IJCOT -Special Issue– The Malla Reddy National Conference on Information System and Knowledge Engineering (MRNC-ISKE 2013) - July 2013

Target Domains Data Extraction Using Margin Technology

K.Madhusudan Reddy*1, S.Md.Ibrahim*2
M.Tech, Dept of CSE, SKDEC, Gooty, D.t: Anantapuram, A.P, India
M.Tech, Assistant Professor, Dept of CSE, SKDEC, Gooty, D.t: Anantapuram, A.P, India

ABSTRACT

The quality measures used in information retrieval are particularly difficult to optimize directly, since they depend on the model scores only through the sorted order of the documents returned for a given query. Thus, the derivatives of the cost with respect to the model parameters are either zero, or are undefined. In this paper, we propose a class of simple, flexible algorithms, called LambdaRank, which avoids these difficulties by working with implicit cost functions. We describe LambdaRank using neural network models, although the idea applies to any differentiable function class. We give necessary and sufficient conditions for the resulting implicit cost function to be convex, and we show that the general method has a simple mechanical interpretation. We demonstrate significantly improved accuracy, over a state-of-the-art ranking algorithm, on several datasets. We investigate using gradient descent methods for learning ranking functions; we propose a simple probabilistic cost function, and we introduce RankNet, an implementation of these ideas using a neural network to model the underlying ranking function. We present test results on toy data and on data from a commercial internet search engine.

KEYWORDS: Rank Net, Bounded Continuous Function, Learning Functions.

I. INTRODUCTION

Any system that presents results to a user, ordered by a utility function that the user cares about, is performing a ranking function. A common example is the ranking of search results, for example from the Web or from an intranet; this is the task we will consider in this paper. For this problem, the data consists of a set of queries, and for each query, a set of returned documents. In the training phase, some query/document pairs are labeled for relevance (\excellent match", \good match", etc.). Only those documents returned for a given query are to be ranked against each other. Thus, rather than consisting of a single set of objects to be ranked amongst each other; the data is instead partitioned by query. In this paper we propose a new approach to this problem. Our approach follows in that we train on pairs of examples to learn a ranking function that maps to the real (having the model evaluate on pairs would be prohibitively slow for many applications). However cast the ranking problem as an ordinal regression problem; rank boundaries play a critical role during training, as they do for several other algorithms (Crammer & Singer, 2002; Harrington, 2003). For our application, given that item A appears higher than item B in the output list, the user concludes that the system ranks A higher than, or equal to, B; no mapping to particular rank values, and no rank boundaries, are needed; to cast this as an ordinal regression problem is to solve an unnecessarily hard problem, and our approach avoids this extra step. We also propose a natural probabilistic cost function on pairs of examples. Such an approach is not specific to the underlying learning algorithm; we chose to explore these ideas using neural networks, since they are exible (e.g. two layer neural nets can approximate any bounded continuous function, and since they are often faster in test phase than competing kernel methods (and test speed is critical for this application); however our cost function could equally well be applied to a variety of machine learning algorithms. For the neural net

case, we show that back propagation is easily extended to handle ordered pairs; we call the resulting algorithm, together with the probabilistic cost function we describe below, RankNet. We present results on toy data and on data gathered from a commercial internet search engine. For the latter, the data takes the form of 17,004 queries, and for each query, up to 1000 returned documents, namely the top documents returned by another, simple ranker. Thus each query generates up to 1000 feature vectors. In many inference tasks, the cost function1 used to assess the final quality of the system is not the one used during training. For example for classification tasks, an error rate for a binary SVM classifier might be reported, although the cost function used to train the SVM only very loosely models the number of errors on the training set, and similarly neural net training uses smooth costs, such as MSE or cross entropy. Thus often in machine learning tasks, there are actually two cost functions: the desired cost, and the one used in the optimization process. For brevity we will call the former the 'target' cost, and the latter the 'optimization' cost. The optimization cost plays two roles: it is chosen to make the optimization task tractable (smooth, convex etc.), and it should approximate the desired cost well. This mismatch between target and optimization costs is not limited to classification tasks, and is particularly acute for information retrieval. For example, [10] list nine target quality measures that are commonly used in information retrieval, all of which depend only on the sorted order of the documents2 and their labeled relevance. The target costs are usually averaged over a large number of queries to arrive at a single cost that can be used to assess the algorithm. These target costs present severe challenges to machine learning: they are either flat (have zero gradient with respect to the model scores), or are discontinuous, everywhere. It is very likely that a significant mismatch between the target and optimizations costs will have a substantial adverse impact on the accuracy of the algorithm.

