LEARNING TO RANK

Yasin Ozan KILIÇ

B.S, Computer Engineering, Işık University, 2008

Submitted to the Graduate School of Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Graduate Program in Computer Engineering

Işık University

2011

IŞIK UNIVERSITY
GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

LEARNING TO RANK

Yasin Ozan KILIÇ

APPROVED BY:

Assist Prof. Olcay Taner YILDIZ	Işık University	____________________
(Thesis Supervisor)		
Assist Prof. Ali INAN	Işık University	____________________
Asist Prof. Çağlar AKSEZER	Işık University	____________________

DATE OF APPROVAL: 28.04.2011

Acknowledgements

I take it as an honour to work with Assist. Prof. Olcay Taner Yıldız. He shared his deep knowledge with me and guided me throughout my thesis. Thanks a million to him for his help an patience.

LEARNING TO RANK

Abstract

The web has grown so rapidly in the last decade and it brought the need for proper ranking. Learning to rank (LTR) is the collection of machine learning technologies that construct a ranking model using training data. The model can sort documents according to their degrees of relevance or preference.

In this thesis, we introduce LTR technologies and divide them into three approaches: the point-wise, pair-wise and list-wise. We review the theoritical aspects of each category and introduce the representative algorithms of them.

We also introduce a new LTR method GRwC which uses classification and graph algorithms. We reduce the ranking problem to a two class classification problem and apply KNN algorithm on a modified LTR dataset. We compared it with the popular ranking algorithm RankingSVM.

Experiments on the well-known ranking datasets show that our proposed method gives slightly worse results than RankingSVM.

SIRALAMA ÖĞRENİMİ

Özet

Sıralama öğrenimi örnek verileri kullanarak bunlardan bir sıralama modeli oluşturan makine öğrenimi metotlarıdır. Bu model dökümanları önemine ya da uygunluğuna bağlı olarak sıralayabilir. Birçok Bilgiye Erişim teknolojisinin temelinde sıralama vardır. Bu yüzden Sıralama öğrenimi teknolojisi ile varolan bu teknolojiler daha da iyileştirilebilir.

Sıralama öğrenimi son yıllarda artan bir popülariteye sahip olmuştur. Bunun temel sebebi Sıralama öğrenimi metotlarının arama motorları tarafından kullanılmaya başlanmış olmasıdır. Büyük arama motoru şirketleri son zamanlarda bir çok Sıralama öğrenimi algoritmaları geliştirmiş ve bu algoritmaları arama sistemlerinde kullanarak iyi sonuçlar almışlardır.

Bu tezde, Sıralama öğrenimi teknolojilerini inceledik ve üç ayrı kategoriye ayırdık: nokta-bazlı, çift-bazlı ve liste-bazlı yaklaşımlar. Ayrıca yeni bir Sıralama öğrenimi algoritması tasarlayıp bunu popüler bir algoritma olan RankingSVM ile karşılatırdık.

Table of Contents

List of Figures

List of Tables

List of Symbols

$\mathbf{X}$	Document collection
$\mathbf{Y}$	Relevance labels
$\mathbf{Q}$	Collection of queries
x_i	Document i in collection X
y_i	Relevance label of x_i
$\mathbf{f}$	Feature vector of a document
L	Loss function
$\mathbf{H}$	Final collection of feature functions
h_i	Weak ranker i in $\mathbf{H}$
α_i	Weight of weak ranker h_i
ϕ	Feedback function of relevance labels
$\pi(x_i)$	Potential of document x_i
$\mathbf{g(q)}$	Ground truth ranking of query q
n(q)	Number of documents in query q
$\mathbf{W}$	Query weight vectors
π	Each possible permutation of documents in a query

Chapter 1
Introduction

The web develops so rapidly that everyone experience a flood of information. It is estimated that there are 14 billion pages on the web as of December 2010. This situation makes it very hard for everyone to find the desired information by browsing the web. As a result, effective and efficient information retrieval (IR) has become very important and today search engines are an essential tool for many people.

Ranking is an important problem in IR. Most of the IR problems are naturally ranking problems, i.e. document retrieval, collaborative filtering [1], key term extraction [2], definition finding [3], important email routing [4], sentiment analysis [5], product rating [6], and anti web spam [7]. In this thesis, we will focus on document retrieval where academic papers, web pages, news articles, emails are just a few examples of documents. There are many kinds of interesting ranking problems in document retrieval:

- The documents are ranked only in terms of relevance to the query.
- Relational ranking [8] which gives importance to the relationship of web site structure, completeness of documents in the ranking.
- Combining more than one candidate ranking lists to create a better one. The lists can come from different index servers or vertical search engines and the final ranking list is the result presented to users.
- Finding in what degree a feature of a web page effects the ranking result. This

approach is called reverse engineering in search engine optimization (SEO).

Ranking in document retrieval is not an easy problem, so many ranking algorithms have been proposed in IR. In recent years, to create good ranking models, researchers started to use machine learning techniques due to possibility to reach large set of training data. The data come from either machine learning conferences like TREC or commercial web search engines. The methods that learn the ranking model from the training data are called "learning to rank" (LTR) methods. These methods learn how to combine the specified features of documents for ranking.

One can define LTR as the following: LTR is a set of algorithms that utilize machine learning techniques to solve the ranking problems. Usually the existing ranking models include human calculated equations of features and weights for feature parameters of score calculations. LTR algorithms try to learn the best way of combining the features of a document query pair. If a ranking method has following two features, we call them learning to rank methods.

- Method is feature based: The documents of the queries are represented as feature vectors. Most commonly used features in LTR are the frequency information of the query terms, PageRank and BM25 scores etc.
- Training: The learning of the model is in four components of discriminative learning. LTR algorithm has an input, output and hypothesis spaces and a loss function.

Some LTR methods also have online learning process which is preferred by commercial search engines. Given hundreds of thousands feedbacks from the users, it is very important to learn from the feedback to improve the ranking mechanisms.

LTR has become an active research topic in the latest years. Due to this trend, significiant number of algorithms are proposed ([9–28]). It is predicted that LTR will have a bigger impact on IR in the next years.

Since learning to rank is a new research area, there are some questions arisen by the researchers:

- What are the similar and different aspects of the LTR algorithms?
- Which LTR algorithm performs the best? How can we compare different LTR algorithms?
- Can ranking be defined as a new kind machine learning problem or can it be reduced to existing machine learning problems? Are there any unique theoretical problems of ranking?

In this thesis, we will briefly review LTR algorithms and try to answer the first question. We also reduce the ranking problem to a two-class classifier problem and solve it using KNN classification algorithm. This thesis is organized as follows: In Chapter 2, a brief introduction on ranking in IR will be given. The ranking models and the evaluation measures will be studied. In Chapter 3, LTR algorithms will be studied more detailed. In Chapter 4, we propose a classification based LTR algorithm and in Chapter 5 we compare its performance with the populer LTR algorithm RankingSVM. We conclude in Chapter 6.

Chapter 2
Ranking in Information Retrieval

2.1. Ranking Models in IR

In information retrieval, there are many known ranking models. To give a brief introduction, we will divide them into two categories as query-dependent ranking models and query-independent ranking models.

2.1.1. Query-dependent Ranking Models

The very first ranking models rank documents according to the occurrences of query terms in documents. We can give the Boolean model as an example. These kinds of models cannot predict the degree of relevance truly but they can predict if the document is relevant to the document or not. To improve the relevance degree, vector space model was introduced. In this model, documents and queries are modeled as vectors in the Euclidean space. The inner product of these two vectors can be used to measure the similarity between the document and the query. To get a reasonable similarity, an effective vector representation of query and documents should be built. For this purpose TF-IDF weighting is widely used. The TF (Term Frequency) of term t in a vector is the normalized number of its occurrences in the document. IDF (Inverse Document Frequency) is

$$IDF(t) = \log \frac{N}{n(t)} \tag{2.1}$$

where N is the number of documents and *n(t)* is the number of documents with term t.

In addition above, researchers developed ranking models based on the probabilistic ranking principle [29]. Examples of this kind of models include BM25 and language model for information retrieval.

BM25 is used to rank the documents by the log-odds of their relevance. Actually, it is not a single ranking model; it uses a variety of ranking models with slightly different components and parameters. General representation of the model is

$$BM25(d,q) = \sum_{i=1}^{M} \frac{IDF(t_i)TF(t_i, D(k_1+1)}{TF(t_i, D) + k_1(1 - b + b\frac{LEN(d)}{avdl})} \tag{2.2}$$

where q is the given query, containing the terms $t_1, ..., t_M$, d is given the document, *TF(t,d)* is the term frequency of term t in the document d, *LEN(d)* is length of the document d (number of words), *avdl* is the average document length of the main index, k_1 and b are free parameters, *IDF(t)* is the *IDF* of t.

