Learning to Rank from Clickthrough Data

Qiuyan Xu

Abstract

This paper presents an analysis over current algorithms of learning to rank, with a focus on the input of the learning approaches. There are many algorithms of learning to rank available. The most basic one is linear regression, which learns single documents. RankingSVM is classic while MPRank is relative new. Both of them use pair-wise preferences as input. RBA and Cofi-Rank, however, learn lists instead of preferences. Almost all the learning approaches nowadays use one of the above three types of input. Five learning to rank algorithms are briefly introduced with focus on their input data. In order to learn to rank with those learning approaches, a large amount of training data is required. For learning to rank algorithm evaluations some training data is labeled by experts. However, in large scale learning, it is more practical to extract required training dataset from logging data, i.e. the implicit feedback from users which mainly contains the information that on which documents a user has clicked. Considering the importance of the training data, we will especially concentrate on, what kind of input is required and how to obtain such an input using implicit or explicit feedback, with respect to popular learning approaches.

1 Motivation

Nowadays it is a trend that machine learning approaches are used in learning to rank. Compared with conventional ranking approaches such as TF-IDF [22] and PageRank [2], machine learning approaches show the advantage that people do not need to assign the parameters of the models. Instead, optimal parameters can be found out during the learning process, which is far more intelligent and make it possible to build a complex model.

However, the introduction of machine learning gives rise to a new problem that a large amount of training data is required to support the learning process. It is a crucial problem, how to obtain the training data. In some experiment, e.g. [20], some experts

labeled an amount of documents and the labeled documents were later formed into a dataset, which fits the designed approach. Unfortunately, it is not practical to label a large quantity of data manually, because it is too expensive and the data is still rare. To solve such a problem, some rules are introduced in [20], with which pair-wise preferences can be extracted from the users' clickthrough data that exists in the logging data of a search engine. In comparison with other source of training data such as those generated from the experts, logging data has the advantage that it is free available and exists in a large amount. Furthermore, as the search engine wins new logging data every day, the logging data is always up-to-date.

Considering the advantage of clickthrough data, in this paper, we will concentrate on how to construct appropriate input for popular learning to rank approaches. Moreover, some typical approaches will be introduced, which take advantage of clickthrough data to learn to rank.

2 Logging

A logging server can collect a lot of logging data every day, which mainly includes the users, the queries, the displayed results and the clicks of the users. Intuitively we may conclude that clicked results are relevant to the query, since the users don't click at random [12]. Instead, they tend to click those results which can meet their information requirement. As clickthrough date reflects the quality of the results, logging data is regarded as the most important source of implicit feedback, which may be used to optimize as well as personalize the rankings.

2.1 Logging Server

In order to acquire the logging data, a logging server can be established to log the user behavior. In [12] Joachims introduced an easy way to set up a proxy server for logging. A unique id will be assigned to each query. The links of the documents presented to the users lead to the proxy server rather than directly to the target. Useful information such as query id, target link will be encoded in the link. Once the server received a request, it stores the information into the database and redirect to the target site, so that the whole process remain transparent to the users.

Moreover, Joachims also suggested how to present the data in the database. The clickthrough data can be regarded as triplets (q, r, c), where q is the query, r is the ranking for the query and c is a set of clicked results.

2.2 From Feedback to Dataset

Both explicit and implicit feedback reflect the relevance of the documents. In order to apply them into learning approaches, they need to be organized in a form that fits a specific approach for learning to rank. Typically there are three types of input, i.e. point-wise, pair-wise and list-wise.

2.2.1 Ratings

When we use rating to determine if a document is relevant to a given query, it is called point-wise. Each document corresponds a point with an assigned score. Each document is regarded independent and there exists no correlation among documents. In [18], information retrieval was regarded as a binary classification problem that a document is either relevant or irrelevant to a given query. However, in the reality, there is barely absolute relevance or irrelevance. It is more common that some document is reasonably relevant or somewhat relevant to a query. So it is hard to classify documents into only two classes.

Since binary classification cannot comprehensively cover all the situations, it is reasonable to use multi-level score to rate the documents. For instance documents can be classified into highly relevant, partially relevant, definitively irrelevant and so on [24].

Ratings for documents can be won from both explicit and implicit feedback. Some commercial websites collect explicit feedback from users in such a way that they let users click stars to score a result, e.g. YouTube and Amazon. Fig 1 shows how YouTube collects ratings from users. In the video page there are five stars under the video, on which users may click to score. As shown in the example, when the mouse stays on four stars, that means the video is "pretty cool".

