Learning from Click Model and Latent Factor Model for Relevance Prediction Challenge

Botao Hu
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
botao@cse.ust.hk

Nathan N. Liu
Hong Kong University of Science and Technology
Clear Water Bay, Hong Kong
nliu@cse.ust.hk

Weizhu Chen
Microsoft Research Asia
Hong Kong University of Science and Technology
wzchen@microsoft.com

ABSTRACT

How to accurately interpret user click behaviour in search log is a key but challenging problem for search relevance. In this paper, we describe our solution to the relevance prediction challenge which achieves the first place among eligible teams. There are three stages in our solution: feature generation, feature augmentation and learning a ranking function. In the first stage, we extract features in relation to query-document pairs as well as individual queries and documents from the click log data. In the second stage, we induce additional features by click model techniques and learning latent factor models to correct different biases and discover the correlations between different queries or documents respectively. In the final stage, we apply supervised learning models on the limited labelled data to induce a model for predicting relevance based on the features generated in the previous two stages.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Retrieval Models

General Terms

Algorithms, Experimentation

1. INTRODUCTION

Search engine query logs record enormous amount of data about users' interaction with the search engine result pages (SERPs). It is well recognized as one of most important type of data and has been proven to be invaluable in a variety of Web search applications such as query understanding and query suggestions, etc. Among these applications, the most important and direct task will be to understand user preference for search results based on their click behaviour and then learn a user-perceived relevance of documents with respect to queries.This enables the search engine to automatically learn to improve its ranking from the click logs. Due to its importance, considerable amount of research has attempted to derive user-perceived document relevance from click logs.

In this paper, we describe our solution to the Relevance Prediction Challenge, which requires building a relevance prediction model given both large-scale click logs and a small portion of human assessed relevance labels, which can be regarded as a semi-supervised learning problem or a missing value problem, similar as the problem setting in work [5]. The highlight of our solution lies in the leverage of click model techniques and latent factor models as a new approach of feature argumentation. The whole solution consists of three stages: statistical feature construction, feature augmentation and learning a ranking function. In the first stage, statistical features, such as click-through rate (CTR), with respect to different queries are derived separately, which implies that the relevancies of a particular document with respect to different queries are independent on each other. There are two typical limitations of the statistical features. First, user click behaviour are often biased, such as the position bias. This is not characterized by them. Second, the independent assumption makes it hard to derive the relevance of the low-frequent queries which only appear in the logs with very few times, i.e, the sparseness problem.

To alleviate these two limitations, we conduct a second stage called feature argumentation. It focuses on deriving new features from click model and latent factor models. Click model has been positioned as an effective technique to correct biases in user click behaviour. Typical works includes [7, 9, 16, 15] for Web search and [17] for Online advertising. In our solution, we model the position bias [6] and intent bias [10]. Position bias means that a document appearing in a higher position is more likely to attract user clicks even though it is not as relevant as other documents in lower positions. Intent bias illustrates that even under a same query, users with different search intents may expect and prefer different search results. The unbiased relevance learnt from click models are regarded as features for next stage. The initial motivation of latent factor model is to address the sparseness problem by bridging the high-frequent queries and the low-frequent queries. This bridge is built based on an assumption that different users often issue different but related queries to find the same or similar documents. We achieve this goal via building some sparse matrices with rows and columns corresponding to queries and documents and entries being some statistics, such as CTR. We then perform a low rank approximation of each sparse matrix to generate the latent factors for each query and document.

WSCD '12 Washington, Seattle USA

The approximated latent factors is treated as an additional set of features to supplement the click model features and statistical features to represent the whole feature set.

In the last stage, we utilize the provided labelled data to train a ranking function based on the generated features. We tried linear models such as ranking svm as well as non-linear models including neural network (NN) and multiple additive regression tree (MART) and found MART led to superior performance. Several advanced techniques were attempted to further optimize the MART model. Firstly, we tried to directly optimize different ranking objective such as regression loss and NDCG using the several learning to rank algorithms, such as RankNet and LambdaRank. We found that optimizing NDCG actually led to the best AUC result in submission. Secondly, to accommodate the distributional difference between training and test sets, we tried several data resampling and weighting scheme to adjust the distribution of training data towards that of test data and found that discarding one-class queries while downweighting queries with too many distinct URLs would help improve the performance.

2. STATISTICAL FEATURES

Click log is an invaluable source of *implicit feedback*, that could convey users' perceived relevance of the results shown to them by the search engine. Intuitively, a user would only click on a URL if he finds the result plausibly relevant via examining the presented information such as title, URL and search snippet. However, clicks are inherently noisy and also subject to various biases such as the position it was show on the SERP page. Thus, correctly interpreting click data would require the consideration of various other factors such as the position of the URL when being clicked, the amount of time spent on the result page after click (i.e. dwelling time), etc. In this section, we describe a variety of statistical quantities that can be directly computed from click logs to capture various properties with respect to individual queries or URLs or the relation between them.