Language model for information retrieval [30] is another ranking model which utilizes statistical language model on information retrieval. A statistical language model developed to assign a probability to the terms. When we use it in information retrieval, the language model is associated with the document. Documents are ranked based on the probability that document's language model generates the terms in the query. We define this probability as follows:

$$P(q|d) = \prod_{i=1}^{M} P(t_i|d) \tag{2.3}$$

where q is the query which contains the terms $t_1, ..., t_M$.

Training the document's language model is done with the maximum likelihood method. One point we should be careful is the smoothing of the estimate. For this puporse, a background language model estimated using the entire index. Then the language model of the document is

$$p(t_i|d) = (1-\lambda)\frac{TF(t_i,d)}{LEN(d)} + \lambda p(t_i|C) \tag{2.4}$$

where $p(t_i|C)$ is the background language model for t_i and $\lambda \in [0,1]$ is the smoothing factor.

There are also other ranking models that try to find the relevance between the documents and queries. These models use different approaches. Some give more importance to the relationship between documents in terms of content similarity, web site structure, hyper-link structure, and topic diversity; some consider the proximity of the query terms more.

2.1.2. Query-independent Ranking Models

The models we mentioned so far rank documents based on the query and document relationship, this is why we call them query-dependent models. On the other hand, query-independent ranking models rank the documents based on their importance compared to other documents. We will explain PageRank as an example.

PageRank [31] is mostly applicable to web searching because it uses the hyperlink structure of the web. The probability that a user randomly clicks through the links and arrives to a web page is

$$PR(d_u) = \sum_{d_v \in B_u} \frac{PR(d_v)}{U(d_v)} \tag{2.5}$$

where *PR(d_u)* is the PageRank value of page *d*, B_u is the collection of the web pages linking to d_u, and $U(d_v)$ is the number of outer links from the page d_v.

In addition above, there is a possibility that a user does not follow the hyperlink structure of the web and visits a web page directly. The probability of this event is called the damping factor which is calculated as $(1 - \alpha)$. So the PageRank of the document d_v is

$$PR(d_u) = \alpha \sum_{d_v \in B_u} \frac{PR(d_v)}{U(d_v)} + \frac{1-\alpha}{N} \tag{2.6}$$

where N is the total number of pages in the index.

In addition to PageRank, there are other link analysis models like hyperlink induced topic search [32] and TrustRank [7]. Some of these use content or topic information in addition to the link analysis.

2.2. Query-level Evaluation in Information Retrieval

From the large collection of ranking models a standard and correct evaluation function is required to select the best model. The general procedure for doing this evaluation is as follows:

- A sufficient number of queries are collected and they form a test set.
- The following steps are applied for each query in the set:
  - Gather the documents in the queries' result set
  - Use human judgments for getting the relevance value of each document
  - Apply a ranking model on queries.
  - Compute the difference between the ground truth ranking list and the result of the ranking model using an evaluation measure.

- Evaluate the performance of the given ranking model using the average measure.

Collecting the documents for the test set is a critical job, because we need quality data for the ranking models. There are many different ways to collect the data. One example is the pooling strategy. This method is used in TREC (Text REtrieval Conference). The procedure of this strategy is as follows:

- Create sample queries.
- Collect documents associated with each query. At this step many predefined ranking models can be used to gather the relevant documents, for example existing search engines.
- Merge the documents of each query from different sources to a pool for human judgment.
- Use human assessment to create a ground truth ranking list for each query from the associated documents in the pool.

Assessments made by human are also a critical factor. There are three methodologies used in IR for this purpose:

- Defining the document as relevant or not relevant. The relevance can be defined in multiple levels like perfect, good, bad.
- Specifying the relative preference between the documents of the query. The judgments are done only in document pairs.
- Specifying the total orders of the queries' document.

Human assessment is a time and resource consuming process. Due to this fact, it is not always possible to do right judgments on the documents of a query. As a consequence, there are always not-judged documents in the index. These documents are labeled as irrelevant in common practice.

There are other evaluation measures introduced in IR. These measures also state the true objectives of the ranking. Some of the measures are as follows:

Mean reciprocal rank (MRR) : The ranking position of the first relevant document of a query is shown as *r(q)*. So, $\frac{1}{r(q)}$ is used to formulate the *MRR* for the query *q*. The documents with the position below *r(q)* is not in *MRR*.

Precision at position k (P@k): The precision at position k is defined as

$$P@k(q) = \frac{\#\{\text{relevant documents in the top } k \text{ positions}\}}{k} \tag{2.7}$$

Mean average precision (MAP): The average precision calculated as

$$AP(q) = \frac{\sum_{k=1}^{m} P@k(q) l_k}{\#\{\text{relevant documents}\}} \tag{2.8}$$

where m is the number of the documents of the query q, l_k is the judgment made about the relevance of the document at position *k*. This judgment is binary. MAP is defined as the mean AP value of all queries in the collection.

Discounted cumulative gain (DCG): *MRR* and *MAP* are used for binary judgment and thus they are not very useful when documents are ranked as in multiple levels of judgments or in total order as list. DCG [33] is defined to be used in these situations. Let π be the ranked document list of query q, then *DCG* at position k is

$$DCG@k(q) = \sum_{r=1}^{k} G(\pi^{-1}(r))\mu(r) \tag{2.9}$$

where $\pi^{-1}(r)$ is the *rth* document in the document list π. $G(\cdot)$ is called the rating of the document and $\mu(r)$ is the position discount factor which is calculated as

$$\mu(r) = \frac{1}{\log_2(r+1)} \tag{2.10}$$

Normalized DCG ($NDCG$) is the normalized value of $DCG@k$

$$NDCG@k(q) = \frac{1}{Z_k}\sum_{r=1}^{k} G(\pi^{-1}(r))\mu(r) \tag{2.11}$$

where Z_k is the maximum value of $DCG@k$

These evaluation measures have some common properties:

- They are all query level. The values of the measures are different for each query, since they are calculated separately. The average value of these calculations on all the queries is used. This way the quality of the test set is increased, because the loss from the poorly selected relevant documents is decreased on overall collection.
- Rank position is used for all of them so they are position based. So they are not sensitive to the small changes in the documents scores but they give more importance to the order in the list.

Chapter 3
Learning to Rank

Learning to rank algorithms can be designed with a variety of methodologies. Looking at the literature, we see that there are three main approaches in LTR. The differences derive from the input, output, hypothesis spaces, and loss functions. Now we will review the main differences of the approaches and give example algorithms associated with these approaches.

3.1. The point-wise approach

The goal of this approach is to determine the relevance degree of each document. Thus, the input space of this approach is the feature vector of all documents. The algorithm's output space is formed with the relevance degrees of every document in the collection. The collection of functions that take the feature vector as input and give the relevance degree of a document are the hypothesis space of this approach. Loss functions of point-wise approach analyze the relevance prediction of every document to the ground truth degree. Different point-wise approach algorithms have different loss functions as the ranking model can be different like regression, ordinal regression and classification. So, the loss becomes regression loss, classification loss, etc. Example algorithms of point-wise approach are ([17, 19, 22–24, 26, 34–36]).

3.1.1. Multi-class Classification for Ranking

McRank was proposed by Li *et al* [23] where multi-class classification is used to build ranking models. Lets say document collection $\mathbf{X} = \{\mathbf{x_j}\}_{j=1}^{m}$ and the relevance labels $\mathbf{Y} = \{\mathbf{y_j}\}_{(j=1)}^{m}$ are associated with query q. Then we are able to say that we have a multi-class classifier which predicts $\hat{y}_j$ on x_j. The loss function for this classifier is defined as

$$L(\hat{y}_j, y_j) = I_{y_j \neq \hat{y}_j} \tag{3.1}$$

where I is the indicator function.

To convert the classification results into ranking scores, the classifier results are converted to a probability distribution, which shows the probability of a document to be under a category. We can define this probability as $P(\hat{y}_j = k), k = 0, ..., K-1$ where K is the number of the categories. Then the final ranking score function is

$$H(x_j) = \sum_{k=0}^{K-1} kP(\hat{y}_j = k) \tag{3.2}$$

3.1.2. Subset Ranking with Regression

Cossock and Zahn [19] propose to solve the ranking problem by reducing it to a regression problem. Suppose scoring function f is used to rank a set of documents $\mathbf{X} = \{\mathbf{x_j}\}_{j=1}^{m}$ belonging to query q with the relevance labels $\mathbf{Y} = \{\mathbf{y_j}\}_{j=1}^{m}$. They formulate the loss function as regression loss

$$L(f; \mathbf{x_j}, \mathbf{y_j}) = (\mathbf{y_j} - f(\mathbf{x_j}))^2 \tag{3.3}$$

3.1.3. Other Point-wise Algorithms

Chu and Ghahramani [34] propose a probabilistic kernel approach to ordinal regression Gaussian process. They use two interference techniques which are based on Laplace approximation and the expectation propagation algorithm.They compare these techniques with the previous ordinal regression models and report that the data sets they use satisfy the usefulness of their approach.