Figure 1: User may click the stars to give an explicit feedback.

Although explicit feedback works well, it is not easy to collect it. Usually speaking, users are not willing to give explicit feedback. It is more practical, if we can learn implicit feedback. For example, binary score can also be learnt from users' click in such a way that clicked results are assigned as relevant while non-clicked results are assigned as irrelevant. However it has also its disadvantages. The assumption that non-clicked results are irrelevant is not so convincing, since such results with a low ranking seem to never be seen by the users. As a result, they have no chance to be clicked at all, no matter how relevant they are. Furthermore, as the amount of non-clicked results is much more than clicked ones, in other words, there are much more irrelevant documents

than relevant ones in the training data, which may cause that the error is small even when a function always returns false [24].

2.2.2 Preferences

Although ratings are easy to apply, it is regarded unresonable to treat the relevance learnt from clickthrough data in a whole range [20], as clickthrough data has nothing to do with those results that the user has never seen. In [20] some rules were introduced, with which pair-wise preferences can be extracted from clickthrough data.

Before we introduce the rules, it is necessary to define the concept of query chains. When searching with a search engine, it is quite common that the users reformulate their query key words. Those queries with a same purpose build a query chain. The idea is that all queries in a query chain share the same information requirement. With query chains, more queries can be associated together and thus more preferences can be learnt.

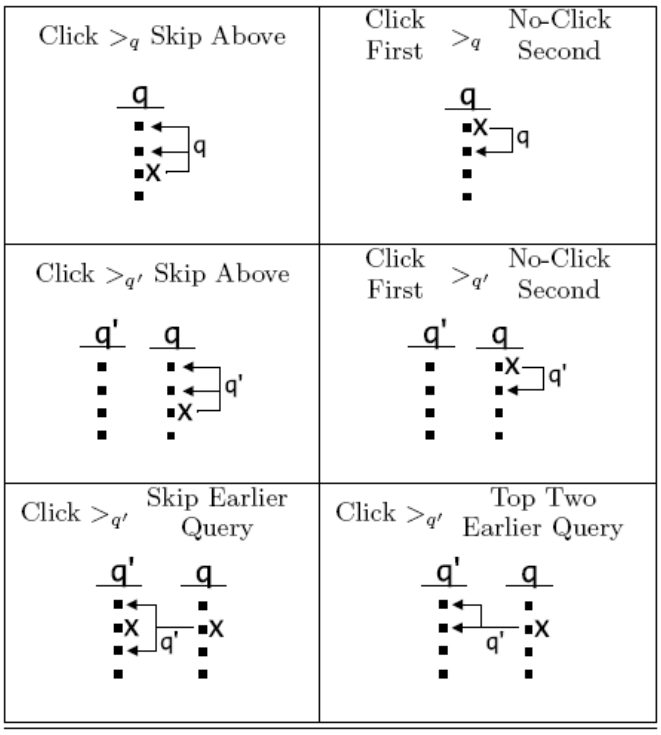


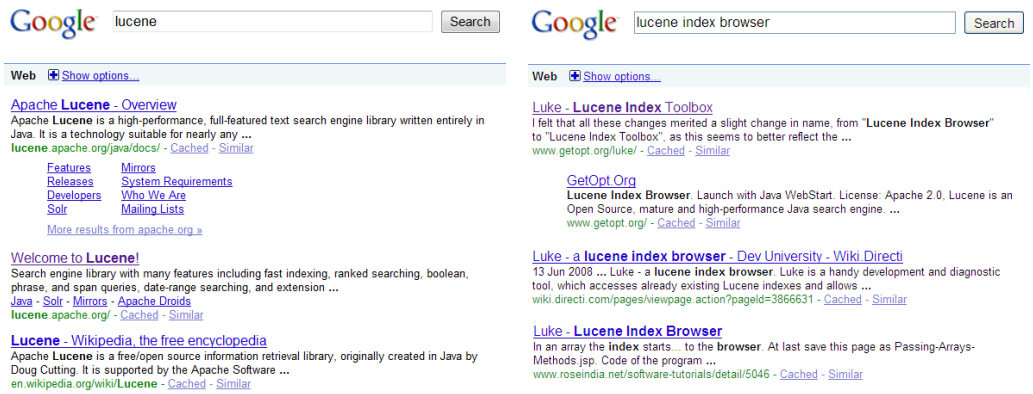
Figure 2: Clickthrough Rules [20]

Fig. 2 shows six clickthrough rules from [20]. The first one presents that a clicked result is better than the skipped results that have a higher position but are not clicked. An eye tracking study [10] has proved that the users usually read the results from above and one after another, and the users always click a result after they read it. It is also

understandable that they tend to click those results that are relevant to the query. In conclusion, when a user clicks a result, he has read all the results shown above and finally he decides to click a more relevant one. The second rule shows that when a user click the first result of a page without clicking the second one, a first result is then better than the second one, since the users read at least first two results of a page, according to the eye tracking study.

