

---

# Ranking as Function Approximation

Christopher J.C. Burges

Microsoft Research [cburges@microsoft.com](mailto:cburges@microsoft.com)

**Summary.** An overview of the problem of learning to rank data is given. Some current machine learning approaches to the problem are described. The cost functions used to assess the quality of a ranking algorithm present particular difficulties: they are non-differentiable (as a function of the scores output by the ranker) and multivariate (in the sense that the cost associated with one ranked object depends on its relations to several other ranked objects). I present some ideas on a general framework for training using such cost functions; the approach has an appealing physical interpretation. The paper is tutorial in the sense that it is not assumed that the reader is familiar with the methods of machine learning; my hope is that the paper will encourage applied mathematicians to explore this topic.

## 1 Introduction

The field of machine learning draws from many disciplines, but ultimately the task is often one of function approximation: for classification, regression estimation, time series estimation, clustering, or more complex forms of learning, an attempt is being made to find a function that meets given criteria on some data. Because the machine learning enterprise is multi-disciplinary, it has much to gain from more established fields such as approximation theory, statistical and mathematical modeling, and algorithm design. In this paper, in the hope of stimulating more interaction between our communities, I give a review of approaches to one problem of growing interest in the machine learning community, namely, ranking. Ranking is needed whenever an algorithm returns a set of results upon which one would like to impose an order: for example, commercial search engines must rank millions of URLs in real time to help users find what they are looking for, and automated Question-Answering systems will often return a few top-ranked answers from a long list of possible answers. Ranking is also interesting in that it bridges the gap between traditional machine learning (where, for example, a sample is to be classified into one of two classes), and another area that is attracting growing interest, namely that of modeling structured data (as inputs, outputs, or both), for example for data structures such as graphs. In this light, I will also present some new ideas on models for handling structured output data.

## 1.1 Notation

To make the discussion concrete and to establish notation, I will use the example of ranking search results. There, the task is the following: a query  $Q$  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 previously gathered documents, and for each document  $D$ , constructs a feature vector  $F(Q, D) \in \mathcal{R}^n$ . Thus, the  $i$ th element of  $F$  is itself a function  $f_i : \{Q, D\} \mapsto \mathcal{R}$ , and  $f_i$  has been constructed to encapsulate some aspect of how relevant the document  $D$  is to the query  $Q$ <sup>1</sup>. The feature vector  $F$  is then input to a ranking algorithm  $\mathcal{A}$ , which outputs a scalar “score”:  $\mathcal{A} : F \in \mathcal{R}^n \mapsto s \in \mathcal{R}$ . We will denote the number of queries for a given dataset by  $N_Q$  and the number of documents returned for the  $i$ ’th query by  $n_i$ . During the training phase, a set of labeled data  $\{Q_i, D_{ij}, l_{ij}, i = 1, \dots, N_Q, j = 1, \dots, n_i\}$  is used to minimize a cost function  $C$ . Here the labels  $l$  encode the relevance of document  $D_{ij}$  for the query  $Q_i$ , and take integer values, where for a given query  $Q$ ,  $l_1 > l_2$  means that the document with label  $l_1$  is more relevant to  $Q$  than that with label  $l_2$  (note that the labels  $l$  really attach to document-query pairs, since a given document may be relevant for one query but not for another). The form that the cost function  $C$  takes varies from one algorithm to another, but its range is always the reals; the training process aims to find those parameters in the function  $\mathcal{A}$  that minimize the sample expectation of the cost over the training set. Once such a function  $\mathcal{A}$  has been found, its parameters are fixed, and its output scores  $s$  are used to map feature vectors  $F$  to the reals, where  $A(F(Q, D_1)) > A(F(Q, D_2))$  is taken to mean that, for query  $Q$ , document  $D_1$  is to be ranked higher than document  $D_2$ . We will encapsulate this last relation using the symbol  $\triangleright$ , so that  $A(F(Q, D_1)) > A(F(Q, D_2)) \Rightarrow D_1 \triangleright D_2$ .

## 1.2 Representing the Ranking Problem as a Graph

[DMS04] provide a very general framework for ranking using directed graphs, where an arc from A to B means that A is to be ranked higher than B. Note that for ranking algorithms that train on pairs, all such sets of relations can be captured by specifying a set of training pairs, which amounts to specifying the arcs in the graph. This approach can represent arbitrary ranking functions, in particular, ones that are inconsistent - for example  $A \triangleright B$ ,  $B \triangleright C$ ,  $C \triangleright A$ . Such inconsistent rankings can easily arise when mapping multivariate measurements to one dimensional ranking, as the following toy example illustrates: imagine that a psychologist has devised an aptitude test<sup>2</sup>. Mathematician  $A$  is considered stronger than mathematician  $B$  if, given three particular theorems,  $A$  can prove at least two theorems faster than  $B$ . The psychologist finds the measurements shown in Table 1.

