JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 1-22 (2019)
DOI: 10.6688/JISE.201901_35(1).0001

Cost-Sensitive ListMLE Ranking Approach Based on Sparse Representation

DANQI DU, FENG ZHOU AND WEI XIONG
Department of Computer Science and Technology
GuiZhou University
GuiYang, 550025 P.R. China
E-mail: wawjddq@163.com

Learning-to-rank plays a pivotal role in information retrieval. To emphasize the top training of the permutation and improve the accuracy of the ranking model, several cost-sensitive listwise ranking algorithms have been proposed by incorporating the cost-sensitive learning idea into the ranking model. However, these methods ignore the impact of the high-dimensional features of the sample on the complexity of model, which results in low computational efficiency of the model. In this article, we proposed a cost-sensitive ListMLE ranking algorithm based on sparse representation which takes into account both the accuracy and computational efficiency of the ranking model. For the sake of achieving sparsity, the ℓ_1 regularized sparse term is added to the existing cost-sensitive ListMLE ranking model, and the global optimal parameters of the model are obtained by a simple yet efficient proximal gradient descent (PGD) learning method. Experiments performed on several benchmark datasets demonstrate that the proposed algorithm can improve empirical performance accuracy in building sparse model.

Keywords: learning to rank, cost-sensitive, sparse representation, ListMLE, proximal gradient descent

1. INTRODUCTION

Learning-to-rank plays a pivotal role in document retrieval. Given document relevance to a query, learning-to-rank can learn a ranking function automatically via machine learning techniques. Nowadays, the existing ranking approaches mainly fall into three categories: pointwise, pairwise and listwise. The pointwise and pairwise algorithms such as [1-4] formulate ranking problem into ordinal regression or classification problem. The listwise ranking algorithm such as AdaRank [5], ListNet [6], ListMLE [7], which directly modeling the ranking lists. For instance, ListMLE utilized the likelihood loss of the probability distribution based on Plackett-Luce model for optimization. Previous studies [6] show that the listwise approaches tend to perform better than the other two approaches on benchmark datasets.

However, both theoretical results and experiments [8, 9] show that listwise approaches cannot well capture the importance information of position. In other words, listwise approaches treat all documents equally, however, making errors between higher ranks and lower ranks should be punished more than making errors among lower ranks [3]. Thus, some improved listwise methods has been proposed, for instance, Jun Xu *et al.* construct different losses for misclassification of instance pairs between different rank pairs, and proposed cost-sensitive SVM for Ranking which proves to outperform Rank-

Received July 24, 2017; revised October 28, 2017; accepted January 2, 2018.
Communicated by Meng Chang Chen.

ing SVM in practice [10]. M. Lu *et al.* [11] put forward a cost-sensitive listwise framework by imposing weights for different documents and give a detail theoretical analysis about it. L. Yan [12] proposed a position-aware listwise method by imposing weights for different positions in ranking lists. Despite the success of these methods in emphasizing the top training of the permutation and improving the accuracy of the ranking model, they all ignore the impact of the high-dimensional features of the sample on the complexity of model. In real application, we face the situation that samples have high-dimensional features, the algorithm mentioned above make it more time-consuming for feature computation. Studies show that the entire ranking performance is dominated by a few decisive features, which indicate that current listwise methods suffer from major limitations of redundant features, time-consuming and the lower ranking accuracy [13, 14]. Therefore, it is desirable to incorporate sparse learning into cost-sensitive listwise model to improve computational efficiency.

Recently, many machine learning applications have been successfully implemented using sparse learning, which also works well in learning-to-rank, especially for tacking with the high-dimensional datasets. In practice, dealing with high-dimensional datasets faces two major challenges. In the one hand, it is very time-consuming to compute large number of features, such as features in Yahoo! learning to rank challenge [15]. In the other hand, inevitably, some noise or redundant features exists in datasets, in which situation models with sparsity constraints are very applicable for processing the high dimensional datasets. They hold several superiorities, such as high computational efficiency and strong generalization performance. However, only several algorithms have been proposed to tackle the issues of learning sparse models for ranking. Sun *et al.* put forward a ranking algorithm which incorporates ℓ_1 regularization into importance weighted binary classification [16]. Lai *et al.* formulated the sparse ranking as a convex optimization problem with the ℓ_1 regularization and proposed an efficient primal-dual framework based on Fenchel Duality theory [13]. Later, Lai *et al.* focus on efficient optimization approach of sparse learning to rank model and put forward a simple yet effective iterative algorithm to solve the sparse learning to rank [17]. However, all of the sparse ranking models mentioned above are based on pairwise ranking, no effort has been made to formulate sparse listwise ranking algorithms except the work [18], in which the authors proposed an expert listwise sparse learning to rank model through directly optimizing the loss function based on evaluation measure, and adopt a two-stage sparse learning including setting feature threshold and adding ℓ_1 norm, which may cause the learned features to be too sparse to produce a good performance. Besides, [18] has not provided a detail description about how to efficiently optimize the sparse model.

In order to improve ranking accuracy and computational efficiency of the listwise ranking model, this paper seeks to incorporate sparsity into cost-sensitive listwise method. In one hand, we consider the cost-sensitive ListMLE to improve the top ranking accuracy. In the other hand, to amend the impact of the high-dimensional features of the sample on the computational complexity of the model, we consider incorporating the sparsity into the cost-sensitive ListMLE model by adding a ℓ_1 regularization term. Our algorithm applies a simple yet efficient optimization strategy called proximal gradient descent (PGD) to get a global solution of the model. Moreover, we provide theoretical analysis and experiments of feature selection to justify that only a few strong features dominate the whole performance, which indicates the necessity of incorporating the sparsity into cost-sensitive listMLE.

Our contributions in this paper can be summarized in the following aspects:

(1) We propose a novel listwise ranking algorithm to incorporate sparsity into cost-sensitive listMLE ranking algorithm.
(2) We provide an efficient learning method for the proposed model, which is Proximal gradient descent (PGD) method with adaptive Lipschitz constant.
(3) We provide the feature selection algorithm designed for the proposed ranking model, and conduct experiments to explore the effects of sparse features on model.
(4) We conduct several comparison experiments to demonstrate that incorporating sparsity into cost-sensitive listwise algorithm will contribute to improving ranking accuracy and computational efficiency of the listwise ranking model.

The paper is organized as follows. In section 2, we provide a conclusion about some state-of-the-art methods related to our work. In Section 3, firstly, we describe the original cost-sensitive listwise ranking model and a proximal Gradient Descent learning approach. Secondly, we introduce the proposed algorithm. Lastly, an efficient feature selection algorithm for ranking model is introduced. Section 4 presents the experiments and analysis, including feature filtering process and comparison of different models. Section 5 concludes the paper.

2. RELATED WORK

2.1 Learning to Rank

Learning to rank is a new and popular topic in machine learning, which can be applied to wide fields including document retrieval [2], collaborate filtering [19], natural language processing [20] and so on. This paper focuses on the application of learning to rank for document retrieval. The methods of Learning-to-rank mainly fall into three categories: pointwise method, pairwise method and listwise method [1-3, 5, 6, 21]. Our proposed method belongs to the third category.

The listwise method contains two main streams: (1) Training model through minimizing the listwise loss function defined on the predicted list and the ground truth list. For example, ListNet [6] uses cross entropy loss as the surrogate loss to approximate the true loss. RankCosine [21] uses cosine loss to estimate the true loss. ListMLE [7] uses the negative logloss of a probability model to train model; (2) Directly optimize IR evaluation measures which amounts to minimizing different loss functions based on measures, such as AdaRank [5] and SVM_{map} [22] which aim to minimize upper bound of measure loss to obtain the optimal solution of the model.

2.2 Cost-Sensitive Learning to Rank

Though many learning to rank algorithms has been proposed, most of them ignore the importance of position that errors between higher ranks and lower ranks should be punished more than errors among lower ranks [3]. Few works are conducted on utilizing position information in ranking except [10-12]. For pairwise ranking, J. Xu *et al.* first attempted to incorporate cost-sensitive learning into the Ranking SVM and proposed cost-sensitive Ranking SVM which retains different losses for different rank pairs by bringing into penalty weights into rank pairs [10]. M. Lu *et al.* extend the cost-sensitive to listwise ranking [11], different from J. Xu [10], they modify the probability distribution by imposing different weights for documents. Y. Lan *et al.* proposed a position-aware ListMLE, which impose weights for different ranking position in loss function so that ListMLE can capture the position information [12].