The third and fourth rules are in principle identical with the first two ones, except that they are applied in a query chain. When a query is in a query chain, the preferences can be also applied to the previous query.

The last two rules take use of the concept of query chains intensively. The fifth rule states that when a query is in a query chain, a clicked result is also better than the skipped results in the page of the previous query. Here the skipped results were defined a little different that the result shown one position below the last click is also regarded as skipped, since the eye tracking study also showed that when a user clicks a result, he has already read the result shown one position below. The sixth rule presents that when a query is in a query chain and no results were clicked in the previous query, a clicked result is better than the first two results of the previous query.



(a) Search for "lucene" and click on the second result

(b) Search refined to "lucene index browser" and click on the first result

Figure 3: An example illustrating clickthrough behaviour and query chains

An example is shown in Fig. 3. At first we search for "lucene" and click on the second result, and then we refine the search to "lucene index browser" and click on the first result. We denote the first search as $S1$, second as $S2$, Di as the i'th result, the query "lucene" as q' and "lucene index browser" as q. According to the first rule, we may get the preference: $S1D2 >_{q'} S1D1$. According to the second rule, we get the preference: $S2D1 >_q S2D2$. Taking use of the fourth rule, the preference in $S2$ can also be applied in $S1$, i.e. $S2D1 >_{q'} S2D2$. From the fifth rule, we may further get

some preferences with respect to both searches: $S2D1 >_{q'} S1D1$, $S2D1 >_{q'} S1D3$. As shown we have won five preferences altogether from the above query chain.

2.2.3 Rankings

Ranking as a form of list is exactly what an original information retrieval problem is like. However there is so far no available rules that extracts a list of ranking from the logging data. So the way of extract a ranking from logging data is similar to constructing ratings. We take the count of each document as its score, so that documents with more clicks achieve a higher score. In this way we may get a list of documents ordered by score which presents the ranking.

Since it's the same way as to construct a point-wise dataset. They share the disadvantage that it is difficult to carry out in a personalization.

Google collects ranking data in another way. If the user has logged in, there will show a button "promote" and a button "remove" beside each result, so that the user is allowed to adjust the rankings themselves. And google also benefits as they win explizit feedback on rankings.

3 Learning to Rank

Thus far there are lots of approaches for learning to rank. TF-IDF [22], for example, takes use of a statistic over term frequency within a document and document frequency which counts in how many documents a term appears. PageRank [2] as a very successful approach, however, estimates the importance of each web page according to the number of the pages that link to the page and the importance of those pages as well. Nowadays almost all new approaches use machine learning technologies, which have the advantage that they can adjust parameters themselves to achieve a best result.

3.1 Approach

In most cases, documents are represented as vectors. The example from [1] has explained clearly, how it works.

Suppose we have a dictionary constituted of six terms: bak(e,ing), recipes, bread, cake, pastr(y,ies) and pie(s), note that all the words belong to a same stem share a term. And we have five document titles listed in Fig. 4:

D1: How to Bake Bread Without Recipes
D2: The Classic Art of Viennese Pastry
D3: Numerical Recipes: The Art of Scientific Computing
D4: Breads, Pastries, Pies and Cakes: Quantity Baking Recipes
D5: Pastry: A Book of Best French Recipes

Figure 4: Document Titles [1]

From the above documents we can construct a matrix:

$$\hat{A} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{1}$$

where each row is a vector corresponds to a document and each column corresponds to a term. The numbers in the matrix present how often a term appears in the document. After that, we usually normalize each row, so that each document vector has a length of 1, thus we can avoid the bias between long documents and short ones. So the result matrix would look like:

$$A = \begin{pmatrix} 0.5774 & 0.5774 & 0.5774 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & 0 \\ 0 & 1.0000 & 0 & 0 & 0 & 0 \\ 0.4082 & 0.4082 & 0.4082 & 0.4082 & 0.4082 & 0.4082 \\ 0 & 0.7071 & 0 & 0 & 0.7071 & 0 \end{pmatrix} \tag{2}$$