---

<sup>1</sup> In fact, some elements of the feature vector may depend only on the document  $D$ , in order to capture the notion that some documents are unlikely to be relevant for any possible query.

<sup>2</sup> Of course this “magic-square” example is not serious, although it illustrates the perils of one-dimensional thinking.

Mathematician	Minutes Per Proof		
	Theorem 1	Theorem 2	Theorem 3
Archimedes	8	1	6
Bryson	3	5	7
Callippus	4	9	2

**Table 1.** Archimedes is stronger than Bryson; Bryson is stronger than Callippus; but Callippus is stronger than Archimedes.

## 2 Measures of Ranking Quality

In the information retrieval literature, there are many methods used to measure the quality of ranking results. Here we briefly describe four. We observe that there are two properties that are shared by all of these cost functions: none are differentiable, and all are multivariate, in the sense that they depend on the scores of multiple documents. The non-differentiability presents particular challenges to the machine learning approach, where cost functions are almost always assumed to be smooth. Recently, some progress has been made tackling the latter property using support vector methods [Joa05]; below, we will outline an alternative approach.

### *Pair-wise Error*

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.

### *Normalized Discounted Cumulative Gain (NDCG)*

The normalized discounted cumulative gain measure [JK00] is a cumulative measure of ranking quality (so a suitable cost would be 1-NDCG). For a given query  $Q_i$  the NDCG is computed as

$$\mathcal{N}_i \equiv N_i \sum_{j=1}^L (2^{r(j)} - 1) / \log(1 + j) \tag{1}$$

where  $r(j)$  is the relevance level of the  $j$ 'th document, and where the normalization constant  $N_i$  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 averaged over the query set.

### *Mean Reciprocal Rank (MRR)*

This metric applies to the binary relevance task, where for a given query, and for a given document returned for that query, label “1” means “relevant” and “0”, “not relevant”. 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:

$$\text{MRR} = \frac{1}{N_Q} \sum_{i=1}^{N_Q} 1/r_i \quad (2)$$

MRR was used, for example, in TREC evaluations of Question Answering systems, before 2002 [Voo01].

#### *Winner Takes All (WTA)*

This metric 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; for  $N_Q$  queries we again take the mean:

$$\text{WTA} = \frac{1}{N_Q} \sum_{i=1}^{N_Q} \delta(l_{i1}, 1) \quad (3)$$

where  $\delta$  here is the Kronecker delta. WTA is used, for example, in TREC evaluations of Question Answering systems, after 2002 [Voo02].

### 3 Support Vector Ranking

Support vector machines for ordinal regression were proposed by [HGO00] and further explored by [Joa02] and more recently by [CK05]. The approach uses pair-based training. For convenience let us write the feature vector for a given query-document pair as  $\mathbf{x} \equiv F(Q, D)$ , where indices  $Q$  and  $D$  on  $\mathbf{x}$  are understood, and let us represent the training data as a set of pairs  $\{\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}\}$ ,  $i = 1, \dots, N$ , where  $N$  is the total number of pairs in the training set, together with labels  $z_i \in \{\pm 1\}$ ,  $i = 1, \dots, N$ , where  $z_i = 1$  ( $-1$ ) if  $\mathbf{x}_i^{(1)}$  is to be ranked higher (lower) than  $\mathbf{x}_i^{(2)}$ . Note that each query can generate training pairs (and that a given feature vector  $\mathbf{x}$  can appear in several pairs), but that once the pairs have been generated, all that is needed for training is the set of pairs and their labels.

To solve the ranking problem we solve the following QP:

$$\min_{\mathbf{w}, \xi_i} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_i \xi_i \right\} \quad (4)$$

subject to:

$$z_i \mathbf{w} \cdot (\mathbf{x}_i^{(1)} - \mathbf{x}_i^{(2)}) > 1 - \xi_i \quad (5)$$

$$\xi_i \in \mathcal{R}_+ \quad (6)$$

In the separable case, by minimizing  $\|\mathbf{w}\|$ , we are maximizing the gap, projected along  $\mathbf{w}$ , between items that are to be ranked differently; the slack variables  $\xi_i$  allow for non-separable data, and their sum gives a bound on the number of errors. This is similar to the original formulation of Support Vector Machines for classification [CV95, Bur98], and enjoys the same advantages: the algorithm can be implicitly mapped to a feature space using the kernel trick (see, for example, [SS02]), which gives the model a great deal of expressive freedom, and uniform bounds on generalization performance can be given [HGO00].