Cooper *et al* [35] combine the methods of statistical independence assumptions and multiple regression analysis without causing additional computational complexity. The most important element in their approach is that the regression analysis is carried out in two or more levels (stages). Such approach allows composite or grouped retrieval clues to be analyzed in an order, first within groups and then between the groups.

Crammer and Singer [17] discuss the problem of ranking. In their framework, each instance is associated with a rank or a rating which from one to k. The goal is to find a rank prediction rule which has the minimum loss value. They explain an online ranking algorithm to minimize the cost of their rank prediction rule, PRank.

Gey [36] proposes a model for probabilistic text and document retrieval and this model utilizes logistic regression techniques. They call the model as logistic inference. They use the principle that when one transforms the distribution of each statistical clue into its standardized distribution ($\mu = 0$ and $\sigma = 1$), their method allows to apply logistic coefficients derived from a training collection to the other one. They apply the model to the well-known data sets and have good results.

Nallapati [24] explores the applicability of discriminative classifiers for IR. They compare two popular discriminative models, maximum entropy and support vector machines with language modeling. The experiments show that maximum entropy is significantly worse than language model and SVM's are on par with language modeling.

3.2. Pair-wise Approach

Instead of finding each document's relevance to the query like point-wise approach, pair-wise approach finds the relative order between two documents. We can say that pair-wise approach is closer to the ranking concept than point-wise approach. The ranking problem is taken as a classification on document pairs. The goal of learning is to have minimum amount of miss-classified document pairs. So, if all the document pairs are correctly classified, then it means that the whole document list of a query q is correctly ranked.

The pair-wise approach has an input space of document pairs that are represented as feature vectors. The output space is the preference label between the documents of the pairs. Then we can say the output space has the values of 1,-1 to represent the preference. The hypothesis space has the functions h that compute the relative order between the given document pairs. The loss function examines the inconsistency between the output of h functions and the ground truth labels. There are many algorithms in this category such as [9, 11, 12, 15, 21, 27].

3.2.1. RankBoost

RankBoost [18] inherits the AdaBoost [37] algorithm for classification on document pairs. AdaBoost classifies each single document, where RankBoost classifies the document pairs. The algorithm defines a distribution over the document pairs and updates it according to the current selected weak ranker's loss value. The pseudocode of RankBoost is give in Figure 3.1.

```
1 Given: initial distribution $D$ over $\mathbf{X}x\mathbf{X}$
2 Initialize: $D_1 = D$
3 For  $t = 1, ....., T$:
4     Train weak learner using distribution $D_1$
5     Get weak ranking $h_t : X \rightarrow \mathbb{R}$
6     Choose $\alpha_t \in \mathbb{R}$
7     Update $D_{t+1}(\mathbf{x_0}, \mathbf{x_1}) = \frac{D_{t+1}(\mathbf{x_0},\mathbf{x_1}) exp(\alpha_t(h_t(\mathbf{x_0})-h_t(\mathbf{x_1})))}{Z_t}$ ,
      where $Z_t$ is the normalization factor
8 End For
9 Output the final ranking $\mathbf{H}(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$
```

Figure 3.1. Algorithm RankBoost.

For a document pair $(\mathbf{x_1}, \mathbf{x_2})$, the distribution D value is defined as

$$D(x_1, x_2) = cmax\{0, \phi(\mathbf{x_1}, \mathbf{x_2})\} \tag{3.4}$$

where ϕ is the feedback function that outputs the ground truth pair-wise preference between x_1 and x_2 which either -1 if x_1 is preferred over x_2, 0 if both preferred equally and +1 if x_2 is preferred over x_1. c is the normalization factor since

$$\sum_{\mathbf{x_1},\mathbf{x_2}} D(\mathbf{x_1}, \mathbf{x_2}) = 1 \tag{3.5}$$

The final ranking is formed by selecting weak rankings during the training and weights assigned to them. RankBoost defines the rankers as functions that output the linear ordering of each feature in the document's feature vector.

In each iteration, a weak ranker h_t is selected and a weight α_t is assigned to it. There are three approaches to assign the weight α_t

1. For any weak ranker h_t, the normalization factor Z_t can be represented as a function of α_t. The unique minimum of the factor value gives as the value of α_t.
2. This approach is applicable when h_t has the value in range of [0, 1]. Z_t is minimized as follows: For $b \in \{-1, 0, 1\}$

$$\mathbf{W}_{t,b} = \sum_{i=1}^{n} \sum_{u,v:y_{u,v}^{(i)}=1} D_t(\mathbf{x_u}^{(i)}, \mathbf{x_v}^{(i)}) I_{\{h_t(\mathbf{x_u}^{(i)}) - h_t(\mathbf{x_v}^{(i)}) = b\}} \tag{3.6}$$

   Then

$$\alpha_t = \frac{1}{2} \log(\frac{\mathbf{W}_{t,-1}}{\mathbf{W}_{t,1}}) \tag{3.7}$$

3. This approach is based on the approximation of Z_t, which is the applicable when f_t has the value in range of [0,1]. We define :

$$r_t = \sum_{i=1}^{n} \sum_{u,v:y_{u,v}^{(i)}=1} D_t(\mathbf{x_u}^{(i)}, \mathbf{x_v}^{(i)})(f_t(\mathbf{x_u}^{(i)}) - f_t(\mathbf{x_v}^{(i)})) \tag{3.8}$$

   Then

$$\alpha_t = \frac{1}{2} \log\left(\frac{1 + r_t}{1 - r_t}\right) \tag{3.9}$$

The designers of the algorithm selected the third method for finding α_t. r value for a given ranker is defined as

$$r = \sum_{\mathbf{x_0},\mathbf{x_1}} D(\mathbf{x_0},\mathbf{x_1})(h(\mathbf{x_1}) - h(\mathbf{x_0})) \tag{3.10}$$

$$= \sum_{\mathbf{x}} h(\mathbf{x})\pi(\mathbf{x}) \tag{3.11}$$

where $\pi(\mathbf{x}) = \sum_{\mathbf{x'}}(D(\mathbf{x},\mathbf{x'}) - D(\mathbf{x'},\mathbf{x}))$ is called the potential of the document $\boldsymbol{x}$. Note that, $\pi(\mathbf{x})$ is calculated from the current distribution in each iteration.

To minimize α_t at each iteration step, we select the weak ranker which maximizes r. At the end of the iterations, the selected weak rankers along with their weights form the final ranking function.

3.2.2. Ranking SVM

Ranking SVM [21] uses SVM for pair-wise classification. Given the query set of $\{q_i\}_{i=1}^{n}$ we generate the document pairs of $(\mathbf{x_u^i},\mathbf{x_u^i})$. Note that the document pairs are generated for each query. So we don't pair the documents from different queries. The ground truth label of pair-wise preference between $(\mathbf{x_u^i},\mathbf{x_u^i})$ is denoted as $y_{u,v}^i$.

The formulation of Ranking SVM is given as

$$\begin{aligned} min 1/2 \| \mathbf{w} \|^2 + C \sum_{i=1}^{n} \sum_{u,v;y_{u,v}^i=1} \xi_{u,v}^i \\ s.t. \mathbf{w^T}(\mathbf{x_u^i} - \mathbf{x_u^i}) \geq 1 - \xi_{u,v}^i, if y_{u,v}^i = 1, \\ \xi_{u,v}^i \geq 0, i = 1, ..., n \end{aligned}$$

The term $\frac{1}{2} \parallel \mathbf{w} \parallel^2$ controls the complexity of the model w. Unlike SVM, the constraints are constructed from the document pairs. The loss function in Ranking SVM is the hinge loss defined on the document pairs.

Since Ranking SVM inherits many features from SVM framework, with the help of margin maximization, it can have good generalization. Also, Kernel tricks can be applied to Ranking SVM, so it can handle complex non-linear problems.

3.2.3. Other Pair-wise Algorithms

Burges *et al* [11] investigate the gradient descent methods for learning ranking functions. They propose a simple probabilistic cost function and a neural network based learning algorithm RankNet to minimize their cost function.