All these methods adopt the weighted approach to implement cost-sensitive. Despite the success of these cost-sensitive ranking methods, they all ignore the impact of the high-dimensional features of the sample on the complexity of model. In this paper, we bring in sparsity to further improve the cost-sensitive listwise ranking methods.

2.3 Sparse Learning in Learning to Rank

Sparse learning has been successfully applied to many areas for coping with high-dimensional features. However, only several algorithms have been proposed to tackle the issues of learning sparse models for ranking. Sun *et al.* put forward a ranking algorithm which incorporates ℓ_1 regularization into importance weighted binary classification [16]. Lai *et al.* formulated the sparse ranking as a convex optimization problem with the ℓ_1 regularization and proposed an efficient primal-dual framework based on F. Duality theory [13]. Later, Lai *et al.* focus on efficient optimization approach of sparse learning to rank model and put forward a simple yet effective iterative algorithm to solve the sparse learning to rank [17]. To formulate sparse listwise ranking algorithms for expert search, Wang *et al.* [18] proposed an expert listwise sparse learning to rank model through directly optimizing the loss function based on evaluation measure, and adopt a two-stage sparse learning including setting feature threshold and adding ℓ_1 norm.

All of the sparse ranking algorithms mentioned above are based on that adding ℓ_1 penalty term into the loss function, the main difference is the learning methods for optimizing models. But Olga Krasotkina *et al.* point out that in certain cases, though adding ℓ_1 lasso penalty indeed produce sparsity, but it may cause the inconsistency for variable selection and biased [23], so they put forward a Hierarchical Bayesian Model for Sparse Learning to Rank [24]. Specifically, they use a Bayesian approach to variable approach in learning-to-rank which gives the strong probabilistic statement of shrinkage criterion for predictor selection, the experiment results demonstrate the effectiveness of their proposed method. Despite the success of these methods, it needs a further exploration to find more efficient sparse learning to rank methods.

3. PROPOSED METHOD

3.1 Cost-Sensitive ListMLE

The listwise approach minimizes the loss function defined between the ranking lists and the ground truth lists. However, most listwise approaches ignore the position importance that is documents with high levels ought to be ranked closer to the top position in permutation. If the document with high level is ranked close to bottom position, a bigger penalty should be imposed into the loss function. Besides, the ranking order on the top of the permutation should be emphasized. In order to carry out the above idea, a natural way is to consider incorporating cost-sensitive idea into the listwise losses. Specifically, to set different weights for the documents, the loss function of the cost-sensitive ListMLE on a query is defined as (M. Lu *et al.* [11])

$$L(w)=\frac{1}{DCG_{\hat{g}}@k}\sum_{j=1}^{n}\beta_j lb(1+\sum_{t=j+1,y_j>y_t}^{n}\frac{\alpha_{j,t}}{\alpha_{j,j}}\exp(f(x_t)-f(x_j))). \tag{1}$$

In which the ranking function (score) is linear, *i.e.*, $f(x)$ is the inner product between x and $\mathbf{w}$ (model parameters), *i.e.*,

$$f(x)=\langle\mathbf{w},\mathbf{x}\rangle=\mathbf{w}^T\mathbf{x}. \tag{2}$$

$\alpha_j=(\alpha_{j,1},\ldots,\alpha_{j,n})$ and its components are non-negative. $\alpha_{j,i}$ denotes the weight of document d_i corresponding to query q_j, β_j is the weight brought into the cost-sensitive ListMLE. And

$$DCG_{\hat{g}}@k=\sum_{j=1}^{n}\frac{2^{y_j}-1}{lb(1+\hat{g}(j))}I[\hat{g}(j)\le k], \tag{3}$$

in which $\hat{g}(j)$ denotes an approximation of the true position of the document d_i, and

$$\hat{g}(j)=1+\textstyle\sum_{i=1}^{n}I[y_i>y_j], \tag{4}$$

and y_i denotes the true relevance label of the document d_i, $I[x]$ is the indicate function, and when x is true, $I[x]$ equals 1, otherwise 0.

A critical issue arising here is to design a reasonable approach to compute the weights of documents, *i.e.*, compute α and β. Lu *et al.* deduced the weights of the document pair in theory. They brought into two functions,

$$a(i)=2^{y_i}-1 \tag{5}$$

$b(j)$ and it's gradient are defined as

$$b(j)=\begin{cases}\frac{1}{lb(1+j)} & j\le k\\ \frac{1}{lb(1+k)} & j>k\end{cases}\quad \nabla b(j)=\begin{cases}\frac{-\log 2}{(1+j)[\log(1+j)]^2} & j\le k\\ 0 & j>k\end{cases}. \tag{6}$$

Thus β_j and $\alpha_{j,i}$ are computed as:

$$\beta_j=\sum_{y_j>y_i}[a(i)\nabla b(\hat{g}(i))-a(j)\nabla b(\hat{g}(j))], \tag{7}$$

$$\frac{\alpha_{j,i}}{\alpha_{j,j}}=\begin{cases}\frac{a(i)\nabla b(g(i))-a(j)\nabla b(\hat{g}(j))}{\beta_j} & \beta_j\ne 0\\ 0 & \beta_j=0\end{cases}. \tag{8}$$

3.2 An Efficient Proximal Gradient Descent (PGD) Method for Sparse Model

3.2.1 Sparse learning-to-rank algorithm

In the real application of learning to rank, the feature dimension of the sample tends to be too high to compute efficiently. Recently researchers have found that only a few strong features control the whole performance of the ranking model, thus they brought in the sparse learning to amend the problem [13, 16-18]. Sparse learning has been widely used in many machine learning areas, such as compressed sensing and computer vision, the core idea of it is to let most coefficients equal zero while ensuring that the model error continues to decline. Finally a model with only a few nonzero coefficients is by decomposing high-dimensional data linearly.

At first, we give a description of the notations used in this article. Without loss of generality, we use lower case letters to denote scalars, and use bold letters to denote vectors. $\langle x, y\rangle$ represents the inner product between x and y. A norm of a vector $\mathbf{w}$ is denoted by $||\mathbf{w}||$. The ℓ_1 norm is defined as $||\mathbf{w}||_1 = \sum_i|\mathbf{w}_i|$. For any function L, let $L'(x)$ denote the gradient of L at x. We give a simple visit to the general sparse learning-to-rank problem. The training data is given by $S := (x_i, y_i, q_i)$, $i = 1, \ldots, n$ with $x_i = (x_i^{(1)}, x_i^{(2)}, \ldots, x_i^{(n)})\in\mathbf{R}^d$, y_i denotes relevance label, and q_i is a query. Given the training data, linear ranking model aims to find the coefficients vector $\mathbf{w}\in\mathbf{R}^d$, so that given a new query, the model can predict where all the documents related to the query ought to be ranked. Let $L(\mathbf{w})$ denotes the loss function, which represents the gap between the $\langle\mathbf{w}, x_i\rangle$ and label y_i. Therefore, to improve the prediction accuracies of the model, it is natural to minimize the loss function $L(\mathbf{w})$.

To achieve sparsity, we incorporate ℓ_1 regularization into the ranking model and optimize the following problem

$$\min_{\mathbf{w}} Loss(\mathbf{w}) = \min_{\mathbf{w}} L(\mathbf{w}) + \lambda \,||\, \mathbf{w} \,||_1. \tag{9}$$

Where λ is the balance factor to trade off the training error and the model complexity. Here $L(\cdot)$ is a continuously differentiable convex function.

Note that the weight coefficients $\mathbf{w}$ are sparse means that only the features corresponding to the non-zero coefficients of $\mathbf{w}$ will affect the model performance. Thus, the result of solving the ℓ_1 norm regularization is that we obtain the model with few dominant features of the original features; In other words, the learning method based on ℓ_1 regularization is intrinsically an embedded feature selection method. The feature selection process is integrated with the learner training process, and two processes are completed simultaneously.