- Table 1: *Query-URL Features*
  They can be generated by simply summing the query-region-URL-based features over all region r. The performance and usefulness of the query-URL-based features is not too much different from the query-region-URL-based features.

- Table 2: *Query Features*
  Because for every URL u, query-based features are identical, the AUC evaluation of them is meaningless. But query-based features are very important.Experimentally, disabling this features will lead the performance of the final result to drop about 0.5% in terms of AUC.

- Table 3: *URL Features*
  They can be generated by simply summing Query-region-URL-based features over all query q. However, their usefulness is negligible as compared to other features.

3. CLICK-MODEL FEATURES

The statistical features described in previous section are easy to compute, but due to their arbitrary nature, such

Feature	Description	AUC
#Impressions	# times u is impressed on SERPs of q	0.565
#Clicks	# times u is clicked on SERPs of q	0.620
AvgPos	average rank position of u when impressed on SERPs of q	0.617
CTR	#Clicks / #Impressions	0.628
FirstCTR	CTR on u when it is the first clicked URL on SERPs of q	0.622
LastCTR	CTR on u when it is the last clicked URL on SERPs of q	0.631
OnlyCTR	CTR on u when it is the only clicked URL on SERPs of q	0.621
AvgDwellTime	average dwelling time on u after being clicked on SERPs of q	0.606

Table 1: Statistical features for Query-URL Pairs.

Feature	Description
#Shows	how many times query q is searched in the data
#Clicks	how many clicks on all SERPs of query q
AvgClickPos	average position of all clicks in SERPs
#NoClickNum	# of SERP with no clicks
AvgClickNum	average number of click on each SERP
AvgIndexInSession	average index of SERP on query q in the sequence of SERPs of a session
AvgTime2FirstClick	average time from SERP's opening to the first click
AvgTime2LastClick	average time from SERP's opening to the last click
%ClickURL$_i$	percentage of total clicks on the i-th most clicked URL ($i \in \{1, 2, 3\}$)
CTR$_k$	CTR for rank position k of the query ($1 \leq k \leq 10$)
ClickEntropy	$-\sum_i \text{CTR}_i \log(\text{CTR}_i)$

Table 2: Statistical Features for Invidual Queries

Feature	Description	AUC
#Shows	# of impressions of u irrespective of queries	0.552
#Clicks	# of clicks on u irrespective of queries	0.605
CTR	click through rate on u irrepsective of queries	0.620
LastCTR	click through rate on u when it is the last clicked URL irrespective of queries	0.618

Table 3: Statistical features for individual URLs

features may fail to capture some more intricate aspects of users' click behavior that may be query or URL dependent. This motivated many recent works on developing more formal and structural statistical models of user click behavior. These models often treat the intrinsic relevance of documents to queries as free parameters that can be adaptively learnt from data, therefore providing a more principle way for deriving features from click logs.

The key problem in click modeling is to separate document's intrinsic relevance from various kinds of biases present in click data. It is tempting to think that documents that get clicked often for a query should have higher relevance. However, a well-known problem of clicks is the so-called *position bias*: a document appearing in a higher position is more likely to attract user clicks even though it is not as relevant as other documents in lower positions[6]. The position bias leads the relevance of a document in a lower position to be probably underestimated by its CTR value.Recently, Hu et at. [10] proposed an another kind of bias – intent bias, which characterizes the diversity of user preference to one URL. Users with different search intents may submit the same query to the search engine but expect different search results. Such phenomenon is omnipresent in real click data and results in diversified CTR values.

In our solution to the Relevance Prediction Challenge, we adopt three state-of-the-art click models to infer document relevances from click logs: the baseline model (BLM) [6], the dynamic Bayesian network (DBN) model [4] and the user browsing model (UBM) [8] and implement them with different inference methods: expectation maximization (EM) and probit Bayesian inference (PBI) [16]. Further, we extract intent bias to derive the corresponding intent-unbiased versions of the above models.

Before going into the model details, we introduce definitions and notations which will be used throughout the paper. Once a user enters the search engine under region r, a *session* starts. The session records all activities of the user's behaviors until s/he leaves the search engine. The user submits a *query* q and the search engine returns a *search result page* (SERP) containing M (usually $M = 10$) URLs, denoted by $\{u_{\pi_i}\}_{i=1}^{M}$, where π_i is the index of the URL at the i-th position. The user *examines* the summary of each search result and *clicks* some or none of them.

3.1 Baseline Model

Craswell et al. [6] formalized the idea of position bias as *Examination Hypothesis*: a document is clicked if and only if it is both examined and relevant, which can be formulated as $\Pr(C_i = 1|E_i = 1) = r_{\pi_i}$ and $\Pr(C_i = 1|E_i = 0) = 0$, where the binary random variables, C_i, E_i denote user click and user examination events at the i-th position and the parameter r_{π_i} represents the *document relevance*.

