
Ranking as Learning Structured Outputs

Christopher J.C. Burges
Microsoft Research
One Microsoft Way
Redmond, WA 98052-6399
cburges@microsoft.com

Abstract

We propose a new method for learning structured outputs using gradient descent.

1 Introduction

We examine ranking from the point of view of learning structured outputs. To make the discussion concrete we will use the example of ranking search results. There, the task is the following: a query $Q \in \{Q_i : i = 1, \dots, N_Q\}$ is issued by a user. Q may be thought of as a text string, but it may also contain other kinds of data. The search engine examines a large set of documents, and for each document D , constructs a feature vector $\mathbf{x}(Q, D) \in \mathcal{R}^n$. The feature vector \mathbf{x} is then input to a ranking algorithm \mathcal{A} , which outputs a scalar score s : $\mathcal{A} : \mathbf{x} \in \mathcal{R}^n \mapsto s \in \mathcal{R}$. For training, a set of labeled data $\{Q_i, D_{ij}, l_{ij}, i = 1, \dots, N_Q, j = 1, \dots, n_i\}$, where n_i is the number of documents returned for the i 'th query, is used to minimize a real-valued cost function C . Here the labels l encode the relevance of document D_{ij} with respect to the query Q_i , and take integer values, where for a given query Q , $l_1 > l_2$ means that the document with label l_1 has been judged more relevant to Q than that with label l_2 . Once training is complete, the scores s output by \mathcal{A} are used to map feature vectors \mathbf{x} to the reals, where $\mathcal{A}(\mathbf{x}(Q, D_1)) > \mathcal{A}(\mathbf{x}(Q, D_2))$ is taken to mean that, for query Q , document D_1 is to be ranked higher than document D_2 .

2 Ranking Measures

In the information retrieval literature, there are many methods used to measure the quality of ranking results. Here we briefly describe four of the simplest. The pair-wise error counts the number of pairs that are in the incorrect order, as a fraction of the maximum possible number of such pairs. The Mean Reciprocal Rank (MRR) applies to the binary relevance task: for a given query, any returned document is labeled 'relevant' or 'not relevant', and if r_i is the rank of the highest ranking relevant document for the i 'th query, then the reciprocal rank measure for that query is $1/r_i$, and the MRR is just the reciprocal rank, averaged over queries. 'Winner Takes All' also applies to the binary relevance task: if the top ranked document for a given query is relevant, the WTA cost is zero, otherwise it is one; WTA is again averaged over queries. MRR and WTA have been used, for example, in past TREC evaluations of Question Answering systems. Finally, the normalized discounted cumulative gain measure (NDCG) [9] is a cumulative measure of ranking quality (so a suitable cost would be $1 - \text{NDCG}$). For a given query Q_i the NDCG is computed as $\mathcal{N}_i \equiv N_i \sum_{j=1}^L (2^{r(j)} - 1) / \log(1 + j)$, where $r(j)$ is the relevance level of the j 'th ranked document, and where the normalization constant \mathcal{N}_i is chosen so that a perfect ordering would result in $\mathcal{N}_i = 1$. Here L is the ranking level at which the NDCG is computed. The \mathcal{N}_i are then again averaged over the query set.

3 Ranking as Learning Structured Outputs

Many different approaches to learning ranking algorithms have been proposed: from SVMs [6, 10, 3] to Perceptrons [4, 5] to Neural Networks [2, 1]. Are these algorithms solving the right problem? They are certainly attempting to learn an ordering of the data. However, in this Section we argue that, in general, the answer is no. Let's revisit the cost metrics described above. We assume throughout that the documents have been ordered by decreasing score.

These metrics present two key challenges. First, they all depend on not just the output s for a single feature vector \mathbf{x} , but on the outputs of all feature vectors, for a given query; for example for WTA, we must compare all the scores to find the maximum. Second, none are differentiable functions of their arguments; in fact they are flat over large regions of parameter space, which makes the learning problem much more challenging. By contrast, note that the algorithms mentioned above have the property that, in order to make the learning problem tractable, they use smooth costs. This smoothness requirement is, in principle, not necessarily a burden, since in the ideal case, when the algorithm can achieve zero cost on the some dataset, using, say, the RankNet cost [1], it has also achieved zero cost using any of the above measures. Hence, the problems that arise from using a simple, smooth approximation to one of the above cost functions, arise because in practice, learning algorithms cannot achieve perfect generalization.

For a concrete example of where using an approximate cost can lead to problems, suppose that we use a smooth approximation to pair-wise error [1], but that what we really want to minimize is the WTA cost. Consider a training query with 1,000 returned documents, and suppose that there are two relevant documents D_1 and D_2 , and 998 irrelevant documents, and that the ranker puts D_1 in position 1 and D_2 in position 1000. Then the ranker can reduce the pair-wise error, for that query, by 996 errors, by moving D_2 up to rank 3 and by moving D_1 down to rank 2. However the WTA cost has gone from zero to one. A huge decrease in the pairwise error rate has resulted in the maximum possible increase in the WTA cost.

