td>Additive Groves</td>
<td>0.8010</td>
<td>0.8018</td>
</tr>
<tr>
<td>Ensemble</td>
<td>0.7995</td>
<td>0.8041</td>
</tr>
<tr>
<td>LambdaMART</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We move on to answer our last question (RQ3). In comparison with the Yahoo! Learning to Rank Challenge (see Table 4), the results are quite different. Even the RankSVM that serves as baseline can outperform the LambdaMART and AdditiveGroves with proper feature selection. There are two possible reasons for this. Firstly, the implementation of the LambdaMART and Additive Groves algorithms that are used in the challenge is quite different with the one used in this experiment. The LambdaMART algorithm employed in the Yahoo! Learning to Rank Challenge is the ensemble version, including several LambraMART trained on different parameters, and other related algorithms such as LambdaRank and logistic regression. The version of Additive Groves algorithm in the challenge used additional preprocessing and feature selection. Secondly, the dataset that is used in the challenge is much larger, with hundreds of features. Smaller dataset, with much less features possibly made the performance difference of these state-of-the-art algorithms not too noticeable against the baseline (RankSVM). These differences may explain the difference to final algorithm performance in our experiment.

V. CONCLUSION

We performed an experimental comparison of the basic version of state-of-the-art learning to rank algorithms. Evaluated against the MQ2008 dataset, the performances of the algorithms were slightly different from the Yahoo! Learning to Rank Challenge results. The best algorithms in order of NDCG performance measure are: Additive Groves, LambdaMART, and RankSVM. We show that simple feature selection successfully improved the performance of RankSVM and LambdaMART, changing the order of the algorithm by performance: RankSVM, LambdaMART, and Additive Groves. Based on our experiments, filter approach, with less computational cost, is slightly better than the wrapper approach in improving the underlying algorithm’s performance.

REFERENCES

This page is left blank on purpose