According to this hypothesis, the document click-through rate can be represented by

$$\begin{aligned}\Pr(C_i = 1) &= \sum_{e\in\{0,1\}} \Pr(E_i = e)\Pr(C_i = 1|E_i = e) \\ &= \underbrace{\Pr(E_i = 1)}_{\text{position bias}}\underbrace{\Pr(C_i = 1|E_i = 1)}_{\text{document relevance}}\end{aligned}$$

where the position bias and the document relevance are decomposed. Naturally, the *baseline model* (BLM) simply assumes that the position bias in every position is a free parameter and suggest to employ the *expectation maximization* algorithm to estimate these parameters. After learning, the resulted r_{π_i} in used as a feature for the corresponding query-document pair.

3.2 Dynamic Bayesian Network Model

The *dynamic Bayesian network model* (DBN) [4] is designed based on the fact that a click does not necessarily indicate that the user is satisfied with this document. Thus, the DBN model distinguishes the document relevance as the perceived relevance (a.k.a. the document attractiveness) and the real relevance (a.k.a. the user satisfaction), where whether the user clicks a document depends on its perceived relevance while whether the user is satisfied with this document and examines the next document depends on the real relevance.

Formally speaking,

$$\begin{aligned}\Pr(C_i = 1|E_i = 1) &= a_{\pi_i} \\ \Pr(C_i = 1|E_i = 0) &= 0 \\ \Pr(S_i = 1|C_j = 1) &= s_{\pi_i} \\ \Pr(S_i = 1|C_j = 0) &= 0 \\ \Pr(E_1 = 1) &= 1 \\ \Pr(E_{i+1} = 1|E_i = 0) &= 0 \\ \Pr(E_{i+1} = 1|S_i = 1) &= 0 \\ \Pr(E_{i+1} = 1|E_i = 1, S_i = 0) &= \gamma\end{aligned}$$

where the parameter γ is the jumping probability of that the user examines the next document without satisfaction. S_i is a binary variable indicating whether the user is satisfied with the document π_i at position i, and the parameters a_{π_i} and s_{π_i} measure the document attractiveness and the user satisfaction respectively. In our solution, we regard the document attractiveness a_{π_i} and the user satisfaction s_{π_i} and their product $a_{\pi_i}s_{\pi_i}$ as three features.

3.3 User Browsing Model

The *user browsing model* (UBM) [8] is based on the examination hypothesis but does not follow the cascade hypothesis, that the user scans the search results in order from top to down. Instead, it assumes that the examination probability E_i depends on the previous clicked position $l_i = \max\{j \in \{1, \cdots, i-1\} \,|\, C_j = 1\}$ as well as the distance between the i-th position and the l_i-th position:

$$\Pr(E_i = 1|C_{1:i-1}) = \beta_{l_i,i-l_i} \tag{1}$$

If there are no clicks before the position i, l_i is set to 0. The likelihood of a search session under UBM can be stated in a quite simple form:

$$\Pr(C_{1:M}) = \prod_{i=1}^{M} \left(r_{\pi_i}\beta_{l_i,i-l_i}\right)^{C_i}\left(1 - r_{\pi_i}\beta_{l_i,i-l_i}\right)^{1-C_i} \tag{2}$$

where there are $10 \times (10 + 1)/2$ $\{\beta_{i,j}\}$ parameters shared across all search sessions. In our solution, we regard r_{π_i} as feature.

3.4 Intent-unbiased Models

Users with different search intents may submit the same query to the search engine but expect different search results. Thus, there might be a bias between user search intent and the query formulated by the user, which can lead to

the diversity in user clicks. Hu et al. [10] named this gap as the *intent bias* which measures how well the query matches the intent, i.e., the degree of match between the intent and the query and further proposed the *intent hypothesis* as a complement to the original examination hypothesis. It assumes that a document is clicked only after it meets the user's search intent, i.e. it is needed by the user. Whether a relevant document is needed is uniquely influenced by the gap between the user's intent and the query. Formally, the intent hypothesis can be formulated as

$$\Pr(C_i = 1|E_i = 1) = \mu_s r_{\pi_i} \quad (3)$$

$$\Pr(C_i = 1|E_i = 0) = 0 \quad (4)$$

where μ_s is the parameter to represent the intent bias, which is local for each session s. Every session maintains its own intent bias, and the intent biases for different sessions are mutually independent.

The new hypothesis is very general and can be fit into most existing click models to improve their capacities for learning unbiased relevance. All click models based on the examination hypothesis can be easily modified to its *intent-unbiased* version by introducing the intent hypothesis to replace the examination hypothesis. In our solution, we choose DBN and UBM to apply the intent hypothesis. The new models based on DBN and UBM are called Intent-unbiased DBN and Intent-unbiased UBM respectively.