Cohen *et al* [15] introduce an on-line learning algorithm based on the "Hedge" algorithm to find a good linear combination of ranking experts. They also show that finding an ordering that agrees best with a preference function is NP-complete.

Tsai *et al* [27] conduct further study on RankNet [11] and propose a new loss function named Fidelity to measure loss of ranking. To efficiently minimize Fidelity loss, they propose a learning algorithm called FRank.

3.3. List-wise approach

Unlike the two other approaches that does predictions on document level or document pair level, list-wise approach try to minimize the difference between the ground truth ranking list of a document and the output ranking list of the ranking model.

The input space of this approach is formed of the entire collection of the documents belonging to query q. The output space contains the ranked list of the documents of a query. The hypothesis space contains the set of ranking functions that predict the ranking order of the given documents. The loss function computes the difference between the output ranking list of the hypothesis and the ground truth list from the input space. Example algorithms of this approach include [10, 13, 25, 38–41].

3.3.1. RankCosine

Qin *et al* proposed a list-wise LTR algorithm called RankCosine [41] which uses cosine similarity as the loss function.

Suppose there are *n(q)* documents for a query q. The ground truth ranking list for this query is ***g**(q)* which is a *n(q)-dimension* vector. The documents are listed from one to n and the kth element of the ranking list is the rating of document k in the list. For example, if we have document collection such as *A, B, C, D*; one instance of the ranking list of this collection would be 2, 3, 1, and 4. The absolute value of the scores is not important, in RankCosine (and in most of the other list-wise algorithms) only the difference between scores matters which is the linear ordering of the documents. The output of learning algorithm for a query q is defined as ***H**(q)*. This output is also an *n(q)*-dimension vector same as ***g**(q)*. Also, the kth element in the output list is the rating of the kth document given by the algorithm.

The ranking loss for a query q is defined as follows:

$$L(\mathbf{g}(p), \mathbf{H}(q)) = \frac{1}{2}(1 - cos(\mathbf{g}(q), \mathbf{H}(q))) = \frac{1}{2}(1 - \frac{\mathbf{g}(q)^T \mathbf{H}(q)}{\parallel \mathbf{g}(q) \parallel \parallel \mathbf{H}(q) \parallel}) \tag{3.12}$$

where $\parallel \cdot \parallel$ is L-2 norm of a vector. Since they use cosine similarity in this function, it is called as cosine loss.

The goal of learning in this setting is to minimize the total loss function value over all the queries that are in training set Q;

$$L(H) = \sum_{q \in Q} L(\mathbf{g}(p), \mathbf{H}(q)) \tag{3.13}$$

For the final ranking function an additive model is used

$$\mathbf{H}(q) = \sum_{t=1}^{T} \alpha_t \mathbf{h_t}(q) \tag{3.14}$$

where α_t is the combination weight and $h_t(q)$ is the weak learner which maps an input matrix whose one row is the feature vector of a document, to an output vector whose elements are the scores of the documents.

$$\mathbf{h_t}(q) : R^{n(q)xd} \to R^{n(q)} \tag{3.15}$$

where d is the dimension of the feature vector.

To train the parameters of the final ranking, a stage-wise greedy search strategy is used. At each step a new weak learner is selected whose cosine loss is minimum. When the new weak learner is selected, it is combined with the previously selected weak learners.

If we define the final ranking function as

$$\mathbf{H_k}(q) = \sum_{t=1}^{k} \alpha_t \mathbf{h_t}(q) \tag{3.16}$$

where $\mathbf{h_t}(q)$ is the weak learner selected at step t, we can define the cosine loss at step

t as

$$L(\mathbf{H_k}) = \sum_q \frac{1}{2}\left(1 - \frac{\mathbf{g}(q)^T(\mathbf{H_{k-1}}(q) + \alpha_t \mathbf{h_k}(q))}{\sqrt{(\mathbf{H_{k-1}}(q) + \alpha_t \mathbf{h_k}(q))^T(\mathbf{H_{k-1}}(q) + \alpha_t \mathbf{h_k}(q))}}\right) \tag{3.17}$$

When we set the derivative of $L(\mathbf{H_k})$ with respect to α_t to zero, we get the optimal value of α_t as

$$\alpha_t = \frac{\sum_q \mathbf{W_{1,k}}^T(q)\mathbf{h_k}(q)}{\sum_q \mathbf{W_{2,k}}^T(q)(\mathbf{h_k}(q)\mathbf{g}^T(q)\mathbf{h_k}(q) - \mathbf{g}(q)\mathbf{h_k}^T(q)\mathbf{h_k}(q)} \tag{3.18}$$

where $\mathbf{W_{1,k}}(q)$ and $\mathbf{W_{2,k}}(q)$ are given as

$$\begin{aligned} \mathbf{W_{1,k}}(q) &= \frac{\mathbf{g}^T(q)\mathbf{H_{k-1}}(q)\mathbf{H_{k-1}}(q) - \mathbf{H_{k-1}}^T(q)\mathbf{H_{k-1}}(q)\mathbf{g}(q)}{\| \mathbf{H_{k-1}}(q) \|^{3/2}} && (3.19) \\ \mathbf{W_{2,k}}(q) &= \frac{\mathbf{H_{k-1}}(q)}{\| \mathbf{H_{k-1}}(q) \|^{3/2}} && (3.20) \end{aligned}$$

Using these equations we calculate the optimal weight of α_t, evaluate cosine loss for each weak learner candidate, and select the one with the smallest loss as the kth weak learner. At the end, we get a collection of weak learners and their combination coefficient which makes the final ranking function.

Figure 3.2 shows the pseudocode of RankCosine. Here, $e_{n(q)}$ is an *n(q)*-dimensional vector with all elements equal to one.

```
1  Given: ground truth $g(q)$ for all queries $Q$, weak learner candidates $h_i(q), i = 1, 2...$
2  Initialize: $\mathbf{W_{1,1}}(q) = \mathbf{W_{2,1}}(q) = \frac{\mathbf{e_{n(q)}}}{n(q)}$
3  For $t = 1, 2, ..., T$
4      For each weak learner candidate $h_i$
5          Compute optimal $\alpha_{t,i}$
6          Compute the cosine loss $\varepsilon_{t,i}$
7      End For each
8      Choose weak learner $h_{t,i}$ which has the minimal loss as $h_t$.
9      Choose coefficient $\alpha_{t,i}$ as $\alpha_t$.
10     Update query weight vectors $\mathbf{W_{1,t}}(q)$ and $\mathbf{W_{2,t}}(q)$
11 End For
12 Output the final ranking function $\mathbf{H}(q) = \sum_{t=1}^{T} \alpha_t \mathbf{h_t}(\mathbf{q})$
```

Figure 3.2. Algorithm RankCosine

3.3.2. ListNet

ListNet [13] is another LTR algorithm that has a list-wise ranking loss based on the probability distribution on permutations. The distribution on permutations is well-studied in probability theory. There are famous distribution models like Luce model and the Mallows model. Since we can treat a ranked list as a permutation of the document list, these models can be applied easily. ListNet applies Luce model to the LTR.

The relevance scores are given by the scoring function h like $s = {s_j}_{j=1}^{m}$ where $s_j = h(\mathbf{x_j})$. The Luce model assigns a probability to each possible permutation π of the documents.

$$P(\pi|s) = \prod_{j=1}^{m} \frac{\varphi(s_{\pi^1(j)}}{\sum_{u=j}^{m} \varphi(s - 1_{\pi(u)})} \tag{3.21}$$

where $\pi^{-1}(j)$ points the document that is ranked at position j of the list, or permutation π. φ is a transformation function. Each item $\frac{\varphi(s_{\pi^{-1}(j)})}{\sum_{u=1}^{m} \varphi(s_{\pi^{-1}(u)})}$ is a conditional probability shown as the following example.

Assume that we have three documents $\boldsymbol{X}, \boldsymbol{Y}$ and $\boldsymbol{Z}$ belonging to the query q. The permutation probability $\pi = (\mathbf{X}, \mathbf{Y}, \mathbf{Z})$ is the product of the following three probabilities, $P_\pi = P_1 P_2 P_3$

- P1: The probability $\boldsymbol{X}$ being ranked at the top position in π
- P2: The conditional probability of $\boldsymbol{Y}$ being at second in the permutation after $\boldsymbol{X}$.
- P3: The conditional probability of $\boldsymbol{Z}$ being at third position after $\boldsymbol{X}$ and $\boldsymbol{Y}$. In this simple case, P_3 equals to 1.

