Learning to Rank

Hang Li

1 Abstract

Many tasks in information retrieval, natural language processing, and data mining are essentially ranking problems. These include document retrieval, expert search, question answering, collaborative filtering, and keyphrase extraction. Learning to rank is a sub-area of machine learning, studying methodologies and theories for automatically constructing a model from data for a ranking problem [1, 2, 3].

Learning to rank is usually formalized as a supervised learning task, while unsupervised learning and semi-supervised learning formulations are also possible. In learning, training data consisting of sets of objects as well as the total or partial orders of the objects in each set is given, and a ranking model is learned using the data. In prediction, a new set of objects is given, and a ranking list of the objects is created using the ranking model.

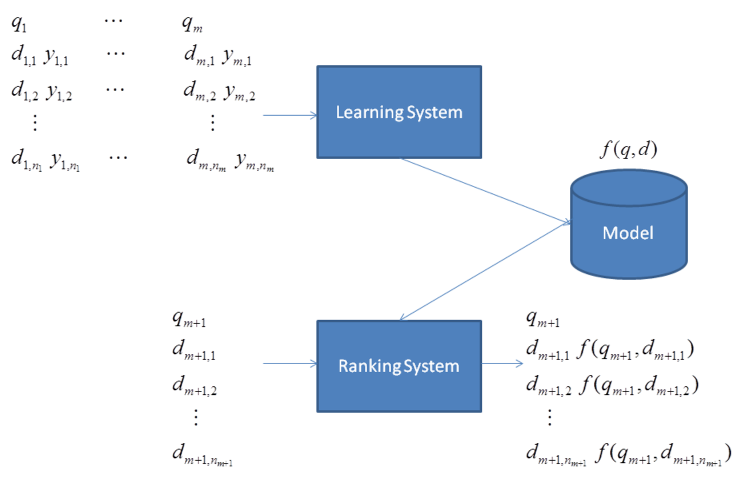
Learning to rank has been intensively studied in the past decade and many methods of learning to rank have been proposed. Popular methods include Ranking SVM, IR SVM, AdaRank, LambdaRank, and LambdaMART. The methods can be categorized into the pointwise, pairwise, and listwise approaches according to the loss functions which they use. It is known that learning to rank methods, such as LambdaMART, are being employed in a number of commercial web search engines.

In this article, we describe the formulation as well as several methods of learning to rank. Without loss of generality, we take document retrieval as example.

2 Solution

2.1 Problem Formulation

In the supervised learning setting, learning to rank includes training and testing phases (see Fig. 1). In training, the learning system learns a ranking model from given training data, and in testing, given a query the ranking system assigns scores to documents with respect to the query using the ranking model and sorts the documents on the basis of the scores.



q_1 $\cdots$ q_m
$d_{1,1}\ y_{1,1}$ $\cdots$ $d_{m,1}\ y_{m,1}$
$d_{1,2}\ y_{1,2}$ $\cdots$ $d_{m,2}\ y_{m,2}$
$d_{1,n_1}\ y_{1,n_1}$ $\cdots$ $d_{m,n_m}\ y_{m,n_m}$
Learning System
$f(q,d)$
Model
q_{m+1}
$d_{m+1,1}$
$d_{m+1,2}$
$d_{m+1,n_{m+1}}$
Ranking System
q_{m+1}
$d_{m+1,1}\ f(q_{m+1},d_{m+1,1})$
$d_{m+1,2}\ f(q_{m+1},d_{m+1,2})$
$d_{m+1,n_{m+1}}\ f(q_{m+1},d_{m+1,n_{m+1}})$

Figure 1: An overview of learning to rank for document retrieval.

The training data consists of queries and documents. Each query is associated with a number of documents. The relevance of the documents with respect to the query is also given. The relevance can be represented in several ways. Here, we take the most widely used approach and assume that the relevance of a document with respect to a query is represented by a label, while the labels denote several grades (levels). The higher grade a document has, the more relevant the document is.