3.5 Probit Bayesian Inference

To improve the traditional expectation maximization(EM), which is computationally expensive and highly sensitive for very sparse data, Zhang et al. [16] proposed the Probit Bayesian Inference (PBI) to implement click models. PBI connects each parameter θ in the click model with an auxiliary variable x through the probit link $\theta = \Phi(x)$ where Φ is the cumulative distribution function of the normal distribution, and restricts $p(x)$ always to the Gaussian family. Thus, in order to update $p(\theta)$, it is sufficient to derive $p(x)$ from $p(\theta)$ and approximate it by a Gaussian density. Then we use the approximation to update $p(x)$ and further update $p(\theta)$. This method facilitates the online Bayesian updating process within a linear computational cost and achieves the robustness for the sparse data due to the prior of parameters. In our solution, we implement both PBI and EM for above three click models.

3.6 Results

In our solution to the Challenge, we adopt three click models described above: BLM, DBN and UBM and implement them in both EM and PBI as the inference approaches, and further implement the intent-unbiased versions of DBN and UBM. We regards all the relevance scores extracted by these models as features of the next stage of training. Table 4 lists the AUC scores obtained by various click models using different implementations. From the data in the table, it is clear to see that all models of intent-unbiased versions outperform their original ones. The intent hypothesis helps the models to extract accurate relevance and generate more useful features for the next stage of training. It is worth noting that the individual performance of click models is not as good as some statistical feature like CTR. However, our experimental results show that with the removal of the click model features, the final performance will drop about 1%. This convinces us to believe that click model features are actually very complementary features to the general statistical features in terms of relevance ranking.

Relevance	EM	PBI	Intent-unbiased
BLM r_{π_i}	0.595	– [1]	–
UBM r_{π_i}	0.597	0.569	0.612
DBN $a_{\pi_i} s_{\pi_i}$	0.524	0.546	0.594
DBN a_{π_i}	0.561	0.567	0.598
DBN s_{π_i}	0.501	0.525	0.565

Table 4: Comparison of results of various click models with different learning algorithms

4. FEATURE GENERATION VIA LATENT FACTOR MODELS

In the previous section, we describe various ways for deriving a score characterizing a document's relevance with respect to a query from click logs. Since it is not clear which method is the most effective, we propose to learn a function to combine the different measures rather than just relying on any single measure. Suppose there are a total of p methods for scoring a pair of document and query, then given a pair of document d and query q, we can encode such a set of scores produced by different models by a feature vector $\mathbf{x}_{qd} = \{x^1_{qd}, x^2_{qd}, \ldots, x^p_{qd}\}$, where $x^i_{qd} \in \mathbb{R}$ is the score produced by method i. Given these features, we can then directly apply any supervised learning methods on the set of labeled (q, d) pairs. However, the set of labeled data is rather limited, simply learning the model this way would render all those unlabeled (q, d) pairs useless. The typical way of utilizing unlabeled data is via the use of *semi-supervised learning*, which lets the decision boundary of the model be jointly determined based on both labeled and unlabled data. However, directly applying these semi-supervised learning methods can be problematic here due to two reasons. Firstly, most existing methods in this area are proposed for modeling data sets consisting of independently and identically distributed (I.I.D.) instances, whereas for the instances in our problem which corresponds to query document pairs are clearly non-i.i.d. Different instances may be related to the same query or document, which will very likely introduce correlations into the data set. Secondly, the amount of unlabeled data in our problem is enormous, making it simply infeasible to apply most existing semi-supervised learning models.

In our solution, rather than resorting to more complicated semi-supervised learning methods, we apply several unsupervised learning methods on the combined set of labeled and unlabeled instances in order to refine the set of raw features as well as inducing some additional features. We call this the *feature refinement and augmentation* step. Rather than directly applying off-the-shelf unsupervised learning algorithms like k-means, or PCA on the data set, an interesting aspect of our solution is in devising a way to exploit the non-i.i.d. nature of the data instances during this process. In particular, we borrow ideas from the field of *collaborative filtering* to model the click based features so that the features of related queries or related documents would be naturally coupled based on certain low rank structure imposed by matrix or tensor factorization models.

When looking back to the preceding section on the statis-

# Disctinct Queries	# Documents
<5	167,412
5-20	94,370
20-100	80,257
>100	53,460

Table 5: Click Number of different queries on a document

tical feature generation and click models used in deriving the features for (q, d), we can see two limitations. It was these limitations that inspire us to apply collaborative filtering ideas to post-process these raw features. Firstly, the parameters associated each query including both position biases and document relevance is learnt very independent on other queries. Secondly, the relevance of a document with respect to a query is also learnt independently of all other documents. While such independence assumptions could lead to simpler models that could be efficiently learnt from large amount of data, they are clearly not appropriate in the case of documents and queries arising the Web search applications. It is commonly known that different users may use different queries to look for the same documents, which is the main motivation for the extensive works on query suggestions. It would be natural to assume the parameters associating these related queries to be highly correlated. Similarly, there often exist many different Web documents with related content, which would make their relevance with respect to the same query highly correlated.

