Learning to Rank Algorithms and their Application in Machine Translation

Tian Xia
Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Computer Engineering Commons, and the Computer Sciences Commons

Repository Citation
https://corescholar.libraries.wright.edu/etd_all/1643

This Dissertation is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact corescholar@www.libraries.wright.edu, library-corescholar@wright.edu.
Learning to Rank Algorithms and Their Application in Machine Translation

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

by

Tian Xia
M.S., Chinese Academy of Sciences, 2011
B.S., Xidian University, 2007

2015
Wright State University

Shaojun Wang, Ph.D.
Dissertation Director

Michael Raymer, Ph.D.
Director, Department of Computer Science and Engineering

Robert E. W. Fyffe, Ph.D.
Vice President for Research and Dean of the Graduate School

Committee on Final Examination

Keke Chen, Ph.D.

Xinhui Zhang, Ph.D.

Michael Raymer, Ph.D.
ABSTRACT


In this thesis, we discuss two issues in the learning to rank area, choosing effective objective loss function, constructing effective regression trees in the gradient boosting framework, as well as a third issue, applying learning to rank models into statistical machine translation.

First, list-wise based learning to rank methods either directly optimize performance measures or optimize surrogate functions of performance measures that have smaller gaps between optimized losses and performance measures, thus it is generally believed that they should be able to lead to better performance than point- and pair-wise based learning to rank methods. However, in real-world applications, state-of-the-art practical learning to rank systems, such as MART and LambdaMART, are not from list-wise based camp. One cause may be that several list-wise based methods work well in the popular but very small LETOR datasets but fail in real-world datasets that are often used for training practical systems.

We propose a list-wise learning to rank method that is based on a list-wise surrogate function, the Plackett-Luce (PL) model. The PL model has convex loss to ensure a global optimal guarantee, and is proven to be consistent to certain performance measures such as NDCG score. When we conduct experiments on the PL model, we observe that it is actually unstable in performance; when the data has rich enough features, it gives very good results, but for data with scarce features, it fails horribly. For example, when we
apply the PL with a linear model on the Microsoft 30K dataset, it gives 7.6 points worse NDCG@1 score than an average performance of several linear systems. This motivates us to propose our new ranking system, PLRank, that is suitable for any data sets through a mapping from feature space into tree space to gain more expressive power. PLRank is trained based on the gradient boosting framework, and it is simple to implement. It has the same time complexity as the LambdaMART, and runs a little bit faster in practice. Moreover, we extend three other list-wise surrogate functions in a gradient boosting framework for a fair and full comparison, and we find that the PL model has special advantages.

Our experiments are conducted on the two largest publicly available real-world datasets, Yahoo challenge 2010 and Microsoft 30K. The results show this is the first time in the single model level for a list-wise based system to match or overpass state-of-the-art point- and pair-wise based ones, MART, LambdaMART, and McRank, in real-world datasets.

Second, industry-level applications of learning to rank models have been dominated by gradient boosting framework, which fits a tree using least square error (SE) principle. Another tree fitting principle, (robust) weighted least square error ((R)WSE), has been widely used in classification, such as LogitBoost and its variants, but hasn’t been reformulated to fulfill learning the rank tasks. For both principles, there is a lack of deep analysis on their relationship in the scenario of learning to rank. Motivated by AdaBoost, we propose a new principle named least objective loss based error (OLE) that enables us to analyze several important learning to rank systems: *we prove that (R)WSE is actually a special case of OLE for derivative additive loss functions; OLE, (R)WSE and SE are equivalent for MART system*. Under the guidance of OLE principle, we implement three typical and strong systems and conduct our experiments in two real-world datasets. Experimental results show
that our proposed OLE principle improves most results over SE.

Third, Margin infused relaxed algorithms (MIRAs) dominate model tuning in statistical machine translation in the case of large scale features, but also they are famous for the complexity in implementation. We introduce a new method, which regards an N-best list as a permutation and minimizes the Plackett-Luce loss of ground-truth permutations. Experiments with large-scale features demonstrate that, the new method is more robust than MERT; though it is only matchable with MIRAs, it has a comparatively advantage, easier to implement.
Contents

1 Introduction 1
  1.1 Learning to Rank ......................................................... 1
    1.1.1 Methodologies ..................................................... 1
    1.1.2 Basic Notations .................................................. 3
    1.1.3 Metrics ............................................................. 4
    1.1.4 Gradient Boosting, Square Error (SE) and Regression Tree .... 5
    1.1.5 Classification and Learning to Rank ............................. 7
  1.2 Our Work ...................................................................... 10
    1.2.1 Plackett-Luce Model for Learning-to-Rank Task ................. 10
    1.2.2 Analysis of Regression Tree Fitting Algorithms in Learning to Rank 12
    1.2.3 A Simple Discriminative Training Method for Machine Translation with Large-Scale Features ...................... 14

2 Plackett-Luce Model for Learning-to-Rank Task 16
  2.1 Plackett-Luce Loss for Learning to Rank ............................. 16
    2.1.1 Plackett-Luce Loss with Linear Features ....................... 17
    2.1.2 Plackett-Luce Loss with Regression Trees ...................... 18
    2.1.3 Training with Plackett-Luce Loss ................................ 24
    2.1.4 Transformation between PL and McRank .......................... 25
    2.1.5 Comparison with Other Consistent List-wise Methods .......... 27
  2.2 Experiments .................................................................. 28
    2.2.1 Datasets .................................................................... 29
    2.2.2 l-ListMLE vs. Other Linear Systems ............................... 33
    2.2.3 Different Number of Ground Truth Permutations ............... 35
    2.2.4 PLRank vs. l-ListMLE, MART, McRank and LambdaMART ...... 36
2.2.5 PLRank vs. Other Consistent List-wise Method with Boosted regression trees ................................................................. 38
2.2.6 Influence of Different Feature Number ................................................................. 39
2.2.7 Industry-level Comparison .................................................................................. 41
2.2.8 Running time ................................................................................................. 42

3 Analysis of Regression Tree Fitting Algorithms in Learning to Rank 43
3.1 Greedy Tree Fitting Algorithm in Learning to Rank ........................................... 43
3.1.1 Objective Loss Based Error (OLE) ................................................................. 44
3.1.2 Derivative Additive Loss Functions .................................................................. 46
3.1.3 (R)WSE ⊂ OLE .......................................................................................... 50
3.1.4 SE = (R)WSE = OLE for MART .................................................................. 51
3.2 Experiments ....................................................................................................... 54
3.2.1 Datasets and Systems ..................................................................................... 54
3.2.2 Experimental Comparison with Three Systems ............................................. 60

4 A Simple Discriminative Training Method for Machine Translation with Large-Scale Features 72
4.1 Plackett-Luce Model ......................................................................................... 72
4.2 Plackett-Luce Model in Statistical Machine Translation .................................. 74
4.3 Evaluation .......................................................................................................... 77
4.3.1 Plackett-Luce Model for SMT Tuning ......................................................... 78
4.3.2 Plackett-Luce Model for SMT Reranking .................................................. 80

5 Conclusion ......................................................................................................... 82

Bibliography ........................................................................................................ 84
List of Figures

1.1 Three node splitting principles in decision tree construction, namely Square Error (SE) in Gradient Boosting framework, Robust Weighted Square Error (RWSE) in the LogitBoost system, and Objective Loss Based Error (OLE) in this paper, as well as their relations. All real-line arrows denote known relations, and dotted-line arrows denote our contribution. 13

2.1 The performance of \( l \)-ListMLE and the selected linear reference system Coordinate Ascent (CA) on Yahoo data (left) and Microsoft 30K (right). CA is capable of representing the mainstream linear systems on these datasets Tan et al. (2013a). 33

2.2 \( l \)-ListMLE and CA with different number of features. 34

2.3 Comparison of several tree-based systems. 36

2.4 With different feature number, the comparison of the PLRank and the LambdaMART systems on the synthetic dataset. The feature number in the real distribution is 1000, and per query document number is 250. 40

3.1 Histogram of total improvements of OLE over SE. 60

3.2 Improvements (absolute %) of OLE over SE with McRank system when the learning rate is set as 0.06.  Each point in the figures has been averaged over a five-fold or three-fold cross-validation. 63

3.3 Improvements (absolute %) of OLE over SE with McRank system when the learning rate is set as 0.1.  Each point in the figures has been averaged over a five-fold or three-fold cross-validation. 64

3.4 Improvements (absolute %) of OLE over SE with McRank system when the learning rate is set as 0.12.  Each point in the figures has been averaged over a five-fold or three-fold cross-validation. 65
3.5 Improvements (absolute %) of OLE over SE with LambdaMart system when the learning rate is set as 0.06. *Each point in the figures has been averaged over a five-fold or three-fold cross-validation.* .......................... 66

3.6 Improvements (absolute %) of OLE over SE with LambdaMart system when the learning rate is set as 0.1. *Each point in the figures has been averaged over a five-fold or three-fold cross-validation.* .......................... 67

3.7 Improvements (absolute %) of OLE over SE with LambdaMart system when the learning rate is set as 0.12. *Each point in the figures has been averaged over a five-fold or three-fold cross-validation.* .......................... 68

3.8 Improvements (absolute %) of OLE over SE with RankBoost system when the learning rate is set as 0.06. *Each point in the figures has been averaged over a five-fold or three-fold cross-validation.* .......................... 69

3.9 Improvements (absolute %) of OLE over SE with RankBoost system when the learning rate is set as 0.1. *Each point in the figures has been averaged over a five-fold or three-fold cross-validation.* .......................... 70

3.10 Improvements (absolute %) of OLE over SE with RankBoost system when the learning rate is set as 0.12. *Each point in the figures has been averaged over a five-fold or three-fold cross-validation.* .......................... 71

4.1 PL(\(k\)) with 500 L-BFGS iterations, \(k=1,3,5,7,9,12,15\) compared with MIRA in reranking. .......................... 79
List of Tables

2.1 The top two datasets are used, while the others are as a reference. Ohsumed is of LETOR 3.0. #D./#Q. means average document number per query. 30

2.2 Main results on the Yahoo Challenge dataset. Results on the standard five splits of Microsoft data are averaged, and we follow the standard one split for the Yahoo data to compare to published results. System 1 is trained towards optimizing NDCG. System 10, CA, is a Coordinate Ascent based method directly maximizing NDCG. We provide results marked as “ref” reported in other papers. LambdaMART-Aug70 is trained on resampled training data, where our experiments are conducted on full training data. As PLRank(obj=6) in Microsoft data starts to decrease, we did not test more objectives. CosMART got an abnormally low score in Microsoft data as /-ListMLE, thus it is less meaningful to list them. 31

2.3 Main results on the Microsoft dataset. Results on the standard five splits of Microsoft data are averaged, and we follow the standard one split for the Yahoo data to compare to published results. System 1 is trained towards optimizing NDCG. System 10, CA, is a Coordinate Ascent based method directly maximizing NDCG. We provide results marked as “ref” reported in other papers. LambdaMART-Aug70 is trained on resampled training data, where our experiments are conducted on full training data. As PLRank(obj=6) in Microsoft data starts to decrease, we did not test more objectives. CosMART got an abnormally low score in Microsoft data as /-ListMLE, thus it is less meaningful to list them. 32

2.4 Compression ratio of denominator terms in Eqn. (2.3) in considering multiple ground-truth permutations. 35

2.5 An industry level comparison in the standard Yahoo Challenge set 1 data with LambdaMART. 41
2.6 Sorted running time in the Microsoft dataset in a single computing core.

3.1 The top two datasets are used in this work and others are just a reference.
   λ-MART is LambdaMART. Ohsumed is of letor 3.0. #D/#Q. means average document number per query. .......................... 55

3.2 Performances (%) of SE / OLE in the Yahoo Data with McRank model.
   All results reported are averaged over self-defined three-fold. All results with over 0.1 point improvement are marked. .......................... 56

3.3 Performances (%) of SE / OLE in the Yahoo Data with LambdaMART model.
   All results reported are averaged over self-defined three-fold. All results with over 0.1 point improvement are marked. .......................... 57

3.4 Performances (%) of SE / OLE in the Yahoo Data with RankBoost model.
   All results reported are averaged over self-defined three-fold. All results with over 0.1 point improvement are marked. .......................... 58

4.1 The probability of the rank $\pi = (2, 3, 1)$ is $p(r_2) \cdot p(r_3)/(1 - p(r_2))$ in a simplified form, as $\frac{p(r_2)}{Z_1} = p(r_2)$ and $\frac{p(r_1)}{Z_3} = 1$. .......................... 73

4.2 PL(k): Plackett-Luce model optimizing the ground-truth permutation with length $k$. The significant symbols (+ at 0.05 level) are compared with MERT. The bold font numbers signifies better results compared to M(1) system. .......................... 77
Acknowledgment

First, I would like to give my appreciation to my Advisor, Prof. Shaojun Wang, who is like an amiable father but also strict on our work, and keeps giving endless encouragement and support, in the past several years.

Second, I would like to give my gratitude to Prof. Keke Chen, Prof. Michael Raymer, and Prof. Xinhui Zhang, for their invaluable time and advice as well as serving as my committees.

Third, I would like to give my thanks to my colleagues, Shaodan Zhai, Ming Tan, Zhongliang Li, Raymond Kulhanek, Lu Chen, Wenbo Wang and other awesome guys, who accompanied me to go through the happiest days and the hardest days.

Finally, I would like to give my special thanks to my family, and wife, for their supporting my work, understanding my decision in the work and life.
Dedicated to
Introduction

1.1 Learning to Rank

1.1.1 Methodologies

The learning to rank task arises from practical real-world applications such as Google, Yahoo, Bing and other search engines, and has been flourishing for a decade. To put it simply, after a user inputs a query, the ranking system is designed to return a set of documents and rank them by their relevance to the input. Commonly used measures to quantify a rank quality of retrieved documents include MAP, ERR, NDCG, and etc.

The learning to rank task arises from real-world applications such as Google, Yahoo, and other search engines. A ranking system returns a set of documents and ranks them by their relevance to the query from a user.

Learning to rank techniques are influencing traditional natural language processing applications, such as model parameter training Hopkins and May (2011b), and non-linear feature extraction Sokolov et al. (2012); Toutanova and Ahn (2013).

Generally, ranking models fall into three methodologies based on how they model
basic ranking objects. *This definition would not be affected by how to utilize features*, e.g., linear and non-linear features.

The first methodology, **point-wise based**, breaks relationship between documents related to different queries Cossock and Zhang (2006); Crammer et al. (2001); Friedman (2001); Li et al. (2007), then uses traditional machine learning regression and classification techniques for training. For example, MART Friedman (2001) uses the regression tree technique to fit model outputs to their relevance scores; McRank Li et al. (2007) converts the rank procedure as a multi-class classification.

The second methodology, **pair-wise based**, considers the relationship among documents related to the same query Cohen et al. (1999); Freund et al. (2003); Hazan et al. (2010); Herbrich et al. (1999); Joachims (2002); Quoc and Le (2007); Rudin (2009); Tsai et al. (2007); Wu et al. (2010), then adopts mature classification techniques to minimize the inversion number of documents by considering document pairs. For example, RankBoost Freund et al. (2003) plugs the exponential loss of document pairs into a framework of Adaboost; RankSVM Herbrich et al. (1999); Joachims (2002) uses SVM to perform a binary classification on the document pairs; LambdaRank Quoc and Le (2007) and LambdaMART Wu et al. (2010) take into account the influence of a correctly classified document pair to the objective measures, and achieve a big success.