Suppose that $\mathcal{Q}$ is the query set and $\mathcal{D}$ is the document set. Suppose that $\mathcal{Y} = \{1, 2, \cdots, l\}$ is the label set, where labels represent grades. There exists a total order between the grades $l \succ l-1 \succ \cdots \succ 1$, where $\succ$ denotes the order relation. Further suppose that $Q = \{q_1, q_2, \cdots, q_m\}$ is the set of queries for training and q_i is the i-th query. $D_i = \{d_{i,1}, d_{i,2}, \cdots, d_{i,n_i}\}$ is the set of documents associated with query q_i and $\mathbf{y}_i = \{y_{i,1}, y_{i,2}, \cdots, y_{i,n_i}\}$ is the set of labels associated with query q_i, where n_i denotes the sizes of D_i and $\mathbf{y}_i$; $d_{i,j}$ denotes the j-th document in D_i; and $y_{i,j} \in Y$ denotes the j-th grade label in $\mathbf{y}_i$. The original training set is denoted as $S = \{(q_i, D_i), \mathbf{y}_i\}_{i=1}^{m}$.

A feature vector $x_{i,j} = \phi(q_i, d_{i,j})$ is created from each query-document pair $(q_i, d_{i,j}), i = 1, 2, \cdots, m; j = 1, 2, \cdots, n_i$, where ϕ denotes the feature functions. That is to say, features are defined as functions of a query document pair. For example, BM25 and PageRank are typical features. Letting $\mathbf{x}_i = \{x_{i,1}, x_{i,2}, \cdots, x_{i,n_i}\}$, the training data set is also represented as $S' = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$. Here $x \in \mathcal{X}$ and $\mathcal{X} \subseteq \Re^d$, where d denotes number of features.

We aim to train a (local) ranking model $f(q, d) = f(x)$ that can assign a score to a given query document pair q and d, or equivalently to a given feature vector x. More generally, we can also consider training a global ranking model $F(q, D) = F(\mathbf{x})$.

Let the documents in D_i be identified by the integers $\{1, 2, \cdots, n_i\}$. We define a permutation (ranking list) π_i on D_i as a bijection from $\{1, 2, \cdots, n_i\}$ to itself. We use Π_i to denote the set of all possible permutations on D_i, use $\pi_i(j)$ to denote the rank (or position) of the j-th document (i.e., $d_{i,j}$) in permutation π_i. Ranking is nothing but to select a permutation $\pi_i \in \Pi_i$ for the given query q_i and the associated documents D_i using the scores given by the ranking model $f(q_i, d_i)$.

The test data consists of a new query q_{m+1} and associated documents D_{m+1}. We create feature vector $\mathbf{x}_{m+1}$, use the trained ranking model to assign scores to the documents D_{m+1}, sort them based on the scores, and give the ranking list of documents as output π_{m+1}.

The training and testing data is similar to, but different from the data in conventional supervised learning such as classification and regression. Query and its associated documents form a group. The groups are i.i.d. data, while the instances within a group are not i.i.d. data. A local ranking model is a function of a query and a document, or equivalently, a function of a feature vector derived from a query and a document.

Evaluation on the performance of a ranking model is carried out by comparison between the ranking lists output by the model and the ranking lists given as the ground truth. Several evaluation measures are usually used in information retrieval, such as NDCG (Normalized Discounted Cumulative Gain), DCG (Discounted Cumulative Gain), MAP (Mean Average Precision), and Kendall's Tau.

2.2 Evaluation Measures: DCG and NDCG

We give definitions of DCG and NDCG; both are most utilized in document retrieval. NDCG is normalized and thus it is easier to use NDCG in comparison.

Given query q_i and associated documents D_i, suppose that π_i is the ranking list (permutation) on D_i and $\mathbf{y}_i$ is the set of labels (grades) of D_i. DCG measures the goodness of the ranking list with the labels. Specifically, DCG at position k is defined as

$$DCG(k) = \sum_{j:\pi_i(j)\leq k} G(j)D(\pi_i(j)),$$

where $G_i(\cdot)$ is a gain function, $D_i(\cdot)$ is a position discount function, $\pi_i(j)$ is the position of $d_{i,j}$ in π_i, and the summation is taken over the top k positions in the ranking list π_i. The gain function is normally defined as an exponential function of grade