For a given query q, ListNet computes the permutation probability distribution based on the scores of function f. Also, it defines another permutation probability distribution $p_y(\pi)$ based on the ground truth list. Then it uses K-L divergence between these two distributions and computes the loss function

$$L(f; \mathbf{x}, \pi_y) = D(P(\pi|\varphi(f(w, \mathbf{x}))) \parallel P_y(\pi)) \tag{3.22}$$

Then, the algorithm uses gradient descent based neural network to minimize the K-L divergence loss.

3.3.3. Other List-wise Algorithms

LambdaRank [10] is a class of simple flexible algorithms which use implicit const functions. Burges *et al* [10] note that the quality measures in IR are difficult to optimize since they depend on the model scores only through the sorted order of the documents returned for a given query. Derivatives of the cost with respect to the model parameters are either zero or undefined. Thus they give the necessary conditions for the resulting implicit cost function to be convex.

Taylor *et al* [38] present a new family of training objectives that are derived from the ranks distributions of the documents. They call this approach SoftRank. Most IR applications use evaluation metrics that depend only upon the ranks of documents. However, most ranking functions generate document scores, which are sorted to produce a ranking. Hence IR metrics are non-smooth with respect to the scores, due to the sorting. They also did approximation on Normalized Discounted Cumulative Gain (NDCG) and proposed SoftNDCG.

Yue *et al* [40] states that few learning tecniques have been developed to directly optimize for mean average precision (MAP) and the existing methods don't find a globally optimal solution. They present a general SVM learning algorithm that efficiently finds a globally optimal solution to a straightforward relaxation of MAP.

Chapter 4
Graph Ranking with Classification

In this chapter, we will give the details of our novel LTR algorithm which first transforms the ranking problem into a two-class classification problem and then use KNN to classify the documents.

Every query q has the set of associated documents $\boldsymbol{X}$ which is either ordered linearly or labeled in terms of relevancy. Every document $\mathbf{x_i} \in \mathbf{X}$ has a set of features $f_1, ..., f_k$ which have numerical values of the corresponding ranking models. The goal here is to find the weights of each feature on the final scoring function and linearly combine them. Since a document $\mathbf{x}$ is the vector of its feature values, we will do classification on feature values of the documents.

Before doing any training, we need to prepare the data set for classification. Thus, we create two groups of pairs, + and -. Every entry under a group is a feature vector pair aligned side by side. For example, a pair consists of two documents $\mathbf{x_i}$ and $\mathbf{x_j}$. Then each entry in our data set is the features of two documents combined together as $f_{0,i}, f_{1,i}, ..., f_{k,i}, f_{0,j}, f_{1,j}, ..., f_{k,j}$. If a pair is in group +, it means that the ranking of the first document is higher than the second. If a pair is in group -, then the second document is ranked higher than the first document.

The document pairs should be created from the same query. So, we can't pair the documents from different queries because this would be same as comparing apples

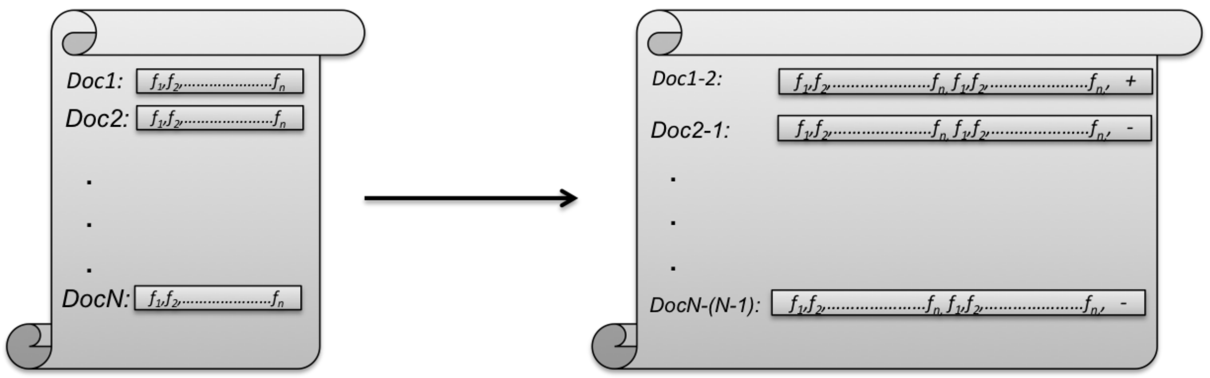
Doc1:
$f_1, f_2, \ldots f_n$
Doc2:
$f_1, f_2, \ldots f_n$
DocN:
$f_1, f_2, \ldots f_n$
Doc1-2:
$f_1, f_2, \ldots f_n, f_1, f_2, \ldots f_n,$ +
Doc2-1:
$f_1, f_2, \ldots f_n, f_1, f_2, \ldots f_n,$ -
DocN-(N-1):
$f_1, f_2, \ldots f_n, f_1, f_2, \ldots f_n,$ -

Figure 4.1. Conversion of dataset for GRwC

to oranges. Since we prepare the data set from the scratch, we can work on two types of data sets, relevance labeled queries and linear ordered queries. In relevance labeled queries, we can easily generate the pairs and put them to the corresponding group by comparing their relevance label. If they are equal, we don't put them to any group. If the data set is formed of linear ordered queries, then we can do the same as labeled data. The difference is that if a query has n documents, then there are n different relevance labels, meaning the ranking of the document in the ordered list. To generate the pairs, we only take the top k documents of each query, avoiding large memory footprint. Figure 4.1 illustrates the conversion of one query to GRwC format.

After preparing the dataset, a classification algorithm (in our case KNN) is applied which predicts the pair-wise preferences for each document pair. Since we have to output the ranked list of the documents, we have to generate a linear ordering of the documents from the prediction results of the classification algorithm. To do this, we build a document graph in which each node is a document in the query and each directed edge indicates the value of preference function between two documents. The direction of the edges are determined from the prediction results of the classification algorithm. For example, if the document A is preferred over document B according to the prediction, then there should be an edge from node A to node B in the document graph.

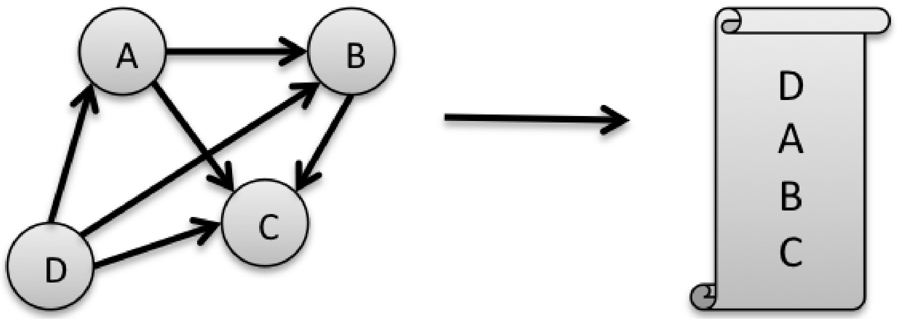

Figure 4.2. Generating ordered list of documents from the document graph

After building the document graph, we run topological sort algorithm to get the linear order of the documents. To sort a graph topologically, we need to have at least one node without incoming edges. However, since we have prediction errors, we will have cycles in the document graph, thus we can't apply topological sort directly. So we use a modified topological sort algorithm which selects the node with minimum incoming edge instead of the node with no incoming edges. At the end, the result of our algorithm is the output of the topological sort on the document graph.

Figure 4.2 shows an example of topological sort. We see four document (A,B,C,D) with incoming edges count as (1,2,3,0) respectively. At the first iteration, we choose D as it has the minimum number of incoming edges. We remove D and its outgoing edges from the graph. After D, documents (A,B,C) remain in the graph with incoming edges count as (0,1,2) respectively. This time we remove A and its outgoing edges. Documents (B,C) remain in the graph with incoming edges count as (0,1) respectively. We remove B from graph and lastly we remove C which is the last document remaining in the graph. At the end we have the linear order of document as (D,A,B,C).

Chapter 5
Experiments and Results

5.1. Setup

We use Letor dataset [42] in our experiments. Letor contains two differents datasets, namely OHSUMED [43] and TREC Web Track [44]. OHSUMED is a dataset on medicine. It has three levels of relevance judgements: “definitely relevant”, “possibly relevant” and “not relevant”. TREC dataset contains web pages crawled from the .gov domain in the early 2002. The ground truths of the sample queries in the dataset are provided by TREC committee as binary judgements. Table 5.1 shows the list of the features we used in these datasets.