The third methodology, **list-wise based**, treats a permutation of a set of documents as a basic unit, and builds loss functions on them Cao et al. (2007); Metzler and Croft (2007); Ravikumar et al. (2011); Tan et al. (2013a); Xia et al. (2009, 2008); Xu and Li (2007); Xu et al. (2008). Because exact losses of performance measures are step-wise, non-differentiable as well as non-convex with respect to model parameters, most work
in this methodology resort to suitable surrogate functions. These surrogate functions are
either not directly related to ranking performance measures Cao et al. (2007); Qin et al.
(2008); Xia et al. (2009, 2008), or just continuous and differentiable approximation bounds
of ranking measures Chakrabarti et al. (2008); Chapelle and Wu (2010); Le and Smola
(2007); Qin et al. (2010a); Taylor et al. (2008); Valizadegan et al. (2009); Wu et al. (2010);
Xu et al. (2008); Xu and Li (2007); Yue et al. (2007). To further decrease the gap be-
tween optimization objectives and performance measures, some work attempt to directly
optimize objective measures and show promising results. For example, in Metzler and
Croft (2007); Tan et al. (2013a), the authors use a coordinate ascent framework to directly
optimize performance measures, and DirectRank in Tan et al. (2013a) is much faster in
practice. However, both their work still can not match the state-of-the-art systems in large
data sets when decision trees are used.

1.1.2 Basic Notations

Given a set of queries $Q = \{q_1, \ldots, q_{|Q|}\}$, each query $q_i$ is associated with a set of can-
didate relevant documents $D_i = \{d_{i1}, \ldots, d_{i|D_i|}\}$ and a corresponding vector of relevance
scores $r_i = \{r_{i1}, \ldots, r_{i|D_i|}\}$ for each $D_i$. The relevance score is usually an integer, and
greater value means more related for the document to the query. An $M$-dimensional
feature vector $h(d) = [h_1(d|q), \ldots, h_M(d|q)]^T$ is created for each query-document pair,
where $h_i(\cdot)$s are predefined real-value feature functions.

A ranking function $f$ is designed to score each query-document pair, and the doc-

\footnote{Tan et al. Tan et al. (2013a) use a mixed strategy, which borrows boosted trees generated from MART,
to compete with LambdaMART. Their strategy should be treated as a system combination technique rather
than a single ranking model.}
Documents associated with the same query are ranked by their scores and returned to users. Since these documents have a fixed ground truth rank with its corresponding query, our goal is to find an optimal ranking function which returns such a rank of related documents that is as close to the ground truth rank as possible. Industry-level applications often adopt regression trees to construct the ranking function, and use Newton Formula to calculate the output values of leaves of trees.

Generally, ranking functions use only linear information of original features $h(d|q)$ or their nonlinear information. The linear form is as $f(d|q) = w^T \cdot h(d)$, where $w = [w_1, \ldots, w_M]^T \in \mathcal{R}^M$ is the model parameter. The nonlinear form often adopts regression trees, kernel technique, and neural network.

Several measures have been used to quantify the quality of a rank, such as NDCG@K, ERR, MAP etc. In this paper, we use the most popular NDCG@K and ERR Chapelle and Chang (2011) as the performance measures.

1.1.3 Metrics

Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen (2002)) is a popular metric for relevance judgments. It assigns exponentially high weight to highly relevant documents.

Given a training query $q_i$ and the ranking function $f$, the relevant documents $D_i$ is sorted by their ranking scores. NDCG value is computed as following.
\[ \text{NDCG}(q_i, f) = \frac{\text{DCG}(q_i, f)}{\text{Idea DCG}} \] (1.1)

\[ \text{DCG} = \sum_{j=1}^{K} \frac{2^{r_j} - 1}{\log(1 + j)} \] (1.2)

Usually, NDCG is trimmed at a certain ranking level \( K \), such as, 1, 3, 10.

ERR is a novel metric based on the cascade user model (Chapelle et al. (2009)). It is defined as the expected reciprocal rank at which the user will stop his search under this model. The resulting formula is:

\[ \text{ERR} = \sum_{j=1}^{n} \frac{1}{j} \text{P(user stops at } j) = \sum_{j=1}^{n} \frac{1}{j} R(j) \prod_{t=1}^{j-1} (1 - R(t)) \] (1.3)

\[ R(t) = \frac{2^{y_t} - 1}{16} \] (1.4)

1.1.4 Gradient Boosting, Square Error (SE) and Regression Tree

We review gradient boosting Friedman (2001) as a general framework for function approximation using regression trees as the weak learners, which has been the most successful approach for learning to rank models.

Gradient boosting iteratively finds an additive predictor \( f(\cdot) \in \mathcal{H} \) that minimizes a loss function \( \mathcal{L} \). At the \( t \)th iteration, a new weak learner \( g_t(\cdot) \) is selected to be added to
current predictor $f_t(\cdot)$ to construct a new predictor,

$$f_{t+1}(\cdot) = f_t(\cdot) + \alpha g_t(\cdot)$$  \hspace{1cm} (1.5)

where $\alpha$ is the learning rate.

$$r(\cdot) = -\mathcal{L}'(f_t(\cdot))$$  \hspace{1cm} (1.6)

In gradient boosting, according to the following squared loss, $g_t(\cdot)$ is chosen as the one most parallel to the pseudo-response $-\frac{\partial \mathcal{L}}{\partial f_t(\cdot)}$, which is negative derivative of the loss function in functional space.

$$g_t(\cdot) = \arg \min_{g \in \mathcal{H}} \| -\frac{\partial \mathcal{L}}{\partial f_t(\cdot)} - g(\cdot) \|_2^2$$  \hspace{1cm} (1.7)

To fit a regression tree, the data in each internal tree node is greedily split into two parts by minimizing Eqn. (1.7), and this procedure recursively iterates until a predefined condition is satisfied. This tree construction procedure is applicable for any differentiable loss function. The complexity of a regression tree is usually controlled by the tree height or leaf number. In learning to rank, the latter is more flexible, thus is adopted in this work by default.

In one node, we enumerate all features as well as their possible thresholds, and find the best feature-threshold pair with the smallest error to conduct binary splitting on the current node.

Regarding a feature, suppose a threshold $v$ to split samples on the current node into
two parts. The samples, whose feature values are less than \( v \), are denoted as \( D_l \), and others are denoted as \( D_r \). Then the squared error is defined as

\[
SE(v) = \sum_{d \in D_l} (r(d) - \bar{r}_1)^2 + \sum_{d \in D_r} (r(d) - \bar{r}_2)^2
\]

(1.8)

where \( \bar{r}_1, \bar{r}_2 \) are average pseudo-response of samples on the left and right respectively.

### 1.1.5 Classification and Learning to Rank

Cossock et al. Cossock and Zhang (2006); Li et al. (2007) proved that the negative unnormalized NDCG value is upper-bounded by multi-class classification error, where NDCG is an important measure in learning to rank. Thus Li et al. Li et al. (2007) proposed a multi-class classification based ranking systems called McRank in gradient boosting framework.

McRank utilizes classic logistic regression, which models class probability \( p_k(d) \) as

\[
p_k(d) = p(y(d) = k | d) = \frac{\exp f^k(d)}{\sum_{c=0}^{K} \exp f^c(d)}
\]

(1.9)

where \( f^c(\cdot) \) is an additive predictor function for the \( c \)th class.

The objective loss function is the negative log-likelihood, defined as

\[
\mathcal{L} = -\sum_{d} \sum_{c=0}^{K} \mathbb{I}(y(d) = c) \log p_c(d)
\]

(1.10)

where \( \mathbb{I}(\cdot) \) is an indicator function.
Weighted Square Error (WSE) in Logitboost

In classification, this loss function (Eqn. 1.10) resulted in the well-known system LogitBoost, which first used WSE to fit a regression tree. WSE utilizes both first- and second-order derivative information.

WSE uses a different definition of the response value $r(\cdot)$ from that in SE (Eqn. 1.6), and defines an extra weight $w(\cdot)$ for each sample.

$$r(\cdot) = -\frac{L'(f_i(\cdot))}{L''(f_i(\cdot))}$$
$$w(\cdot) = L''(f_i(\cdot))$$

(1.11)

The splitting principle is minimizing the following weighted error

$$WSE(v) = [\sum_{d \in D_l} w(d)(r(d) - \bar{r_1})^2 + \sum_{d \in D_r} w(d)(r(d) - \bar{r_2})^2]$$
$$- \sum_{d \in D} w(d)(r(d) - \bar{r})^2$$

(1.12)

where

$$\bar{r_1} = \frac{\sum_{d \in D_l} w_i \cdot r(d)}{\sum_{d \in D_l} w_i}$$
$$\bar{r_2} = \frac{\sum_{d \in D_r} w_i \cdot r(d)}{\sum_{d \in D_r} w_i}$$
$$\bar{r} = \frac{\sum_{d \in D} w_i \cdot r(d)}{\sum_{d \in D} w_i}$$

(1.13)

Regarding LogitBoost, the response and weight values, by Eqn. 1.11, are set as $w(d) = p_k(d)(1 - p_k(d))$, $r(d) = \frac{I[y(d)=k] - p_k(d)}{p_k(d)(1 - p_k(d))}$, in fitting a tree for the $k$th classification.
The response \( r(d) \) might become huge and lead to unsteadiness when \( p_k(d) \) is approaching 0 or 1. Though Friedman et al. Friedman et al. (2000) described some heuristics to smooth the response values, LogitBoost was still believed numerically unstable Friedman (2001); Li (2010b); Friedman et al. (2000). As a result, McRank is actually using SE and gradient boosting to fit regression trees (Section 1.1.4), rather than LogitBoost.

**Robust Weighted Square Error (RWSE) in Logitboost**

Li Li (2010b) rederived Eqn. 1.12 and proposed a stable version of WSE for LogitBoost, which is shown below

\[
RWSE(v) = \frac{\left(\sum_{d \in D} w(d) r(d)\right)^2}{\sum_{d \in D} w(d)} - \left[ \frac{\left(\sum_{d \in D_t} w(d) r(d)\right)^2}{\sum_{d \in D_t} w(d)} + \frac{\sum_{d \in D_c} w(d) r(d)\right)^2}{\sum_{d \in D_c} w(d)} \right] 
\]  

(1.14)

Since the denominators in Eqn. 1.14 are summation of a set of weights \( w(\cdot) \), which are less likely to be close to zero in practical applications, and RWSE is hence more stable than WSE.

After fitting a regression tree by either SE or (R)WSE, the data aggregated in the same leaf is assigned with a value by weighted averaging responses (Eqn. 1.13).

Li mentioned, Eqn. 1.13 could be interpreted as a weighted average in (R)WSE; while in gradient boosting, it is interpreted as a one-step Newton update. It looks like a coincidence. In next section, we propose a unified splitting principle, which not only
clearly explains the relationship of these principles, but also could be extended to more complex loss functions. Also, our method generates Li’s robust version directly.

1.2 Our Work

1.2.1 Plackett-Luce Model for Learning-to-Rank Task

Our work utilizes an elegant list-wise surrogate function called Plackett-Luce (PL) loss, which was first proposed in 1975 Plackett (1975) for horse gambling. Cao et al. Cao et al. (2007) introduce it to the learning to rank task by using it to model the probabilistic distribution of a set of documents given a query, where the training is conducted by minimizing the KL distance between the probability distribution for the ranking model and that for the ground truth. Later Xia et al. Xia et al. (2009, 2008) provide a model called ListMLE, which instead maximizes the likelihood of ground-truth permutations defined in the PL loss. ListMLE could be viewed as a general framework to utilize linear and non-linear features, however, as its non-linear system has not been developed, we refer to $l$-ListMLE as its linear version hereafter. Because public large-scale datasets were not available until 2010, many properties of the PL loss are not revealed in $l$-ListMLE. Even though $l$-ListMLE performs pretty well on some datasets, it is rather unstable in many other cases, especially when compared with direct optimization based models, e.g. DirectRank Tan et al. (2013a) and LambdaRank Quoc and Le (2007). Although not necessarily the best, DirectRank and LambdaRank often show reasonable good performance, while $l$-ListMLE under some circumstances performs far more poorly than average performance.
For example, on the Microsoft 30K data, the largest publicly available real world dataset, \(l\)-ListMLE is approximately 7.6 points worse than the coordinate ascent based method Metzler and Croft (2007) in terms of NDCG scores. Although, Xia et al. Xia et al. (2009) further proved the PL loss is consistent with NDCG@K under certain assumptions, it is not guaranteed to achieve a reasonable performance on practical applications that use data sets with limited size, and the unstable performance behavior greatly limits wide spread real-world applications for the ListMLE model.

Understanding why the PL loss fails in some datasets is important to design more effective algorithms, thus we conduct experiments to analyze these datasets, and figure out one principle as the condition for the PL loss, which states that as compared to average document number per query, the number of features should be large enough. Therefore in order to gain better performance, we have to use more features for PL loss. There are several ways to enrich features of datasets: kernel mapping, neural network mapping, and gradient boosting. We select the gradient boosting with decision trees as weak rankers in this work due to the convenient comparison with LambdaMART, and leave the others for further work. A merit of the PL loss is its concise formula to compute functional gradients, Eqn. (2.11), which results in our ranking system, called PLRank.

As suggested in Chapelle and Chang (2011), real-world datasets are closer to the scenario of search engine applications and have much smaller fluctuations in terms of performance. We conduct experiments on two publicly released real-world datasets. As far as we know, these datasets are larger than any used in previous research papers, except Wu et al. (2010) \(^2\). To compare with other list-wise based methods, we also extend three

\(^2\)They adopted a larger but proprietary one
extra consistent list-wise surrogate functions in Ravikumar et al. (2011) in the gradient boosting framework. We find that PLRank not only maintains the merits of the PL loss, but also greatly alleviates the instability problem of $l$-ListMLE. PLRank has the same time complexity with LambdaMART, and is $M$ times as fast as McRank $^3$.

1.2.2 Analysis of Regression Tree Fitting Algorithms in Learning to Rank

Top practical learning to ranking systems are adopting gradient boosting framework and using regression trees as weak learners. These systems performed much better than linear systems on real-world datasets such as in Yahoo challenge 2010 Chapelle and Chang (2011), and another real-world dataset Microsoft 30K Tan et al. (2013a). Among these systems, LambdaMART Wu et al. (2010); Burges et al. (2011), a pair-wise based model, gained an excellent reputation in Yahoo challenge; MART Friedman (2001), a point-wise based, utilizes least square loss as objective loss function, and McRank Li et al. (2007)$^4$, a point-wise based, uses multi-class classification technique and converts predictions into ranking. For industry applications, gradient boosting combined with regression trees appears to be a standard practice.

An important finding was made by Cossock et al. Li et al. (2007); Cossock and Zhang (2006) that has created a bridge between learning to rank and classification. They proved that an important measure NDCG in learning to rank is bounded by multi-class classification error. This leads to McRank system. This insight opens a door for learning

$^3$ $M$ is the number of different relevance scores in measuring a document.

$^4$ Li et al. call the model of McRank as MART in the scenario of classification.
to rank, as we could borrow state-of-the-art techniques from those developed for multi-class classification.

In multi-class classification area, there is a work that fits a regression tree using weighted least square error (WSE) principle. It uses both first- and second-order information, not only first-order like SE in gradient boosting. LogitBoost Friedman et al. (2000) and its robust versions Li (2010a,b) are examples of such applications. A comparison between gradient boosting using SE, and LogitBoost using WSE for classification task Friedman (2001) shows that the latter is slightly better. As WSE is empirically considered as unstable in practice Friedman et al. (2000); Li (2010b); Friedman (2001), Li et al. Li (2010b) obtained a stable form of WSE, called RWSE.

However, both WSE and RWSE are somewhat hard to understand and have no clear theoretical explanation. Li et al. thus proposed an interesting question in Section 2.3 of Li (2010b): in determining the output of a leaf of a regression tree, the one-step Newton formula from gradient boosting coincides with the weighted averaging from WSE. Moreover,
RWSE looks pretty concise, which might be considered to be applicable for any ranking models in addition to LogitBoost and its variants. These issues drive us to consider from another point of view.