However, the correlations between pairs of queries or documents are not easy to be discovered using the Relevance Prediction Challenge data set since all documents and queries are completely annonymized as numeric IDs, making it impossible to compare queries based on their spellings or compare documents based on their content. All that are observable are just how different documents got clicked for different queries when being shown at different positions. This situation closely resembles that of *collaborative filtering* based recommendation, where only users' ratings on items are observed and the similarities between pairs of users or items need to be determined based on their associated ratings. However, in order for collaborative filtering to work, different users need to have rated some common items. To examine such possibility, we tried to count the number of distinct queries each document has been clicked for and summarized the statistics in Table 5. It can be seen that there is a significant number of documents which have been clicked for many queries.

To transform click modeling into a collaborative filtering problem, we simply treat the queries and documents as users and items respectively, and the scores $\{x^1_{qd}, x^2_{qd}, \ldots, x^P_{qd}\}$ as a set of ratings for the pair (q, d). A key difference from traditional collaborative filtering is that there are multiple ratings associated with a pair of query and document in this setting. In the following subsections, we would describe several strategies to adapt the widely used matrix factorization model for this setting with multiple ratings. Before doing that, we will present a brief overview of the matrix factorization model, which is a fundamental building block of machine learning, underlying many popular regression, dimensionality reduction and clustering algorithms. It has been shown to achieve state of the art performances in many recommendation problems such as movie recommendation[12] and tag recommendation[13].

A matrix factorization model can be written in the general form $X \approx f(UV^T)$. The two matrices $U \in \mathbb{R}^{M\times K}$ and $V \in \mathbb{R}^{N\times K}$ are known as the factor matrices with each row vector $\mathbf{u}_p$ and $\mathbf{v}_d$ capturing the characteristics about a particular row and column of the matrix X respectively. The function $f : \mathbb{R}^{M\times N} \to \mathbb{R}^{M\times N}$ is commonly known as the link function, the choice of which depends on the underlying distributional assumption about $P(x_{dq}|\widehat{x}_{dq})$. Some common forms of the link function are identity link $f(x) = x$, logistic link $f(x) = (1 + e^{-x})^{-1}$ and exponential link $f(x) = e^x$. The dimensionality of the factor matrices K controls the complexity of the factorization model. Larger K leads to more flexible models but are also subject to higher risk of overfitting.

The learning of a matrix factorization is ofen via solving the following optimization problem:

$$\arg\min_{U,V} \mathcal{D}(X \| f(UV^T)) + \mathcal{R}(U, V) \tag{5}$$

where $\mathcal{D}$ is a loss function that measures the consistency between the data X and the predictions $f(UV^T)$ and $\mathcal{R}$ is a regularization penalty used to address overfitting. As show in [14], via choosing the appropriate combination of loss and link function based on the nature of the data in X, we can obtain several well known variations of the matrix factorization model as shown in Table **??**.

The most common regularizers used in matrix factorization are ℓ_p norm regularizers: $\mathbb{R}(U, V) = \lambda \cdot (\sum_{dk} |u_{dk}|^p + \sum_{qk} |v_{qk}|^p)$, where λ controls the strength of the regularizer, are decomposable. In this work, we choose to use the ℓ_2-norm regularization:

$$\mathbb{R}(U, V) = \lambda \cdot (\sum_{dk} u^2_{dk} + \sum_{qk} v^2_{qk}) \tag{6}$$

which is smoothly differentiable and can be easily optimized with gradient descent algorithms.

4.1 Independent Matrix Factorization

Let M and N denote the number of distinct queries and documents in the data set respectively. Then for each score type i, we can encode all the feature values in a matrix $X^i \in \mathbb{R}^{M\times N}$ with entries equal to x^i_{qd}'s. Since there are p different scoring methods, we could define a collection of p matrices $\{X^1, X^2, \ldots, X^p\}$ following the above methods. A straightforward way to handle the many different rating matrices is simply to apply the matrix factorization model on each X^i separately to induce two factors U^i and V^i for each score type i.

After learning the factorization models, we can use the approximated scores $\{\widehat{x}^1_{dq}, \ldots, \widehat{x}^p_{dq}\}$ and the set of latent factors $\{\mathbf{u}^1_d, \ldots, \mathbf{u}^p_d, \mathbf{v}^1_q, \ldots, \mathbf{v}^p_q\}$ as additional features for each query document pair.

4.2 Tensor Factorization

Another way to encode the different collections of score matrices $\{X^1, \ldots, X^P\}$ is via tensor based representation. In particular, we can pack the P matrices into a three dimensional tensor $\mathcal{X} \in \mathbb{R}^{M\times N\times P}$ with each entry x_{qdi} equal to the score x^i_{dq}. Tensor factorization (TF) models have been studied in several fields for many years and successfully used in applications like personalized tag recommendation.