We used Weka [45] for training and predicting the pair-wise preference of the documents. We have to convert the dataset from Letor format to the ARFF format. In the ARFF format, we need to provide the attributes values in the header. So, we define 93 attributes from which 92 attributes are features of documents and 1 attribute is class attribute which has values Positive and Negative. In Letor dataset, there are 46 features. The reason we have 92 features for Weka tool is that we align the features of two documents side by side. If the first document is preferred over the second, then 93th attribute is labeled as Positive, otherwise Negative. Note that we don’t put documents side by side if they are equally judged. After converting the dataset, we train the model using K-nearest neighbor (KNN) classification algorithm, where we save the model for later use.

To test our model, we need to do processing on the test dataset. We prepare 80 test queries from test dataset and converted each of them to ARFF format. The result of tests from Weka don't contain the identities of the documents so we have to generate another file which contains the document id of all pairs in the test queries. For example, if the first line of the test query file contains the feature values of documents **X** and **Y**, then the identity files' first line contains the id of documents **X** and **Y**. So this way, we know what documents are pointed with each prediction.

When the prediction is finished for a test query, we still don't have the linear ordering of the documents for that query. We have only the pair-wise preference judgements for each document pair. So, to get a linear ordering, we build the graph of all documents in that query and apply topological ordering. To use topological ordering the graph should have at least one vertex which does not have any incoming edges. However, since our prediction has some error, the document graph don't have any vertex obeying this rule. So, we use a modified version of topological ordering algorithm. In this version, we chose the next vertex having minimum number of incoming edges. This way, we construct the linear ordering of our test queries.

5.2. Results

We compare GRwC to the well-known LTR algorithm RankingSVM. We use SVMlight tool [46] to train RankingSVM on Letor datasets. For RankingSVM, we use the same training and test queries that we prepared for GRwC.

To evaluate the results, MAP and P@k metrics are used. Figures 5.1 and 5.3 show MAP comparison of the algorithms. Numerical values of MAP metric can be seen in Tables 5.2 and 5.3. Figures 5.2 and 5.4 show P@k comparison. Numerical P@k values are given in Tables 5.3 and 5.5. We see that both algorithms perform better on OHSUMED dataset.

From the results we see that, RankingSVM performs better than GRwC. One reason is that the classification error rate causes to form cycles on document graph of GRwC. Since we choose the vertex that has minimum incoming edge, some lower ranked documents have higher positions in the ordering. This causes lower values in the evaluation metrics. Also, from the prediction results, we see that relevant documents are predicted better than the documents that are irrelevant. Since the irrelevant documents have very small feature values and since their feature values are almost identical, KNN can not classify them correctly. To avoid this situation, we try to decrease the number of irrelevant documents in the queries. Although this approach is more successful, we don't count on them in favor of fairness.

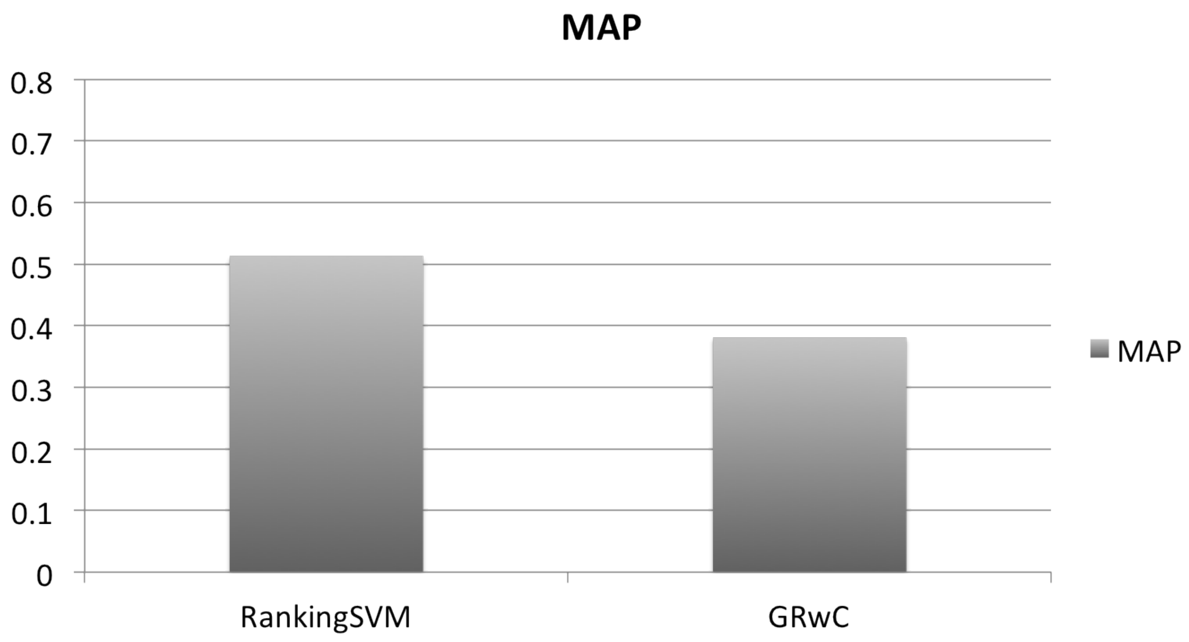

Figure 5.1. MAP comparison on OHSUMED

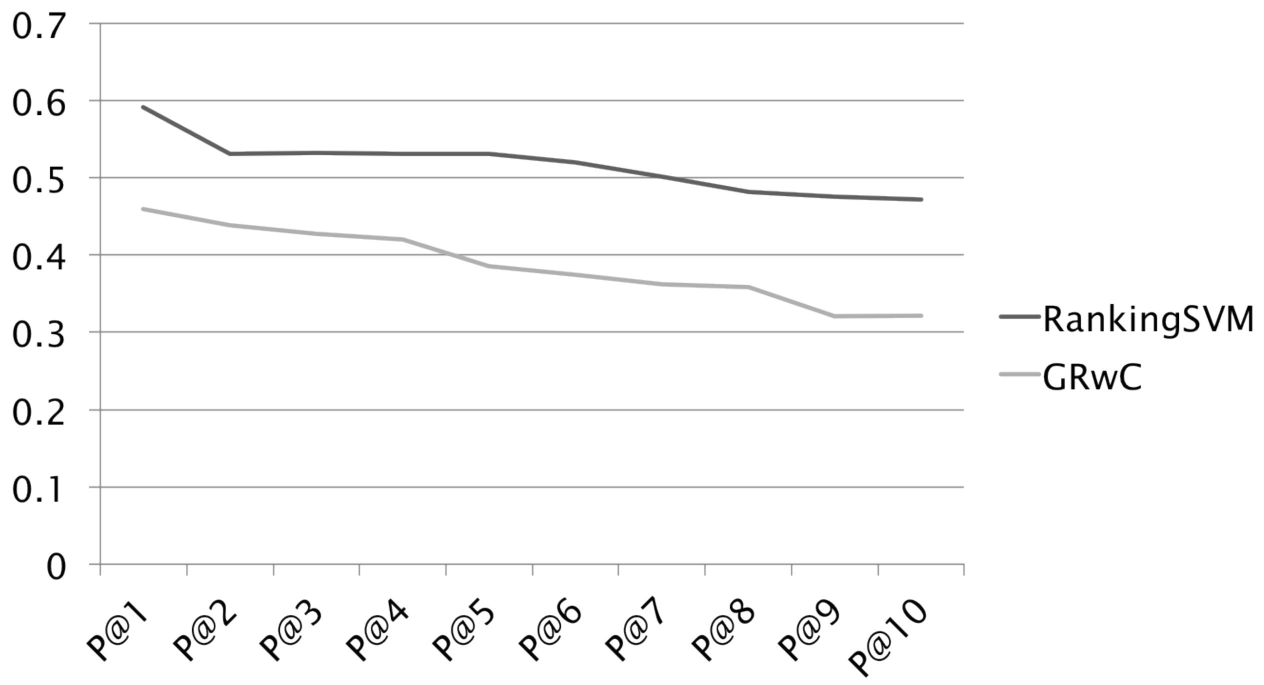

Figure 5.2. P@k comparison on OHSUMED

Table 5.1. List of the document features in Letor.