In web scale problems, a great amount of terms results that the dimension of the vector can be very large, which may consume memory and slow the learning process. As all the ranking problem are related to a particular query, we usually map such a large vector into smaller ones, in which, for example, only terms relevant to the query, or in the simplest situation, only such terms that also appear in the query are selected for the learning process. For instance, if the query is "bread recipe", in the above example, we may get a mapped matrix as:

$$A' = \begin{pmatrix} 0.5774 & 0.5774 \\ 0 & 0 \\ 1.0000 & 0 \\ 0.4082 & 0.4082 \\ 0.7071 & 0 \end{pmatrix} \tag{3}$$

In case of TF-IDF, we get the final score by:

$$S = A'\mathbf{w} \tag{4}$$

where $\mathbf{w}$ is a column vector which contains the weighting factor of each document in A', according to the IDF function.

As stated above, in traditional information retrieval, documents are presented with term vectors in the vector space model. In case of machine learning, vectors are feature vectors with static features, i.e. metadata, or query dependent features, such as term frequency and so on.

3.2 Datasets

In employing machine learning technologies in learning to rank, a large dataset is required for the learning approaches to learn with.

3.2.1 LETOR

LETOR [16] is a benchmark dataset distributed to the search community. In the up to date version LETOR4.0, it makes use of the Gov2 web page collection and two query sets from TREC 2007 and TREC 2008. It extracts not only such conventional feature data as term frequency, inverse document frequency, but also recently proposed features such as HostRank, feature propagation, topical PageRank and so on. Those features with queries and relevance judgment constitute LETOR dataset.

The purpose of LETOR is to provide a benchmark for learning to rank. With such a preprocessed dataset, new approaches benefit as they can directly be evaluated. Moreover, it can be used to measure the performance of a new developed learning approach as well as compare two approaches. As both approaches learn a same dataset, the comparison is relative fair.

4 Measure

In information retrieval, there are some common standards for evaluation, such as MAP(Mean Average Precision), NDCG(Normalized Discounted Cumulative Gain), MRR(Mean Reciprocal Rank), WTA(Winners Take All) and so on.

4.1 MAP

MAP is the most standard measure among the TREC community currently [17], which is based on binary judgement of documents. The calculation can be described in the following three steps:

For a specific query, the precision at position n is calculated as:

$$P@n = \frac{\#\{\text{relevant documents in top } n \text{ results}\}}{n} \tag{5}$$

For each query, the average precision is defined as:

$$AP = \frac{\sum_n P@n \cdot I\{\text{document } n \text{ is relevant}\}}{\#\{\text{relevant documents}\}} \tag{6}$$

where I is 1 if the n'th document is relevant and is 0 otherwise.

MAP is the mean value over all queries in the test set:

$$MAP = \frac{\sum_{q \in \text{queries}} AP_q}{\#\{\text{queries}\}} \tag{7}$$

4.2 NDCG

NDCG [13] is designed for non-binary notions of relevance. It assumes that highly relevant documents are more valuable than marginally relevant documents and the greater the ranked position of a relevant document, the less valuable it is for the user.

Let $G[i]$ be the score of document i, the cumulated gain is defined recursively as:

$$CG[i] = \begin{cases} G[1], & \text{if } i = 1 \\ CG[i-1] + G[i], & \text{otherwise} \end{cases} \tag{8}$$

The discounted cumulated gain is then:

$$DCG[i] = \begin{cases} CG[i], & \text{if } i < b \\ DCG[i-1] + G[i]/\log_b i, & \text{if } i \geq b \end{cases} \tag{9}$$

where b is the base of the algorithm.

DCG can be normalized by dividing them by the corresponding ideal DCG. IDCG is basically calculated the same way as DCG, expect that the scores are resorted monotonically decreasing. So normalized DCG is given as:

$$NDCG[i] = \frac{DCG[i]}{IDCG[i]} \tag{10}$$

4.3 Abandonment

Abandonment is the situation that a user does not click any result in a page [21]. It is also an important measure of user satisfaction since it indicates that users were presented with results of no potential interest. To decrease the abandonment rate, it is required that the results contain at least one useful result for a random user.

5 Learning to Rank Approaches

In the past decade a lot of machine learning approaches for learning to rank have been studied and presented, which learn to rank with different kinds of input. In the following we will introduce some of them.

5.1 Point-wise

In information retrieval, point-wise input is usually in such a form that each document corresponds to an independent point. For instance, we may assign each document a binary value, either relevant or irrelevant to a given query. Another possibility is that for a certain query, the documents are given a score which presents how relevant the document is. Typical learning methods which use point as input are linear regression, Discriminative model for IR [18], McRank [15].