Feature	Orignal	IMF	TF
# Impressions	0.565	0.552	0.561
# Clicks	0.620	0.609	0.617
AvgClickPos	0.617	0.622	0.631
CTR	0.628	0.629	0.636

Table 6: Ranking performances based on individual features with original and calibrated values

In contrast to matrix factorization models which can only model interactions between two groups of entities, the tensor model can be used to capture higher order interactions involving more than two entities. For example, social annotation data about which user used which tags on which items can be encoded by a 3D tensor with the three dimensions corresponding to users, items and tags respectively.

A general model is the tucker decomposition[11], which has previously been used for personalized tag recommendation[13]. A special case of the Tucker decomposition is the canonical decomposition (CD)[11], which is also known as the PARAFAC model. In this work, we adopt the PARAFAC model to decompose the tensor $\mathcal{X}$ containing different types of scores on query document pairs. Under the PARAFAC model, each query q, document d and score type i is associated with their respective latent factors $\mathbf{u}_d$, $\mathbf{v}_q$ and $\mathbf{o}_i$ and the factorization is of the following form:

$$\widehat{x}_{dqi} = \mathbf{u}_d \circ \mathbf{v}_q \circ \mathbf{o}_i = \sum_k u_{d,k} \cdot v_{q,k} \cdot o_{i,k} \tag{7}$$

The objective function used for training a TF model is via solving the following optimization problem:

$$\arg\min_{U,V,O} \sum_{dqi} \ell(x_{dqi}, \mathbf{u}_d \circ \mathbf{v}_q \circ \mathbf{o}_i) + \mathcal{R}(U,V,O) \tag{8}$$

which optimizes over the three factor matrices U,V and O rather than just U and V as in the CMF model. Compared with the CMF model, TF introduces an additional factor matrix O to absorb specific properties about each type of score. So in addition to producing the U and V factors as new features, TF is also able to approximate every type of original scores for any query document pair using equation 7. Given a learnt TF model, we can obtain a set of $2K + P$ additional parameters for each query document pair including the factors for the query and document, $\mathbf{u}_d$ and $\mathbf{v}_q$ and the set of approximate scores $\{\widehat{x}^1_{dq}, \ldots, \widehat{x}^p_{dq}\}$.

4.3 Results

Table 6 summarizes the results obtained using each individual type of feature scores, in which we compare the AUC obtaining using the original feature values and the calibrated feature values based IMF and TF respectively. We can see that the effectiveness of the calibrated values via TF tend to be comparable and better than the original values. TF also consistently outperforms IMF.

5. LEARNING TO RANK

In the previous two sections, we have describe various ways to derive features about query-document pairs from raw search query logs and methods based collaborative filtering to post-process the click based raw features. Most techniques that we have used so far have not considered the use of labeled data at all. In this section, we describe the final step of our solution, which is the use of learning to rank methods to obtain a ranking function to combine all the features generated in previous steps based on the provided labeled data.

We begin by formalizing the problem of learning to rank. Our training data consists of a set N queries $Q = \{q_1, \ldots, q_N\}$. Each query i is associated with a set of $m^{(i)}$ documents $D_i = \{d^{(i)}_j\}^{m^{(i)}}_{j=1}$. Each document $d^{(i)}_j$ is represented as a query dependent feature vector $\mathbf{x}^{(i)}_j \in \mathbb{R}^p$ and has a corresponding relevance judgement $y^{(i)}_j$. In this work, we focus on binary relevance judgement (i.e. $y^{(i)}_j \in \{-1, +1\}$). The goal of learning to rank is to create a function $f : \mathbb{R}^p \to \mathbb{R}$ for scoring documents with respect to queries based on the feature vector $\mathbf{x}^{(i)}_j$, so that the ranking of documents based on scores produced by f is maximally consistent with the ranking given by the true labels.

There are two major components of a learning to rank algorithm. One is the functional class used for f, which may be learning functions, neural networks or trees, etc. The other is the loss function used to measure the inconsistency between rankings produced by f and the ground truth, which is optimized to learn the function f. Typical types of losses include regression or classification loss over individual documents (i.e., pointwise approach), classification of pairs of documents (pairwise approach) or permutations over sets of documents (listwise approach). In this work, we emloyed both neural network and multiple additive tree based ranker and tried to learn these models using a variety of loss functions

5.1 RankNet

RankNet[1] is a neural network based ranker, whose loss function is defined over pairs of documents. Given two documents u and v associated with the same query i, a target probability $\bar{P}^q_{u,v}$ is defined based on the two documents labels. For example, we can define $\bar{P}^q_{u,v} = 1$ if $y^{(q)}_u = +1$ and $y^{(q)}_v = -1$; $\bar{P}^q_{u,v} = 0$, othewise. Then the probability of this outcome under the model is defined based on the difference between the scores on these two documents given by the scoring function:

$$P_{u,v}(f) = \frac{\exp(f(\mathbf{x}^{(i)}_u) - f(\mathbf{x}^{(i)}_v))}{1 + \exp(f(\mathbf{x}^{(i)}_u) - f(\mathbf{x}^{(i)}_v))}$$