2. RELATED WORK

RankProp is also a neural net ranking model. RankProp alternates between two phases: an MSE regression on the current target values, and an adjustment of the target values themselves to reect the current ranking given by the net. The end result is a mapping of the data to a large number of targets which reect the desired ranking, which performs better than just regressing to the original, scaled rank values. Rank prop has the advantage that it is trained on individual patterns rather than pairs; however it is not known under what conditions it converges, and it does not give a probabilistic model. cast the problem of learning to rank as ordinal regression, that is, learning the mapping of an input vector to a member of an ordered set of numerical ranks. They model ranks as intervals on the real line, and consider loss functions that depend on pairs of examples and their target ranks. The positions of the rank boundaries play a critical role in the final ranking function. (Crammer & Singer, 2002) cast the problem in similar form and propose a ranker based on the perceptron ('PRank'), which maps a feature vector x 2 Rd to the real with a learned w 2 Rd such that the output of the mapping function is just w _ x. PRank also learns the values of N increasing thresholds1 br = 1; _ _ _ ;N and declares the rank of x to be minrfw _ x □ br < 0g. PRank learns using one example at a time, which is held as an advantage over pair-based methods (e.g. (Freund et al., 2003)), since the latter must learn using O(m2) pairs rather than m examples. However this is not the case in our application; the number of pairs is much smaller than m2, since documents are only compared to other documents retrieved for the same query, and since many feature vectors have the same assigned rank. We find that for our task the memory usage is strongly dominated by the feature vectors themselves. Although the linear version is an online algorithm2, PRank has been compared to batch ranking algorithms, and a quadratic kernel version was found to outperform all such algorithms

described in (Herbrich et al., 2000). (Harrington, 2003) has proposed a simple but very effective extension of PRank, which approximates finding the Bayes point by averaging over PRank models. Therefore in this paper we will compare RankNet with PRank, kernel PRank, large margin PRank, and RankProp. 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 (which here and below we write as A B B). This approach can represent arbitrary ranking functions, in particular, ones that are inconsistent - for example A B B, B B C, C B A. We adopt this more general view, and 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. In addition, we introduce a probabilistic model, so that each training pair fA;Bg has associated posterior P(A B B). This is an important feature of our 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.) Finally, we use cost functions that are functions 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; again, the goal is to avoid unnecessary learning.

RankBoost (Freund et al., 2003) is another ranking algorithm that is trained on pairs, and which is closer in spirit to our work since it attempts to solve the preference learning problem directly, rather than solving an ordinal regression problem. In (Freund et al., 2003), results are given using decision stumps as the weak learners. The cost is a function of the margin over reweighted examples. Since boosting can be viewed as gradient descent (Mason et al., 2000), the question naturally arises as to how combining RankBoost with our pair-wise differentiable cost function would compare. Due to space constraints we will describe this work elsewhere.

The ranking task is the task of finding a sort on a set, and as such is related to the task of learning structured outputs. Our approach is very different, however, from recent work on structured outputs, such as the large margin methods of. There, structures are also mapped to the reals (through choice of a suitable inner product), but the best output is found by estimating the argmax over all possible outputs. The ranking problem also maps outputs (documents) to the real, but solves a much simpler problem in that the number of documents to be sorted is tractable. Our focus is on a very different aspect of the problem, namely, finding ways to directly optimize the cost that the user ultimately cares about. As in, we handle cost functions that are multivariate, in the sense

that the number of documents returned for a given query can itself vary, but the key challenge we address in this paper is how to work with costs that are everywhere either flat or non-differentiable. However, we emphasize that the method also handles the case of multivariate costs that cannot be represented as a sum of terms, each depending on the output for a single feature vector and its label.