5.1.1 Linear Regression

We assume that the relevance between a document and a given query has an approximately linear relation with some properties of the documents, e.g. term frequency. So for each document, the score can be presented as a linear function of the term frequencies like:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_p + \epsilon \tag{11}$$

In the above function, y is the score, x_i is the frequency of a certain term, ϵ contains all the random factors which are not observed and β_i is the parameter to estimate. For a dataset with a great deal of documents and known score, the linear relation can be formed with matrices:

$$Y = X\beta + \epsilon \tag{12}$$

where

$$Y = \begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix}^T \tag{13}$$

$$X = \begin{pmatrix} 1 & x_{11} & \dots & x_{1p} \\ 1 & x_{21} & \dots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \dots & x_{np} \end{pmatrix} \tag{14}$$

$$\beta = \begin{pmatrix} \beta_0 & \beta_1 & \cdots & \beta_n \end{pmatrix}^T \tag{15}$$

Each row of X begins with 1 and is followed with the vector of a document, as was introduced. As a result, we got the estimation:

$$\hat{\beta} = (X^T X)^{-1} X^T Y \tag{16}$$

in which we get a minimal sum of squared residuals, in another word, a best linear fitting is found so that the error is minimized. In a 2-dimension view, the result can be described like showed in Fig. 5.

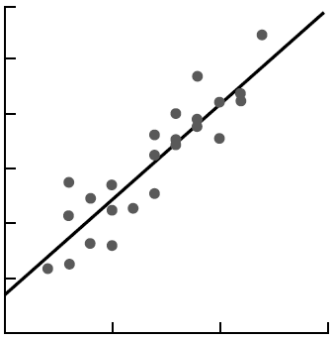


Figure 5: Linear Regression [14]

5.2 Pair-wise

Currently most learning methods learn pair-wise input. Typical examples are RankingSVM [12], MPRank [6], FRank [24], LDM [9], RankBoost [8], RankNet [3], GBRank [30], QBRank [31] and so on.

5.2.1 RankingSVM

Before we learn RankingSVM, it would be helpful to take a look at SVM at first.

Naive SVM solves a classification problem in such a way that it tries to find out a hyperplane which separates the two given classes ($d_i = +1$ and $d_i = -1$) of points with a maximal margin, as is showed in Fig. 6. All the given points satisfy the constraints:

$$\begin{array}{ll} \mathbf{w}_0^T \mathbf{x}_i + b_0 \geq 1 & \text{for } d_i = +1 \\ \mathbf{w}_0^T \mathbf{x}_i + b_0 \leq -1 & \text{for } d_i = -1 \end{array} \tag{17}$$

The margin is then

$$\rho = \frac{2}{||\mathbf{w_0}||} \tag{18}$$

To maximize the margin, we only need to minimize $\frac{||\mathbf{w_0}||}{2}$.

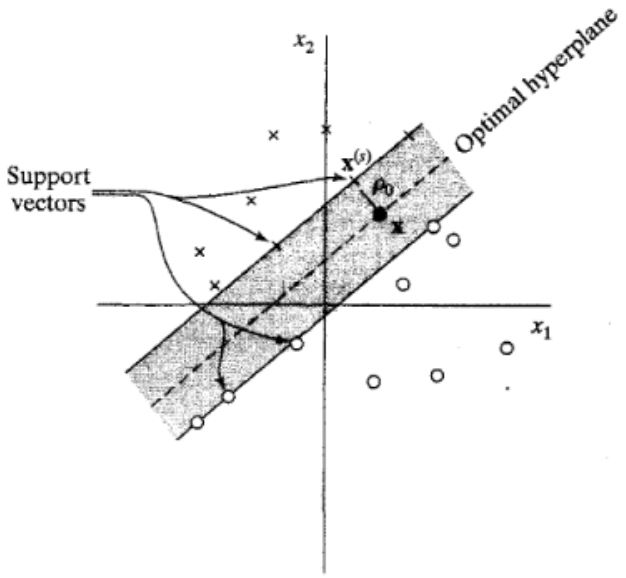


Figure 6: SVM [11]

An extension of SVM is called soft margin, in which mislabels are allowed. Slack variables are ξ_i introduced to measure the degree of misclassification. The hyperplane is then in a form like:

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \tag{19}$$

The problem is now to minimize the following function:

$$V(\mathbf{w}, \xi) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{N}^{i=1}\xi_i \tag{20}$$

RankingSVM [12] inherits the method of SVM with the difference that we try to minimize the misorder instead of misclassification. Given a query q and two document d_i, d_j, the ranking function satisfies:

$$d_i >_q d_j \Leftrightarrow \mathbf{w}\Phi(q, d_i) > \mathbf{w}\Phi(q, d_j) \tag{21}$$

where $\Phi(q, d)$ presents the match between the query q and the document d.