## 4 Perceptron Ranking

[CS02] propose a ranker based on the Perceptron ('PRank'), which maps a feature vector  $\mathbf{x} \in \mathcal{R}^d$  to the reals with a learned vector  $\mathbf{w} \in \mathcal{R}^d$  and increasing thresholds<sup>3</sup>  $b_r = 1, \dots, N$  such that the output of the mapping function is just  $\mathbf{w} \cdot \mathbf{x}$ , and such that the declared rank of  $\mathbf{x}$  is  $\min_r \{\mathbf{w} \cdot \mathbf{x} - b_r < 0\}$ . An alternative way to view this is that the rank of  $\mathbf{x}$  is defined by the bin into which  $\mathbf{w} \cdot \mathbf{x}$  falls. The learning step is modeled after the Perceptron update rule (see [CS02] for details): a newly presented example  $\mathbf{x}$  results in a change in  $\mathbf{w}$  (and in the  $b_r$ ) only if it falls in the wrong bin, given the current values of  $\mathbf{w}$  and the  $b_r$ . If this occurs,  $\mathbf{w}$  is updated by a quantity proportional to  $\mathbf{x}$ , and those thresholds whose movement could result in  $\mathbf{x}$  being correctly ranked are also updated. The linear form of PRank is an online algorithm<sup>4</sup>, in that it learns (that is, it updates the vector  $\mathbf{w}$ , and the thresholds that define the rank boundaries) using one example at a time. However, PRank can be, and has been, compared to batch ranking algorithms, and a quadratic kernel version was found to outperform all such algorithms described in [HGO00]. [Har03] has proposed a simple but very effective extension of PRank, which approximates finding the Bayes point (that point which would give the minimum achievable generalization error) by averaging over PRank models.

## 5 Neural Network Ranking

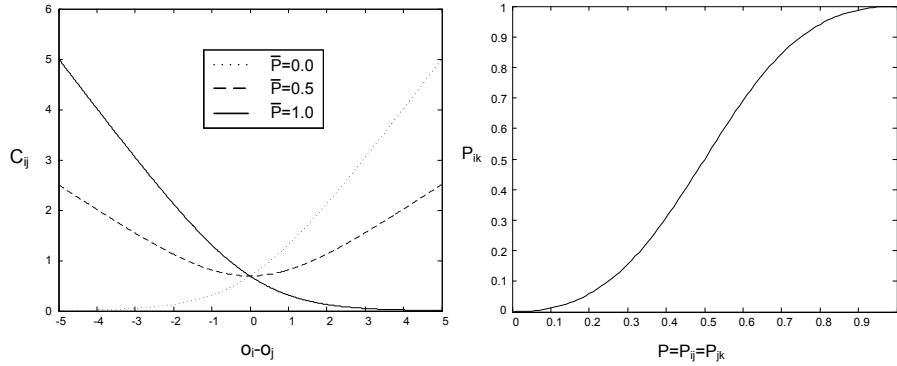
In this Section we describe a recent neural net based ranking algorithm that is currently used in one of the major commercial search engines [BSR<sup>+</sup>05]. Let's begin by defining a suitable cost.

### 5.1 A Probabilistic Cost

As we have observed, most machine learning algorithms require differentiable cost functions, and neural networks fall in this class. To this end, in [BSR<sup>+</sup>05] the following probabilistic model was proposed for modeling posteriors, where each training pair  $\{A, B\}$  has associated posterior  $P(A \triangleright B)$ . The probabilistic model is an important feature of the approach, since ranking algorithms often model preferences, and the ascription of preferences is a much more subjective process than the ascription of, say, classes. (Target probabilities could be measured, for example, by measuring multiple human preferences for each pair.) We consider models where the learning algorithm is given a set of pairs of samples  $[A, B]$  in  $\mathcal{R}^d$ , together with target probabilities  $P_{AB}$  that sample  $A$  is to be ranked higher than sample  $B$ . As described above, this is a general formulation, in that the pairs of ranks need not be complete (in that taken together, they need not specify a complete ranking of the training data), or even consistent. We again consider models  $\mathcal{A} : \mathcal{R}^d \mapsto \mathcal{R}$  such that the rank order of a set of test samples is specified by the real values that  $\mathcal{A}$  takes, specifically,  $\mathcal{A}(\mathbf{x}_1) > \mathcal{A}(\mathbf{x}_2)$  is taken to mean that the model asserts that  $\mathbf{x}_1 \triangleright \mathbf{x}_2$ .

<sup>3</sup> Actually the last threshold is pegged at infinity.

<sup>4</sup> The general kernel version is not, since the support vectors must be saved.