Col.	Description	Col.	Description
1	TF(Term frequency) of body	24	BM25 of URL
2	TF of anchor	25	BM25 of whole document
3	TF of title	26	LMIR.ABS of body
4	TF of URL	27	LMIR.ABS of anchor
5	TF of whole document	28	LMIR.ABS of title
6	IDF(Inverse document frequency) of body	29	LMIR.ABS of URL
7	IDF of anchor	30	LMIR.ABS of whole document
8	IDF of title	31	LMIR.DIR of body
9	IDF of URL	32	LMIR.DIR of anchor
10	IDF of whole document	33	LMIR.DIR of title
11	TF*IDF of body	34	LMIR.DIR of URL
12	TF*IDF of anchor	35	LMIR.DIR of whole document
13	TF*IDF of title	36	LMIR.JM of body
14	TF*IDF of URL	37	LMIR.JM of anchor
15	TF*IDF of whole document	38	LMIR.JM of title
16	DL(Document length) of body	39	LMIR.JM of URL
17	DL of anchor	40	LMIR.JM of whole document
18	DL of title	41	PageRank
19	DL of URL	42	Inlink number
20	DL of whole document	43	Outlink number
21	BM25 of body	44	Number of slash in URL
22	BM25 of anchor	45	Length of URL
23	BM25 of title	46	Number of child page

Table 5.2. MAP comparison of RankingSVM and GRwC on OHSUMED.

	MAP
RankingSVM	0.4134
GRwC	0.3810

Table 5.3. P@k comparison of RankingSVM and GRwC on OHSUMED.

	RankingSVM	GRwC
P@1	0.5914	0.4602
P@2	0.5314	0.4391
P@3	0.5327	0.4281
P@4	0.5313	0.4201
P@5	0.5309	0.3854
P@6	0.5203	0.3743
P@7	0.5017	0.3621
P@8	0.4813	0.3581
P@9	0.4760	0.3209
P@10	0.4714	0.3210

Table 5.4. MAP comparison of RankingSVM and GRwC on TREC.

	MAP
RankingSVM	0.3613
GRwC	0.2094

Table 5.5. P@k comparison of RankingSVM and GRwC on TREC.

	RankingSVM	GRwC
P@1	0.4325	0.3752
P@2	0.3381	0.2861
P@3	0.3264	0.2374
P@4	0.3104	0.2301
P@5	0.3012	0.2153
P@6	0.2792	0.1926
P@7	0.2647	0.1635
P@8	0.2562	0.1357
P@9	0.2501	0.1227
P@10	0.2035	0.1183

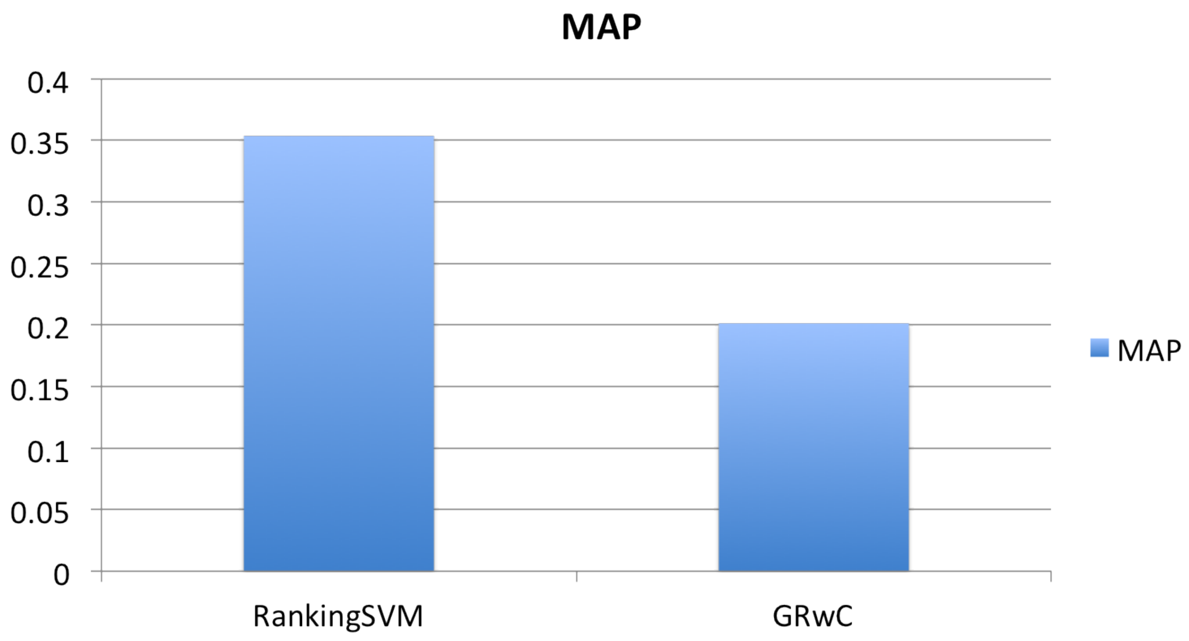

Figure 5.3. MAP comparison on TREC

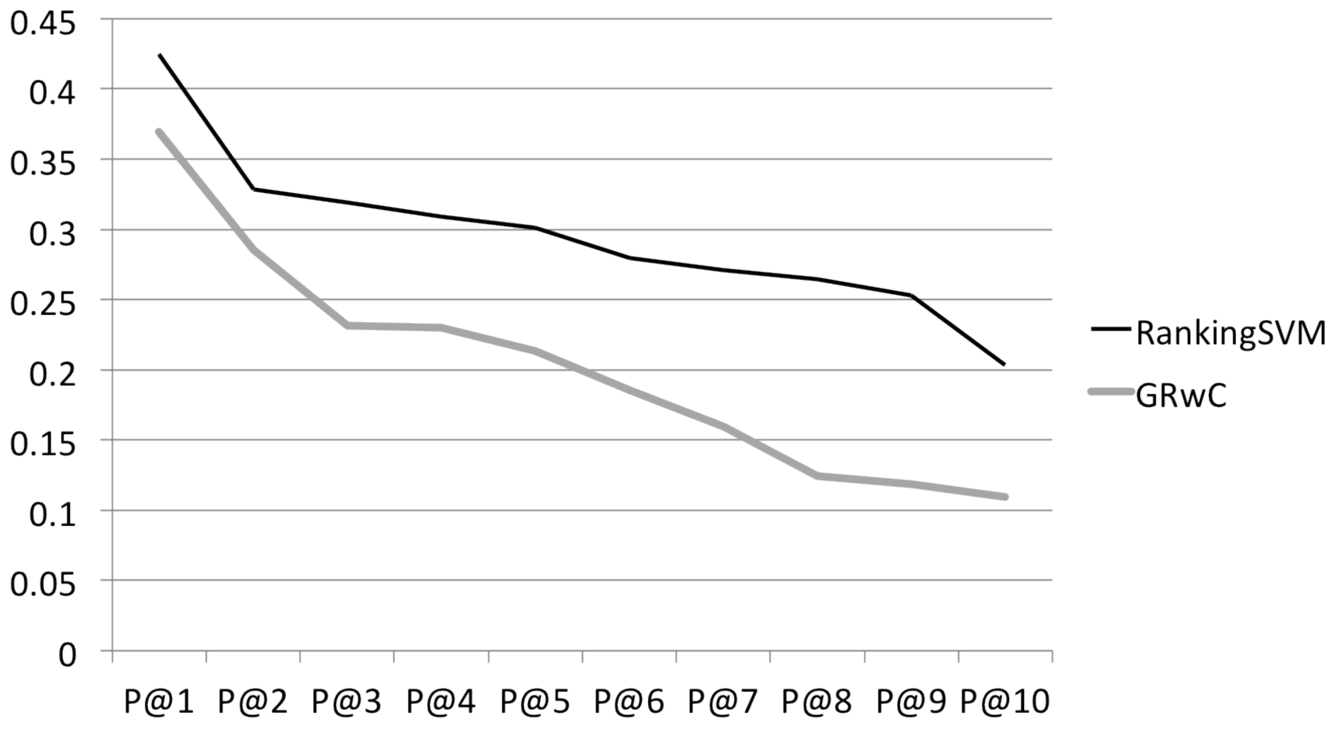

Figure 5.4. P@k comparison on TREC

Chapter 6
Conclusion

In this thesis, we review three approaches to learning to rank. The first approach is the point-wise approach which reduces ranking problem to regression or classification on single documents. The second is the pair-wise approach which reduces ranking to pair-wise classification problem. The third is the list-wise approach which takes ranking as a completely new problem and tries to minimize ranking losses. We review the most important algorithms of these three approaches.

We also propose a new LTR method GRwC, where we use a different kind of dataset format than those popular LTR algorithms use, by changing the structure of the dataset. We create positive and negative instances in terms of preference, by putting the document features side by side. Then we use KNN classifier to get predictions on pair-wise preference over all document pairs. By using these predictions, we build the graph of all documents for each query and at the end, we rank the documents by applying a modified toplogical sort algorithm on this graph.

Experiments on two LTR datasets OHSUMED and TREC show that our algorithm does not give as good results as well-known LTR algorithm RankingSVM. But with using distinct classification algorithm and preprocessing the features using feature selection and/or feature extraction algorithms one can get better results.