5.2.2 MPRank

Magnitude-Preserving Ranking(MPRank) [6] learns from pairwise preferences and it employs the magnitude of the relevance as well.

Four cost functions are introduced in [6]. The first is called hinge rank loss, which defined as:

$$c^n_{HR}(h, x, x') = \begin{cases} 0, & \text{if } (h(x') - h(x))(y_{x'} - y_x) \geq 0 \\ |(h(x') - h(x))|^n, & \text{otherwise} \end{cases} \tag{22}$$

where h is a hypothesis, x and x' are documents, n is either 1 or 2. It penalizes misranking without taking the magnitude of the preference into account. Another cost function is defined as:

$$c^n_{MP}(h, x, x') = |(h(x') - h(x)) - (y_{x'} - y_x)|^n \tag{23}$$

where any difference between true magnitude of preferences and predicted ones will cause penalty. A one-side version of c_{MP} is:

$$c^n_{HMP}(h, x, x') = \begin{cases} 0, & \text{if } (h(x') - h(x))(y_{x'} - y_x) \geq 0 \\ |(h(x') - h(x)) - (y_{x'} - y_x)|^n, & \text{otherwise} \end{cases} \tag{24}$$

where cost is only counted in case of misordering. The last cost function is:

$$c^n_{SVR}(h, x, x') = \begin{cases} 0, & \text{if } (h(x') - h(x))(y_{x'} - y_x) \leq \epsilon \\ ||(h(x') - h(x)) - (y_{x'} - y_x)| - \epsilon|^n, & \text{otherwise} \end{cases} \tag{25}$$

which tolerates small errors.

The MPRank algorithm minimizes the following function:

$$F(h, S) = ||h||^2_K + C\frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m} c(h, x_i, x_j) \tag{26}$$

When defining $h(x) = w \cdot \Phi(x)$ and using c_{MP} as cost function, the above objective function can be represented as:

$$F(h, s) = ||w||^2 + C\frac{1}{m^2}\sum_{i=1}^{m}\sum_{j=1}^{m}[(w \cdot \Phi(x_j) - w \cdot \Phi(x_i)) - (y_j - y_i)]^2 \tag{27}$$

5.3 List-wise

Besides point-wise inputs und pair-wise ones, list-wise input can also be applied to a machine learning approach. Typical examples of list-wise approaches are RBA [21], Cofi-Rank [25], ListNet [5], LambdaRank [4], AdaRank [27], SVM-MAP [29], SoftRank [23], GPRank [28], CCA [7], RankCosine [19], ListMLE [26] and so on.

5.3.1 RBA

Unlike other approaches, RBA(Ranked Bandits Algorithm) [21] tries to minimize the abandonment rate. Since the interest of a user differs from that of another, the relevant set of document for a given queries is also different for different users. The basic idea of RBA is trying to find out the diverse rankings so that for a random user there exists at least one result that satisfies him.

Algorithm 1 Ranked Explore and Commit

input: Documents $(d_1, .., d_n)$, parameters ϵ, δ, k.
$x \leftarrow \lceil 2k^2/\epsilon^2 \log(2k/\delta) \rceil$
$(b_1, \ldots, b_k) \leftarrow k$ arbitrary documents.
for i=1 ... k **do** *At every rank*
  $\forall j.\ p_j \leftarrow 0$
  for counter=1 ... x **do** *Loop x times*
    for j=1 ... n **do** *over every document d_j*
      $b_i \leftarrow d_j$
      display $\{b_1, \ldots, b_k\}$ to user; record clicks
      if user clicked on b_i **then** $p_j \leftarrow p_j + 1$
    end for
  end for
  $j^* \leftarrow \text{argmax}_j\, p_j$ *Commit to best document at this rank*
  $b_i \leftarrow d_{j^*}$
end for

Figure 7: REC [21]

Fig. 7 shows the pseudo code of REC(Randed Explore and Commit). It is a greedy strategy that from the first rank on, the rank is assigned the document which achieves the most clicks.