Then the cross entropy between the target and model probability is used as the loss function:

$$\ell(f; u, v, i) = -\bar{P}^{(i)}_{u,v} \log P_{u,v}(f) - (1 - \bar{P}_{u,v}) \log(1 - P_{u,v}(f))$$

It can be seen that such loss function has its advantages over the pointwise regression or classification approach in that it focus on modeling the relative order between documents rather than the absolute magnitude of relevance labels associated with individual documents. Theoretically, when the labels are binary, optimizing this pairwise loss is indeed equivalent to optimizing the Area Under Curve (AUC) evaluation measure. In RankNet, a neural network is used to model f and gradient descent is used as the optimization algorithm to learn parameters in the neural network. In this work, we adopt a two layer network as the underlying scoring

function, which embodies the following scoring function:

$$f(\mathbf{x}) = \sigma\Big(\sum_i w_i^2 \sigma\Big(\sum_j w_{ij}^1 x_j + b_i^1\Big) + b^2\Big) \quad (9)$$

where w_{ij}^1's and b_i^1's are the weight and bias parameters for the layer connecting the inputs and the hidden units, and w_i^2's and b^2 are the weights and bias for the layer connecting the hidden units and the ouput. The function $\sigma(\cdot)$ is the sigmoid function, $\sigma(x) = (1 + e^{-x})^{-1}$.

5.2 LambdaRank

Most ranking evaluation measures are usually based the positions of relevant and irrelevant documents in the resulted ranked list with top positions being emphasized more significantly in metrics like NDCG. This motivates the design of *listwise* loss functions that are dependent upon the resulted ranked list and could serve as a good proxy for the concerned evaluation measures, such as NDCG, MAP, etc. In this work, we used the listwise ranking framework LambdaRank[3]. The LambdaRank framework is based on the λ-gradient technique, which assign an additional weight to the gradient comuted using a document pair by the difference in the target performance metric obtained if the positions of the two documents are swapped. To apply LambdaRank, at the beginning of each iteration of the optimization, the documents are first sorted according to scores under the current model. Then the difference in a target performance metric like NDCG is computed for each pair of documents by keeping the ranks of all other documents constant and swapping only the selected pair. The full gradient over model parameters are then computed as a weighted sum of the gradients over all possible pairs. The λ-rank framework places stronger emphasis on those pairs that have the largest impact on the target evaluation measure.

In the original LambdaRank paper[3], a neural network model is learnt using λ-gradient, which can be viewed as an extension of the RankNet model for optimizing different target evaluation measures besides AUC. In fact, RankNet is indeed a special case of LambdaRank with uniform weights over all document pairs. LambdaMART[2] is an algorithm that implements the LambdaRank ide using the multiple additive regression tree (MART), a boosted tree model where each tree tries to approximates the gradients of the loss function with respect to the model predictions for each training instance. LambdaMART is a state of the art ranking model, recently winning the Yahoo! Learning to Rank Challenge. The scoring function produced by LambdaMART after T iterations can be written as

$$f(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t f_t(\mathbf{x}) \quad (10)$$

where $f_t(\mathbf{x}) : \mathbb{R}^p \to \mathbb{R}$ is modeled by a single regression tree that best fits the current gradients and $\alpha_i \in \mathbb{R}$ is the weight associated each regression tree and playes the role as step lengths in gradient descent steps.

5.3 Results

The relevance prediction challenge provided a collection of 41,276 labeled query-document pair associated with 4,991 distinct queries and 36,395 documents. As noted by the challenge organizer, the final test set differs from the training set in two respects. First, the training set contains queries with document labels of only one kind (all relevant or all irrelevant). Second, the training set contains queries with more than 300 unique clicked documents. Both types of queries were deliberately excluded from the test set. So a natural question is whether or not to utilize these two types of queries during the training and how to utilize them. There seems to be a natural trade off here. Excluding these invalid queries would make the training data more consistent with the test data, but would reduce the amount of labeled data at the same time. In our experiments, we consider two training set, one with and another without the invalid queries. The ranking models we compared are listed in Table 5.3 whereas the different feature representations are listed in Table 5.3.

Model	Description
M1	RankNet
M2	MART optimized for regression loss
M3	MART optimized for classification loss
M4	LambdaMART optimized for NDCG

Table 7: Different learning to rank models used in our experiments

Feature	Description
D1	Original features
D2	Original features plus IMF features
D3	Original features plus TF features

Table 8: Different feature representaitons

From the results, we can make the following observations:

- LambdaMART was consistently the best ranker under different settings. Two two variations of MART also outperformed RankNet, which seemed to suggest that additive tree based nonlinear model is particularly suited to the task of ranking based on click features. The superior performance of LambdaMART also suggested the importance of choosing the right objective function. The NDCG measure is query-normalized and top-sensitive, which appeared to be an effective proxy measure for AUC measure used in the official evaluation.