**Fig. 1.** Left: the cost function, for three values of the target probability. Right: combining probabilities.

Denote the modeled posterior  $P(\mathbf{x}_i \triangleright \mathbf{x}_j)$  by  $P_{ij}$ ,  $i, j = 1, \dots, m$ , and let  $\bar{P}_{ij}$  be the desired target values for those posteriors. The cost function is a function of the difference of the system's outputs for each member of a pair of examples, which encapsulates the observation that for any given pair, an arbitrary offset can be added to the outputs without changing the final ranking. Define  $o_i \equiv \mathcal{A}(\mathbf{x}_i)$  and  $o_{ij} \equiv \mathcal{A}(\mathbf{x}_i) - \mathcal{A}(\mathbf{x}_j)$ . The cost is a cross entropy cost function

$$C_{ij} \equiv C(o_{ij}) = -\bar{P}_{ij} \log P_{ij} - (1 - \bar{P}_{ij}) \log (1 - P_{ij}) \quad (7)$$

where the map from outputs to probabilities are modeled using a logistic function

$$P_{ij} \equiv \frac{1}{1 + e^{-o_{ij}}} \quad (8)$$

The cross entropy cost has been shown to result in neural net outputs that model probabilities [BW88].  $C_{ij}$  then becomes

$$C_{ij} = -\bar{P}_{ij} o_{ij} + \log(1 + e^{o_{ij}}) \quad (9)$$

Note that  $C_{ij}$  asymptotes to a linear function; for problems with noisy labels this is likely to be more robust than a quadratic cost. Also, when  $\bar{P}_{ij} = \frac{1}{2}$  (when no information is available as to the relative rank of the two patterns),  $C_{ij}$  becomes symmetric, with its minimum at the origin. This gives us a principled way of training on patterns that are desired to have the same rank. We plot  $C_{ij}$  as a function of  $o_{ij}$  in the left hand panel of Figure 1, for the three values  $\bar{P} = \{0, 0.5, 1\}$ .

### Combining Probabilities

The above model puts consistency requirements on the  $\bar{P}_{ij}$ , in that we require that there exist 'ideal' outputs  $\bar{o}_i$  of the model such that

$$\bar{P}_{ij} \equiv \frac{1}{1 + e^{-\bar{o}_i - \bar{o}_j}} \quad (10)$$

where  $\bar{o}_{ij} \equiv \bar{o}_i - \bar{o}_j$ . This consistency requirement arises because if it is not met, then there will exist no set of outputs of the model that give the desired pair-wise

probabilities. The consistency condition leads to constraints on possible choices of the  $\bar{P}$ 's. For example, given  $\bar{P}_{ij}$  and  $\bar{P}_{jk}$ , Eq. (10) gives

$$\bar{P}_{ik} = \frac{\bar{P}_{ij}\bar{P}_{jk}}{1 + 2\bar{P}_{ij}\bar{P}_{jk} - \bar{P}_{ij} - \bar{P}_{jk}} \quad (11)$$

This is plotted in the right hand panel of Figure 1, for the case  $\bar{P}_{ij} = \bar{P}_{jk} = P$ . We draw attention to some appealing properties of the combined probability  $\bar{P}_{ik}$ . First,  $\bar{P}_{ik} = P$  at the three points  $P = 0$ ,  $P = 0.5$  and  $P = 1$ , and only at those points. For example, if we specify that  $P(A \triangleright B) = 0.5$  and that  $P(B \triangleright C) = 0.5$ , then it follows that  $P(A \triangleright C) = 0.5$ ; complete uncertainty propagates. Complete certainty ( $P = 0$  or  $P = 1$ ) propagates similarly. Finally confidence, or lack of confidence, builds as expected: for  $0 < P < 0.5$ , then  $\bar{P}_{ik} < P$ , and for  $0.5 < P < 1.0$ , then  $\bar{P}_{ik} > P$  (for example, if  $P(A \triangleright B) = 0.6$ , and  $P(B \triangleright C) = 0.6$ , then  $P(A \triangleright C) > 0.6$ ). These considerations raise the following question: given the consistency requirements, how much freedom is there to choose the pairwise probabilities? We have the following<sup>5</sup>

**Theorem 1.** *Given a sample set  $\mathbf{x}_i$ ,  $i = 1, \dots, m$  and any permutation  $\mathcal{Q}$  of the consecutive integers  $\{1, 2, \dots, m\}$ , suppose that an arbitrary target posterior  $0 \leq \bar{P}_{kj} \leq 1$  is specified for every adjacent pair  $k = \mathcal{Q}(i), j = \mathcal{Q}(i+1)$ ,  $i = 1, \dots, m-1$ . Denote the set of such  $\bar{P}$ 's, for a given choice of  $\mathcal{Q}$ , a set of 'adjacency posteriors'. Then specifying any set of adjacency posteriors is necessary and sufficient to uniquely identify a target posterior  $0 \leq \bar{P}_{ij} \leq 1$  for every pair of samples  $\mathbf{x}_i, \mathbf{x}_j$ .*