3.2.2 An efficient Proximal Gradient Descent (PGD) method for ℓ_1 regularization learning

Motivated by proximal forward-backward splitting method proposed in [25], in this paper, we describe an efficient and convergence-provable optimization algorithm to solve the problem in Eq. (9), called proximal gradient descent (PGD). Specifically, let ∇ denotes the differential operator, as to the optimization problem in Eq. (9), if $L(\mathbf{w})$ is differentiable, and $\nabla L(\cdot)$ satisfies the L-Lipschitz condition, then there exists a constant $L > 0$ such that

$$\| \nabla L(\mathbf{w}') - \nabla L(\mathbf{w}) \|_2^2 \le L \| \mathbf{w}' - \mathbf{w} \|_2^2 \ (\forall \mathbf{w}, \mathbf{w}'). \tag{10}$$

In that way $L(\mathbf{w})$ can be estimated by a second-order Taylor expansion to be

$$\begin{aligned} \hat{L}(\mathbf{w}) &\simeq L(\mathbf{w}_k) + < \nabla L(\mathbf{w}_k),\ \mathbf{w} - \mathbf{w}_k > + \frac{L}{2} \| (\mathbf{w} - \mathbf{w}_k) \|^2 \\ &= \frac{L}{2} \| \mathbf{w} - (\mathbf{w}_k - \frac{1}{L} \nabla L(\mathbf{w}_k)) \|_2^2 + const, \end{aligned} \tag{11}$$

where *const* is a constant independent of **w**, and $\langle \cdot\ ,\ \cdot \rangle$ denotes inner product. Obviously, the minimum value of Eq. (10) is taken at $\mathbf{w}_{k+1}$ as follows

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \frac{1}{L} \nabla L(\mathbf{w}_k). \tag{12}$$

So, if $L(\mathbf{w})$ is minimized by the gradient descent method, each iteration of the gradient descent is actually equivalent to minimizing the quadratic function $\hat{L}(\mathbf{w})$. If we extend the idea to Eq. (9), then we can similarly obtain the update formula of each iteration as follows

$$\mathbf{w}_{k+1} = \arg\min_{\mathbf{w}} \frac{L}{2} \| \mathbf{w} - (\mathbf{w}_k - \frac{1}{L} \nabla L(\mathbf{w}_k)) \|_2^2 + \lambda \| \mathbf{w} \|_1 . \tag{13}$$

Eq. (13) shows that when conducting each iteration of the gradient descent, the ℓ_1 norm is taken into consideration at the same time.

As to Eq. (13), we can compute $\mathbf{z} = \mathbf{w}_k - \frac{1}{L}\nabla L(\mathbf{w}_k)$ at first, then to solve

$$\mathbf{w}_{k+1} = \arg\min_{\mathbf{w}} \frac{L}{2} \| \mathbf{w} - \mathbf{z} \|_2^2 + \lambda \| \mathbf{w} \|_1. \tag{14}$$

Let $\mathbf{w}^i$ denotes the ith component of **w**, we expand the Eq. (14) and find that there exists no component like $\mathbf{w}^i\ \mathbf{w}^j$, which indicates that components of **w** are independent of each other. So Eq. (14) has a closed-form solution as follows

$$\mathbf{w}_{k+1}^i = \begin{cases} \mathbf{z}^i - \frac{\lambda}{L} & \text{if}(\lambda / L < z^i) \\ 0 & \text{if}(|\mathbf{z}^i| \le \lambda / L) \\ \mathbf{z}^i - \frac{\lambda}{L} & \text{if}(\mathbf{z}^i < -\lambda / L) \end{cases} . \tag{15}$$

In which $\mathbf{w}_{k+1}^i$ and $\mathbf{z}^i$ are the ith component of $\mathbf{w}_{k+1}$ and **z**, respectively. Thus the minimization problem with ℓ_1 regularization can be solved efficiently via PGD. Note that we did not specify how to choose the appropriate Lipschitz constant L what we will see in section 3.3.2.

3.3 Cost-Sensitive ListMLE Ranking Approach Based on Sparse Representation

3.3.1 Solving the objective loss function

In our work, we propose a novel ranking model called Sparse cost-sensitive List-

MLE. We assume that a linear model $f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$, where $\mathbf{w} \in \mathbf{R}^d$ is a parameter vector of the model, $\mathbf{x}$ is the feature vector. Without regularization constraints, the result of $\mathbf{w}$ in the model parameter space tends to over fitting, so in this paper we introduce a sparse factor to solve. Since ℓ_0 norm is hard to compute and optimize, while the ℓ_1 norm is the optimal convex approximation of the R_0 norm, thus we use the ℓ_1 norm as the regularization parameter. Given training instances $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$, considering the cost-sensitive ListMLE and combine the sparse learning theory, we define an objective loss function of Sparse cost-sensitive ListMLE ranking Approach as follows

$$Loss(\mathbf{w}) = \frac{1}{DCG_{\hat{g}} @ k} \sum_{j=1}^{n} \beta_j lb\Big(1 + \sum_{t=j+1, y_j > y_t}^{n} \frac{\alpha_{j,t}}{\alpha_{j,j}} \exp(f(x_t) - f(x_j))\Big) + \lambda \, \| \mathbf{w} \|_1 . \tag{16}$$

In which the weights β_j and $\alpha_{j,t}/\alpha_{j,j}$ can be computed based on Eqs. (7) and (8), which is the key factor to emphasis the top training of the ranking list. λ is the balance factor to trade off the training error and the model complexity. We can see that loss function $L(\mathbf{w})$ like Eq. (1) is continuously differentiable convex function without ℓ_1 regularization, it's enough to use the gradient descent method to obtain the global optimal solution.

However, when we add the ℓ_1 regularization to the target function like Eq. (16), gradient descent method is no longer valid since ℓ_1 is not differentiable everywhere, so we cannot apply the Newton descent algorithm used in cost-sensitive ListMLE to optimize our Loss function. To optimize the convex but nondifferentiable loss function, a common pattern is to use subgradient descent learning algorithm. However, the convergence rate of the subgradient descent method is as low as $O(1/\varepsilon^2)$, which is not the expected optimization precision, thus this paper applys the aforementioned PGD method to obtain the solution of the proposed model.

3.3.2 Setting the Lipschitz constant L

Before using the PGD method, it's necessary to consider how to set the Lipschitz constant L. Some researchers have provided some strategies to estimate L [26, 27]. However, given a function $f(\cdot)$, computing the large eigenvalue of the Hessian is costly. To overcome this weakness, we introduce an efficient method to find an upper bound of L, denoted by L_0. The truth is, for any $\mathbf{w}$ and $\mathbf{x} \in \mathbf{R}^d$, proving $\|f''(\mathbf{w})\| \leq L$ is equivalent to proving $(\mathbf{x} f''(\mathbf{w}), \mathbf{x}) \leq L \sum_{i=1}^{d} (\mathbf{x}_i)^2$. According to that, we can determine the parameter L_0. Theorem 1 is given to compute the parameters of the cost-sensitive ListMLE loss function.

Theorem 1: The Lipschitz constant L of the cost-sensitive *ListMLE* loss function $L(\mathbf{w}) = \frac{1}{DCG_{\hat{g}} @ k} \sum_{j=1}^{n} \beta_j lb(1 + \sum_{t=j+1, y_j > y_t}^{n} \frac{\alpha_{j,t}}{\alpha_{j,j}} \exp(f(x_t) - f(x_j)))$ is less than or equal to $\frac{\boldsymbol{R}^T \boldsymbol{R}}{4 DCG_{\hat{g}} @ k} \sum_{j=1}^{n} \beta_j$, in which $\boldsymbol{R}$ denotes the upper bound of the feature vector of the document.

The detailed steps for parameters setting are given in Appendix A5, and the steps are also applicable to the calculation of other loss function.

To further improve the Lipschitz constant L of Sparse cost-sensitive ListMLE, we adopt the adaptive Lipschitz constant L proposed by Lai *et al.* [17]. In Algorithm 1, we give the pseudo-code for adaptively finding Lipschitz constant.

Algorithm 1: Sparse cost-sensitive ListMLE algorithm with adaptive L

Input: γ is the decay speed of Lipschitz constant L in each iteration, p indicates that the algorithm performs p times at most.
Output: the vector $\mathbf{w}$ of estimations parameters of the model.

```
1: Calculate $L_0$ according to Theorem 1.
2: Initialize: $\mathbf{w}_0 = \mathbf{0}$
3: Let $L = L_0/\gamma^p$
4: for $k = 1$ to $p$ do
5:     Calculate the gradient of loss in the last iteration: $grad = \nabla L(\mathbf{w}_k)$
6:     while true do
7:         $\mathbf{w}_{k+1} = \arg\min_{\mathbf{w}} \frac{L}{2} \| \mathbf{w} - (\mathbf{w}_k - \frac{1}{L} grand) \|_2^2 + \lambda \| \mathbf{w} \|_1$
8:         if $L(\mathbf{w}_{k+1}) \le L(\mathbf{w}_k) + < grad, \mathbf{w}_{k+1} - \mathbf{w}_k > + \frac{L}{2} ||\mathbf{w}_{k+1} - \mathbf{w}_k||^2$ then
9:             break;
10:        end if
11:        $L = \gamma L$
12:    end while
13: end for
```

In our algorithm, we first calculate the Lipschitz constant L_0 to be the upper bound of L given in Theorem 1. Then for obtaining adaptive Lipschitz constant L, we initialize L as L_0/γ^p, in each iteration, we reduce L by multiplying γ. And p is the maximum number of iterations, in each iteration, for computational efficiency, we calculate the gradient of loss in the last iteration based on the loss and weight vector $\mathbf{w}_k$ in in last iteration. Then by minimizing the loss function in Eq. (14), we obtain the weight vector $\mathbf{w}_{k+1}$ in this iteration, and update it until satisfying the condition of Eq. (10), this procedure would finally stops and performs p times at most.