We propose a general regression tree fitting principle for ranking models, called least objective loss based error (OLE). It only requires simple computation to derive exact formula and is easy-to-understand. Under this principle, besides clearly answering the aforementioned question, we analyze a variety of ranking systems to build a relationship between SE, (R)WSE and OLE, which is shown in Figure 1.1. Experiments in real-world datasets show OLE improves most of results over SE.

1.2.3 A Simple Discriminative Training Method for Machine Translation with Large-Scale Features

Since Och Och (2003) proposed minimum error rate training (MERT) to exactly optimize objective evaluation measures, MERT has become a standard model tuning technique in statistical machine translation (SMT). Though MERT performs better by improving its searching algorithm Macherey et al. (2008); Cer et al. (2008); Galley and Quirk (2011); Moore and Quirk (2008), it does not work reasonably when there are lots of features. As a result, margin infused relaxed algorithms (MIRA) dominate in this case McDonald et al. (2005); Watanabe et al. (2007); Chiang et al. (2008); Tan et al. (2013b); Cherry and Foster (2012).

In SMT, MIRAs consider margin losses related to sentence-level BLEUs. However,

---

5The regularized MERT seems promising from Galley et al. Galley et al. (2013) at the cost of model complexity.
since the BLEU is not decomposable into each sentence, these MIRA algorithms use some heuristics to compute the exact losses, e.g., pseudo-document Chiang et al. (2008), and document-level loss Tan et al. (2013b).

Recently, another successful work in large-scale feature tuning include force decoding based Yu et al. (2013), classification based Hopkins and May (2011a).

We aim to provide a simpler tuning method for large-scale features than MIRAs. Out motivation derives from an observation on MERT. As MERT considers the quality of only top1 hypothesis set, there might have more-than-one set of parameters, which have similar top1 performances in tuning, but have very different topN hypotheses. Empirically, we expect an ideal model to benefit the total N-best list. That is, better hypotheses should be assigned with higher ranks, and this might decrease the error risk of top1 result on unseen data.

Plackett (1975) offered an easy-to-understand theory of modeling a permutation. An N-best list is generated by sampling without replacement. The $i$th hypothesis to sample relies on those ranked after it, instead of on the whole list. This model also supports a partial permutation which accounts for top $k$ positions in a list, regardless of the remaining. When taking $k$ as 1, this model reduces to a standard conditional probabilistic training, whose dual problem is actual the maximum entropy based Och and Ney (2002). Although Och Och (2003) substituted direct error optimization for a maximum entropy based training, probabilistic models correlate with BLEU well when features are rich enough. The similar claim also appears in Zhu and Hastie (2001). This also make the new method be applicable in large-scale features.
Plackett-Luce Model for Learning-to-Rank Task

2.1 Plackett-Luce Loss for Learning to Rank

The Plackett-Luce model was first proposed by Plackett [Plackett (1975)] to predict the ranks of horses in gambling. Consider a horse racing game with five horses. Suppose a probability distribution $P$ on their abilities to win a race, then a rank of these horses can be understood as a generative procedure. Suppose we want to know the probability of a top3 rank $2, 3, 5$. The result can be computed as follows:

Being the champion for the 2nd horse, the probability is $p_2$ among five candidates. Being the runner-up for the 3rd horse, the probability $p_3$ has to be normalized among the remaining four horses, which leads to $p_3/(p_1 + p_3 + p_4 + p_5)$. Being the third winner for the 5th horse, its probability among the remaining three horses becomes $p_5/(p_1 + p_4 + p_5)$. So the probability of the rank $2, 3, 5$ is their product. It is not difficult to see that the most likely rank is all horses are ranked by their winning probability in a descending order.
The key idea for the Plackett-Luce model is the choice in the $i$th position in a rank $\pi$ only depends on the candidates not chosen at previous positions.

### 2.1.1 Plackett-Luce Loss with Linear Features

In learning to rank, each training sample has been labeled with a relevance score, so the ground-truth permutation of documents related to the $i$th query can be easily obtained and denoted as $\pi_i$, where $\pi_i(j)$ denotes the index of the document in the $j$th position of the ground-truth permutation. We note that $\pi_i$ is not obligatory to be a full rank, as we may only care about the top $K$ documents.

Consider a ranking function with linear features, the probability of a set of candidate relevant documents $D_i$ associated with a query $q_i$ is defined as

$$
p(d_i^e) = \frac{\exp\{h(d_i^e)^T \cdot w\}}{\sum_{d \in D_i} \exp\{h(d)^T \cdot w\}}
$$

The probability of the Plackett-Luce model to generate a rank $\pi_i$ is given as

$$
p(\pi_i, w) = \prod_{j=1}^{\pi_i} p(d_{\pi_i(j)}^i | C_{i,j})
$$

$$
p(d_i^e | C_{i,j}) = \frac{p(d_i^e)}{\sum_{d \in C_{i,j}} p(d)}
$$

where $C_{i,j} = D_i - \{d_{\pi_i(1)}^i, \ldots, d_{\pi_i(j-1)}^i\}$.

The training objective is to maximize the log-likelihood of all expected ranks over all queries and retrieved documents with corresponding ranks in the training data with a
zero-mean and unit-variance Gaussian prior parameterized by \( w \).

\[
\mathcal{L} = \log \prod_i p(\pi_i, w) - \frac{1}{2} w^T w \quad (2.4)
\]

The gradient can be calculated as follows,

\[
\frac{\partial \mathcal{L}}{\partial w} = \sum_i \sum_j \{ h(d_i^{(j)}) - \sum_{d \in C_{i,j}} (h(d) \cdot p(d|C_{i,j})) \} - w
\]

Since the log-likelihood function is smooth, differentiable, and concave with the weight vector \( w \), global optimum guarantee is satisfied.

### 2.1.2 Plackett-Luce Loss with Regression Trees

In this paper, we build ensemble regression trees for the Plackett-Luce loss in the gradient boosting framework, Alg. 1 summarizes the main procedure. We first describe how to compute the pseudo response and output value for fitting a regression tree, and then we provide more analysis for this new model.

At the \( t \)th iteration, all fitted regression trees constitute the current predictor \( f_t(\cdot) \), and the Eqn. (2.1) can be rewritten as

\[
p(d_i^e) = \frac{\exp\{f_t(d_i^e)\}}{\sum_{k=1}^{[D_i]} \exp\{f_t(d_i^k)\}} \quad (2.5)
\]

We limit \(|\pi| = K\), and adopt Eqn. (2.4) without a normalization as our objective \(^1\).

\(^1\)The model complexity of regression trees is often controlled by the learning rate \( \alpha \), different from the
Plugging Eqn. (2.5) into Eqn. (2.4), and taking derivative with respect to $f_t(\cdot)$, we obtain

$$L'(f_t(d)) = I(d \in \text{topK ground-truth}) - \sum_{C \text{ s.t. } d \in C} p(d|C)$$

(2.6)

where $I(\cdot)$ denotes the indicator function. When $I(\cdot)$ returns 0 for the current document, the size of $\{C\}$ equals $K$, otherwise it is smaller.

We follow Eqn. (1.7) to fit a regression tree $g_t(\cdot)$. Denotes the documents falling in the leaf $U$ as $U_d$. We set the output of the leaf $U$ as $g_t(d \in U_d) = -v$, and $v$ is optimized independently from other leaves. Following Eqn. (1.5), we construct $f_{t+1}(\cdot)$ for documents in $U_d$.

We adjust $v$ to maximize the log-likelihood $L$. Thus $L$ has been reinterpreted as a function of $v$. We rewrite Eqn. (2.5) as

$$p(d^k_e) = \frac{\exp\{f_t(d^k_e) - I(d^k_e \in U_d) \cdot \alpha v\}}{\sum_{k=1}^{[D]} \exp\{f_t(d^k_e) - I(d^k_e \in U_d) \cdot \alpha v\}}$$

(2.7)

By the Newton method, we have

$$v = \frac{L'(v = 0)}{L''(v = 0)}$$

(2.8)

$$L'(v = 0) = \sum_{d \in U_d} L'(f_t(d))$$

(2.9)

$$L''(v = 0) = \sum_C p'_j \cdot (p'_j - 1)$$

(2.10)

$$p'_j = \sum_{d \in U_d \cap C} p(d|C)$$

(2.11)

normalization factor used in a linear model.
Algorithm 1 PLRank

Require: Documents $D = \{D_1, D_2, \ldots\}$; $K$ defines top $K$ documents of a ground-truth rank; $T$ defines regression tree number; $L$ defines leaf number; $\alpha$ defines learning rate.

1: $f_1(\cdot) \leftarrow$ BackGroundModel(\cdot) \triangleright Initialization for model adaptation. None by default.
2: for $D_i$ in $D$ do
3: Randomly shuffle $D_i$
4: Sort $D_i$ by relevances. \triangleright We could build several ground-truth permutations.
5: end for
6: for $t = 1$ to $T$ do
7: $\text{Resp}(d \in \bigcup D_i) \leftarrow -\mathcal{L}'(f_t(d))$ \triangleright Compute pseudo response following Equ. 2.6.
8: Fit a $L$-leaf tree $g_t$ on $\text{Resp}$. \triangleright By Eqn. (1.7) by default.
9: for leaf $U$ in $g_t$ do
10: $v \leftarrow \mathcal{L}'(v = 0)/\mathcal{L}''(v = 0)$ \triangleright Set output of current leaf by Eqn. (2.11)
11: $g_t(d \in U_d) \leftarrow -v$
12: end for
13: $f_{t+1} \leftarrow f_t + \alpha g_t$ \triangleright Eqn. (1.5)
14: end for
return $f_{T+1}$

To clarify this procedure, we take one query with four related documents as an example. Suppose the four documents $d_1, d_2, d_3, d_4$ are sorted in a descending order with their relevance scores. In an other word, the ground-truth permutation is $d_1, d_2, d_3, d_4$. Let their scores after some iterations, from current predictor $f_t(\cdot)$, be $s_1, s_2, s_3, s_4$ respectively for abbreviation. Considering the top 2 documents of the ground-truth permutation, the log-likelihood is

$$
\mathcal{L} = s_1 - \log\{\exp s_1 + \exp s_2 + \exp s_3 + \exp s_4\} \\
+ s_2 - \log\{\exp s_2 + \exp s_3 + \exp s_4\}
$$
Taking derivatives with respect to their scores, we obtain

\[ L'(s_1) = 1 - p(s_1|s_1, s_2, s_3, s_4) \]
\[ L'(s_2) = 1 - p(s_2|s_1, s_2, s_3, s_4) - p(s_2|s_2, s_3, s_4) \]
\[ L'(s_3) = 0 - p(s_3|s_1, s_2, s_3, s_4) - p(s_3|s_2, s_3, s_4) \]
\[ L'(s_4) = 0 - p(s_4|s_1, s_2, s_3, s_4) - p(s_4|s_2, s_3, s_4) \]

In this toy example, the samples \( s_3, s_4 \) have \( K = 2 \) contextual probabilities.

Suppose \( s_1, s_3 \) fall into the same leaf of a regression tree, then

\[ L'(v = 0) = 1 - p(s_1|C_1) + 0 - \{p(s_3|C_1) + p(s_3|C_2)\} \]
\[ L''(v = 0) = (p(s_1|C_1) + p(s_3|C_1)) \cdot (p(s_1|C_1) \]
\[ + \ p(s_3|C_1) - 1) + p(s_3|C_2) \cdot (p(s_3|C_2) - 1) \]

where \( C_1 = \{s_1, s_2, s_3, s_4\}, \ C_2 = \{s_2, s_3, s_4\} \).

In the following, we describe more details of Alg. 1 that relate to initialization of models (line 1), selection of ground-truth permutation (line 3-4).

**Initialization of Models**

As a statistical model is sensitive to data genres, a trivial yet effective way is to use more data for training. In some applications, data in the objective genre may be sparse, while general genre data is plenty, then it is useful to explore model adaptation. Borrowing the idea from adaptive LambdaMART Wu et al. (2010), our model could also first train a background model on plenty of general genre data. Then we assign the resulting model to
initialize our Alg 1 (line 1), and continue to train our model using on objective genre data. In this paper, we are not focusing on the adaptation experiments, and we initialize to zero.

Selection of Ground-Truth Permutations

In learning to rank, as the relevance scores are scattered among limited integers, e.g., 0 to 10 inclusively, there are many ties in the scores, this would impact the determination of ideal permutations and our training objective. We analyze and compare three strategies on toy documents $d_1, d_2, d_3, d_4$ with relevance scores $4, 0, 4, 4$, and consider top 4 ground-truth documents.

The first one is a stable sort compatible with input. If two documents are equal in relevance scores, their relative rank in the ground-truth permutation should comply with the original input. Hence, the ground truth permutation in the first strategy should be $d_1, d_3, d_4, d_2$.

The second one is random selection. This strategy has been straightforwardly implemented in Alg. 1 by firstly randomly shuffling input data and then invoking the first strategy. This strategy is also the default setting in Xia et al. Xia et al. (2009, 2008).

The third one considers multiple ground-truth permutations (looping lines 2-5 in Alg. 1). As the number of all permutation possibilities is huge, we randomly select several ground-truth ranks and store them compactly in terms of data structure. For instance, the ground truth permutation $d_1, d_3, d_4, d_2$ consists of three contextual terms, $C_1 = \{d_1, d_2, d_3, d_4\}$, $C_2 = \{d_2, d_3, d_4\}$, $C_3 = \{d_2, d_4\}$, while adding a second permutation $d_1, d_4, d_3, d_2$ leads to merely one extra term $C_4 = \{d_2, d_3\}$, rather than new three terms. The statistics about this issue on Yahoo 2010 and Microsoft 30k datasets are re-
ported in Table 2.4. We use PLRank(obj=\textit{num}) to denote different number of objectives.

**Greedy Construction of Regression Tree**

Since all tree fitting algorithms are conducted node by node, we only need to focus on the smallest sub-problem: given some features $f$ and pseudo-response $Resp(d)$ of each document $d$, how do we select the best threshold to optimize the objective? Suppose documents are sorted in an ascending order, denoted as $d_1, \ldots, d_N$, then there are $N - 1$ positions to define the threshold. Eqn. (1.7) defines such an optimal choice under least square loss.

$$I^* = \arg\min_I \sum_{i \leq I} (Resp(d_i) - \bar{r}_1)^2 + \sum_{I < j} (Resp(d_j) - \bar{r}_2)^2$$  \quad (2.12)

where $\bar{r}_1$ is the average pseudo-response of documents in the left part, and $\bar{r}_2$ is the average in the right part. If the resulting two nodes are leaf nodes, then $\bar{r}_1, \bar{r}_2$ are set as the output of leaves respectively.

Currently almost all tree based systems in learning to rank follow Eqn. (1.7) to fit a regression tree, which only uses the first order gradient of the least square loss rather than directly minimizing surrogate loss. In next section, we would introduce an effective construction algorithm which utilizes the second-order gradient information, and many systems such as McRank and our proposed PRRank could be further improved.

Theorem 8 suggests that the classic tree fitting algorithm in gradient boosting is very suitable for the least-square loss function, on which MART system is based, and this could explain why MART system actually performs extremely well in our experiments.
By default, in our experiments, all systems use the classic tree fitting algorithm to explore the differences resulting from loss functions only.

**Stable Gradients**

In the implementations of ranking systems, we often face the following numerical computation problem that leads to unstable performance: when the second derivative is small, the Newton iteration returns an abnormally huge quantity. Li et al. Li (2010b) explored a method to somewhat overcome this problem. Here we adopt another simple yet efficient tactic, exerting a small variation (line 12).