$$G(j) = 2^{y_{i,j}} - 1,$$

where $y_{i,j}$ is the label (grade) of document $d_{i,j}$ in ranking list π_i. The position discount function is normally defined as a logarithmic function of position

$$D(\pi_i(j)) = \frac{1}{\log_2(1+\pi_i(j))},$$

where $\pi_i(j)$ is the position of document $d_{i,j}$ in ranking list π_i. DCG represents the cumulative gain of accessing the information from position one to position k with discounts on the positions. Hence, DCG at position k becomes

$$DCG(k) = \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1+\pi_i(j))}.$$

NDCG is normalized DCG and NDCG at position k is defined as

$$NDCG(k) = G^{-1}_{max,i}(k) DCG(k),$$

where $G_{max,i}(k)$ is the normalizing factor and is chosen such that a perfect ranking π_i^*'s $NDCG$ score at position k is 1. In a perfect ranking, the documents with higher grades are always ranked higher. Note that there can be multiple perfect rankings for a query and associated documents. Then, NDCG at position k becomes

$$NDCG(k) = G^{-1}_{max,i}(k) \sum_{j:\pi_i(j)\leq k} \frac{2^{y_{i,j}} - 1}{\log_2(1+\pi_i(j))}.$$

In evaluation, DCG and NDCG values are further averaged over queries.

2.3 Objective Function of Learning

Learning to rank is generally formalized as a supervised learning task. Suppose that $\mathcal{X}$ is the input space (feature space) consisting of lists of feature vectors, and $\mathcal{Y}$ is the output space consisting of lists of grades. Further suppose that $\mathbf{x}$ is an element of $\mathcal{X}$ representing a list of feature vectors and $\mathbf{y}$ is an element of $\mathcal{Y}$ representing a list of grades. Let $P(X, Y)$ be an unknown joint probability distribution where random variable X takes $\mathbf{x}$ as its value and random variable Y takes $\mathbf{y}$ as its value.

Assume that $F(\cdot)$ is a function mapping from a list of feature vectors $\mathbf{x}$ to a list of scores. The goal of the learning task is to automatically learn a function $\hat{F}(\mathbf{x})$ given training data $(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_m, \mathbf{y}_m)$. Each training instance is comprised of feature vectors $\mathbf{x}_i$ and the corresponding grades $\mathbf{y}_i$ $(i = 1, \cdots, m)$. Here m denotes the number of training instances.

$F(\mathbf{x})$ and $\mathbf{y}$ can be further written as $F(\mathbf{x}) = (f(x_1), f(x_2), \cdots, f(x_n))$ and $\mathbf{y} = (y_1, y_2, \cdots, y_n)$. The feature vectors represent objects to be ranked. Here $F(\mathbf{x})$ denotes the global ranking function, $f(x)$ denotes the local ranking function, and n denotes the number of feature vectors and grades.

A loss function $L(\cdot, \cdot)$ is utilized to evaluate the prediction result of $F(\cdot)$. First, feature vectors $\mathbf{x}$ are ranked according to $F(\mathbf{x})$, then the top n results of the ranking are evaluated using their corresponding grades $\mathbf{y}$. If the feature vectors with higher grades are ranked higher, then the loss will be small. Otherwise, the loss will be large. The loss function is specifically represented as $L(F(\mathbf{x}), \mathbf{y})$. Note that the loss function for ranking is slightly different from the loss functions in other statistical learning tasks, in the sense that it makes use of sorting.

The loss function can be defined, for example, based on NDCG (Normalized Discounted Cumulative Gain) and MAP (Mean Average Precision). In the former case, we have

$$L(F(\mathbf{x}), \mathbf{y}) = 1 - NDCG.$$

The risk function, i.e., the objective function in learning, is further defined as the expected loss function with respect to the joint distribution $P(X, Y)$,

$$R(F) = \int_{\mathcal{X} \times \mathcal{Y}} L(F(\mathbf{x}), \mathbf{y}) dP(\mathbf{x}, \mathbf{y}).$$

Given training data, the empirical risk function is defined as follows,

$$\hat{R}(F) = \frac{1}{m} \sum_{i=1}^{m} L(F(\mathbf{x}_i), \mathbf{y}_i). \tag{1}$$

The learning task then becomes minimization of the empirical risk function, as in other learning tasks. The minimization could be difficult due to the nature of the loss function (it is not continuous and it uses sorting). We can consider using a surrogate loss function $L'(F(\mathbf{x}), \mathbf{y})$.