In the next section, we will give our analysis about the reason to set a smaller Lipschitz constant. According to that analysis, we know the number of iterations required to achieve ε-solution is $T = O(\frac{L}{2\varepsilon} \| \mathbf{w}^* - \mathbf{w}_0 \|_2^2)$. The larger L, the more the number of iterations. Hence, in order to speed up the training of the proposed model, it's necessary to find a smaller Lipschitz constant.

3.4 Convergence Analysis

In this section, we give a simple analysis about the convergence rate of Sparse cost-sensitive ListMLE. Let $\mathbf{w}_t$ denotes the coefficient sequence obtained by our algorithm. We denote $\mathbf{w}^*$ as the optimal solution of $Loss(\mathbf{w})$, that is $\mathbf{w}^* = \arg\min_{\mathbf{w}} Loss(\mathbf{w})$. We also denote $A_t = \max_{\mathbf{w}} \langle L'(\mathbf{w}_t), \mathbf{w}_t - \mathbf{w} \rangle - \frac{L}{2} \| \mathbf{w}_t - \mathbf{w} \|_2^2 + \lambda \| \mathbf{w}_t \|_1 - \lambda \| \mathbf{w} \|_1$, and define $\varepsilon_t = L(\mathbf{w}_t) - L(\mathbf{w}^*)$ as errors between the optimal solution and the solution at iteration t. The algorithm obtains an ε-accurate solution at iteration t when $\varepsilon_t \le \varepsilon$.

Since $L(\mathbf{w})$ and $||\mathbf{w}||_1$ are all convex function with respect to $\mathbf{w}$, so $Loss(\mathbf{w})$ is also convex function. Because our algorithm satisfies the same condition as [17], thus we obtain a similar Theorem 2.

Theorem 2:
(a) $L(\mathbf{w}_t) - L(\mathbf{w}_{t+1}) \geq A_t \geq 0$.
(b) If $A_t = 0$ then $\mathbf{w}_t$ is an optimal model parameters vector of $L(\mathbf{w})$.
(c) $\mathbf{w}_t \rightarrow \mathbf{w}^*$, shows that our sparse algorithm eventually converges to the optimum of $L(\mathbf{w})$.
(d) The sparse cost-sensitive ListMLE algorithm terminates after at most $T = \frac{L}{2\varepsilon}(\| \mathbf{w}^* - \mathbf{w}_0 \|_2^2)$ iterations, and at termination $L(\mathbf{w}_T) - L(\mathbf{w}^*) \leq \varepsilon$.

The (a), (b) and (c) in Theorem 2 indicate that our loss function continues to decline, and eventually converges to an optimal value, and (d) gives the upper bound of required iterations to obtain ε-accurate solution.

3.5 Feature Selection for Ranking Model

Feature selection plays a pivotal role in ranking, it contributes to enhance accuracy and improve the efficiency of training [14], moreover, it is a powerful means to avoid over-fitting [28]. In order to explore whether sparse learning works, we give a simple example to describe it at first. Then we demonstrate a feature selection method for ranking, which is intrinsically a greedy search algorithm.

3.5.1 The effectiveness of features for sparse ranking model

We make experiment on MQ2008 data set in Letor4.0 [29], which contains 46 features. We take a feature of 46 features to train sparse cost-sensitive ListMLE each time, until all the 46 features are taken once, if we use NDCG@10 as evaluation metric, we will get 46 NDCG@10 values (showed in Fig. 1), and then we construct a random predictor which use all the features as follows: we randomly initialize a feature weight vector $\mathbf{w}(\mathbf{w} = (w_1, w_2, ..., w_{46}), \forall i, w_i \geq 0, \sum_{i=1}^{46} w_i = 1)$, and we can predict the scores of all documents, and rank the documents in the decrease order, so we can compute NDCG @10. We repeated this procedure 10 times, then we get the average of the 10 NDCG@ 10 values. The results are shown in Fig. 1.

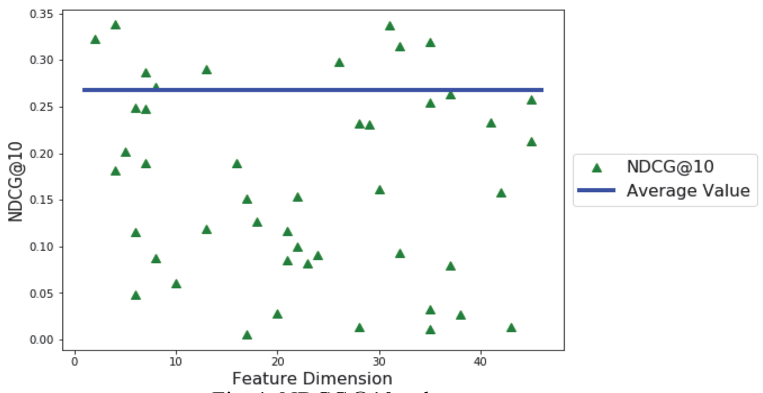


Fig. 1. NDCG@10 value.

According to the result of Fig. 1, only a few strong features (such as TFIDF, BM25, PageRank) have obviously higher NDCG@10 value than the average of ten random ranking values, while many poor features whose NDCG@10 value are much lower than the average value. The result indicates that only a few strong features are the key factors in controlling the overall performance.

3.5.2 Greedily selecting features for ranking

Algorithm 2: Greedy feature selection algorithm for Sparse Cost-Sensitive ListMLE

```
Input: an expected feature dimension $K$, an error threshold $\varepsilon$, a selected feature set $R$,
and the BaseRanker $S$.
Output: The set of features be selected: $R$
1: Calculate $\mathbf{A}$ by Pearson Correlation method.
2: Initialize: $\mathbf{w}_0 = \mathbf{0}$, $\varepsilon = 0.005$, $R = \varnothing$.
3: for $t = 1$ to $d$ do
4:     Select a feature $i$ which is not in $R^{t-1}$.
5:     Use the feature set $R^{t-1} \cup i$ to train model (update $\mathbf{w}$)
6:     if $S(R^{t-1} \cup \{i\}) - S(R^{t-1}) \le \varepsilon$ then
7:         $R^t = R^{t-1}$
8:     else
9:         $R^t = R^{t-1} \cup i$
10:    end if
11:    while true do
12:        Choose a feature $j$ from the selected features $R^t$
13:        Use the feature set $R^t \setminus \{j\}$ to train model (update $\mathbf{w}$)
14:        if $S(R^t \setminus \{j\}) - S(R^t) \le \varepsilon$ then
15:            $R^t = R^t \setminus \{j\}$
16:        end if
17:    end while
18:    if $NumberOf(R^t) \ge K$ then
19:        return $R^t$;
20:    end if
21: end for
```

Due to the gap between classification and ranking, the feature filter approach for classification is not suitable for ranking, thus a few techniques for feature selection in ranking has been proposed, such as [14, 30, 31], all of which take account two principles of feature selection: maximized the importance of features and minimized similarity between features. In this paper, we adopt the FBPCRank algorithm [30] as the feature selection method for its computational efficiency and good performance, different from it, we use our sparse cost-sensitive ranker as the BaseRanker, which is denoted by S to measure the loss produced by feature set F, $S(F) = \min CL(\mathbf{w}) + \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w}$, in which F is the set of all features, $L(\mathbf{w})$ denotes the loss of Sparse Cost-Sensitive ListMLE, $\mathbf{w}$ is the weight vector of the model, $\mathbf{A}$ is the similarity matrix between features, C is the balance factor between the two terms. The pseudo-code of greedy feature selection algorithm for Sparse Cost-Sensitive ListMLE is as Algorithm 2.