References

1. Harrington, E. F., “Online Ranking/Collaborative Filtering Using the Perceptron Algorithm”, *Proceedings of the 20th International Conference on Machine Learning*, pp. 250–257, 2003.

2. Bikel, D. M., R. Schwartz, and R. M. Weischedel, “An Algorithm that Learns What is in a Name”, *Machine Learning*, Vol. 34, pp. 211–231, February 1999.

3. Xu, J., Y. Cao, H. Li, and M. Zhao, “Ranking definitions with supervised learning methods”, *Special Interest Tracks and Posters of The 14th International Conference on World Wide Web*, pp. 811–819, 2005.

4. Chirita, P.-A., J. Diederich, and W. Nejdl, “MailRank: Using Ranking for Spam Detection”, *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, pp. 373–380, 2005.

5. Pang, B. and L. Lee, “Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales”, pp. 115–124, 2005.

6. Dave, K., S. Lawrence, and D. M. Pennock, “Mining the peanut gallery: opinion extraction and semantic classification of product reviews”, *Proceedings of the 12th International Conference on World Wide Web*, pp. 519–528, 2003.

7. Gyöngyi, Z., H. Garcia-Molina, and J. Pedersen, “Combating web spam with trustrank”, VLDB ’04, pp. 576–587, 2004.

8. Qin, T., T.-Y. Liu, X.-D. Zhang, D.-S. Wang, W.-Y. Xiong, and H. Li, "Learning to rank relational objects and its application to web search", *Proceeding of the 17th International Conference on World Wide Web*, pp. 407–416, ACM, 2008.

9. Bartell, B., E. Britannica, R. Belew, G. Cottrell, and R. Belew, "Learning to Retrieve Information", *Proceedings of the Swedish Conference on Connectionism*, 1995.

10. Burges, C. J. C., R. Ragno, and Q. V. Le, "Learning to Rank with Nonsmooth Cost Functions", pp. 193–200, MIT Press, 2006.

11. Burges, C., T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, "Learning to rank using gradient descent", *NIPS*, pp. 89–96, 2005.

12. Cao, Y., J. Xu, T.-Y. Liu, H. Li, Y. Huang, and H.-W. Hon, "Adapting ranking SVM to document retrieval", *Proceedings of the 22nd International Conference on Machine Learning*, pp. 186–193, 2006.

13. Cao, Z., T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach", *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 129–136, ACM, New York, NY, USA, 2007.

14. Chu, W. and S. S. Keerthi, "Support Vector Ordinal Regression", *Neural Computation*, Vol. 19, No. 3, pp. 792–815, 2007.

15. Cohen, W. W., R. E. Schapire, and Y. Singer, "Learning to Order Things", *Journal of Artificial Intelligence Research*, Vol. 10, pp. 243–270, 1998.

16. Cossock, D. and T. Zhang, "Subset Ranking Using Regression", *Learning Theory*,

Vol. 4005, pp. 605–619, 2006.

17. Crammer, K. and Y. Singer, "Pranking with Ranking", *Advances in Neural Information Processing Systems 14*, pp. 641–647, 2001.

18. Freund, Y., R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences", *Journal of Machine Learning Research*, Vol. 4, pp. 933–969, December 2003.

19. Fuhr, N., "Optimum polynomial retrieval functions based on the probability ranking principle", *ACM Transactions on Information Systems*, Vol. 7, pp. 183–204, July 1989.

20. Harrington, E. F., "Online Ranking/Collaborative Filtering Using the Perceptron Algorithm", *Proceedings of the 20th International Conference on Machine Learning*, pp. 250–257, 2003.

21. Herbrich, R., T. Graepel, and K. Obermayer, "Large Margin Rank Boundaries for Ordinal Regression", *Advances in Large Margin Classifiers*, pp. 115–132, Cambridge, MA, 2000.

22. Kramer, S., G. Widmer, B. Pfahringer, and M. De Groeve, "Prediction of Ordinal Classes Using Regression Trees", *Fundam. Inf.*, Vol. 47, pp. 1–13, September 2001.

23. Li, P., C. J. C. Burges, and Q. Wu, "McRank: Learning to Rank Using Multiple Classification and Gradient Boosting.", *NIPS*, MIT Press.

24. Nallapati, R., "Discriminative models for information retrieval", *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 64–71, 2004.

25. Qin, T., 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.*, Vol. 44, pp. 838–855, March 2008.

26. Shashua, A. and A. Levin, "Ranking with Large Margin Principle: Two Approaches", *NIPS*, 2003.

27. Tsai, M.-F., T.-Y. Liu, T. Qin, H. hsi Chen, and W.-Y. Ma, "Frank: A ranking method with fidelity loss", *Proceedings of the 30th Annual International ACM SIGIR Conference*, 2007.

28. Liu, T.-Y., J. Wang, W. Zhang, and H. Li, "Listwise approach to learning to rank - theory and algorithm", *Proceedings of 25th International Conference on Machine Learning*, pp. 1192–1199, 2008.

29. Maron, M. E. and J. L. Kuhns, "On Relevance, Probabilistic Indexing and Information Retrieval", *J. ACM*, Vol. 7, pp. 216–244, July 1960.

30. Ponte, J. M. and W. B. Croft, "A Language Modeling Approach to Information Retrieval", pp. 275–281, 1998.

31. Page, L., S. Brin, R. Motwani, and T. Winograd, "The PageRank Citation Ranking: Bringing Order to the Web.", Technical Report 1999-66, Stanford InfoLab, 1999.

32. Kleinberg, J. M., "Authoritative sources in a hyperlinked environment", *J. ACM*, Vol. 46, pp. 604–632, September 1999.

33. Järvelin, K. and J. Kekäläinen, "IR evaluation methods for retrieving highly relevant documents", *Proceedings of the 23rd Annual International ACM SIGIR Con-*

ference on Research and Development in Information Retrieval, pp. 41–48, New York, NY, USA, 2000.

34. Chu, W. and Z. Ghahramani, "Gaussian processes for ordinal regression", *Journal of Machine Learning Research*, Vol. 6, p. 2005, 2004.

35. Cooper, W. S., F. C. Gey, and D. P. Dabney, "Probabilistic retrieval based on staged logistic regression", *Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 198–210, 1992.

36. Gey, F. C., "Inferring probability of relevance using the method of logistic regression", *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 222–231, 1994.

37. Freund, Y. and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting", *Proceedings of the Second European Conference on Computational Learning Theory*, pp. 23–37, 1995.

38. Taylor, M., J. Guiver, S. Robertson, and T. Minka, "SoftRank: optimizing non-smooth rank metrics", *Proceedings of the International Conference on Web Search and Web Data Mining*, WSDM '08, pp. 77–86, 2008.

39. Xia, F., T.-Y. Liu, J. Wang, W. Zhang, and H. Li, "Listwise approach to learning to rank: theory and algorithm", *Proceedings of the 25th International Conference on Machine Learning*, pp. 1192–1199, 2008.

40. Yue, Y., T. Finley, F. Radlinski, and T. Joachims, "A support vector method for optimizing average precision", *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR

'07, pp. 271–278, 2007.

41. Qin, T., 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.*, Vol. 44, pp. 838–855, March 2008.

42. Qin, T., T.-Y. Liu, J. Xu, and H. Li, "LETOR: A benchmark collection for research on learning to rank for information retrieval", *Information Retrieval*, Vol. 13, pp. 346–374, 2010.

43. Buckley, C., "OHSUMED: An interactive retrieval evaluation and new large test collection for research", pp. 192–201, 1994.

44. Craswell, N., D. Hawking, R. Wilkinson, and M. Wu, "Overview of the TREC 2003 Web Track", *TREC*, pp. 78–92, 2003.

45. Project, W. M. L., "Weka", http://www.cs.waikato.ac.nz/~ml/weka.

46. Joachims, T., "Making Large-Scale SVM Learning Practical", *Advances in Kernel Methods - Support Vector Learning*, 1998.

Curriculum Vitae

Yasin Ozan KILIC was born on 18 July 1985, in Ordu. He received his B.S. degree in Computer Science & Engineering in 2008 from Işık University. He worked as a research assistant at the department of Computer Engineering of Işık University from 2008 to 2011.

Publications

[1] Kilic, O., Aydin M., "Automatic Xml Schema Matching", *European and Mediterranean Conference on Information Systems*, July 13-14 2009

[2] Efrat, A., Forrester, D., Iyer, A., Kobourov, S. G., Erten, S., Kilic, O. "Force-directed approaches to sensor localization", *ACM Trans. Sensor Netw.* 7, 3, Article 28 (September 2010)