As the optimization objective is concave with respect to the output $v$ of a leaf, the second derivative is guaranteed to be negative by optimization theory Bazaraa et al. (2006). If we set $\beta$ to be a positive number, it forces $G$ to have an at least $\beta$ margin from zero. Thus we simply set $\beta$ as 1 which works well in practice.

We find that there is another explanation for this $\beta$, which can be viewed as an effect of $l_2$ norm of $\alpha$ in Eqn. (2.4). The first derivative of Eqn. (2.4) in $\alpha = 0$ equals $L'(v = 0)$ in Eqn. (2.11); its second derivative equals $L''(v = 0)$ in Eqn. (??) minus $\alpha$. This tactic doesn’t influence system performance as $L''(v = 0)$ is constantly much greater than $\beta$ in large dataset.

### 2.1.3 Training with Plackett-Luce Loss

Regarding linear features, Xia et al. Xia et al. (2009, 2008) adopt a neural network to maximize the log-likelihood of expected ranks. The neural network works well in small
datasets, e.g. LETOR, while it also requires suitable settings on hidden layer structure and the number of hidden neurons.

As our experiments are conducted on real-world datasets, we instead use L-BFGS Byrd et al. (1995) for parameter tuning to gain faster convergence speed. It is observed that overfitting often occurs in small data sets, while in large datasets the log-likelihood correlates with ranking measures very well.

Regarding non-linear features, kernel technique could map them into a linear form in a high dimensional space, and then the neural network based training in Xia et al.’s work or LBFGS are applicable, provided that the new dimension is acceptable in practice. However, in the case of regression trees, it is impractical to expand all dimensions, which is why we propose our new algorithm. We are following the boosting framework, which iteratively fits high-quality decision trees, to maximize the objective log-likelihood.

2.1.4 Transformation between PL and McRank

McRank is a multi-class classification based ranking system first proposed by Li et al. Li et al. (2007). They proved that the unnormalized NDCG is bounded by classification errors, it is also well known that the log-loss is an upper bound of classification error, thus maximizing the log-likelihood is nothing but minimizing an upper bound of the unnormalized NDCG. As shown in Li et al. (2007), the ordinal classification version performs quite closely to its classification version; we use the latter hereafter.

We discover an interesting relationship between the PL model and the McRank model after a suitable transformation of datasets. Being a point-wise based method, McRank deals with data in a form of \((H, r)\), where \(H\) is a feature vector of a document, and \(r\) is
its relevance score which is used as its category. Suppose the maximum relevance in the
dataset is $T$, then in the view of McRank, the ranking problem is actually a $(T + 1)$-class
classification.

We now further transform this problem as the following: Each original document
$(H, r)$ extends the feature dimension by $T$ times, filling with zero, and rewrites its new
relevance as 1.

$$
\begin{align*}
&|H| \cdot r \\
&|H| \cdot (T - r) \\
&((0, \ldots, 0, H, 0, \ldots, 0), 1)
\end{align*}
$$

Insert extra $T$ pseudo-documents for each original document with the feature dimension
as $(T + 1) \cdot |H|$, and rewrite their new relevances as 0. The pseudo-documents are then
defined as

$$
\begin{align*}
&|H| \cdot r' \\
&|H| \cdot (T - r') \\
&((0, \ldots, 0, H, 0, \ldots, 0), 0)
\end{align*}
$$

where $r' \in [0, T]$, and $r' \neq r$.

In the resulting dataset, a new pseudo-query is created to relate to $T + 1$ documents,
and actually exclusively relates to one document of the original data. Among these $T + 1$
documents, only one is redefined as relevant, with others as irrelevant to the pseudo-query.
We invoke our PL ranker, setting top 1 document of a ground truth permutation as the
optimization objective to solve the McRank model equivalently.

The discussion above only provides a theoretical analysis of these two ranking mod-
els, and in practice we do not adopt this transformation since it expands the dataset by
$(T + 1)^2$ times in term of feature dimension and document number. Moreover the McRank
model specifies an additional sum-to-one restriction, even though the two methods indeed
utilize the same optimization objective.
In other areas such as natural language processing, many problems can be formulated as a ranking problem, while the definition of relevance scores is often no longer a discrete integer as in learning to rank. In this case, the McRank model may be not applicable, and the PL model is more promising instead.

2.1.5 Comparison with Other Consistent List-wise Methods

Calauzenes et al. Calauzènes et al. (2012) have proved that no consistent surrogate function exists for ERR and MAP. However, regarding NDCG, Xia et al. Xia et al. (2009) proved that the ListMLE model is consistent with NDCG@K. They also modified two other losses, cosine and KL divergence, to make them NDCG@K consistent. As Xia et al. have compared them in their work, we thus compare the PL loss with three other consistent versions proposed in Ravikumar et al. (2011), squared loss, cosine, and KL divergence, which were proved to be consistent with the whole list, in the case of boosted trees.

We pay special attention to the first one since it has three different implementations. Let \( s \) denote a score vector of all documents, \( r \) denote the corresponding relevance vector, and \( G(r) = 2^r - 1 \). The consistent and inconsistent equations in terms of square loss in Ravikumar et al. (2011) are

\[
\phi_{sq}^{\text{consistent}}(s, r) = \left\| s - \frac{G(r)}{\| G(r) \|_D} \right\|_2^2 \tag{2.13}
\]

and

\[
\phi_{sq}^{\text{inconsistent}}(s, r) = \left\| s - G(r) \right\|_2^2 \tag{2.14}
\]
where the norm $\| \cdot \|_D$ defines the DCG value of a ground-truth permutation per query.

A third equation in *Cossock and Zhang (2006)* is also inconsistent with NDCG.

$$
\phi_{sq}^{\text{inconsistent}}(s, r) = \| s - r \|_2^2
$$

All boosting systems with the least-squares loss are called MART in this paper. The two inconsistent versions are point-wise based, and the consistent one is list-wise based since the norm $\| \cdot \|_D$ is operated by query. We remove detailed discussion about the functional gradients for all surrogates above due to space limitation.

### 2.2 Experiments

We studied the performance of the proposed algorithm in two real world datasets, Yahoo challenge 2010 and Microsoft 30K. We implemented 9 baseline ranking systems in C++, which use boosted trees as features. System 1 is LambdaMART. System 2 is McRank. System 3 is MART-1 which is the first inconsistent version of MART (Eqn. (2.14)). System 4 is MART-2 which is the second inconsistent version of MART (Eqn. 2.15). System 5 is c-MART-1 which is a consistent version of MART-1 (Eqn. (2.13)). System 6 is CosMART which is an inconsistent version of cosine distance loss with boosted trees. System 7 is c-CosMART which is a consistent version of CosMART. System 8 is KLMART which is a MART using the KL distance. System 9 is c-KLMART which is a consistent version of KLMART.

Moreover, in order to compare tree features and linear features, we add two linear
systems. System 10 is based on a heuristic coordinate ascend (CA) based optimization method proposed by Metzler and Croft (2007) which uses linear features and optimizes NDCG directly. CA is used as a reference system to represent the average performance of linear systems due to its relatively stable and good performances among a variety of linear models in different datasets, including the datasets used in this work, as shown in the experiments of Tan et al. Tan et al. (2013a). This system is akin to the one proposed by Tan et al., but the latter is an exact coordinate ascent optimization ranking method. We also used the experimental results in Tan et al. (2013a) as a reference here. System 11 is $l$-ListMLE that optimizes top 10 retrieved documents.

We set up the same parameters as in Wu et al. (2010) for all systems. The learning rate $\alpha$ is 0.1 (line 15 in Alg. 1). We set the number of decision tree leaves as 30, which is a classic setting. As in real world datasets, McRank requires more iterations to converge, thus we use 2500 boosted trees as a final model, and use 1000 boosted trees for other systems. Regarding to PLRank, as we mainly concentrate on NDCG@10, we set $K$ to 10 to optimize top 10 documents of ground-truth permutations. All results are reported with NDCG@(1,3,10) and ERR scores.

In order to examine the industry-level performance of our system, we search exhaustively parameters to compare to the Yahoo Challenge results Chapelle and Chang (2011) in Table 2.5.

### 2.2.1 Datasets

The LETOR benchmark datasets released in 2007 Qin et al. (2010b) have significantly boosted the development of learning to rank algorithms since researchers could compare
their algorithms on the same datasets for the first time. But unfortunately, the sizes of the datasets in LETOR are several orders of magnitude smaller than the ones used by search engine companies. Several researchers have noticed that the conclusions drawn from experiments based on LETOR datasets are unstable and quite different from the ones based on large real datasets Chapelle and Chang (2011). Thus in this work, we attempt to make stable system comparisons by using as large datasets as possible, and we use two real world datasets, Yahoo challenge 2010 and Microsoft 30K. The statistics oh these three data sets are reported in Table 2.1 which might a bit different from those in Chapelle and Chang (2011) as we only give the statistics of training datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Query</th>
<th>#Doc.</th>
<th>#D./#Q.</th>
<th>#Feat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft 30K</td>
<td>18.9K</td>
<td>2270K</td>
<td>120</td>
<td>136</td>
</tr>
<tr>
<td>Yahoo 2010 (Set 1)</td>
<td>20K</td>
<td>473K</td>
<td>23</td>
<td>519</td>
</tr>
<tr>
<td>McRank Li et al. (2007)</td>
<td>10-26K</td>
<td>474-1741K</td>
<td>18-88</td>
<td>367-619</td>
</tr>
<tr>
<td>LambdaMART Wu et al. (2010)</td>
<td>31K</td>
<td>4154K</td>
<td>134</td>
<td>416</td>
</tr>
<tr>
<td>Ohsumed</td>
<td>106</td>
<td>16K</td>
<td>150</td>
<td>45</td>
</tr>
<tr>
<td>LETOR 4.0</td>
<td>2.4K</td>
<td>85K</td>
<td>34</td>
<td>46</td>
</tr>
</tbody>
</table>

Table 2.1: The top two datasets are used, while the others are as a reference. Ohsumed is of LETOR 3.0. #D./#Q. means average document number per query.

Microsoft 30K is the largest publicly released dataset in terms of the document number. As its official release has provided a standard 5-fold split, we report average results. Regarding the Yahoo dataset, it only provides a 1-fold split. In order to compare to other released systems, we report results on the standard 1-fold split in Table 2.3, and report average results on a randomly generated 3-fold split in Figure 2.3.
<table>
<thead>
<tr>
<th>System</th>
<th>Yahoo 2010 NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
<th>ERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLRank(obj=1)</td>
<td>0.7210</td>
<td>0.7267</td>
<td>0.7885</td>
<td>0.4598</td>
</tr>
<tr>
<td>PLRank(obj=3)</td>
<td>0.7228</td>
<td>0.7290</td>
<td>0.7895</td>
<td>0.4611</td>
</tr>
<tr>
<td>PLRank(obj=6)</td>
<td><strong>0.7240</strong></td>
<td><strong>0.7295</strong></td>
<td><strong>0.7902</strong></td>
<td>0.4609</td>
</tr>
<tr>
<td>PLRank(obj=9)</td>
<td><strong>0.7239</strong></td>
<td><strong>0.7298</strong></td>
<td><strong>0.7903</strong></td>
<td><strong>0.4610</strong></td>
</tr>
<tr>
<td>PLRank(obj=15)</td>
<td>0.7205</td>
<td>0.7291</td>
<td>0.7896</td>
<td>0.4601</td>
</tr>
<tr>
<td>LambdaMART</td>
<td>0.7160</td>
<td>0.7187</td>
<td>0.7809</td>
<td>0.4589</td>
</tr>
<tr>
<td>McRank</td>
<td>0.7213</td>
<td>0.7257</td>
<td>0.7871</td>
<td>0.4586</td>
</tr>
<tr>
<td>MART-1</td>
<td>0.7112</td>
<td>0.7211</td>
<td>0.7831</td>
<td>0.456</td>
</tr>
<tr>
<td>MART-2</td>
<td>0.7166</td>
<td>0.7230</td>
<td>0.7858</td>
<td>0.4586</td>
</tr>
<tr>
<td>MART ²</td>
<td>-</td>
<td>-</td>
<td>0.782-0.789</td>
<td>0.458-0.461</td>
</tr>
<tr>
<td>c-MART-1</td>
<td>0.7123</td>
<td>0.7221</td>
<td>0.784</td>
<td>0.454</td>
</tr>
<tr>
<td>CosMART</td>
<td>0.6979</td>
<td>0.6967</td>
<td>0.7638</td>
<td>0.4521</td>
</tr>
<tr>
<td>c-CosMART</td>
<td>0.6981</td>
<td>0.7100</td>
<td>0.7669</td>
<td>0.4510</td>
</tr>
<tr>
<td>KLMART</td>
<td>0.7012</td>
<td>0.7111</td>
<td>0.7710</td>
<td>0.4520</td>
</tr>
<tr>
<td>c-KLMART</td>
<td>0.7020</td>
<td>0.7120</td>
<td>0.7732</td>
<td>0.4525</td>
</tr>
<tr>
<td>CA</td>
<td>0.6933</td>
<td>0.6879</td>
<td>0.7549</td>
<td>0.444</td>
</tr>
<tr>
<td>l-ListMLE</td>
<td>0.7017</td>
<td>0.7014</td>
<td>0.7673</td>
<td>0.4520</td>
</tr>
</tbody>
</table>

Table 2.2: Main results on the Yahoo Challenge dataset. Results on the standard five splits of Microsoft data are averaged, and we follow the standard one split for the Yahoo data to compare to published results. System 1 is trained towards optimizing NDCG. System 10, CA, is a Coordinate Ascent based method directly maximizing NDCG. We provide results marked as “ref” reported in other papers. LambdaMART-Aug70 is trained on resampled training data, where our experiments are conducted on full training data. As PLRank(obj=6) in Microsoft data starts to decrease, we did not test more objectives. CosMART got an abnormally low score in Microsoft data as l-ListMLE, thus it is less meaningful to list them.
<table>
<thead>
<tr>
<th>System</th>
<th>Microsoft 30K</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NDCG@1  @3 @10 ERR</td>
</tr>
<tr>
<td>PLRank(obj=1)</td>
<td>0.4947 0.4814 0.5045 0.3770</td>
</tr>
<tr>
<td>PLRank(obj=3)</td>
<td>0.4967 <strong>0.4835</strong> 0.5063 <strong>0.3781</strong></td>
</tr>
<tr>
<td>PLRank(obj=6)</td>
<td>0.4949 0.4828 <strong>0.5069</strong> 0.3778</td>
</tr>
<tr>
<td>PLRank(obj=9)</td>
<td>- - - -</td>
</tr>
<tr>
<td>PLRank(obj=15)</td>
<td>- - - -</td>
</tr>
<tr>
<td>LambdaMART</td>
<td>0.4942 0.4793 0.4995 <strong>0.3774</strong></td>
</tr>
<tr>
<td>McRank</td>
<td>0.4913 0.4815 <strong>0.5057</strong> 0.3735</td>
</tr>
<tr>
<td>MART-1</td>
<td>0.4856 0.4734 0.4985 0.3769</td>
</tr>
<tr>
<td>MART-2</td>
<td>0.4924 0.4788 0.5021 0.3736</td>
</tr>
<tr>
<td>ref MART in Tyree et al. (2011)</td>
<td>- - - -</td>
</tr>
<tr>
<td>c-MART-1</td>
<td>0.4860 0.4730 0.4990 0.3750</td>
</tr>
<tr>
<td>CosMART</td>
<td>- - - -</td>
</tr>
<tr>
<td>c-CosMART</td>
<td>- - - -</td>
</tr>
<tr>
<td>KLMART</td>
<td>- - - -</td>
</tr>
<tr>
<td>c-KLMART</td>
<td>- - - -</td>
</tr>
<tr>
<td>CA</td>
<td>0.4596 0.4366 0.4597 0.3401</td>
</tr>
<tr>
<td>l-ListMLE</td>
<td>0.3838 0.3880 0.4230 0.3234</td>
</tr>
</tbody>
</table>

Table 2.3: Main results on the Microsoft dataset. Results on the standard five splits of Microsoft data are averaged, and we follow the standard one split for the Yahoo data to compare to published results. System 1 is trained towards optimizing NDCG. System 10, CA, is a Coordinate Ascent based method directly maximizing NDCG. We provide results marked as “ref” reported in other papers. LambdaMART-Aug70 is trained on re-sampled training data, where our experiments are conducted on full training data. As PLRank(obj=6) in Microsoft data starts to decrease, we did not test more objectives. CosMART got an abnormally low score in Microsoft data as l-ListMLE, thus it is less meaningful to list them.
2.2.2 \( l \)-ListMLE vs. Other Linear Systems

We first examine the performance of \( l \)-ListMLE (System-11) compared to another linear system CA (System-10). Their results are shown in Table 2.3 and Figure 2.1. \( l \)-ListMLE obtains 0.7673 in NDCG@10 in the Yahoo 2010 dataset after 100 iterations of quasi-Newton optimization, but performs unsatisfactorily in Microsoft 30K even after 1000 iterations, approximately 8 percent lower in NDCG@1, and several percent lower in other measures. Tan et al. Tan et al. (2013a) also compared several linear systems in these two datasets, except \( l \)-ListMLE. Our implementation of \( l \)-ListMLE outperforms their best result 0.760 from DirectRank in the Yahoo datasets, while performs significant worse in the Microsoft 30K.