The process of Algorithm 2 is as follows: (1) Before training model, calculate similarity matrix **A** in order to avoid computing **A** each iteration and improve computational efficiency; (2) Initial several hyper-parameters such as the weight vector **w**, the threshold ε of error, the selected feature set R; (3) Using Forward Propagation and Pearson Correlation greedy algorithm to select a feature; (4) Using Backward Propagation and Pearson Correlation greedy algorithm to remove the current useless features; (5) Repeat the process of (3) and (4) until the number of features achieves the expected K.

4. EXPERIMENT AND ANALYSIS

To evaluate the performance of our proposed method, we conduct three contrast experiments on several publicly available benchmark datasets: the proposed method compare to some state-of-the-art dense model, compare to dense cost-sensitive dense model compare to learning-to-rank models.

The results show that: (1) The proposed Sparse Position-Sensitive ListMLE ranking model, with the document weight and the sparse-inducing ℓ_1 constraint, significantly outperforms Cost-Sensitive ListMLE (CS_ListMLE) and Cost-Sensitive RankSVM (CS_ RankSVM) ranking on TD2003 and TD2004, while their performance is comparable on OHSUMED; (2) When compared with other sparse ranking algorithms, Sparse_CS_ListMLE can be sparser and achieve excellent performance on both accuracies and efficiency; (3) Compared to other state-of-the-art algorithms for ranking, Sparse_CS_ListMLE also achieves competitive performance on ranking accuracies.

4.1 Data Sets

We make contrast experiments on the Letor 3.0 [32] and Letor 4.0 datasets [29], which are publicly available benchmarks for learning-to-rank. In our experiments, we evaluate the performances of ranking approaches on two small datasets, TD2004 and OHSUMED in Letor 3.0 and one large-scale dataset MQ2008 in Letor 4.0. In each dataset, many features are related to the query-document pairs, which are extracted from the documents and covers different levels such as TF, IDF (low level) and BM25 (high level).

Each dataset we used in our experiment have already been separated into 5 folds. In each experiment, three folds were merged as training dataset, one fold was used as validation datasets, the left one was used as test datasets. For each experiment, we randomly combined the training set and the test set, and the final sorting performance was the average of 5 sub experimental performance tests.

4.2 Evaluation Measures

MAP [33] and NDCG@k(N@k) [34] are used as the evaluation measures to evaluate the performance of ranking models. NDCG@k is used to evaluate performance of the top permutation in a sorted list. We set the truncation level k in NDCG@k as 1, 3, 5, 10.

4.3 Experiment Setting

We set the parameters of Sparse position-sensitive ListMLE as follows, we fix $\gamma = 2$, $p = 8$ in all our later experiments, but the balance factor λ is chosen in the set 10^{-7}, 10^{-6}, …, 10^{-2}, 10^{-1} by 5-folds cross validation, and our algorithm stops until satisfying the condition of $||L'(\mathbf{w}_t)||_\infty \leq 0.001 \times ||L'(\mathbf{w}_0)||_\infty$ or reaches the maximum iteration $T = 1000$.

4.4 Feature Filtering Contrast Experiment Before Training Model

We choose MQ2008 as the dataset and use 5-fold cross validation to determine C. According to Algorithm 2, we initialize several hyper-parameters such as $\mathbf{w}_0 = \mathbf{0}$, $\varepsilon = 0.005$, $R = \varnothing$, $K = 15$. Compute the similarity matrix $\mathbf{A}$ between features. The algorithm outputs a set R which is composed of dominated features and is shown in Table 1.

Table 1. Features be chosen in MQ2008 (including 46 features).

MQ2008	Features be chosen
Fold 1	23 19 29 18
Fold 2	39 29 28 19 21 38
Fold 3	29 26 39 19 12 32
Fold 4	23 29 19
Fold 5	36 25 42 18

We make the union of the above 5-fold results, which shows that about 28% features are selected. We conduct a contrast experiment to explore whether using the selected sparse features to train model outperforms using all features, the result is as Fig. 2. According to the result, it is better to use all the features to train the model. We explain why this happened, on one hand, the datasets such as MQ2008 in Letor4.0 has been streamlined compared to the original dataset in practice, the feature dimension is not particularly large, so the advantage of sparse features is not obvious. On the other hand, the proposed sparse model with l_1 regularization already has feature selection ability in some degree, so the selected features used for training the proposed sparse model is too sparse so that it is easy to cause the model to be overfitting, in this case, it is necessary to set an appropriate feature dimension, *i.e.*, a suitable K.

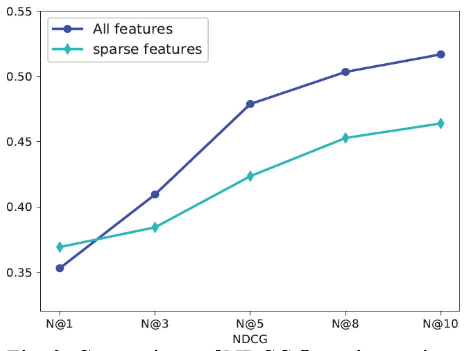


Fig. 2. Comparison of NDCG@n values using different feature sets.

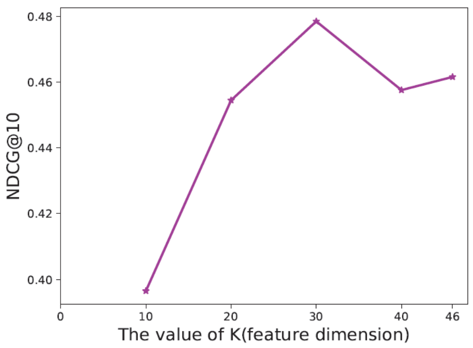


Fig. 3. The effect of different feature dimension on NDCG@10. K is the parameter of feature dimension.

To explore the effects of the feature sparse degree for ranking, the paper compare NDCG@10 values of Sparse Cost Sensitive ListMLE in different feature dimensions, the feature dimension *K* is set to be 10, 20, …, 40, 46, and we conduct Algorithm 2 to obtain the five selected feature sets, which then were put into model respectively, the NDCG@10 are displayed in Fig. 3.

In Fig. 3, the value of abscissa represents dimension of the selected features, and the value of ordinate represents the value of NDCG@10. If *K* is fixed on a certain value that means the selected features with *K* dimension are the most influential features in dataset, and the result in Fig. 3 indicates that when choosing 30 as the feature dimension on MQ2008 dataset, the NDCG@10 value is the maximum. So it's desirable to apply sparse algorithm to select strong features to train model, but whose good performance is based on the condition that an approximate feature dimension is chosen. It also justify that the whole ranking performance is dominated by a few strong features.

4.5 Comparing to Dense Cost-Sensitive Ranking Model

We focus on exploring if our sparse model is superior than the dense model. We conduct two contrast experiments on between cost-sensitive ListMLE (CS_ListMLE) [11] and our proposed algorithm (Sparse_CS_ListMLE). One is concerned about the ranking accuracies, another is the training time.

4.5.1 Ranking accuracy between CS_ListMLE and sparse_CS_ListMLE

The performance of Sparse_CS_ListMLE, CS_ListMLE [11] and CS_RankSVM [10] on four datasets are shown in Tables 2 and 3. In order to validate the effectiveness of Sparse_CS_ListMLE in a more comprehensive way, we set different parameters during the training process. CS_ListMLE@*k* indicates that cost-sensitive ListMLE concentrates on rank order of the top *k* documents in the ranked list. In other words, CS_ListMLE@k directly optimizes NDCG@*k*, and *k* takes 1, 3, 5, 8 and 10 in the experiment. The statistiCS_for sparse_CS_ListMLE and CS_RankSVM [10] are obtained in the same experimental environment.

Table 2. Ranking accuracies on OSHUMED.