**Proof:** Sufficiency: suppose we are given a set of adjacency posteriors. Without loss of generality we can relabel the samples such that the adjacency posteriors may be written  $\bar{P}_{i,i+1}$ ,  $i = 1, \dots, m-1$ . From Eq. (10),  $\bar{o}$  is just the log odds:

$$\bar{o}_{ij} = \log \frac{\bar{P}_{ij}}{1 - \bar{P}_{ij}} \quad (12)$$

From its definition as a difference, any  $\bar{o}_{jk}$ ,  $j \leq k$ , can be computed as  $\sum_{m=j}^{k-1} \bar{o}_{m,m+1}$ . Eq. (10) then shows that the resulting probabilities indeed lie in  $[0, 1]$ . Uniqueness can be seen as follows: for any  $i, j$ ,  $\bar{P}_{ij}$  can be computed in multiple ways, in that given a set of previously computed posteriors  $\bar{P}_{im_1}, \bar{P}_{m_1m_2}, \dots, \bar{P}_{m_nj}$ , then  $\bar{P}_{ij}$  can be computed by first computing the corresponding  $\bar{o}_{kl}$ 's, adding them, and then using (10). However since  $\bar{o}_{kl} = \bar{o}_k - \bar{o}_l$ , the intermediate terms cancel, leaving just  $\bar{o}_{ij}$ , and the resulting  $\bar{P}_{ij}$  is unique. Necessity: if a target posterior is specified for every pair of samples, then by definition for any  $\mathcal{Q}$ , the adjacency posteriors are specified, since the adjacency posteriors are a subset of the set of all pairwise posteriors.  $\square$

Although the above gives a straightforward method for computing  $\bar{P}_{ij}$  given an arbitrary set of adjacency posteriors, it is instructive to compute the  $\bar{P}_{ij}$  for the special case when all adjacency posteriors are equal to some value  $P$ . Then  $\bar{o}_{i,i+1} = \log(P/(1-P))$ , and  $\bar{o}_{i,i+n} = \bar{o}_{i,i+1} + \bar{o}_{i+1,i+2} + \dots + \bar{o}_{i+n-1,i+n} = n\bar{o}_{i,i+1}$  gives

<sup>5</sup> A similar argument can be found in [RV91]; however there the intent was to uncover underlying class conditional probabilities from pairwise probabilities; here, we have no analog of the class conditional probabilities.

$P_{i,i+n} = \Delta^n / (1 + \Delta^n)$ , where  $\Delta$  is the odds ratio  $\Delta = P / (1 - P)$ . The expected strengthening (or weakening) of confidence in the ordering of a given pair, as their difference in ranks increases, is then captured by:

**Lemma 1.** : *Let  $n > 0$ . If  $P > \frac{1}{2}$ , then  $P_{i,i+n} \geq P$  with equality when  $n = 1$ , and  $P_{i,i+n}$  increases strictly monotonically with  $n$ . If  $P < \frac{1}{2}$ , then  $P_{i,i+n} \leq P$  with equality when  $n = 1$ , and  $P_{i,i+n}$  decreases strictly monotonically with  $n$ . If  $P = \frac{1}{2}$ , then  $P_{i,i+n} = \frac{1}{2}$  for all  $n$ .*

**Proof:** Assume that  $n > 0$ . Since  $P_{i,i+n} = 1 / (1 + (\frac{1-P}{P})^n)$ , then for  $P > \frac{1}{2}$ ,  $\frac{1-P}{P} < 1$  and the denominator decreases strictly monotonically with  $n$ ; and for  $P < \frac{1}{2}$ ,  $\frac{1-P}{P} > 1$  and the denominator increases strictly monotonically with  $n$ ; and for  $P = \frac{1}{2}$ ,  $P_{i,i+n} = \frac{1}{2}$  by substitution. Finally if  $n = 1$ , then  $P_{i,i+n} = P$  by construction.  $\square$

We end this section with the following observation. In [HT98] and [BT52], the authors consider models of the following form: for some fixed set of events  $A_1, \dots, A_k$ , pairwise probabilities  $P(A_i | A_i \text{ or } A_j)$  are given, and it is assumed that there is a set of probabilities  $\hat{P}_i$  such that  $P(A_i | A_i \text{ or } A_j) = \hat{P}_i / (\hat{P}_i + \hat{P}_j)$ . This is closely related to the model described here, where for example one can model  $\hat{P}_i$  as  $N \exp(o_i)$ , where  $N$  is an overall normalization.