![Chart 8: Yahoo challenge 2010](image)

![Chart 9: Microsoft 30K](image)

Figure 2.1: The performance of \( l \)-ListMLE and the selected linear reference system Coordinate Ascent (CA) on Yahoo data (left) and Microsoft 30K (right). CA is capable of representing the mainstream linear systems on these datasets Tan et al. (2013a).

The unexpectedly bad performance of ListMLE in the larger dataset contradicts the proof from Xia et al. (2009), that is ListMLE is consistent with NDCG. In another words, ListMLE theoretically should perform better with more available data. The main reason
may be that the features on Microsoft 30K is not rich enough to ensure the consistency of ListMLE. To verify this, we notice that the features of Yahoo 2010 data set are richer than Microsoft 30k, thus we conduct experiments on Yahoo 2010 dataset by adjusting the number of features and compare the performance of $l$-ListMLE and CA. The results are shown in Figure 2.2. Since the features might not be independent to each other, the NDCG performance curves are not monotonic with the size of features number. However both figures have their own critical points, 200 for NDCG@1 and 100 for NDCG@10: When the feature number is beyond this point, $l$-ListMLE beats CA, otherwise it performs worse than CA.

![Figure 2.2: $l$-ListMLE and CA with different number of features.](image)

To improve the performance of $l$-ListMLE, instead of using a linear feature model, we need to increase the model capacity that have more expressive power. Thus we decide to use decision trees as our basic weak learners, and we grown our model through gradient boosting that maximize the likelihood of ground-truth ranks. The PL loss is not the only one that is consistent with NDCG, there are other three models proposed in Ravikumar et al. (2011) that are also consistent with it, so we extend these three models to boosted
trees versions for a full comparison.

2.2.3 Different Number of Ground Truth Permutations

<table>
<thead>
<tr>
<th>#Obj</th>
<th>Yahoo</th>
<th>Microsoft</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0.8672</td>
<td>0.8986</td>
</tr>
<tr>
<td>6</td>
<td>0.7586</td>
<td>0.8069</td>
</tr>
<tr>
<td>9</td>
<td>0.6965</td>
<td>0.7609</td>
</tr>
<tr>
<td>12</td>
<td>0.6524</td>
<td>0.7277</td>
</tr>
<tr>
<td>15</td>
<td>0.6197</td>
<td>0.7026</td>
</tr>
<tr>
<td>18</td>
<td>0.5926</td>
<td>0.6823</td>
</tr>
<tr>
<td>21</td>
<td>0.5708</td>
<td>0.6661</td>
</tr>
</tbody>
</table>

Table 2.4: Compression ratio of denominator terms in Eqn. (2.3) in considering multiple ground-truth permutations.

In Section 2.1.2, we introduce three strategies in selecting ground-truth permutations as model objective. The first one works well in Yahoo dataset, while it fails in Microsoft dataset: approximately 3 points in NDCG lower. The second one in PLRank(obj=1) works satisfactorily in terms of running time and performance. With respective to the third one, we locate an optimal setting to balance the running time and performance.

We empirically search an optimal setting to balance the running time and performance. Table 2.4 displays actual compression ratio. For example, when objective number is 9, actual number of terms in computing the functional gradient is 69.6 percent of that without compressed storage, and this is equivalent to 6.27 objectives. From the results in
Tables 2.3, 2.6, 2.4, we recommend to use PL(obj=3) in practice to gain stable improvements with acceptable extra training time.

2.2.4 PLRank vs. \(l\)-ListMLE, MART, McRank and LambdaMART

![Comparison of several tree-based systems.](chart)

Figure 2.3: Comparison of several tree-based systems.
Currently, the state-of-the-art learning to rank systems use boosted trees which have been proved to be more powerful than those using linear features in real world datasets. The champion of Yahoo challenge 2010 is a system that combines approximately 12 models, most of which are trained with LambdaMART Burges et al. (2011). The other two state-of-the-art systems using trees are MART and McRank, one optimizes least-square loss and the other treat the ranking as a multi-class classification.

As shown in Table 2.3, PLRank outperforms l-ListMLE, which is a natural result as PLRank is in a more complex function space than the linear space. However, what surprises us is that, in the Yahoo dataset there are moderate improvements, approximately 2 points in NDCG(@1, 3, 10), while in the Microsoft dataset, there are significant 8 to 10 points in NDCG(@1, 3, 10). On one aspect, boosted trees indeed could capture the dependency between features, and on another aspect, it is especially effective for the PL loss when the features are not rich.

As shown in Figure 2.3 and Table 2.3, the tree-based systems obviously perform well over linear feature systems. Among tree-based systems, PLRank demonstrates some moderate improvements over MART, McRank and LambdaMART in the Yahoo dataset, and in the Microsoft dataset, all tree-based systems perform pretty closely to each other.

McRank and PLRank are more close in six NDCG scores except NDCG@1 in the Microsoft dataset. LambdaMART performs well in ERR, and is significantly better than McRank and MART, and close to PLRank(obj=1). Comparatively, three PLRank variants act more stably. PLRank(obj=1) is always in best two systems on all measures when it is compared with McRank, on the other hand, as shown in Table 1 LambdaMART and MART. PLRank(obj=2) is considered to be the best in balancing the performance and
running time.

Two-tailed t-test results show PLRank(obj=*) systems would have significant improvements over others when their differences are greater than about 0.5 point at 95% confidence. Unfortunately, in Table 2.3, most of the improvements of PLRank(obj=*) are not significant, just matchable to these state-of-the-art systems.

Our MART baseline results are close to those reported in Tyree et al. (2011). Tan et al. Tan et al. (2013a) also used the same datasets to compare LambdaMART and MART, and their baselines are about 1 point lower in NDCG than our reported results. We notice that their baselines are from RankLib, which is written in Java, and DirectRank is implemented in C++. In comparison, our 10 tree-based systems are re-implemented in C++ with an identical code template, thus our systems could be better to reflect differences in models rather than being impacted by coding.

2.2.5 PLRank vs. Other Consistent List-wise Method with Boosted regression trees

The list-wise methods discussed in Section 2.1.5 have better performance than their inconsistent counterparts in Yahoo dataset, although the differences are not that much. In contrast, it is reported in Ravikumar et al. (2011) that for all linear systems, the consistent versions improves NDCG scores of the in-consistent counterparts by several points.

As shown in Table 2.3, these consistent methods, after extended to boosted trees versions, unfortunately, have not show competitive performances when compared with LambdaMART, McRank and PLRank, so we did not run them on the larger Microsoft
LambdaMART is a method that considers NDCG loss in optimization, and McRank optimizes unnormalized NDCG, so we only need to further analyze the surrogate functions of PLRank and the three consistent versions, that are not directly related to NDCG. A plausible explanation is the PL loss is consistent with NDCG@$K$, $K$ taken 10, while those of c-MART-1, c-CosMART, c-KLMART are consistent with NDCG with a whole list. We conjecture that when we let $K$ go to the whole list, these systems would show advantages.

### 2.2.6 Influence of Different Feature Number

We observe that PLRank has relatively better performance in Yahoo data than in Microsoft data, though the gaps are small. As from previous analysis, we known that ListMLE prefers feature-rich data. Its performance can be improved significantly when we increase the model capacity such as decision trees. One question we’d like to investigate is whether the richness of original features would impact performances when we use decision models to increase model capacity. Thus we run PLRank and LambdaMART on Yahoo data by varying the size of features, but we couldn’t observe similar phenomenon as reported in Figure 2.2, except that when the size of feature is smalled than 50, these two systems perform closely with quite low NDCG scores. So we explore this question on a large synthetic dataset.
Figure 2.4: With different feature number, the comparison of the PLRank and the LambdaMART systems on the synthetic dataset. The feature number in the real distribution is 1000, and per query document number is 250.

The results measured by NDCG@(1, 3, 10) and ERR on the synthetic dataset are shown in Figure 2.4. We observe that when the size of features reaches 300, PLRank starts to outperform the LambdaMART in terms of these four measures. When the size of features is further increased but less than 700, PLRank shows obvious improvements over other three systems. Comparatively, LambdaMART doesn’t perform as well as McRank and MART-2 in terms of NDCG scorers when the size of features is large, while it is better in terms of ERR score. This observation coincides with those in the two real datasets. MART is not as bad as we expect in the synthetic dataset though.
2.2.7 Industry-level Comparison

Last, in Table 2.5, we examine our PLRank system in the Yahoo Challenge set 1 data in an industry level. To save time, we use PLRank(obj=1) and search its parameters to gain best performance regardless of any cost. We sweep the number of tree leaves from 100 to 1000 in steps of 100, and the learning rate $\alpha$ from 0.01 to 0.1 in steps of 0.03. We notice that Burges et al. (2011) actually did not release results of single LambdaMART systems in the standard test set, but in a self-define test set. Since the final result of a LambdaMART-based system combination in the standard set has been available, we reasonably estimate their single LambdaMART systems in the standard test set.

<table>
<thead>
<tr>
<th>System</th>
<th>NDCG@10</th>
<th>ERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLRank(obj=1)</td>
<td>0.802</td>
<td>0.4660</td>
</tr>
<tr>
<td>LambdaMART in Burges et al. (2011)</td>
<td>0.796</td>
<td>0.4649</td>
</tr>
<tr>
<td>LambdaMART-Aug70 in Burges et al. (2011)</td>
<td>0.804</td>
<td>0.4669</td>
</tr>
</tbody>
</table>

Table 2.5: An industry level comparison in the standard Yahoo Challenge set 1 data with LambdaMART.

LambdaMART with complete training set for tuning parameters reaches 0.796 in NDCG@10, and they use a resampled technique called Aug70 to increase the training data to improve their systems to 0.804. In comparison, our result is acceptable compared with LambdaMART with standard training data for tuning, as our result is obtained in a resources-constrained laboratory environment, which might be better given industry-level computing clusters for larger parameter searching.
2.2.8 Running time

<table>
<thead>
<tr>
<th></th>
<th>MT</th>
<th>PR1</th>
<th>LM</th>
<th>PR3</th>
<th>PR6</th>
<th>MR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hours</td>
<td>83</td>
<td>91</td>
<td>101</td>
<td>106.2</td>
<td>126</td>
<td>250+</td>
</tr>
</tbody>
</table>

Table 2.6: Sorted running time in the Microsoft dataset in a single computing core. MT: MART. PR\(_n\): PLRank(\text{obj} = \_n\). MR: McRank.

The computational costs of tree-based systems are mainly at the stage of tree construction, thus these systems have the same time complexity except that McRank requires more iterations to reach reasonable performance. The running times of PLRank, MART, LambdaMART and McRank in Microsoft dataset are shown in Table 2.6. Their differences are mainly due to the computation of functional gradients.
3.1 Greedy Tree Fitting Algorithm in Learning to Rank

Since fitting a regression tree is conducted node by node, in this paper we only discuss the differences of algorithms running on each node. We call them node-splitting algorithms. To make them distinguish from the objective loss functions of ranking models, we use a special word, called error, to help name the node-splitting algorithms. For example, square error (Section 1.1.4), (robust) weighted square error (Section 1.1.5, 1.1.5), and the objective loss based error (Section 3.1.1) proposed in this work. Obviously, if there is a minor difference in the split of a tree node for two node-splitting algorithms, they would lead to two totally different regression trees.

Several measures have been used to quantify the quality of the rank with respect to a ground-truth rank such as NDCG, ERR, MAP etc. In this paper, we use the most popular NDCG and ERR Chapelle and Chang (2011) as the performance measures.
3.1.1 Objective Loss Based Error (OLE)

We were motivated by the success of AdaBoost, a practical machine learning technique Schapire and Freund (2012). In each iteration, AdaBoost selects the weak learner that has a minimal weighted error\(^1\). Empirically it is observed that AdaBoost converges faster than other strategies of selecting a weak learner, and runs pretty well in many practical applications. It has been proved that the weak learner selected by AdaBoost in each iteration ensures a maximum improvement of the exponential loss, so we are seeking such a similar strategy in fitting a regression tree in learning to rank.

Finding an exact optimal regression is computationally infeasible, as the number of possible trees is combinatorially huge. We thus turn to focus on the most basic unit in fitting a regression tree, that is how to conduct a good binary partition to improve the objective loss most. This is a much acceptable approach.

Given a set of samples \( D = \{d_1, \ldots, d_{|D|}\} \), and a selected feature, we first assume there are at most \( |D| - 1 \) potential positions to define a threshold \( v \) for the selected feature. Based on a threshold \( v \), the samples are split into two parts, \( D_l \) and \( D_r \). Second, we assume once a partition is conducted, the samples on the two sides would receive their updated score. In other words, we fit a temporary two-leaf tree in the current samples and update the outputs of two leaves separately (diagonal approximation). We update samples and calculate the objective loss, and allow the maximum improvement quantity as a measure for the current partition. The best partition and its respective threshold are selected after enumerating at most \( |D| - 1 \) possibilities. We ignore the fact that either side

\(^1\)We do not state too much about the foundation of AdaBoost, and readers may refer to Schapire and Freund (2012).
may not be a real leaf in the final fitted regression tree, so that we have a feasible method.

Regarding a threshold \( v \), let the outputs of the temporary two-leaf tree be \( o_1 \) and \( o_2 \), then the objective loss \( \mathcal{L} \) has become a function of \( o_1 \) and \( o_2 \). Once the values of \( o_1 \) and \( o_2 \) are determined, samples on two sides would be updated, and then the objective loss can be straightforwardly computed. However, even in a moderate size dataset, this computation is still prohibitive. So we approximate the objective loss with the Taylor formula in the second order at the point of 0.

\[
\mathcal{L}(o) = \mathcal{L}(o = 0) + \mathcal{L}'(o = 0) \cdot o + \frac{o^2}{2} \mathcal{L}''(o = 0) \quad (3.1)
\]

where \( o \in \{o_1, o_2\} \).