Methods	N@1	N@3	N@5	N@10	MAP
CS_RankSVM	0.4486	0.3774	0.3638	0.3486	0.3534
CS_ListMLE@1	0.4505	0.3825	0.3792	0.3765	0.3420
CS_ListMLE@3	0.4493	0.3818	0.3785	0.3762	0.3422
CS_ListMLE@5	0.4498	0.3822	0.3788	0.3764	0.3432
CS_ListMLE@10	0.4502	0.3824	0.3790	0.3764	0.3434
Sparse_CS_ListMLE@1	**0.4505**	0.3987	**0.4022**	**0.3986**	**0.3576**
Sparse_CS_ListMLE@3	0.4496	**0.4020**	0.3935	0.3928	0.3553
Sparse_CS_ListMLE@5	0.4493	0.3948	0.3930	0.3922	0.3547
Sparse_CS_ListMLE@8	0.4502	0.3935	0.3928	0.3916	0.3545
Sparse_CS_ListMLE@10	0.4498	0.3939	0.3931	0.3925	0.3558

According to Tables 2 and 3, we can observe that Sparse_CS_ListMLE with sparse inducing ℓ_1 norm consistently outperforms CS_ListMLE and CS_RankSVM in terms of

NDCG at top 10. Compared to CS_RankSVM and CS_ListMLE, Sparse_CS_ListMLE outperforms CS_ListMLE at NDCG@10. We conduct *t*-test on the improvement of Sparse_CS_ListMLE over CS_ListMLE on the two datasets in terms of NDCG@3 to NDCG@10. For TD2004 dataset, the result shows that the improvement of Sparse_CS_ListMLE over CS_ListMLE and CS_RankSVM with a *p*-value of less than 0.036, which demonstrates that the significant effectiveness of our ranking approach based on sparse learning. However, for OHSUMED dataset, Sparse_CS_ListMLE behaves somewhat differently, it is not statistically significant in terms of *t*-test (*p*-value > 0.05). The reason for these results is that Sparse_CS_ListMLE with sparsity constraints can automatically filter redundant features while reducing the impact of noise features on the ranking model, so Sparse_CS_ListMLE perform better than original CS_ListMLE. However, Sparse_CS_ListMLE and CS_ListMLE concerned more on the relevant documents on the top of permutation, that is relevant documents have a greater impact on our algorithm than irrelevant documents, and as to OHSUMED, the relevant documents occupy a very small proportion for most queries, so sparsity cannot make a big difference in results. On the contrary, the relevant documents occupy a relatively large proportion on TD2004, therefore, the difference between Sparse_CS_ListMLE and CS_ListMLE is obvious.

Table 3. Ranking accuracies on TD2004.

Methods	N@1	N@3	N@5	N@10	MAP
CS_RankSVM	0.4573	0.4036	0.3918	0.4436	0.3631
CS_ListMLE@1	0.4678	0.4273	0.4224	0.4445	0.3649
CS_ListMLE@3	0.4672	0.4292	0.4196	0.4497	0.3663
CS_ListMLE@5	0.4673	0.4207	0.4074	0.4449	0.3667
CS_ListMLE@10	0.4663	0.4091	0.4067	0.4451	0.3645
Sparse_CS_ListMLE@1	0.4838	0.4450	**0.4324**	**0.4676**	**0.3844**
Sparse_CS_ListMLE@3	**0.4956**	**0.4507**	0.4304	0.4652	0.3814
Sparse_CS_ListMLE@5	0.4834	0.4486	0.4320	0.4580	0.3775
Sparse_CS_ListMLE@8	0.4884	0.4489	0.4323	0.4585	0.3793
Sparse_CS_ListMLE@10	0.4889	0.4482	0.4295	0.4580	0.3756

4.5.2 Training time between CS_ListMLE and Sparse_CS_ListMLE

To validate the computational efficiency of the sparse cost-sensitive model, we make a comparison about training time between CS_ListMLE [11] and Sparse_CS_ListMLE on MQ2008 (including 5 folds). The main difference between is that the latter is a sparse model with a ℓ_1 regularization term. Since the regularization parameter λ controls the effect of the sparse term on the loss function, so λ is the main factor that makes the difference of training time, and we choose the best λ. We take the early stopping strategy and set a loss threshold *error* = 0.05 for two models, when the loss has been less than *error* for 10 times, the algorithm stop immediately and record the training time. The results are shown in Table 4.

The larger the regularization parameter λ, the sparser the model is. From Table 4, we can observe that: (1) a larger λ corresponding to a shorter training time, which indicates that a sparser model has less computational cost; (2) In 5 folds experiment, the training times of Sparse_CS_ListMLE are less than CS_ListMLE, meanwhile, the NDCG@10 values of the former are larger than the latter. The main reason is that sparse model with ℓ_1 regularization term can filter the redundancy features and simply the model, training time is relatively reduced.

Table 4. Training time on MQ2008.

Method	CS_ListMLE			Sparse_CS_ListMLE	
	Time(s)	NDCG@10	λ	Time(s)	NDCG@10
Fold1	29.63	0.4417	0.01	20.86	0.4673
Fold2	24.16	0.4451	0.1	18.92	0.4782
Fold3	25.73	0.4263	0.1	19.34	0.4528
Fold4	31.62	0.4396	0.01	20.45	0.4635
Fold5	30.32	0.4122	0.01	21.63	0.4512

4.6 Comparing to Sparse Ranking Model

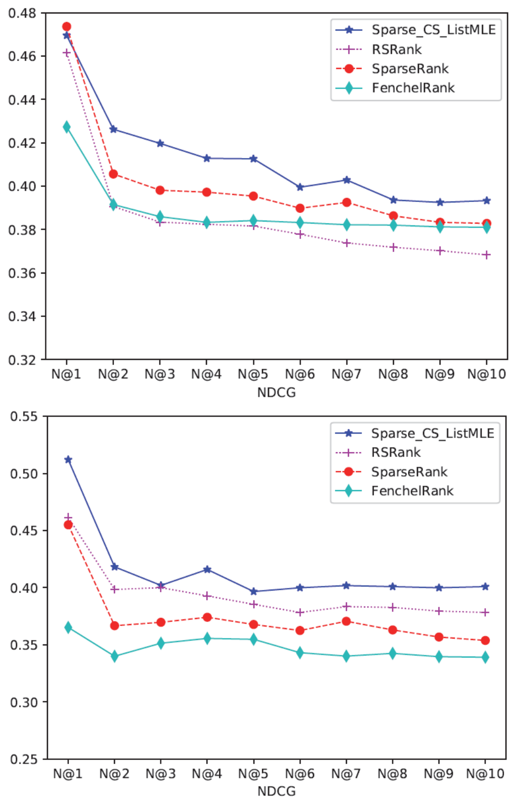
In this section, we are concerned about whether Sparse_CS_ListMLE overperforms other sparse ranking models or not, here we still utilize the Spars_CS_ListMLE@1 to represent Spars_CS_ListMLE to measure the performance. We designed experiments from two aspects; (1) Comparing the ranking accuracies of all sparse ranking models; (2) Comparing the degree of sparsity of all sparse ranking models under current performance. The experiment results indicate that compared to other sparse ranking models, our algorithm can obtain a sparser model while achieving a competitive performance.

4.6.1 Performance comparison experiment of sparse models

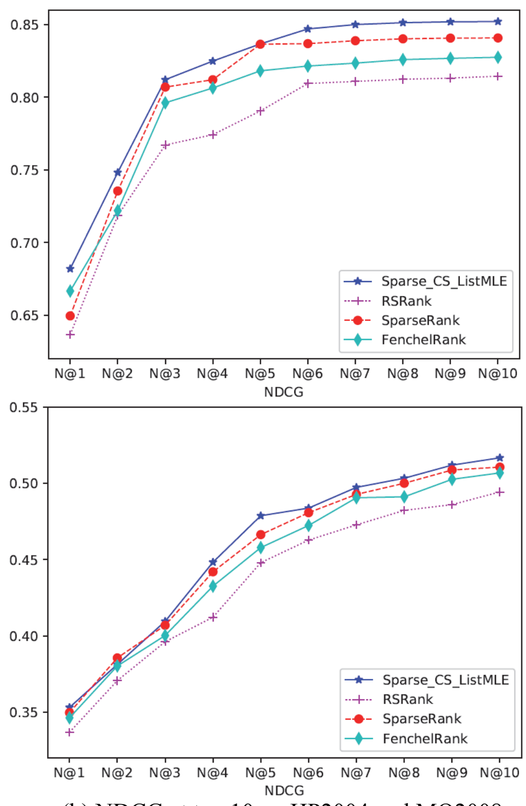
We take Sparse_CS_ListMLE@1 as an example to compare the performance with the other sparse ranking algorithms, including FenchelRank [13], SparseRank [17] and RSRank [16]. Experimental results are presented Fig. 4.

The greater the value of NDCG, the more accurate the ranking is. From Fig. 4, it can be observed that Sparse_CS_ListMLE have the best ranking effect on four datasets. For example, according to Fig. 4 (a), on OHSUMED dataset, the value of NDCG@10 of Sparse_CS_ListMLE has increased 3.4% compared with SparseRank who has the second best performance. And similarly, on TD2003 dataset, NDCG@10 value of Sparse_ CS_ListMLE is 0.4019, a 2.8% percent increase compared with RSRank. According to Fig. 4 (b), Sparse_CS_ListMLE performs 0.8520 at NDCG@10, a 1.3% increase than the second, however, on MQ2008, all the sparse models performs little differently.