## 5.2 RankNet: Learning to Rank with Neural Nets

The above cost function is general, in that it is not tied to any particular learning model; here we explore using it in neural network models. Neural networks provide us with a large class of easily learned functions to choose from. Let us remind the reader of the general back-prop equations<sup>6</sup> for a two layer net with  $q$  output nodes [LBOM98]. For training sample  $\mathbf{x}$ , denote the outputs of net by  $o_i$ ,  $i = 1, \dots, q$ , the targets by  $t_i$ ,  $i = 1, \dots, q$ , let the transfer function of each node in the  $j$ th layer of nodes be  $g^j$ , and let the cost function be  $\sum_{i=1}^q C(o_i, t_i)$ . If  $\alpha_k$  are the parameters of the model, then a gradient descent step amounts to  $\delta\alpha_k = -\eta_k \frac{\partial f}{\partial \alpha_k}$ , where the  $\eta_k$  are positive learning rates. This network embodies the function

$$o_i = g^3 \left( \sum_j w_{ij}^{32} g^2 \left( \sum_k w_{jk}^{21} x_k + b_j^2 \right) + b_i^3 \right) \equiv g_i^3 \quad (13)$$

where for the weights  $w$  and offsets  $b$ , the upper indices index the node layer, and the lower indices index the nodes within each corresponding layer. Taking derivatives of  $C$  with respect to the parameters gives

$$\frac{\partial C}{\partial b_i^3} = \frac{\partial C}{\partial o_i} g_i^3 \equiv \Delta_i^3 \quad (14)$$

$$\frac{\partial C}{\partial w_{in}^{32}} = \Delta_i^3 g_n^2 \quad (15)$$

$$\frac{\partial C}{\partial b_m^2} = g_m^2 \left( \sum_i \Delta_i^3 w_{im}^{32} \right) \equiv \Delta_m^2 \quad (16)$$

$$\frac{\partial C}{\partial w_{mn}^{21}} = x_n \Delta_m^2 \quad (17)$$

<sup>6</sup> *Back-prop* gets its name from the propagation of the  $\Delta$ 's backwards through the network (cf. Eq. 14), by analogy to the 'forward prop' of the node activations.

where  $x_n$  is the  $n$ th component of the input. Thus, 'backProp' consists of a forward pass, during which the activations, and their derivatives, for each node are stored;  $\Delta_1^3$  is computed for the output layer, and is then used to update the bias  $b$  for the output node; the weight updates for the  $w^{32}$  are then computed by simply multiplying  $\Delta_1^3$  by the outputs of the hidden nodes; the  $\Delta_m^2$  are then computed using the activation gradients and the current weight values; and the process repeats for the layer below. This procedure generalizes in the obvious way for more general networks.

Turning now to a net with a single output, the above is generalized to the ranking problem as follows [BSR<sup>+</sup>05]. Recall that the cost function is a function of the difference of the outputs of two consecutive training samples:  $C(o_2 - o_1)$ . Here it is assumed that the first pattern is known to rank higher than, or equal to, the second (so that, in the first case,  $C$  is chosen to be monotonic increasing). Note that  $C$  can include parameters encoding the importance assigned to a given pair. A forward prop is performed for the first sample; each node's activation and gradient value are stored; a forward prop is then performed for the second sample, and the activations and gradients are again stored. The gradient of the cost is then

$$\frac{\partial C}{\partial \alpha} = \left( \frac{\partial o_2}{\partial \alpha} - \frac{\partial o_1}{\partial \alpha} \right) C' \quad (18)$$

where  $C'$  is just the derivative of  $C$  with respect to  $o_2 - o_1$ . We use the same notation as before but add a subscript, 1 or 2, denoting which pattern is the argument of the given function, and we drop the index on the last layer. Thus we have

$$\frac{\partial C}{\partial b^3} = f'(g_2^3 - g_1^3) \equiv \Delta_2^3 - \Delta_1^3 \quad (19)$$

$$\frac{\partial C}{\partial w_m^{32}} = \Delta_2^3 g_{2m}^2 - \Delta_1^3 g_{1m}^2 \quad (20)$$

$$\frac{\partial C}{\partial b_m^2} = \Delta_2^3 w_m^{32} g_{2m}^2 - \Delta_1^3 w_m^{32} g_{1m}^2 \quad (21)$$

$$\frac{\partial C}{\partial w_{mn}^{21}} = \Delta_{2m}^2 g_{2n}^1 - \Delta_{1m}^2 g_{1n}^1 \quad (22)$$

Note that the terms always take the form<sup>7</sup> of the difference of a term depending on  $\mathbf{x}_1$  and a term depending on  $\mathbf{x}_2$ , 'coupled' by an overall multiplicative factor of  $C'$ , which depends on both. A sum over weights does not appear because we are considering a two layer net with one output, but for more layers the sum appears as above; thus training RankNet is accomplished by a straightforward modification of the back-prop algorithm.