RBA, whose algorithm is shown in Fig. 8, has optimized REC in that it is able to adapt itself to the changing mind of users. Each rank i runs an MAB(Multi-armed bandit) instance MAB_i and is initialized according to the selected MAB_i. In case that the document selected is duplicated to that of another rank, the rank will be assigned an arbitrary document. In each iteration with the learning process, the rankings are presented to a user and the user clicks one or none document. Then each rank is updated, no matter it is clicked or not, with the designed algorithm of its MAB.

5.3.2 Cofi-Rank

Cofi-Rank [25] solves the recommendation problem of web shops. It gives a list of recommendation for each user, but can also be applied to general ranking problems. There are three steps to construct the ranking model:

Algorithm 2 Ranked Bandits Algorithm

initialize $\mathsf{MAB}_1(n), \ldots, \mathsf{MAB}_k(n)$ *Initialize MABs*
for t = 1 ... T **do**
  for i = 1 ... k **do** *Sequentially select documents*
    $\hat{b}_i(t) \leftarrow$ select-arm (MAB_i)
    if $\hat{b}_i(t) \in \{b_1(t), .., b_{i-1}(t)\}$ **then** *Replace repeats*
      $b_i(t) \leftarrow$ arbitrary unselected document
    else
      $b_i(t) \leftarrow \hat{b}_i(t)$
    end if
  end for
  display $\{b_1(t), \ldots, b_k(t)\}$ to user; record clicks
  for i = 1 ... k **do** *Determine feedback for* MAB_i
    if user clicked $b_i(t)$ **and** $\hat{b}_i(t) = b_i(t)$ **then**
      $f_{it} = 1$
    else
      $f_{it} = 0$
    end if
    update $(\mathsf{MAB}_i, arm = \hat{b}_i(t), reward = f_{it})$
  end for
end for

Figure 8: RBA [21]

First, NDCG(π, y), which was introduced in section 4.2, is used as a measure of the ranking. It is converted into:

$$\Delta(\pi, y) := 1 - \text{NDCG}(\pi, y) \tag{28}$$

where π is the ranking and y is the rating. As the value of NDCG is greater if the ranking is more accurate, and it achieves 1 in optimal situation, $\Delta(\pi, y)$ is therefore nonnegative and achieves 0 when the ranking is sorted as ideal and our goal is to minimize $\Delta(\pi, y)$.

The second step is to map the ranking into a linear function:

$$\psi(\pi, f) :=< c, f_\pi > \tag{29}$$

where π is the ranking, f is the function to estimate rating, c is a decreasing nonnegative sequence and f_π the the estimated rating of ranking π. As the Polya-Littlewood-Hardy inequality states: for any two vectors $a, b \in \mathbb{R}^n$, their inner product is maximized when a and b are sorted in the same order. Applying the inequality, ψ is maximized by ranking π ideally.

As the third step we define the upper bound of loss:

$$l(f, y) := \max_{\pi}[\Delta(\pi, y) + < c, f_{\pi} - f >] \tag{30}$$

which satisfies:

$$l(f, y) \geq \Delta(\pi^*, y) + < c, f_{\pi^*} - f > \geq \Delta(\pi^*, y) \tag{31}$$

where π^* is the ideal ranking, i.e. the ranking induced by function f. Thus, we get the loss function over all the queries as:

$$L(F, Y) := \sum_{i=1}^{u} l(F^i, Y^i) \tag{32}$$

6 Conclusions

The paper introduced the currently popular machine learning approaches for learning to rank and analyzed the way to construct a suitable training dataset for each type of learning to rank approaches. Basically point-wise, pair-wise and list-wise training dataset are all available from the logging data. Unfortunately, for a given approaches, a certain type of input is required and other types of input are just not fit. As a result, we must determine an approach in advance, which is not so flexible. Due to the introduction of clickthrough data, pair-wise training dataset can more accurately and flexibly be extracted. A number of newly introduced learning to rank approaches require not only the preferences or rankings, but also the magnitude of the relevance to learn the rank. Although they can reach good accuracy, it remain to solve that how the magnitude of the relevance can be won from the implicit feedback, which can be expected as the work in the future.

References

[1] M. W. Berry, Z. Drmac, and E. R. Jessup. Matrices, vector spaces, and information retrieval. *SIAM Rev.*, 41(2):335–362, 1999.

[2] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Comput. Netw. ISDN Syst.*, 30(1-7):107–117, 1998.

[3] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to rank using gradient descent. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 89–96, New York, NY, USA, 2005. ACM.