We provide an explanation about Fig. 4. The reason why Sparse_CS_ListMLE performs better than others lies in two aspects: (1) Sparse_CS_ListMLE belongs to list-wise ranking algorithm, while others optimize the pair-wise loss function, practice shows that list-wise ranking is generally better than pair-wise ranking [6, 7]; (2) By imposing weights for the document, Sparse_CS_ListMLE emphasize the position importance of top k ranking list, which is a key factor for ranking, so our algorithm is superior to the others. Besides, SparseRank and FenchelRank have little difference since their main difference is the optimization method. RSRank formulates ranking as importance weighted pairwise classification, it behaves better than SparseRank and FenchelRank on TD2003, however it performs relatively poor than others on OHSUMED, for that the weights of the documents does not affect much in RSRank.



(a) NDCG at top 10 on OHSUMED and TD2003 dataset.



(b) NDCG at top 10 on HP2004 and MQ2008 dataset.

Fig. 4. NDCG at top 10 on four datasets.

4.6.2 Comparison of sparse degree of sparse models

In Sparse_CS_ListMLE, based on the best performance of cross validation on TD-2003, we fixed the regularization parameter λ as 10^{-2} to control the weight of ℓ_1 norm. The formula of sparse degree is $degree = \frac{\|\mathbf{w}_0\|}{d}$, where d is the dimensionality of the document feature vector. The smaller the *degree*, the sparser the model is. In section 4.6.1, we have compared the performance of Sparse Cost_Sensitive with several other state-of-the-art sparse models, such as FenchelRank [13], SparseRank [17] and RSRank [16]. In this section, we also compare the sparsity of them. The results are given in Table 5, where we can see that the sparse degree of our sparse method is 1.6% smaller than SparseRank on OHSUMED and 0.9% on TD2003. So Sparse_CS_ListMLE is sparser than other sparse models.

Table 5. Sparse degree on two datasets.

	OHSUMED	TD2003
Sparse_CS_ListMLE	**0.23296**	**0.53697**
FenchelRank	0.24290	0.54582
SparseRank	0.23682	0.54204
RSRank	0.34483	0.58898

4.7 Comparing to Other State-of-the-Art Model

In this section, we make a comparison between Sparse_CS_ListMLE and other state-of-the-art algorithms. LambdaMart [35], ListNet [6], AdaRank [5], and RankBoost [4] are chosen as baselines in these experiments. LambdaMart, ListNet, AdaRank attempt to optimize listwise loss functions, while RankBoost belongs to pairwise approach.

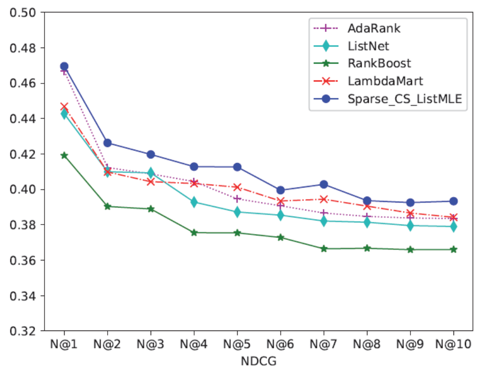


Fig. 5. NDCG at top 10 on OHSUMED.

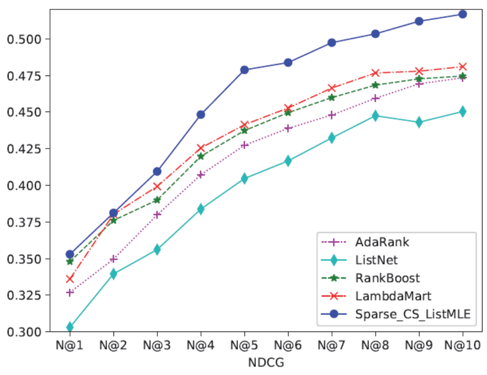


Fig. 6. NDCG at top 10 on MQ2008.

Table 6. MAP on the two datasets from the LETOR.

	Sparse_CS_ListMLE	AdaRank	LambdaMart	ListNet	RankBoost
OHSUMED	0.3514	0.3459	0.3139	0.3192	0.3395
MQ2008	0.4885	0.4571	0.4668	0.4356	0.4674

The results on two datasets with respect to NDCG and MAP are shown in Figs. 5-6 and Table 6. From Table 6, we can observe that Sparse_CS_ListMLE performs the best in two datasets among other state-of-the-art algorithms. For OHSUMED dataset, the improvement of RSRank over the second best model AdaRank is also statistically significant for NDCG@1 with a *p*-value of less than 0.05 in *t*-test, which indicates that the effectiveness of Sparse_CS_ListMLE in emphasizing the top permutation of the ranked list.

5. CONCLUSIONS

In this paper, to address the problem of cost-sensitive ListMLE that long-time training and feature redundancy, we introduced a Sparse cost-sensitive ListMLE which takes into account both the accuracy and computational efficiency. At first, we described the original cost-sensitive ListMLE, and then for the sake of achieving sparsity, we introduced an efficient proximal gradient descent (PGD) learning method via incorporating the ℓ_1 regularized sparse term into the model. We then constructed the object function composed of cost-sensitive ListMLE and ℓ_1 norm, then we theoretically give the initial values of the Lipschitz constant L and adopt the adaptive L method in the trial. Last we give a convergence analysis of the proposed model. Experiments performed on several benchmark datasets demonstrate the effectiveness of the proposed algorithm in constructing sparser model while improving the empirical performance accuracy. In future work, we will focus on more efficient optimization method to solve the sparse model with ℓ_1 regularization and try incorporating the cost-sensitive into other sparse models.

APPENDIX A: PROOF OF THEOREM 1

Proof: Since the Lipschitz constant L of the proposed model is the upper bound of second derivative of the loss function, now we want to prove

$$\| L''(\mathbf{w}) \| \le \frac{1}{4DCG_{\hat{g}}@k}\mathbf{R}^2\sum_{j=1}^{n}\beta_j .$$

Since

$$L(\mathbf{w}) = \frac{1}{DCG_{\hat{g}}@k}\sum_{j=1}^{n}\beta_j lb(1+\sum_{t=j+1,y_j>y_t}^{n}\frac{\alpha_{j,t}}{\alpha_{j,j}}\exp f(x_t)-f(x_j)),$$

let

$$A = 1+\sum_{t=j+1}^{n}\frac{\alpha_{j,t}}{\alpha_{j,j}}\exp(f(x_t)-f(x_j)),$$

$$B = \sum_{t=j+1}^{n}\frac{\alpha_{j,t}}{\alpha_{j,j}}\exp(<\mathbf{w},(x_i-x_j)>)(x_i-x_j),$$

the second derivative of $L(\mathbf{w})$ is

$$L''(\mathbf{w}) = \frac{1}{DCG_{\hat{g}}@k}\sum_{j=1}^{n}\beta_j\frac{A\cdot\sum_{t=j+1,y_t>y_j}^{n}\frac{\alpha_{j,t}}{\alpha_{j,j}}\exp(<\mathbf{w},(x_i-x_j)>)(x_i-x_j)^2-B^2}{A^2}.$$

We use $\mathbf{R}$ to denote the upper bound of the feature vector, that is $\|x_i\|_\infty \le \mathbf{R}$, then we have

$$\begin{aligned}L''(\mathbf{w}) &= \frac{1}{DCG_{\hat{g}}@k}\sum_{j=1}^{n}\beta_j\frac{A\cdot\sum_{t=j+1,y_t>y_j}^{n}\frac{\alpha_{j,t}}{\alpha_{j,j}}\exp(<\mathbf{w},(x_i-x_j)>)(x_i-x_j)^2-B^2}{A^2}\\ &\le \frac{1}{DCG_{\hat{g}}@k}\sum_{j=1}^{n}\beta_j\frac{\mathbf{R}^T\mathbf{R}\sum_{t=j+1,y_t>y_j}^{n}\frac{\alpha_{j,t}}{\alpha_{j,j}}\exp(\mathbf{wR})}{(1+\sum_{t=j+1,y_t>y_j}^{n}\frac{\alpha_{j,t}}{\alpha_{j,j}}\exp(\mathbf{wR})^2}.\end{aligned}$$

Since $(1 + t)^2 = 1 + 2t + t^2 \ge 4t$, $\frac{t}{(1+t)^2} \le \frac{1}{4}$, we have