## 6 Ranking as Learning Structured Outputs

Let's take a step back and ask: are the above algorithms solving the right problem? They are certainly attempting to learn an ordering of the data. However, in this

---

<sup>7</sup> One can also view this as a weight sharing update for a Siamese-like net [BBB<sup>+</sup>93]. However Siamese nets use a cosine similarity measure for the cost function, which results in a different form for the update equations.

Section I argue that, in general, the answer is no. Let’s revisit the cost metrics described in Section 2. 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  $F$ , 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 described 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, 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. This itself has several root causes: the amount of available labeled data may be insufficient; the algorithms themselves have finite capacity to learn (and if the amount of training data is limited, as is often the case, this is a very desirable property [Vap95]); and due to noise in the data and/or the labels, perfect generalization is often not even theoretically possible.

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 (such as the RankNet cost function), 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 error 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 traditional ranking problems. For example, one measure of quality for document retrieval, or in fact of classifiers in general, is the “AUC”, the area under the ROC curve [Bam75]. Maximizing the AUC amounts to learning using a multivariate cost and is in fact also exactly a binary ranking problem: see, for example, [CM05, HR04]. Similarly, optimizing measures that depend on precision and recall can be viewed as optimizing a multivariate cost [Joa05, HR04].

In order to learn using a multivariate, non-differentiable cost function, we propose a general approach, 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 the desired 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

$$\left| \frac{\partial C}{\partial s_1} \right| \gg \left| \frac{\partial C}{\partial s_2} \right| \tag{23}$$

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 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

$$\frac{\partial C}{\partial s_1} = -\lambda_1(s_1, l_1, s_2, l_2) \tag{24}$$

$$\frac{\partial C}{\partial s_2} = -\lambda_2(s_1, l_1, s_2, l_2) \tag{25}$$

We would like the  $\lambda$ 's to take the form shown in Figure 2, 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  $\delta$ , and having nonzero  $\delta$  ensures robustness to small changes in the scores  $s_i$ .

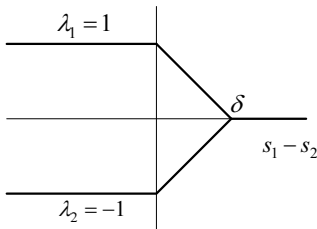


Fig. 2. Choosing the lambda's for a query with two documents.

Letting  $x \equiv s_1 - s_2$ , the  $\lambda$ 's may be written

$$x < 0 : \lambda_1 = 1 = -\lambda_2 \quad (26)$$

$$0 \leq x \leq \delta : \lambda_1 = \delta - x = -\lambda_2 \quad (27)$$

$$x > \delta : \lambda_1 = \lambda_2 = 0 \quad (28)$$

In this case a corresponding cost function exists:

$$x < 0 : C(s_1, l_1, s_2, l_2) = s_2 - s_1 \quad (29)$$

$$0 \leq x \leq \delta : C(s_1, l_1, s_2, l_2) = \frac{1}{2}(s_1 - s_2)^2 - \delta(s_1 - s_2) \quad (30)$$

$$x > \delta : C(s_1, l_1, s_2, l_2) = -\frac{1}{2}\delta^2 \quad (31)$$

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  $\lambda$ 's 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  $\lambda$ 's to best model a given (multivariate, non-differentiable) cost function. Let's call this choice the  $\lambda$ -function. We will not explore here how, given a cost function, to find a particular  $\lambda$ -function, but instead will answer two questions which will help guide the choice: first, for a given choice of the  $\lambda$ '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 [Spi65]:

**Theorem 2.** (*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  $n_i$   $\lambda$ '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

$$\boldsymbol{\lambda} \equiv \sum_i \lambda_i dx^i \quad (32)$$

Then assuming that the scores are defined over  $\mathcal{R}^n$ , the conditions for Theorem 2 are satisfied and  $\boldsymbol{\lambda} = dC$  for some function  $C$  if and only if  $d\boldsymbol{\lambda} = 0$  everywhere. Using classical notation, this amounts to requiring that

$$\frac{\partial \lambda_i}{\partial s_j} = \frac{\partial \lambda_j}{\partial s_i} \quad \forall i, j \quad (33)$$

Thus we have a simple test on the  $\lambda$ '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  $\lambda$ 's that are piecewise constant, the above two conditions (symmetric and positive semidefinite<sup>8</sup>) 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  $\lambda$  then corresponds to a force on the corresponding point. If the conditions of Eq. (33) 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  $\lambda$ '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).