- The incorporation of IMF and TF features tended to lead to superior ranking performance for most models. The TF features consistently outperform the IMF features. The results confirmed the effectiveness of the latent factors based features for characterizing query-document relations.

- Excluding the invalid queries appeared to improve the performance of MART, but have little effect for RankNet and LambdaMART. This is as expected, since the one-class queries would be automatically ignored by both pairwise and listwise loss functions. The NDCG measure being optimized by LambdaMART also does query normalization automatically and won't be biased toward queries with many documents.

We conduct experiments to investigate the choice and parameter settings of the ranking models with different ways to form the feature representation for the relevance prediction task and the handling of invalid queries in training.

	D1	D2	D3
M1	0.6553	0.6544	0.6566
M2	0.6551	0.6593	0.6601
M3	0.6574	0.6609	0.6619
M4	0.6588	0.6621	0.6649

Table 9: 10 fold cross validated AUC of different models based on different feature representations using full training set

	D1	D2	D3
M1	0.6546	0.6518	0.6562
M2	0.6555	0.6570	0.6617
M3	0.6561	0.6612	0.6633
M4	0.6589	0.6619	0.6643

Table 10: 10 fold cross validated AUC of different models based on different feature representations using filtered training set

6. CONCLUSION AND FUTURE WORK

In this paper, we describe our solution to the Relevance Prediction Challenge with various simple statistical quantities and principle models for deriving features. The incorporation of click models and latent factor model as new features brings significant improvements on this challenge. The features are then combined via learning a ranking model based on the provided labeled data. Experimental results demonstrated effectiveness of our proposed solution. In the future, we would look into integrating latent factor model and labeled data into the click model framework, so multiple sources of knowledge could be simultaneously utilized.

7. ACKNOWLEDGE

We would like to thank Prof.Yang Qiang from HKUST, Dong Wang from UCLA and Yuchen Zhang from UC.Berkeley for discussions and advices on this work.

8. REFERENCES

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

[2] C. J. C. Burges. From ranknet to lambdarank to lambdamart : An overview. *Learning*, 11(MSR-TR-2010-82):23–581, 2010.

[3] C. J. C. Burges, R. Ragno, and Q. V. Le. Learning to rank with nonsmooth cost functions. In *NIPS'06*, pages 193–200, 2006.

[4] O. Chapelle and Y. Zhang. A dynamic bayesian network click model for web search ranking. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 1–10, New York, NY, USA, 2009. ACM.

[5] W. Chen, D. Wang, Y. Zhang, Z. Chen, A. Singla, and Q. Yang. A noise-aware click model for web search. In *Proceedings of the international conference on Web search and web data mining*, WSDM '12, 2012.

[6] N. Craswell, O. Zoeter, M. Taylor, and B. Ramsey. An experimental comparison of click position-bias models. In *Proceedings of the international conference on Web search and web data mining*, WSDM '08, pages 87–94, New York, NY, USA, 2008. ACM.

[7] G. Dupret and C. Liao. A model to estimate intrinsic document relevance from the clickthrough logs of a web search engine. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 181–190, New York, NY, USA, 2010. ACM.

[8] G. E. Dupret and B. Piwowarski. A user browsing model to predict search engine click data from past observations. In *Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '08, pages 331–338, New York, NY, USA, 2008. ACM.

[9] F. Guo, C. Liu, A. Kannan, T. Minka, M. Taylor, Y.-M. Wang, and C. Faloutsos. Click chain model in web search. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pages 11–20, New York, NY, USA, 2009. ACM.

[10] B. Hu, Y. Zhang, W. Chen, G. Wang, and Q. Yang. Characterizing search intent diversity into click models. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 17–26, New York, NY, USA, 2011. ACM.

[11] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Rev.*, 51:455–500, August 2009.

[12] Y. Koren, R. Bell, and C. Volinsky. Matrix factorization techniques for recommender systems. *Computer*, 42:30–37, August 2009.

[13] S. Rendle, L. Balby Marinho, A. Nanopoulos, and L. Schmidt-Thieme. Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 727–736, New York, NY, USA, 2009. ACM.

[14] A. P. Singh and G. J. Gordon. A unified view of matrix factorization models. In *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*, ECML PKDD '08, pages 358–373, Berlin, Heidelberg, 2008. Springer-Verlag.

[15] Y. Zhang, W. Chen, D. Wang, and Q. Yang. User-click modeling for understanding and predicting search-behavior. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1388–1396. ACM, 2011.

[16] Y. Zhang, D. Wang, G. Wang, W. Chen, Z. Zhang, B. Hu, and L. Zhang. Learning click models via probit bayesian inference. In *Proceedings of the 19th ACM international conference on Information and knowledge management*, pages 439–448. ACM, 2010.

[17] Z. A. Zhu, W. Chen, T. Minka, C. Zhu, and Z. Chen. A novel click model and its applications to online advertising. In *Proceedings of the third ACM international conference on Web search and data mining*, WSDM '10, pages 321–330, New York, NY, USA, 2010. ACM.