The local optimum \( o \) can be obtained as \(-\frac{\mathcal{L}'(o=0)}{\mathcal{L}''(o=0)}\) by letting the first-order derivative \( \mathcal{L}'(o) \) be zero. More specifically, \( o_1 \) and \( o_2 \) are as following

\[
\begin{aligned}
\begin{cases}
o_1 = -\frac{\mathcal{L}'(o_1=0)}{\mathcal{L}''(o_1=0)} & \text{if } d \in D_l \\
o_2 = -\frac{\mathcal{L}'(o_2=0)}{\mathcal{L}''(o_2=0)} & \text{if } d \in D_r
\end{cases} \quad (3.2)
\end{aligned}
\]

Insert Eqn. 3.2 into Eqn. 3.1, and simplify to obtain our objective loss based error

\[
OLE(v) = \mathcal{L}(o_1 = 0, o_2 = 0)
- \frac{1}{2} \left[ \frac{\mathcal{L}'(o_1 = 0)^2}{\mathcal{L}''(o_1 = 0)} + \frac{\mathcal{L}'(o_2 = 0)^2}{\mathcal{L}''(o_2 = 0)} \right]
\propto - \left[ \frac{\mathcal{L}'(o_1 = 0)^2}{\mathcal{L}''(o_1 = 0)} + \frac{\mathcal{L}'(o_2 = 0)^2}{\mathcal{L}''(o_2 = 0)} \right] \quad (3.3)
\]
This resultant formula is not equivalent to SE or (R)WSE in a general case, and would lead to a totally different regression tree from that using other node-splitting algorithms.

In the case of learning to rank, we analyze their equivalence for point-wise, pair-wise and list-wised based models.

3.1.2 Derivative Additive Loss Functions

In order to calculate all the gradients in Eqn. 3.3 in an efficient left-to-right incremental updating way, we explore the cases the gradient $\mathcal{L}'(o = 0)$, $\mathcal{L}''(o = 0)$ can be decomposed into operations on each sample.

**Definition 1.** A loss function $\mathcal{L}$ is defined as derivative additive if $\mathcal{L}'(o = 0|D) = \sum_{d \in D} \mathcal{L}'(o = 0|d)$ and $\mathcal{L}''(o = 0|D) = \sum_{d \in D} \mathcal{L}''(o = 0|d)$.

**Example 2.** The loss function of MART system is derivative additive, since

$$
\mathcal{L} = \sum_d (f_t(d) - y(d))^2
$$

$$
\mathcal{L}'(o = 0) = \frac{\partial \sum_d (f_t(d) + o - y(d))^2}{\partial o} = \sum_d 2(f_t(d) - y(d))
$$

$$
= \sum_d \mathcal{L}'(o = 0|d)
$$

$$
\mathcal{L}''(o = 0) = \frac{\partial \sum_d (f_t(d)+o-y(d))^2}{\partial^2 o} = \sum_d 2
$$

$$
= \sum_d \mathcal{L}''(o = 0|d)
$$

**Example 3.** The loss function of McRank system is derivative additive, since
\[
\mathcal{L} = \sum_d \sum_{c=0}^{K-1} I(c = y(d)) \log p_c(d)
\]

\[
\mathcal{L}'(o = 0) = \sum_d \sum_{c=0}^{K-1} \{p_c(d) - I(c = y(d))\}
\]

\[
\mathcal{L}''(o = 0) = \sum_d \sum_{c=0}^{K-1} p_c(d)(1 - p_c(d))
\]

**Example 4.** The loss function of RankBoost system is not derivative additive, since

\[
\mathcal{L} = \sum_{D_i} \sum_{d_1,d_2 \in D_i, y(d_1) > y(d_2)} \exp\{f_t(d_2) - f_t(d_1)\}
\]

To clearly explain, we use a toy example with \(d_1, d_2, d_3 \in D_1\), sorted by their relevances \(y(d_1) > y(d_2) > y(d_3)\). Assuming in current \(t\)th iteration, their score is \(f_t(\cdot)\). The exponential loss is

\[
\mathcal{L} = \exp\{f_t(d_2) - f_t(d_1)\} + \exp\{f_t(d_3) - f_t(d_1)\} + \exp\{f_t(d_3) - f_t(d_2)\}
\]

\[
= s_1 + s_2 + s_3
\]

where, to simplify, \(s_1 = \exp\{f_t(d_2) - f_t(d_1)\}\), \(s_2 = \exp\{f_t(d_3) - f_t(d_1)\}\), \(s_3 = \exp\{f_t(d_3) - f_t(d_2)\}\).
as \( \mathcal{L}(o|d_1) = s_1 \exp\{-o\} + s_2 \exp\{-o\} + s_3 \)

so \( \mathcal{L}'(o = 0|d_1) = -s_1 - s_2 \)

\( \mathcal{L}''(o = 0|d_1) = s_1 + s_2 \)

likewise \( \mathcal{L}'(o = 0|d_2) = s_1 - s_3 \)

\( \mathcal{L}''(o = 0|d_2) = s_1 + s_3 \)

likewise \( \mathcal{L}'(o = 0|d_3) = s_2 + s_3 \)

\( \mathcal{L}''(o = 0|d_3) = s_2 + s_3 \)

Suppose one partition is \( \{d_1, d_2\} \) and \( \{d_3\} \), with the output value \( o_1, o_2 \) respectively, then the current loss is

\[
\mathcal{L}(o_1, o_2) = \exp\{(f_t(d_2) + o_1) - (f_t(d_1) + o_1)\} + \exp\{(f_t(d_3) + o_2) - (f_t(d_1) + o_1)\} + \exp\{(f_t(d_3) + o_2) - (f_t(d_2) + o_1)\} + \text{const}
\]

\( \mathcal{L}'(o_1 = 0) = -s_2 - s_3 \)

\( \mathcal{L}''(o_1 = 0) = s_2 + s_3 \)

\( \neq \mathcal{L}''(o_1 = 0|d_1) + \mathcal{L}''(o_2 = 0|d_2) \)

\( = s_1 + s_1 + s_2 + s_3 \)

The key reason is that if two samples appearing in the same \( \exp \) term of the objective loss also are classified into the same leaf, then they would receive the same output of current leaf, which does not contribute to the objective loss. In this example, the two \( s_1 \), coming from \( d_1 \) and \( d_2 \), should be counteracted. As a result, the exponential loss is not
derivative additive.

**Example 5.** The loss function of LambdaMART system is not derivative additive.

This famous system has no explicit objective loss function, but has exact first- and second-order derivatives. Its first derivative has a similar unit with the RankBoost model, both having such terms of \( \exp(f_t(d_i) - f_t(d_j)) \), since LambdaMART is pair-wise based system. Based on the detailed analysis in RankBoost, we could easily know the loss function of LambdaMART, potentially existing, is not derivative additive\(^2\).

**Example 6.** The loss function of ListMLE is not derivative additive.

As the term \( \exp(f_t(d_i) - f_t(d_j)) \) is frequently appearing in its loss function, the loss function of ListMLE is also not derivative additive.

**Example 7.** Some special list-wise models have derivative additive loss functions.

In Ravikumar et al. (2011), several point-wise based systems are modified by using list-wise information, so they are considered to be list-wise based systems, such as consistent-MART, consistent KL divergence based, consistent cosine distance based. As the extra list-wise information is actually utilized in a preprocessing step, and then they are running in a point-wise style, so these so-called list-wise based systems also own derivative additive loss functions.

\(^2\)To examine strictly, the one-step Newton formula in LambdaMART (Line 11 in Alg. 1 of Wu et al. (2010)) is incorrect conceptually, as the denominator in \( \frac{L'(x=x_0)^2}{L''(x=x_0)} \) is tackled as derivative additive. We have not found any explanations from their paper. But it can be viewed as an approximation of the exact formula.
3.1.3 (R)WSE ⊂ OLE

(R)WSE was proposed for LogitBoost, which is viewed as a point-wise based system from the angle of learning to rank. We prove (R)WSE for all point-wise based systems is actually a simplified form of OLE.

Theorem 8. Regarding derivative additive loss function, (R)WSE principle is equivalent to OLE.

Proof.

\[ OLE(v) \text{ (Eqn. 3.3)} \]

\[
= - \left[ \frac{\sum_{d \in D_1} L'(o_1=0|d)}{\sum_{d \in D_1} L''(o_1=0|d)} \right]^2 + \left[ \frac{\sum_{d \in D_r} L'(o_2=0|d)}{\sum_{d \in D_r} L''(o_2=0|d)} \right]^2
\]

by Eqn.1.11

\[
\propto - \left[ \frac{\left(\sum_{d \in D_1} w(d) \cdot r(d)\right)^2}{\sum_{d \in D_1} w(d)} + \frac{\left(\sum_{d \in D_r} w(d) \cdot r(d)\right)^2}{\sum_{d \in D_r} w(d)} \right] + \text{const}
\]

where const = \[
\frac{\left(\sum_{d \in D} w(d) \cdot r(d)\right)^2}{\sum_{d \in D} w(d)}
\]

\[ = RWSE(v) \text{ (Eqn. 1.14)} \]

We simply obtain the robust weighted least square error from Li Li (2010b). As Li proved the robust version is equivalent to original WSE, thus our method is equivalent to WSE, for all derivative additive loss functions.

By the explanation of robustness of Li, our OLE method Eqn. 3.3 is intrinsically robust, as all denominators are less likely to be zero in summing a set of samples.

Recall that a model is classified into the point-wise category if the model does not use the relationship between samples, but only individual samples. See the typical point-wise
based models, Example 1 and 2. So, point-wise based systems have derivative additive loss functions, and in this case, (R)WSE is always equivalent to OLE.

A pair-wise based system considers the relationship only between two samples. If a pair of associated samples are classified into two different tree nodes, the objective loss function is derivative additive; otherwise, it is not. In practical applications, it is not difficult to overcome this inconvenience by using an incremental updating.

A list-wise based system would render more samples interact to each other, and it is relatively more difficult to tackle. But in splitting a tree node, the incremental updating is still working.

We are now able to answer the question from Li Li (2010b), the Eqn. 1.13 appears to have two explanations, one from weighted average, and the other from one-step Newton. As we proved that (R)WSE is special cases derived from optimizing only derivative additive objectives, and LogitBoost uses derivative additive objective, so (R)WSE and SE are equivalent. Moreover, for other complex objective losses, (R)WSE may have no theoretical support, but it may serve as an approximation of our method.

3.1.4 SE = (R)WSE = OLE for MART

SE is generally not equivalent to (R)WSE or OLE, even the objective loss functions used satisfy the condition of Theorem 8. However, we find that MART is an ideal intersection of OLE, (R)WSE and SE.

MART system adopts least square loss as objective loss, and classic gradient boosting framework to fit regression trees. Many commercial search engines are using this model to construct their ranking systems.
Theorem 9. Regarding MART system, whose objective loss is the least-square \( \sum_d |f(d) - y(d)|^2 \), then \( \arg \min_v SE(v) = \arg \min_v (R) WSE(v) = \arg \min_v OLE(v) \).

This theorem means, regarding MART system, the three node-splitting principles lead to the same binary partition in any selected tree node, and then lead to the same regression tree in current iteration of boosting.

Proof. Given some chosen feature function \( f(\cdot) \) and pseudo-response \( r(\cdot) \) of each document \( d \in D \), there are \( |D| - 1 \) positions to define a threshold which is the middle value of two adjacent feature values.

We derive from the objective loss, and prove minimizing it is the same with minimizing the least square error splitting principle (Eqn. 1.8).

\[
L(v) = \sum_{d \in D_t} (f_t(d) + o_1 - y(d))^2 + \sum_{d \in D_r} (f_t(d) - y(d))^2
\]

By the Newton formula,

\[
o_1 = -\frac{L'(o_1=0)}{L''(o_1=0)} = -\frac{\sum_{d \in D_t} L'(f_t(d))}{2|D_t|} = \frac{\sum_{d \in D_t} r(d)}{2|D_t|} = \frac{\bar{r}_1}{2}
\]

\[
o_2 = -\frac{L'(o_2=0)}{L''(o_2=0)} = -\frac{\sum_{d \in D_r} L'(f_t(d))}{2|D_r|} = \frac{\sum_{d \in D_r} r(d)}{2|D_r|} = \frac{\bar{r}_2}{2}
\]
plugging back to get

\[ \mathcal{L}(v) = \sum_{d \in D_i} (f_t(d) + \frac{\bar{r}_1}{2} - y(d))^2 + \sum_{d \in D_i} (f_t(d) + \frac{\bar{r}_2}{2} - y(d))^2 \]

\[ \propto \sum_{d \in D_i} ((2f_t(d) - 2y(d)) + \bar{r}_1)^2 + \sum_{d \in D_i} ((2f_t(d) - 2y(d)) + \bar{r}_2)^2 \]

As

\[ 2f_t(d) - 2y(d) = \mathcal{L}'(f_t(d)) = -r(d) \]

then

\[ \mathcal{L}(v) = \sum_{d \in D_i} (r(d) - \bar{r}_1)^2 + \sum_{d \in D_i} (r(d) - \bar{r}_2)^2 \]

\[ = \text{Eqn. 1.8} \]

So, optimizing objective loss is equivalent to optimizing SE, and as mentioned before MART is a point-wise bases system, which suggests \((R)WSE = OLE\).  

The key step here is, the average pseudo-response, for any of \(\bar{r}_1\) and \(\bar{r}_2\), whose definition is same with that in Eqn. 1.8, is exactly double of the negative optimum computed by Newton equation using the extra second derivative. Regarding other loss functions, this relationship does not necessarily hold.

This theorem suggests the classic tree fitting algorithm in gradient boosting is very suitable for the least-square loss function, on which MART system is based, and this could explain why MART system actually performs excellently in our experiments and many practical applications.

This theorem brings us a more elegant splitting principle than Eqn. 1.8 for the classic
MART system. By plugging its first-order derivative $L'(o_1 = 0) = \sum_{d \in D_t} -r(d), L'(o_2 = 0) = \sum_{d \in D_r} -r(d)$, and second-order derivative $L''(o_1) = 2|D_t|, L''(o_2) = 2|D_r|$ into Eqn. 3.3, we obtain

$$MART(v) = -\left[\sum_{d \in D_t} \frac{r(d)^2}{|D_t|} + \sum_{d \in D_r} \frac{r(d)^2}{|D_r|}\right]$$

(3.4)

This form is more intuitive for incremental computing the optimal threshold $v$ from left to right.

3.2 Experiments

3.2.1 Datasets and Systems

As suggested by Qin et al. (2010b), we use two real-world datasets to make our results more stable, Yahoo challenge 2010 and Microsoft 10K. The statistics of these data sets are reported in Table 2.1.

1. Yahoo Challenge 2010. After Yahoo corporation hosted this far-reaching influence contest of learning to rank in 2010, this dataset has been important for a comparison. It contains two sets, and here we use the bigger one (set 1). Yahoo dataset was released with only one split of training, validating, and testing set, and we add an extra two splits and also report average results.

2. Microsoft 10K. Another publicly released datasets, and even larger than the Yahoo data in terms of the number of documents. As a 5-fold splitting is provided by
Table 3.1: The top two datasets are used in this work and others are just a reference. λ-MART is LambdaMART. Ohsumed is of letor 3.0. #D./#Q. means average document number per query.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Query</th>
<th>#Doc.</th>
<th>#D. / #Q.</th>
<th>#Feat.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yahoo</td>
<td>20K</td>
<td>473K</td>
<td>23</td>
<td>519</td>
</tr>
<tr>
<td>Micro-10K</td>
<td>6K</td>
<td>723K</td>
<td>120</td>
<td>136</td>
</tr>
<tr>
<td>McRankLi et al. (2007)</td>
<td>10-26K</td>
<td>474-1741K</td>
<td>18-88</td>
<td>367-619</td>
</tr>
<tr>
<td>λ-MART Wu et al. (2010)</td>
<td>31K</td>
<td>4154K</td>
<td>134</td>
<td>416</td>
</tr>
<tr>
<td>Ohsumed</td>
<td>106</td>
<td>16K</td>
<td>150</td>
<td>45</td>
</tr>
<tr>
<td>letor 4.0</td>
<td>2.4K</td>
<td>85K</td>
<td>34</td>
<td>46</td>
</tr>
</tbody>
</table>

The two datasets above were empirically found to be different. The Microsoft dataset seems more difficult than Yahoo as some models are reportedly running badly on it Tan et al. (2013a). It has comparatively less features, 136, and larger average number of documents per query 120, compared to 519 and 23 of Yahoo. The two real-world datasets should be capable of providing convincing results.