The corresponding empirical risk function is defined as follows.

$$\hat{R}'(F) = \frac{1}{m} \sum_{i=1}^{m} L'(F(\mathbf{x}_i), \mathbf{y}_i). \tag{2}$$

One can also introduce a regularizer to the minimization. In such case, the learning problem becomes minimization of the regularized empirical risk function based on the surrogate loss.

For the surrogate loss function, there are also different ways to define it, which lead to different approaches to learning to rank, referred to as pointwise loss, pairwise loss, and listwise loss functions.

2.4 Methods

Ranking SVM and IR SVM

A perfect ranking implies perfect classification on all pairs of objects and an imperfect ranking implies existence of incorrect classification on pairs of objects. Therefore, learning of a ranking model can be transformed to learning of a pairwise classification or regression model. This is the idea behind the pairwise approach to learning to rank. Ranking SVM proposed by Herbrich et al. [4] is a typical example.

Ranking SVM takes pairs of objects (feature vectors) at different grades as new objects and takes the orders of the pairs of objects as positive or negative classes. It learns an SVM classifier which can classify the order of pair of objects, and then utilizes the SVM classifier to assign scores to newly given objects and rank the objects on the basis of the scores. One can easily verify that if the classifier is a linear function, then it can be directly utilized as a ranking model.

It can be proved that the ‘pairwise’ loss function in Ranking SVM is an upper bound of (1-NDCG) [2], and therefore Ranking SVM is a method equivalent to minimizing empirical risk based on a surrogate loss function (cf., Eq.(2)).

IR SVM proposed by Cao et al [5]. is an extension of Ranking SVM for information retrieval (IR). Ranking SVM transforms ranking into pairwise classification, and thus it actually makes use of the 0-1 loss in the classification. There exists a gap between the loss function and the IR evaluation measures such as NDCG. IR SVM attempts to bridge the gap by modifying the 0-1 loss, that is, conducting cost sensitive learning of Ranking SVM.

One problem with Ranking SVM is that it equally treats document pairs across different grades (levels). Another issue with Ranking SVM is that it equally treats document pairs from different queries. IR SVM addresses the above two problems by modifying the hinge loss function. Specifically, it sets different losses for document pairs across different grades and from different queries. To emphasize the importance of correct ranking on the top, the loss function heavily penalizes errors related to the top. To increase the influence of queries with less documents, the loss function heavily penalizes errors from the queries.

AdaRank

Since the evaluation measures in IR are defined on lists of objects, it is more natural and effective to directly optimize 'listwise' loss functions defined on lists. This is the presumption in the listwise approach to learning to rank. AdaRank, proposed by Xu & Li [6], is one of the listwise algorithms. One advantage of AdaRank is its simplicity, and it is perhaps one of the simplest learning to rank algorithms.

In learning, ideally we would create a ranking model that can maximize the ranking accuracy in terms of an evaluation measure (e.g., NDCG) on the training data, or equivalently minimize the empirical risk function in Eq. (1) specified as,

$$\sum_{i=1}^{m} (1 - E(\pi_i, \mathbf{y}_i)),$$

where $\mathbf{x}_i$ is a list of feature vectors, $\mathbf{y}_i$ is the corresponding list of grades, π_i is the permutation of feature vectors $\mathbf{x}_i$ by the ranking model $f(x)$, $E(\pi_i, \mathbf{y}_i))$ is the evaluation result of π_i based on $\mathbf{y}_i$ in terms of the evaluation measure, and m is the number of training instances.

The empirical risk function is not smooth and differentiable, and thus straightforward optimization of the evaluation result might not work. Instead, we can consider optimizing an upper bound of the function. For example, one upper bound is the following function

$$\sum_{i=1}^{m} \exp(-E(\pi_i, \mathbf{y}_i)),$$

which is continuous, differentiable, and even convex with respect to E.