The need for the ability to handle multivariate costs is not limited to ranking. For example, one measure of quality for document retrieval is the area under the ROC curve, and maximizing this amounts to learning using a multivariate cost, as does using measures that depend on precision and recall [11, 7].

In order to learn using a multivariate, non-differentiable cost function, we propose the following framework, which for the ranking problem we call LambdaRank. We describe the approach in the context of learning to rank using gradient descent. Here, a general multivariate cost function for a given query takes the form $C(s_{ij}, l_{ij})$, where i indexes the query and j indexes a returned document for that query. Thus, in general the cost function may take a different number of arguments, depending on the query (some queries may get more documents returned than others). In general, finding a smooth cost function that has the desired behaviour is very difficult. Take the above WTA example. It is much more important to keep D_1 in the top position than to move D_2 up 997 positions and D_1 down one: the optimal WTA cost is achieved when either D_1 or D_2 is in the top position. Notice how the finite capacity of the learning algorithm is playing a crucial role here. In this particular case, to better approximate WTA, one approach would be to steeply discount errors that occur low in the ranking. Now imagine that C is a smooth approximation to a cost function that accomplishes this, and assume that at the current learning iteration, \mathcal{A} produces an ordering for a given Q where D_1 is in position 2 and D_2 is in position 1000. Then if $s_i \equiv \mathcal{A}(\mathbf{x}_i)$, $i = 1, 2$, we require that

$$\left| \frac{\partial C}{\partial s_1} \right| \gg \left| \frac{\partial C}{\partial s_2} \right| \quad (1)$$

Notice that we've captured a desired property of C by imposing a constraint on its derivatives. The idea of LambdaRank is to extend this by replacing the requirement of specifying C itself, by the task of specifying its derivative with respect to each s_j , $j = 1, \dots, n_i$, for each query Q_i . Those derivatives can then be used to train \mathcal{A} using gradient descent, just as the derivatives of C normally would be. The point is that it can be much easier, given an instance of a query and its ranked documents, to specify how you would like those documents to move, in order to reduce a non-differentiable cost, than to specify a smooth approximation of that (multivariate) cost. As a simple example, consider a single query with just two returned documents D_1 and D_2 , and suppose they

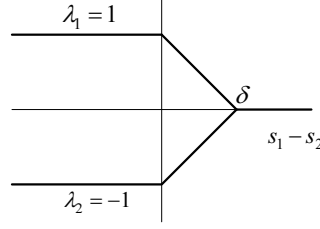


Figure 1: Choosing the λ 's for a query with two documents.

have labels $l_1 = 1$ (relevant) and $l_2 = 0$ (not relevant), respectively. We imagine that there is some $C(s_1, l_1, s_2, l_2)$ such that

$$\frac{\partial C}{\partial s_1} = -\lambda_1(s_1, l_1, s_2, l_2) \quad (2)$$

$$\frac{\partial C}{\partial s_2} = -\lambda_2(s_1, l_1, s_2, l_2) \quad (3)$$

We would like the λ 's to take the form shown in Figure 1, for some chosen margin $\delta \in \mathcal{R}$: thinking of the documents as lying on a vertical line, where higher scores s correspond to higher points on the line, then D_1 (D_2) gets a constant gradient up (or down) as long as it is in the incorrect position, and the gradient goes smoothly to zero until the margin is achieved. Thus the learning algorithm \mathcal{A} will not waste capacity moving D_1 further away from D_2 if they are in the correct position by more than δ , and having nonzero δ ensures robustness to small changes in the scores s_i . Letting $x \equiv s_1 - s_2$, the λ 's may be written

$$x < 0 : \lambda_1 = 1 = -\lambda_2 \quad (4)$$

$$0 \leq x \leq \delta : \lambda_1 = \delta - x = -\lambda_2 \quad (5)$$

$$x > \delta : \lambda_1 = \lambda_2 = 0 \quad (6)$$

In this case a corresponding cost function exists:

$$x < 0 : C(s_1, l_1, s_2, l_2) = s_2 - s_1 \quad (7)$$

$$0 \leq x \leq \delta : C(s_1, l_1, s_2, l_2) = \frac{1}{2}(s_1 - s_2)^2 - \delta(s_1 - s_2) \quad (8)$$

$$x > \delta : C(s_1, l_1, s_2, l_2) = -\frac{1}{2}\delta^2 \quad (9)$$

Note that in addition the Hessian of C is positive semidefinite, so the cost function takes a unique minimum value (although the s 's for which C attains its minimum are not unique). In general, when the number of documents for a given query is much larger than two, and where the rules for writing down the λ 's will depend on the scores, labels and ranks of all the documents, then C can become prohibitively complicated to write down explicitly.