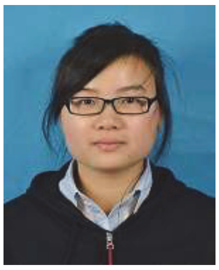
$$L''(\mathbf{w}) \le \frac{1}{DCG_{\hat{g}}@k}\sum_{j=1}^{n}\beta_j\frac{\mathbf{R}^T\mathbf{R}}{4} = \frac{\mathbf{R}^T\mathbf{R}}{4DCG_{\hat{g}}@k}\sum_{j=1}^{n}\beta_j.$$

REFERENCES

1. P. Li, C. J. C. Burges, and Q. Wu, "Mcrank: learning to rank using multiple classification and gradient boosting," in *Proceedings of International Conference on Neural Information Processing Systems*, 2007, pp. 897-904.
2. Y. Cao, J. Xu, T. Y. Liu, H. Li, Y. Huang, and H. W. Hon, "Adapting ranking svm to document retrieval," in *Proceedings of International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2006, pp. 186-193.
3. Y. Cao, J. Xu, T. Y. Liu, H. Li, Y. Huang, and H. W. Hon, "An ordinal regression method for document retrieval," in *Proceedings of ACM SIGIR International Conference of Research and Development on Information Retrieval*, 2006.
4. Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer, "An efficient boosting algorithm for combining preferences," *Journal of Machine Learning Research*, Vol. 4, 2003, pp. 170-178.
5. J. Xu and H. Li, "Adarank: a boosting algorithm for information retrieval," in *Proceedings of International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007, pp. 391-398.
6. Z. Cao, T. Qin, T. Y. Liu, M. F. Tsai, and H. Li, "Learning to rank: from pairwise approach to listwise approach," in *Proceedings of International Conference on Machine Learning*, 2007, pp. 129-136.
7. F. Xia, T. Y. Liu, J. Wang, W. Zhang, and H. Li, "Listwise approach to learning to rank: theory and algorithm," in *Proceedings of ACM International Conference on International Conference on Machine Learning*, 2008, pp. 1192-1199.
8. W. Chen, T. Y. Liu, Y. Lan, Z. Ma, and H. Li, "Ranking measures and loss functions in learning to rank," in *Advances in Neural Information Processing Systems*, Vol. 22, 2009, pp. 315-323.
9. Y. Lan, S. Niu, J. Guo, and X. Cheng, "Is top-*k* sufficient for ranking?" in *Proceedings of ACM International Conference on Information and Knowledge Management*, 2013, pp. 1261-1270.
10. J. Xu, Y. Cao, H. Li, and Y. Huang, "Cost-sensitive learning of svm for ranking," in *Proceedings of European Conference on Machine Learning*, 2006, pp. 833-840.
11. M. Lu, M. Q. Xie, Y. Wang, J. Liu, and Y. L. Huang, "Cost-sensitive listwise ranking approach," in *Advances in Knowledge Discovery and Data Mining*, 2010, pp. 358-366.
12. Y. Lan, Y. Zhu, J. Guo, S. Niu, and X. Cheng, "Position-aware listmle: A sequential learning process for ranking," in *Proceedings of Conference on Uncertainty in Artificial Intelligence*, 2014, pp. 449-458.
13. H. Lai, Y. Pan, C. Liu, L. Lin, and J. Wu, "Sparse learning-to-rank via an efficient primal-dual algorithm," *IEEE Transactions on Computers*, Vol. 62, 2013, pp. 1221-1233.
14. X. Geng, T. Y. Liu, T. Qin, and H. Li, "Feature selection for ranking," in *Proceedings of International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2007, pp. 407-414.

15. O. Chapelle and Y. Chang, "Yahoo! learning to rank challenge overview," *Journal of Machine Learning Research*, Vol. 14, 2013, pp. 1-24.
16. Z. Sun, T. Qin, Q. Tao, and J. Wang, "Robust sparse rank learning for non-smooth ranking measures," in *Proceedings of International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2009, pp. 259-266.
17. H. Lai, Y. Pan, Y. Tang, and N. Liu, "Efficient gradient descent algorithm for sparse models with application in learning-to-rank," *Knowledge-Based Systems*, Vol. 49, 2013, pp. 190-198.
18. L. Wang, Z. Yu, T. Jin, X. Li, and S. Gao, "Expert list-wise ranking method based on sparse learning," *Neurocomputing*, Vol. 217, 2016, pp. 119-124.
19. J. F. Pessiot, T. V. Truong, N. Usunier, M. R. Amini, and P. Gallinari, "Learning to rank for collaborative filtering," in *Proceedings of the 9th International Conference on Enterprise Information Systems*, 2007, pp. 145-151.
20. H. Li, *Learning to Rank for Information Retrieval and Natural Language Processing*, Morgan & Claypool Publishers, CA, 2011.
21. T. Qin, X. D. Zhang, M. F. Tsai, D. S. Wang, T. Y. Liu, and H. Li, "Query-level loss functions for information retrieval," *Information Processing and Management*, Vol. 44, 2008, pp. 838-855.
22. J. Xu, T. Y. Liu, M. Lu, H. Li, and W. Y. Ma, "Directly optimizing evaluation measures in learning to rank," in *Proceedings of International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2008, pp. 107-114.
23. L. Laporte, R. Flamary, S. Canu, S. Djean, and J. Mothe, "Nonconvex regularizations for feature selection in ranking with sparse svm," *IEEE Transactions on Neural Networks and Learning Systems*, Vol. 25, 2015, pp. 1118-1130.
24. O. Krasotkina and V. Mottl, "A bayesian approach to sparse learning-to-rank for search engine optimization," in *Proceedings of International Conference on Machine Learning and Data Mining in Pattern Recognition*, 2015, pp. 382-394.
25. P. L. Combettes and V. R. Wajs, "Signal recovery by proximal forward-backward splitting," *SIAM Journal on Multiscale Modeling and Simulation*, Vol. 4, 2005, pp. 1168-1200.
26. L. Armijo, "Minimization of functions having lipschitz continuous first partial derivatives," *Pacific Journal of Mathematics*, Vol. 16, 1966, pp. 1-3.
27. G. R. Wood and B. P. Zhang, "Estimation of the lipschitz constant of a function," *Journal of Global Optimization*, Vol. 8, 1996, pp. 91-103.
28. A. Y. Ng, "Feature selection, $l1$ vs. $l2$ regularization, and rotational invariance," in *Proceedings of the 21st International Conference on Machine Learning*, Vol. 19, 2004, pp. 379-387.
29. T. Qin and T. Y. Liu, "Introducing letor 4.0 datasets," *Computer Science*, 2013.
30. H. Lai, Y. Tang, H. X. Luo, and Y. Pan, "Greedy feature selection for ranking," in *Proceedings of International Conference on Computer Supported Cooperative Work in Design*, 2011, pp. 42-46.
31. D. X. D. Sousa, T. C. Rosa, and W. S. Martins, "Incorporating risk-sensitiveness into feature selection for learning to rank," in *Proceedings of ACM International on Conference on Information and Knowledge Management*, 2016, pp. 257-266.
32. T. Qin, T. Y. Liu, J. Xu, and H. Li, "Letor: A benchmark collection for research on learning to rank for information retrieval," *Information Retrieval*, Vol. 13, 2010, pp.

346-374.
33. R. Baeza-Yates and B. Ribeiro-Neto, *Modern Information Retrieval*, 2nd ed., Addison Wesley, NY, 1999.
34. K. Jarvelin and J. Kekalainen, "Cumulated gain-based evaluation of IR techniques," *ACM Transactions on Information Systems*, Vol. 20, 2002, pp. 422-466.
35. B. Schlkopf, J. Platt, and T. Hofmann, "Learning to rank with nonsmooth cost functions," in *Proceedings of International Conference on Neural Information Processing Systems*, 2006, pp. 193-200.



DanQi Du received the B.S. degree in Hubei University of Economics, China. Currently, she is a Master student in the Department of Computer Science and Technology at Gui Zhou University. Her research interests include learning-to-rank and recommender system.

Feng Zhou received the B.S. and M.S. degrees in College of Computer Science and Technology from Gui Zhou University, China, in 1999 and 2002. Currently, she is an Assistant Professor in the Department of Computer Science and Technology at Gui Zhou University. Her research interests include web data mining and information security.

Wei Xiong received the B.S. degree in School of Mechanical Engineering from Gui Zhou University, China, in 1999. His research interests include internet of things and workflow software development.