AdaRank minimizes the upper bound, by taking the boosting approach. Mimicking the famous AdaBoost algorithm, AdaRank conducts stepwise minimization of the upper bound. More specifically, AdaRank repeats the process of re-weighting the training instances, creating a weak ranker, and assigning a weight to the weak ranker. Finally, AdaRank linearly combines the weak rankers as the ranking model.

One can prove that AdaRank can continuously reduce the empirical risk during the training process, under certain condition. When the evaluation measure is dot product, AdaRank can reduce the risk to zero.

2.4.1 LambdaRank and LambdaMART

The objective function (empirical risk function) in learning to rank is not continuous and differentiable, and it depends on sorting. This makes it difficult to use gradient decent to optimize the function. LambdaRank and LambdaMART, proposed by Burges et al. [7], manages to solve the problem by directly defining and utilizing a gradient function of the risk function.

Suppose that the ranking model, query, and documents are given. Then each document receives a score from the ranking model, and a ranking list can be created by sorting the documents based on the scores. Since the documents are also assigned ground truth labels, a ranking evaluation result based on an IR measure can be obtained. Suppose that we use a surrogate loss function L to approximate the IR evaluation measure. Then, an evaluation result based on the surrogate loss function L can also be obtained. It is this evaluation result which LambdaRank attempts to continuously optimize.

LambdaRank does not explicitly give the definition of the loss function. Instead it defines the *gradient* function of the surrogate loss function. More specifically, the gradient function is defined as

$$\frac{\partial L}{\partial s_i} = -\lambda_i(s_1, y_1, \cdots, s_n, y_n),$$

where $s_1, s_2, \cdots, s_n$ denote the scores of documents and $y_1, y_2, \cdots, y_n$ denote the labels of documents. Note that the index i is on a single document. That is to say, the gradient of a document depends on the scores and labels of the other documents. The sign is chosen such that a positive value for a document means that the document must reduce the loss. The gradients of documents are calculated after the current model generates a ranking list of documents for the query. The negative gradient function is called lambda function, and that is why the method is called LambdaRank. LambdaRank utilizes a neural network as its ranking model.

LambdaMART follows the idea of LambdaRank, but it utilizes an ensemble of trees as its ranking model and employs the Gradient Tree Boosting algorithm to build the ranking model. Specifically, LambdaMART directly uses the lambda function as the gradient function in the learning process of Gradient Tree Boosting.

In the Yahoo Learning to Rank Challenge, LambdaMART achieved the best performance. It is viewed as one of the state-of-the-art methods for learning to rank, and is being used in a number of commercial search systems.

3 Applications

Learning to rank has been successfully applied to a wide variety of applications, including document retrieval, expert search, definition search, personalized search, online advertisement, collaborative filtering, question answering, keyphrase extraction, document summarization, and machine translation. Particularly, in document retrieval there are many signals which can represent relevance. Incorporating such information into the ranking model and automatically constructing the ranking model by using data becomes a natural choice. In fact, learning to rank has become one of the fundamental technologies for document retrieval.

4 Recommended Reading

References

[1] Tie-Yan Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.

[2] Hang Li. Learning to rank for information retrieval and natural language processing. *Synthesis Lectures on Human Language Technologies*, 4(1):1–113, 2011.

[3] Hang Li. A short introduction to learning to rank. *IEICE TRANSACTIONS on Information and Systems*, 94-D(10):1854–1862, 2011.

[4] Ralf Herbrich, Thore Graepel, Klaus Obermayer. Large margin rank boundaries for ordinal regression. *Advances in Neural Information Processing Systems*, 115-132, 1999.

[5] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, Hsiao-Wuen Hon. Adapting Ranking SVM to document retrieval. *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 186-193, 2006.

[6] Jun Xu, Hang Li. Adarank: a boosting algorithm for information retrieval. *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information retrieval*, 391-398, 2007.

[7] Christopher J.C. Burges. From RankNet to LambdaRank to LambdaMART: an overview. *Microsoft Research Technical Report*, MSR-TR-2010-82, 2010.