As (R)WSE has been shown to be a special case of OLE, we only compare SE and OLE in the scenario of learning to rank. We adopt three famous ranking systems with regression trees as weak learners. To ensure identity in implementation details, we used the same code template. Their only differences are in objective loss functions, and regression tree fitting principles.

1. point-wise based McRank Li et al. (2007). The multi-class classification based system was reported to be strong in real-world datasets Wu et al. (2010), and is natural to be one of our baseline systems.
<table>
<thead>
<tr>
<th>#leaf</th>
<th>α</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
<th>ERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.06</td>
<td>71.32/71.76</td>
<td>71.47/72.22</td>
<td>77.97/78.52</td>
<td>45.44/45.71</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.38/71.73</td>
<td>71.82/72.37</td>
<td>78.25/78.67</td>
<td>45.52/45.77</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.67/71.87</td>
<td>71.96/72.52</td>
<td>78.33/78.77</td>
<td>45.59/45.79</td>
</tr>
<tr>
<td>15</td>
<td>0.06</td>
<td>71.47/72.00</td>
<td>71.70/72.48</td>
<td>78.16/78.73</td>
<td>45.52/45.81</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.64/71.93</td>
<td>72.01/72.71</td>
<td>78.40/78.90</td>
<td>45.61/45.84</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.71/71.96</td>
<td>72.19/72.70</td>
<td>78.51/78.94</td>
<td>45.65/45.85</td>
</tr>
<tr>
<td>20</td>
<td>0.06</td>
<td>71.52/71.90</td>
<td>71.80/72.60</td>
<td>78.26/78.84</td>
<td>45.54/45.83</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.65/72.03</td>
<td>72.14/72.77</td>
<td>78.50/79.00</td>
<td>45.64/45.88</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.80/72.10</td>
<td>72.23/72.79</td>
<td>78.58/78.99</td>
<td>45.66/45.89</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#leaf</th>
<th>α</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
<th>ERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.06</td>
<td>47.43/47.14</td>
<td>46.14/46.46</td>
<td>48.60/49.17</td>
<td>35.90/36.13</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>47.49/47.69</td>
<td>46.41/46.79</td>
<td>49.00/49.47</td>
<td>36.13/36.32</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>47.42/47.67</td>
<td>46.50/46.78</td>
<td>49.02/49.51</td>
<td>36.13/36.30</td>
</tr>
<tr>
<td>15</td>
<td>0.06</td>
<td>47.61/47.77</td>
<td>46.36/46.95</td>
<td>48.95/49.52</td>
<td>36.06/36.41</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>47.69/47.64</td>
<td>46.67/47.02</td>
<td>49.27/49.63</td>
<td>36.26/36.41</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>47.45/47.90</td>
<td>46.65/47.01</td>
<td>49.31/49.67</td>
<td>36.20/36.49</td>
</tr>
<tr>
<td>20</td>
<td>0.06</td>
<td>47.69/47.94</td>
<td>46.44/47.06</td>
<td>49.07/49.67</td>
<td>36.09/36.45</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>47.52/48.04</td>
<td>46.76/47.24</td>
<td>49.36/49.80</td>
<td>36.20/36.54</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>47.87/47.89</td>
<td>46.86/47.12</td>
<td>49.51/49.68</td>
<td>36.34/36.45</td>
</tr>
</tbody>
</table>

Table 3.2: Performances (%) of SE / OLE in the Yahoo Data with McRank model. *All results reported are averaged over self-defined three-fold.* All results with over 0.1 point improvement are marked.
<table>
<thead>
<tr>
<th>#leaf</th>
<th>α</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
<th>ERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.06</td>
<td>71.15/71.62</td>
<td>71.60/71.94</td>
<td>77.82/78.15</td>
<td>45.66/45.80</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.29/71.81</td>
<td>71.76/72.11</td>
<td>77.96/78.29</td>
<td>45.72/45.90</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.30/71.76</td>
<td>71.76/72.19</td>
<td>77.93/78.34</td>
<td>45.67/45.87</td>
</tr>
<tr>
<td>15</td>
<td>0.06</td>
<td>71.30/71.64</td>
<td>71.80/72.13</td>
<td>77.97/78.26</td>
<td>45.69/45.88</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.29/71.61</td>
<td>71.92/72.28</td>
<td>78.07/78.40</td>
<td>45.72/45.91</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.51/71.91</td>
<td>71.96/72.36</td>
<td>78.14/78.46</td>
<td>45.76/45.97</td>
</tr>
<tr>
<td>20</td>
<td>0.06</td>
<td>71.51/71.75</td>
<td>72.02/72.25</td>
<td>78.13/78.40</td>
<td>45.77/45.92</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.37/72.10</td>
<td>71.92/72.56</td>
<td>78.04/78.58</td>
<td>45.72/46.02</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.44/71.76</td>
<td>71.91/72.39</td>
<td>78.06/78.57</td>
<td>45.71/45.96</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#leaf</th>
<th>α</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
<th>ERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.06</td>
<td>47.42/47.98</td>
<td>46.21/46.54</td>
<td>48.22/48.70</td>
<td>36.44/36.68</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>47.79/47.64</td>
<td>46.55/46.57</td>
<td>48.57/48.85</td>
<td>36.54/36.71</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>47.45/47.79</td>
<td>46.32/46.62</td>
<td>48.54/48.94</td>
<td>36.42/36.67</td>
</tr>
<tr>
<td>15</td>
<td>0.06</td>
<td>48.08/47.99</td>
<td>46.64/46.75</td>
<td>48.68/48.88</td>
<td>36.73/36.77</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>47.94/48.03</td>
<td>46.53/46.63</td>
<td>48.90/49.01</td>
<td>36.71/36.78</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>47.73/47.72</td>
<td>46.60/46.64</td>
<td>48.82/49.04</td>
<td>36.65/36.67</td>
</tr>
<tr>
<td>20</td>
<td>0.06</td>
<td>48.01/48.19</td>
<td>46.52/46.87</td>
<td>48.78/49.13</td>
<td>36.74/36.88</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>47.99/48.07</td>
<td>46.66/46.79</td>
<td>48.95/49.10</td>
<td>36.72/36.81</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>47.63/47.67</td>
<td>46.69/46.51</td>
<td>49.02/49.03</td>
<td>36.70/36.61</td>
</tr>
</tbody>
</table>

Table 3.3: Performances (%) of SE / OLE in the Yahoo Data with LambdaMART model. All results reported are averaged over self-defined three-fold. All results with over 0.1 point improvement are marked.
<table>
<thead>
<tr>
<th>#leaf</th>
<th>α</th>
<th>NDCG@1</th>
<th>NDCG@3</th>
<th>NDCG@10</th>
<th>ERR</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.06</td>
<td>71.38/71.47</td>
<td>71.86/71.82</td>
<td>78.31/78.30</td>
<td>45.58/45.58</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.52/71.53</td>
<td>72.03/72.06</td>
<td>78.44/78.45</td>
<td>45.63/45.64</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.50/71.58</td>
<td>72.03/72.09</td>
<td>78.48/78.48</td>
<td>45.63/45.67</td>
</tr>
<tr>
<td>15</td>
<td>0.06</td>
<td>71.60/71.39</td>
<td>72.09/72.14</td>
<td>78.50/78.48</td>
<td>45.65/45.63</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.54/71.75</td>
<td>72.19/72.20</td>
<td>78.62/78.66</td>
<td>45.71/45.73</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.83/71.51</td>
<td>72.34/72.20</td>
<td>78.67/78.61</td>
<td>45.77/45.69</td>
</tr>
<tr>
<td>20</td>
<td>0.06</td>
<td>71.78/71.82</td>
<td>72.29/72.34</td>
<td>78.65/78.65</td>
<td>45.72/45.76</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>71.65/71.75</td>
<td>72.41/72.41</td>
<td>78.76/78.71</td>
<td>45.73/45.77</td>
</tr>
<tr>
<td></td>
<td>0.12</td>
<td>71.73/71.83</td>
<td>72.31/72.37</td>
<td>78.73/78.72</td>
<td>45.76/45.77</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.4: Performances (%) of SE / OLE in the Yahoo Data with RankBoost model. All results reported are averaged over self-defined three-fold. All results with over 0.1 point improvement are marked.
2. pair-wise based LambdaMART Wu et al. (2010). This famous pair-wised system gained its reputation in Yahoo Challenge 2010, as a combined system, mainly constructed on LambdaMART, winning the championship. In our work, we only compare with single LambdaMART systems, which are trained using NDCG loss. Maybe LambdaMART systems could be improved further using different configurations, but here it is not our focus.

3. pair-wise based RankBoost Freund et al. (2003). This classic system has some limitations for large datasets, as it considers all pairs in a set of samples. This is unrealistic for Microsoft data, which has an average of 120 documents per query. We refer to the idea in Xia et al. (2009) to improve original RankBoost into a top\( N \) consistent version, which also has more advantages for learning to rank tasks. We first rank documents per query by their relevance scores, and consider those pairs, at least one document in which appears in the top\( N \) positions. To be consistent with NDCG@10, we take \( N \) as 10.

As shown by the proof, MART is an ideal intersection of these ideas, we did not use the famous MART system. For each system and algorithm, we set configurations as follows: the number of leaves is set as 10, 15, 20; the learning rate \( \alpha \) in Eqn. ?? is set as 0.06, 0.1, 0.12. So there are nine configurations for each system. Examining the testing performance in the real-world datasets, several hundreds of iterations (or regression trees) almost lead to convergence, so I set maximum number of iterations to 1000 for LambdaMART and RankBoost, 2500 for McRank which converges more slowly. We report popular measures, \( NDCG@(1, 3, 10) \) and \( ERR \).
Figure 3.1: Histogram of total improvements of OLE over SE.

3.2.2 Experimental Comparison with Three Systems

Table and Figure Descriptions

There are three types of tables and figures to display our comparison. To save space, we state them here.
The first type reflects a total improvement of OLE over SE considering all iterations, cross-validation, and configurations. We draw a histogram (Figure 3.1) about the frequency of different improvements. The X-axis denotes the differences between using OLE and SE for all NDCG@(1, 3, 10) and ERR, and Y-axis denotes the frequency.

The second type lists exact performances (%) of SE and OLE for nine configurations, at the 1000th iteration for LambdaMART and RankBoost, and the 2500th for McRank. See Table 3.4.

The third type further provides a complete comparison in each iteration and configuration. The Y-axis denotes the differences that OLE minus SE for NDCG@(1, 3, 10) and ERR respectively. See Figure ??.

**Detailed Comparison**

In Figure 3.1, the main parts of McRank and LambdaMART (the left four figures) are on the right of the line $X = 0$, and especially more obvious for LambdaMART. Comparatively, RankBoost has not show significant improvements, and even a slight degrade. This figure reflects in the whole process OLE dominates SE. But to examine the performance close to convergence, we need to look at the following two detailed tables.

In table 3.4 and ??, for McRank and LambdaMART respectively, among 72 comparisons, OLE gains 68 and 56 improvements for at least over 0.1 point, and most of them are 0.3 to 0.4. These improvements are reasonable, as our baselines are strong, and in such large datasets. These statistics are based on nine typical configurations, and demonstrate OLE is workable for the McRank and LambdaMART models in a general case.

However, OLE does not look powerful for RankBoost. There are only observed 4
improvements for at least 0.1 point. As mentioned before, Yahoo dataset was empirically found to be different from Microsoft 10K. Here we also find that OLE performs close to SE in Yahoo dataset, half slightly better. But obviously it is consistently slightly worse in Microsoft data.

We further analyze four measures separately, NDCG@$(1, 3, 10)$ and ERR. Though both McRank (Figure ??, shown in the complementary material due to the space limit) and LambdaMART have been improved consistently with OLE, the NDCG@$1$ (real read line) and ERR (dotted blue line) have relatively smaller improvements. ERR is more difficult to improve than NDCG.

Improvements of NDCG@$3$ and NDCG@$10$ on McRank and LambdaMART are more robust in a variety of configurations. As NDCG@$1$ is computed on the first document predicted by models, and ERR is computed on the whole of ranking documents whose numbers are usually several dozens, in practice, the first page with 10 links returned by a search engine are more desired by users. So we think it may be more useful to improve NDCG@$3$ and NDCG@$10$ measures.

As OLE is supposed to have a faster convergence than SE, we also have a statistics of objective losses in the final iteration. OLE indeed leads to smaller objective losses, but not by that much, about 0.32% - 1%. As this work only focuses on the splitting rule in a single node, we also tried different strategies to generate node. Width-first search and depth-first search. Interestingly, depth-first search runs poorly for both baselines and our method. This is an open question and left to future exploration. We thus adopted the width-first search and limits the number of leaves.

Regarding the running time, there is no loss for the systems with derivative additive
objective losses compared to SE in gradient boosting. Typical such systems are point-wise based. But for pair-wise and list-wise based, OLE suffers from extra overheads of maintaining exact second derivatives of objective loss function. In an incremental updating style, this overhead is about 30% of SE for pair-wise based, and regarding list-wise, there may be more.

Figure 3.2: Improvements (absolute %) of OLE over SE with McRank system when the learning rate is set as 0.06. Each point in the figures has been averaged over a five-fold or three-fold cross-validation.
Figure 3.3: Improvements (absolute %) of OLE over SE with McRank system when the learning rate is set as 0.1. Each point in the figures has been averaged over a five-fold or three-fold cross-validation.
Figure 3.4: Improvements (absolute %) of OLE over SE with McRank system when the learning rate is set as 0.12. Each point in the figures has been averaged over a five-fold or three-fold cross-validation.
Figure 3.5: Improvements (absolute %) of OLE over SE with LambdaMart system when the learning rate is set as 0.06. Each point in the figures has been averaged over a five-fold or three-fold cross-validation.
Figure 3.6: Improvements (absolute %) of OLE over SE with LambdaMart system when the learning rate is set as 0.1. Each point in the figures has been averaged over a five-fold or three-fold cross-validation.
Figure 3.7: Improvements (absolute %) of OLE over SE with LambdaMart system when the learning rate is set as 0.12. Each point in the figures has been averaged over a five-fold or three-fold cross-validation.
Figure 3.8: Improvements (absolute %) of OLE over SE with RankBoost system when the learning rate is set as 0.06. *Each point in the figures has been averaged over a five-fold or three-fold cross-validation.*
Figure 3.9: Improvements (absolute %) of OLE over SE with RankBoost system when the learning rate is set as 0.1. Each point in the figures has been averaged over a five-fold or three-fold cross-validation.
Figure 3.10: Improvements (absolute %) of OLE over SE with RankBoost system when the learning rate is set as 0.12. Each point in the figures has been averaged over a five-fold or three-fold cross-validation.
A Simple Discriminative Training

Method for Machine Translation with Large-Scale Features

4.1 Plackett-Luce Model

Plackett-Luce was firstly proposed to predict ranks of horses in gambling Plackett (1975). Let \( r = (r_1, r_2 \ldots r_N) \) be \( N \) horses with a probability distribution \( \mathcal{P} \) on their abilities to win a game, and a rank \( \pi = (\pi(1), \pi(2) \ldots \pi(|\pi|)) \) of horses can be understood as a generative procedure, where \( \pi(j) \) denotes the index of the horse in the \( j \)th position.