[4] C. J. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 193–200. MIT Press, Cambridge, MA, 2007.

[5] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 129–136, New York, NY, USA, 2007. ACM.

[6] C. Cortes, M. Mohri, and A. Rastogi. Magnitude-preserving ranking algorithms. In *ICML '07: Proceedings of the 24th international conference on Machine learning*, pages 169–176, New York, NY, USA, 2007. ACM.

[7] H. M. de Almeida, M. A. Gonçalves, M. Cristo, and P. Calado. A combined component approach for finding collection-adapted ranking functions based on genetic programming. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 399–406, New York, NY, USA, 2007. ACM.

[8] Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. *J. Mach. Learn. Res.*, 4:933–969, 2003.

[9] J. Gao, H. Qi, X. Xia, and J.-Y. Nie. Linear discriminant model for information retrieval. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 290–297, New York, NY, USA, 2005. ACM.

[10] L. A. Granka, T. Joachims, and G. Gay. Eye-tracking analysis of user behavior in www search. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 478–479, New York, NY, USA, 2004. ACM.

[11] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[12] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, New York, NY, USA, 2002. ACM.

[13] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446, 2002.

[14] M. H. Kutner, C. J. Nachtsheim, and J. Neter. *Applied Linear Regression Models*. McGraw-Hill/Irwin, fourth international edition, September 2004.

[15] P. Li, C. J. C. Burges, and Q. Wu. Mcrank: Learning to rank using multiple classification and gradient boosting. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *NIPS*. MIT Press, 2007.

[16] T.-Y. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *LR4IR 2007, in conjunction with SIGIR 2007*, 2007.

[17] C. D. Manning, P. Raghavan, and H. Schtze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[18] R. Nallapati. Discriminative models for information retrieval. In *SIGIR '04: Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 64–71, New York, NY, USA, 2004. ACM.

[19] T. Qin, X.-D. Zhang, M.-F. Tsai, D.-S. Wang, T.-Y. Liu, and H. Li. Query-level loss functions for information retrieval. *Inf. Process. Manage.*, 44(2):838–855, 2008.

[20] F. Radlinski and T. Joachims. Query chains: learning to rank from implicit feedback. In *KDD '05: Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 239–248, New York, NY, USA, 2005. ACM.

[21] F. Radlinski, R. Kleinberg, and T. Joachims. Learning diverse rankings with multi-armed bandits. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 784–791, New York, NY, USA, 2008. ACM.

[22] G. Salton, A. Wong, and C. S. Yang. A vector space model for automatic indexing. *Commun. ACM*, 18(11):613–620, 1975.

[23] M. Taylor, J. Guiver, S. Robertson, and T. Minka. Softrank: optimizing non-smooth rank metrics. In *WSDM '08: Proceedings of the international conference on Web search and web data mining*, pages 77–86, New York, NY, USA, 2008. ACM.

[24] M.-F. Tsai, T.-Y. Liu, T. Qin, H.-H. Chen, and W.-Y. Ma. Frank: a ranking method with fidelity loss. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 383–390, New York, NY, USA, 2007. ACM.

[25] M. Weimer, A. Karatzoglou, Q. Le, and A. Smola. Cofi rank - maximum margin matrix factorization for collaborative ranking. In *Advances in Neural Information Processing Systems 20*, pages 1600, 1593. MIT Press, 2007.

[26] F. Xia, T.-Y. Liu, J. Wang, W. Zhang, and H. Li. Listwise approach to learning to rank: theory and algorithm. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 1192–1199, New York, NY, USA, 2008. ACM.

[27] J. Xu and H. Li. Adarank: a boosting algorithm for information retrieval. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398, New York, NY, USA, 2007. ACM.

[28] J.-Y. Yeh, J.-Y. Lin, H.-R. Ke, and W.-P. Yang. Learning to rank for information retrieval using genetic programming. In T. Joachims, H. Li, T.-Y. Liu, and C. Zhai, editors, *SIGIR 2007 workshop: Learning to Rank for Information Retrieval*, 27 jul 2007.

[29] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 271–278, New York, NY, USA, 2007. ACM.

[30] Z. Zheng, K. Chen, G. Sun, and H. Zha. A regression framework for learning ranking functions using relative relevance judgments. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 287–294, New York, NY, USA, 2007. ACM.

[31] Z. Zheng, H. Zha, T. Zhang, O. Chapelle, K. Chen, and G. Sun. A general boosting method and its application to learning ranking functions for web search. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 1697–1704. MIT Press, Cambridge, MA, 2008.