## Acknowledgements

I thank J. Levesley and J. Platt for useful discussions.

## References

- [Bam75] D. Bamber. The area above the ordinal dominance graph and the area below the receiver operating characteristic graph. *Journal of Mathematical Psychology*, 12:387–415, 1975.
- [BBB<sup>+</sup>93] J. Bromley, J.W. Bentz, L. Bottou, I. Guyon, Y. LeCun, C. Moore, E. Sackinger, and R. Shah. Signature Verification Using a "Siamese" Time Delay Neural Network. In I. Guyon and P.S.P Wang, editors, *Advances in Pattern Recognition Systems using Neural Network Technologies*, Machine Perception Artificial Intelligence 7, pages 25–44. World Scientific, 1993.
- [BSR<sup>+</sup>05] C.J.C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank using Gradient Descent. In *Proceedings of the Twenty Second International Conference on Machine Learning*, Bonn, Germany, 2005.

---

<sup>8</sup> Some authors define the property of positive semi-definiteness to include the property of symmetry: see [HJ85].

- [BT52] R. Bradley and M. Terry. The Rank Analysis of Incomplete Block Designs 1: The Method of Paired Comparisons. *Biometrika*, 39:324–245, 1952.
- [Bur98] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.
- [BW88] E.B. Baum and F. Wilczek. Supervised learning of probability distributions by neural networks. In D. Anderson, editor, *Neural Information Processing Systems*, pages 52–61. American Institute of Physics, 1988.
- [CK05] W. Chu and S.S. Keerthi. New approaches to support vector ordinal regression. In *Proceedings of the Twenty Second International Conference on Machine Learning*, Bonn, Germany, 2005.
- [CM05] C. Cortes and M. Mohri. Confidence Intervals for the Area Under the ROC Curve. In *Advances in Neural Information Processing Systems 18*. MIT Press, 2005.
- [CS02] K. Crammer and Y. Singer. Pranking with ranking. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2002.
- [CV95] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
- [DMS04] O. Dekel, C.D. Manning, and Y. Singer. Log-linear models for label-ranking. In *Advances in Neural Information Processing Systems 16*. MIT Press, 2004.
- [Har03] E.F. Harrington. Online ranking/collaborative filtering using the Perceptron algorithm. In *Proceedings of the Twentieth International Conference on Machine Learning*, 2003.
- [HGO00] R. Herbrich, T. Graepel, and K. Obermayer. Large margin rank boundaries for ordinal regression. In A.J. Smola, P.L. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*, pages 115–132. MIT Press, 2000.
- [HJ85] R.A. Horn and C.R. Johnson. *Matrix Analysis*. Cambridge University Press, 1985.
- [HR04] A. Herschtal and B. Raskutti. Optimising Area Under the ROC curve using Gradient Descent. In *Proceedings of the Twenty First International Conference on Machine Learning*, Banff, Canada, 2004.
- [HT98] T. Hastie and R. Tibshirani. Classification by pairwise coupling. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.
- [JK00] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 41–48, New York, 2000. ACM.
- [Joa02] T. Joachims. Optimizing search engines using clickthrough data. In David Hand, Daniel Keim, and Raymond Ng, editors, *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-02)*, pages 132–142, New York, 2002. ACM Press.
- [Joa05] T. Joachims. A support vector method for multivariate performance measures. In Luc De Raedt and Stefan Wrobel, editors, *Proceedings of the 22nd International Conference on Machine Learning*, pages 377–384, 2005.

- [LBOM98] Yann LeCun, Leon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient backprop. In G.B. Orr and K-L Müller, editors, *Neural Networks: Tricks of the Trade*, pages 9–50. Springer, 1998.
- [RV91] P. Refregier and F. Vallet. Probabilistic approaches for multiclass classification with neural networks. In *International Conference on Artificial Neural Networks*, pages 1003–1006. Elsevier, 1991.
- [Spi65] M. Spivak. *Calculus on Manifolds*. Addison-Wesley, 1965.
- [SS02] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.
- [Vap95] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, New York, 1995.
- [Voo01] E.M. Voorhees. Overview of the TREC 2001 Question Answering Track. In *TREC*, 2001.
- [Voo02] E.M. Voorhees. Overview of the TREC 2002 Question Answering Track. In *TREC*, 2002.