There is still a great deal of freedom in this model, namely, how to choose the λ 's to best model a given (multivariate, non-differentiable) cost function. Let's call this choice the λ -function. We will not explore here how, given a cost function, to find a particular λ -function, but instead will answer two questions which will help guide the choice: first, for a given choice of the λ 's, under what conditions does there exist a cost function C for which they are the negative derivatives? Second, given that such a C exists, under what conditions is C convex? The latter is desirable to avoid the problem that local minima in the cost function itself will present to any algorithm used for training \mathcal{A} . To address the first question, we can use a well-known result from multilinear algebra [12]:

Theorem (Poincaré Lemma): If $S \subset \mathcal{R}^n$ is an open set that is star-shaped with respect to 0, then every closed form on S is exact.

Note that since every exact form is closed, it follows that on an open set that is star-shaped with respect to 0, a form is closed if and only if it is exact. Now for a given query Q_i and corresponding

set of returned D_{ij} , the n_i λ 's are functions of the scores s_{ij} , parameterized by the (fixed) labels l_{ij} . Let dx^i be a basis of 1-forms on \mathcal{R}^n and define the 1-form

$$\lambda \equiv \sum_i \lambda_i dx^i \quad (10)$$

Then assuming that the scores are defined over \mathcal{R}^n , the conditions for the theorem are satisfied and $\lambda = dC$ for some function C if and only if $d\lambda = 0$ everywhere. Using classical notation, this amounts to requiring that

$$\frac{\partial \lambda_i}{\partial s_j} = \frac{\partial \lambda_j}{\partial s_i} \quad \forall i, j \quad (11)$$

Thus we have a simple test on the λ 's to determine if there exists a cost function for which they are the derivatives: the Jacobian (that is, the matrix $J_{ij} \equiv \partial \lambda_i / \partial s_j$) must be symmetric. Furthermore, given that such a cost function C does exist, the condition that it be convex is that the Jacobian be positive semidefinite everywhere. Under these constraints, the Jacobian is beginning to look very much like a kernel matrix! However, there is a difference: the value of the i 'th, j 'th element of a kernel matrix depends on two vectors $\mathbf{x}_i, \mathbf{x}_j$ (where for example $\mathbf{x} \in \mathcal{R}^d$ for some d , although in general they may be elements of an abstract vector space), whereas the value of the i 'th, j 'th element of the Jacobian depends on all of the scores s_i .

For choices of the λ 's that are piecewise constant, the above two conditions (symmetric and positive semidefinite¹) are trivially satisfied. For other choices of symmetric J , positive definiteness can be imposed by adding regularization terms of the form $\lambda_i \mapsto \lambda_i + \alpha s_i$, $\alpha_i > 0$, which amounts to adding a positive constant along the diagonal of the Hessian.

Finally, we observe that LambdaRank has a clear physical analogy. Think of the documents returned for a given query as point masses. Each λ then corresponds to a force on the corresponding point. If the conditions of Eq. (11) are met, then the forces in the model are conservative, that is, the forces may be viewed as arising from a potential energy function, which in our case is the cost function. For example, if the λ 's are linear in the outputs s , then this corresponds to a spring model, with springs that are either compressed or extended. The requirement that the Jacobian is positive semidefinite amounts to the requirement that the system of springs have a unique global minimum of the potential energy, which can be found from any initial conditions by gradient descent (this is not true in general, for arbitrary systems of springs).

References

- [1] C.J.C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender, Learning to Rank Using Gradient Descent, in *ICML 22*, 2005
- [2] R. Caruana, S. Baluja, and T. Mitchell, Using the Future to "sort out" the Present: Rankprop and Multitask Learning for Medical Risk Evaluation, *NIPS 8*, 1996.
- [3] W. Chu and S.S. Keerthi, New Approaches to Support Vector Ordinal Regression, in *ICML 22*, 2005
- [4] K. Crammer and Y. Singer, Pranking with Ranking, in *NIPS 14*, 2002
- [5] E.F. Harrington, Online Ranking/Collaborative Filtering Using the Perceptron Algorithm, in *ICML 20*, 2003
- [6] R. Herbrich, T. Graepel, and K. Obermayer, Large Margin Rank Boundaries for Ordinal Regression, in A.J. Smola et al., Editors, *Advances in Large Margin Classifiers*, 2000
- [7] A. Herschtal and B. Raskutti, Optimizing Area Under the ROC Curve Using Gradient Descent, in *ICML 21*, 2004.
- [8] R.A. Horn and C.R. Johnson, *Matrix Analysis*, Cambridge University Press, 1985.
- [9] K. Jarvelin and J. Kekalainen, IR Evaluation Methods for Retrieving Highly Relevant Documents, in *Proc. 23rd ACM SIGIR*, 2000.
- [10] T. Joachims, Optimizing Search Engines Using Clickthrough Data, in David Hand, Daniel Keim, and Raymond Ng, editors, *Proc. 8th SIGKDD*, 2002.
- [11] T. Joachims, A Support Vector Method for Multivariate Performance Measures, *ICML 22*, 2005.
- [12] M. Spivak, *Calculus on Manifolds*, Addison-Wesley, 1965.

¹Some authors define the property of positive semi-definiteness to include the property of symmetry: see [8].