In the 1st position, there are \( N \) horses as candidates, each of which \( r_j \) has a probability \( p(r_j) \) to be selected. Regarding the rank \( \pi \), the probability of generating the champion is \( p(r_{\pi(1)}) \). Then the horse \( r_{\pi(1)} \) is removed from the candidate pool.

In the 2nd position, there are only \( N - 1 \) horses, and their probabilities to be selected become \( p(r_j)/Z_2 \), where \( Z_2 = 1 - p(r_{\pi(1)}) \) is the normalization. Then the runner-up in
Table 4.1: The probability of the rank $\pi = (2, 3, 1)$ is $p(r_2) \cdot p(r_3)/(1 - p(r_2))$ in a simplified form, as $\frac{p(r_2)}{Z_1} = p(r_2)$ and $\frac{p(r_1)}{Z_3} = 1$.

the rank $\pi$, the $\pi(2)$th horse, is chosen at the probability $p(r_{\pi(2)}/Z_2$. We use a consistent terminology $Z_1$ in selecting the champion, though $Z_1$ equals 1 trivially.

This procedure iterates to the last rank in $\pi$. The key idea for the Plackett-Luce model is the choice in the $i$th position in a rank $\pi$ only depends on the candidates not chosen at previous stages. The probability of generating a rank $\pi$ is given as follows

$$p(\pi) = \prod_{j=1}^{\left|\pi\right|} \frac{p(r_{\pi(j)})}{Z_j} \quad (4.1)$$

where $Z_j = 1 - \sum_{t=1}^{j-1} p(r_{\pi(t)})$.

We offer a toy example (Table 4.1) to demonstrate this procedure.

**Theorem 1.** The permutation probabilities $p(\pi)$ form a probability distribution over a set of permutations $\Omega_\pi$. For example, for each $\pi \in \Omega_\pi$, we have $p(\pi) > 0$, and $\sum_{\pi \in \Omega_\pi} p(\pi) = 1$.

We have to note that, $\Omega_\pi$ is not necessarily required to be completely ranked permutations in theory and in practice, since gamblers might be interested in only the champion and runner-up, and thus $\left|\pi\right| \leq N$. In experiments, we would examine the effects on different length of permutations, systems being termed $PL(\left|\pi\right|)$. 
Theorem 2. Given any two permutations $\pi$ and $\pi'$, and they are different only in two positions $p$ and $q$, $p < q$, with $\pi(p) = \pi'(q)$ and $\pi(q) = \pi'(p)$. If $p(\pi(p)) > p(\pi(q))$, then $p(\pi) > p(\pi')$.

In other words, exchanging two positions in a permutation where the horse more likely to win is not ranked before the other would lead to an increase of the permutation probability.

This suggests the ground-truth permutation, ranked decreasingly by their probabilities, owns the maximum permutation probability on a given distribution. In SMT, we are motivated to optimize parameters to maximize the likelihood of ground-truth permutation of an N-best hypotheses.

Due to the limitation of space, see Plackett (1975); Cao et al. (2007) for the proofs of the theorems.

4.2 Plackett-Luce Model in Statistical Machine Translation

In SMT, let $f = (f_1, f_2 \ldots)$ denote source sentences, and $e = (\{e_{1,1}, \ldots\}, \{e_{2,1}, \ldots\} \ldots)$ denote target hypotheses. A set of features are defined on both source and target side. We refer to $h(e_{i,s})$ as a feature vector of a hypothesis from the $i$th source sentence, and its score from a ranking function is defined as the inner product $h(e_{i,s})^T w$ of the weight vector $w$ and the feature vector.
We first follow the popular exponential style to define a parameterized probability distribution over a list of hypotheses.

\[ p(e_{i,j}) = \frac{\exp\{h(e_{i,j})^T w\}}{\sum_k \exp\{h(e_{i,k})^T w\}} \]  

(4.2)

The ground-truth permutation of an \( n \) best list is simply obtained after ranking by their sentence-level BLEUs. Here we only concentrate on their relative ranks which are straightforward to compute in practice, e.g. add 1 smoothing. Let \( \pi_i^* \) be the ground-truth permutation of hypotheses from the \( i \)th source sentences, and our optimization objective is maximizing the log-likelihood of the ground-truth permutations and penalized using a zero-mean and unit-variance Gaussian prior. This results in the following objective and gradient:

\[ L = \log\{\prod_i p(\pi_i^*, \mathcal{P}(w))\} - \frac{1}{2} w^T w \]  

(4.3)

\[ \frac{\partial L}{\partial w} = \sum_i \sum_j \{h(e_{i,\pi_i^* (j)}) - \sum_{t=j} (h(e_{i,\pi_i^* (t)}) \cdot \frac{p(e_{i,\pi_i^* (t)})}{Z_{i,j}})\} - w \]  

(4.4)

where \( Z_{i,j} \) is defined as the \( Z_j \) in Formula (1) of the \( i \)th source sentence.

The log-likelihood function is smooth, differentiable, and concave with the weight vector \( w \), and its local maximal solution is also a global maximum. Iteratively selecting one parameter in \( \alpha \) for tuning in a line search style (or MERT style) could also converge into the global global maximum Bertsekas (1999). In practice, we use more fast limited-memory BFGS (L-BFGS) algorithm Byrd et al. (1995).
N-best Hypotheses Resample

The log-likelihood of a Plackett-Luce model is not a strict upper bound of the BLEU score, however, it correlates with BLEU well in the case of rich features. The concept of “rich” is actually qualitative, and obscure to define in different applications. We empirically provide a formula to measure the richness in the scenario of machine translation.

\[ r = \frac{\text{the size of features}}{\text{the average size of N-best lists}} \]  

(4.5)

The greater, the richer. In practice, we find a rough threshold of \( r \) is 5.

In engineering, the size of an N-best list with unique hypotheses is usually less than several thousands. This suggests that, if features are up to thousands or more, the Plackett-Luce model is quite suitable here. Otherwise, we could reduce the size of N-best lists by sampling to make \( r \) beyond the threshold.

Their may be other efficient sampling methods, and here we adopt a simple one. If we want to \( m \) samples from a list of hypotheses \( e \), first, the \( \frac{m}{3} \) best hypotheses and the \( \frac{m}{3} \) worst hypotheses are taken by their sentence-level BLEUs. Second, we sample the remaining hypotheses on distribution \( p(e_i) \propto \exp(h(e_i)^T \mathbf{w}) \), where \( \mathbf{w} \) is an initial weight from last iteration.
Table 4.2: PL(\(k\)): Plackett-Luce model optimizing the ground-truth permutation with length \(k\). The significant symbols (+ at 0.05 level) are compared with MERT. The bold font numbers signifies better results compared to M(1) system.

<table>
<thead>
<tr>
<th></th>
<th>MT02(dev)</th>
<th>MT04(%)</th>
<th>MT05(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MERT</td>
<td>34.61</td>
<td>31.76</td>
<td>28.85</td>
</tr>
<tr>
<td>MIRA</td>
<td>35.31</td>
<td>32.25</td>
<td>29.37</td>
</tr>
<tr>
<td>PL(1)</td>
<td>34.20</td>
<td>31.70</td>
<td>28.90</td>
</tr>
<tr>
<td>PL(2)</td>
<td>34.31</td>
<td>31.83</td>
<td>29.10</td>
</tr>
<tr>
<td>PL(3)</td>
<td>34.39</td>
<td>32.05</td>
<td>29.20</td>
</tr>
<tr>
<td>PL(4)</td>
<td>34.40</td>
<td>32.13</td>
<td>29.46+</td>
</tr>
<tr>
<td>PL(5)</td>
<td>34.46</td>
<td>32.19+</td>
<td>29.42+</td>
</tr>
<tr>
<td>PL(6)</td>
<td>34.37</td>
<td>32.16</td>
<td>29.30</td>
</tr>
<tr>
<td>PL(7)</td>
<td>34.39</td>
<td>32.20+</td>
<td>29.32</td>
</tr>
<tr>
<td>PL(8)</td>
<td>34.70</td>
<td>32.19+</td>
<td>29.10</td>
</tr>
<tr>
<td>PL(9)</td>
<td>34.30</td>
<td>32.07</td>
<td>29.22</td>
</tr>
<tr>
<td>PL(10)</td>
<td>34.30</td>
<td>32.14</td>
<td>29.19</td>
</tr>
</tbody>
</table>

4.3 Evaluation

We compare our method with MERT and MIRA\(^1\) in two tasks, iterative training, and N-best list rerank. We do not list PRO Hopkins and May (2011a) as our baseline, as Cherry et al. Cherry and Foster (2012) have compared PRO with MIRA and MERT massively.

In the first task, we align the FBIS data (about 230K sentence pairs) with GIZA++, and train a 4-gram language model on the Xinhua portion of Gigaword corpus. A hierarchical phrase-based (HPB) model (Chiang, 2007) is tuned on NIST MT 2002, and tested on MT 2004 and 2005. All features are eight basic ones Chiang (2007) and extra 220 group features. We design such feature templates to group grammars by the length of source side and target side, (feat-type,\(a\leq\text{src-side}\leq\text{b},c\leq\text{tgt-side}\leq\text{d}\)), where the feat-type denotes any of the relative frequency, reversed relative frequency, lexical probability and reversed lexical probability, and [\(a, b\), [\(c, d\)] enumerate all possible subranges of [1, 10],

\(^1\)MIRA is from the open-source Moses Koehn et al. (2007)
as the maximum length on both sides of a hierarchical grammar is limited to 10. There are $4 \times 55$ extra group features.

In the second task, we rerank an N-best list from a HPB system with 7491 features from a third party. The system uses six million parallel sentence pairs available to the DARPA BOLT Chinese-English task. This system includes 51 dense features (translation probabilities, provenance features, etc.) and up to 7440 sparse features (mostly lexical and fertility-based). The language model is a 6-gram model trained on a 10 billion words, including the English side of our parallel corpora plus other corpora such as Gigaword (LDC2011T07) and Google News. For the tuning and test sets, we use 1275 and 1239 sentences respectively from the LDC2010E30 corpus.

### 4.3.1 Plackett-Luce Model for SMT Tuning

We conduct a full training of machine translation models. By default, a decoder is invoked for at most 40 times, and each time it outputs 200 hypotheses to be combined with those from previous iterations and sent into tuning algorithms.

In getting the ground-truth permutations, there are many ties with the same sentence-level BLEU, and we just take one randomly. In this section, all systems have only around two hundred features, hence in Plackett-Luce based training, we sample 30 hypotheses in an accumulative $n$best list in each round of training.

All results are shown in Table 4.2, we can see that all PL($k$) systems does not perform well as MERT or MIRA in the development data, this maybe due to that PL($k$) systems do not optimize BLEU and the features here are relatively not enough compared to the size
Figure 4.1: PL($k$) with 500 L-BFGS iterations, $k=1,3,5,7,9,12,15$ compared with MIRA in reranking.
of N-best lists (empirical Formula 4.5). However, PL(k) systems are better than MERT in testing. PL(k) systems consider the quality of hypotheses from the 2th to the kth, which is guessed to act the role of the margin like SVM in classification. Interestingly, MIRA wins first in training, and still performs quite well in testing.

The PL(1) system is equivalent to a max-entropy based algorithm Och and Ney (2002) whose dual problem is actually maximizing the conditional probability of one oracle hypothesis. When we increase the k, the performances improve at first. After reaching a maximum around $k = 5$, they decrease slowly. We explain this phenomenon as this, when features are rich enough, higher BLEU scores could be easily fitted, then longer ground-truth permutations include more useful information.

### 4.3.2 Plackett-Luce Model for SMT Reranking

After being de-duplicated, the N-best list has an average size of around 300, and with 7491 features. Refer to Formula 4.5, this is ideal to use the Plackett-Luce model. Results are shown in Figure 4.1. We observe some interesting phenomena.

First, the Plackett-Luce models boost the training BLEU very greatly, even up to 2.5 points higher than MIRA. This verifies our assumption, richer features benefit BLEU, though they are optimized towards a different objective.

Second, the over-fitting problem of the Plackett-Luce models PL(k) is alleviated with moderately large k. In PL(1), the over-fitting is quite obvious, the portion in which the curve overpasses MIRA is the smallest compared to other k, and its convergent performance is below the baseline. When k is not smaller than 5, the curves are almost above the MIRA line. After 500 L-BFGS iterations, their performances are no less than the
baseline, though only by a small margin.

This experiment displays, in large-scale features, the Plackett-Luce model correlates with BLEU score very well, and alleviates overfitting in some degree.
Conclusion

Our proposed PLRank, as a non-linear algorithm in the boosting framework, enriches the ListMLE framework. As far as we know, PLRank is the first list-wise based ranking system that in real-world datasets could match or outperform suitably the famous LambdaMART and McRank in terms of NDCG and ERR. In order to find out which ranking models would be effective in practice, we also explored the influence brought by different number of features.

We propose a minimum objective loss based tree construction algorithm in the boosting framework, and analyze the square error in the gradient boosting which is widely applied in practical learning to rank systems, as well as the weighted square error in (robust) LogitBoost. We successful build a relationship between our method and WSE in LogitBoost. We prove that when an objective loss function is derivative additive, our method is simplified into it, which provides a theoretical support for (robust) LogitBoost and point-wise based ranking systems. Based on our analysis, we show MART is an ideal connection to SE, WSE, and OLE, and we obtain a more concise formula for MART. Finally, for a fair and full empirical comparison of the three methods, we implement three quite strong ranking models, and examine them with a variety of configurations of regression trees in
two very large public datasets. Our results indicate that our proposed method is better used for McRank, LambdaMART and MART systems.

Last, we apply the Plackett-Luce model into another area, statistical machine translation, which is pretty similar to learning to rank problems in model training. When the number of features is increased beyond some threshold, the traditional training method in machine translation, namely MERT, becomes low efficient. In this case, the Plackett-Luce is applicable as well as simple to implement. More interesting, the Plackett-Luce model is capable of resisting to overfitting. Even the performance in training outperforms baseline by 3 BLEU points, the testing performance is still improved moderately.

Currently, We are working on combing MERT in the machine translation area, and LambdaRank in the learning to rank area, to propose a more efficient direct optimization method on objective loss. The former provides a quick line search algorithm, while the latter provides a gradient of any loss function for optimizing. But, unfortunately, so far we have not obtained reasonable results in the learning to rank datasets. In the further, we would do experiments on machine translation area.
Bibliography


Mark Hopkins and Jonathan May. Tuning as ranking. In EMNLP, 2011b.


Ping Li. Robust logitboost and adaptive base class (abc) logitboost. *UAI*, 2010b.


Donald Metzler and W Bruce Croft. Linear feature-based models for information retrieval. 

Robert C Moore and Chris Quirk. Random restarts in minimum error rate training for 
statistical machine translation. In *Proceedings of the 22nd International Conference on 
Computational Linguistics-Volume 1*, pages 585–592. Association for Computational 


Franz Josef Och and Hermann Ney. Discriminative training and maximum entropy models 


Tao Qin, Xu Dong Zhang, Ming Feng Tsai, De Sheng Wang, Tie Yan Liu, and Hang Li. Query-level loss functions for information retrieval. *Information Processing & 

Tao Qin, Tie Yan Liu, and Hang Li. A general approximation framework for direct opti-

Tao Qin, Tie Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on 


Cynthia Rudin. The p-norm push: A simple convex ranking algorithm that concentrates at the top of the list. JMLR, 2009.


90