We call such functions *irreducible* (such costs are also considered by [7]). Most cost functions used in machine learning are instead reducible (for example, MSE, cross entropy, log likelihood, and the costs commonly used in kernel methods). The ranking problem itself has attracted increasing attention recently (see for example [4, 2, 8]), and in this paper we will use the RankNet algorithm of [2] as a baseline, since it is both easy to implement and performs well on large retrieval tasks.

III. Lambda Rank

One approach to working with a Nonsmooth target cost function would be to search for an optimization function

which is a good approximation to the target cost, but which is also smooth. However, the *sort* required by information retrieval cost functions makes this problematic. Even if the target cost depends on only the top few ranked positions after sorting, the sort itself depends on all documents returned for the query, and that set can be very large; and since the target costs depend on only the rank order and the labels, the target cost functions are either flat or discontinuous in the scores of all the returned documents. We therefore consider a different approach. We illustrate the idea with an example which also demonstrates the perils introduced by a target / optimization cost mismatch.

Let the target cost be WTA and let the chosen optimization cost be a smooth approximation to pair wise error. Suppose that a ranking algorithm A is being trained, and that at some iteration, for a query for which there are only two relevant documents D1 and D2, A gives D1 rank one and D2 rank n. Then on this query, A has WTA cost zero, but a pair wise error cost of n □ 2. If the parameters of A are adjusted so that D1 has rank two, and D2 rank three, then the WTA error is now maximized, but the number of pair wise errors has been reduced by n □ 4. Now suppose that at the next iteration, D1 is at rank two, and D2 at rank n _ 1. The change in D1's score that is required to move it to top position is clearly less (possibly much less) than the change in D2's score required= to move *it* to top position. Roughly speaking, we would prefer A to spend a little capacity moving D1 up by one position, than have it spend a lot of capacity moving D2 up by n □ 1 positions.

3.1 A Boosting Algorithm for the Ranking Task

In this section, we describe an approach to the ranking problem based on a machine learning method called boosting, in particular, Freund and Schapiro's (1997) AdaBoost algorithm and its successor developed by Schapiro and Singer (1999). Boosting is a method of producing highly accurate prediction rules by combining many "weak" rules which may be only moderately accurate. In the current setting, we use boosting to produce a function $H : X$!R whose induced ordering of X will approximate the relative orderings encoded by the feedback function F.

3.2 The Rank Boost Algorithm

We call our boosting algorithm Rank Boost, and its pseudo code is shown in Figure 1. Like all boosting algorithms, RankBoost operates in rounds. We assume access to a separate procedure called the *weak learner* that, on each round, is called to produce a *weak ranking*. RankBoost maintains a distribution *Dt* over X _X that is passed on round t to the weak learner. Intuitively, RankBoost chooses *Dt* to emphasize different parts of the training data. A high weight assigned to a pair of instances indicates a great importance that the weak learner order that pair correctly.

Algorithm RankBoost
Given: initial distribution D over $X \times X$.
Initialize: $D_1 = D$.
For $t = 1,\ldots,T$:

- Train weak learner using distribution D_t.
- Get weak ranking $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update: $D_{t+1}(x_0,x_1) = \dfrac{D_t(x_0,x_1)\exp(\alpha_t(h_t(x_0) - h_t(x_1)))}{Z_t}$

  where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final ranking: $H(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$

Figure 1: The RankBoost algorithm.

Weak rankings have the form $ht : X$!R. We think of these as providing ranking information in the same manner as ranking features and the final ranking. The weak learner we used in our experiments is based on the given ranking features.

The boosting algorithm uses the weak rankings to update the distribution as shown in Figure 1. Suppose that $x0$;

$x1$ is a crucial pair so that we want $x1$ to be ranked higher than $x0$ (in all other cases, Dt will be zero). Assuming for the moment that the parameter at > 0 (as it usually will be), this rule decreases the weight Dt ($x0$; $x1$) if ht gives a correct ranking (ht ($x1$) > ht ($x0$)) and increases the weight otherwise. Thus, Dt will tend to concentrate on the pairs whose relative ranking is hardest to determine. The actual setting of at will be discussed shortly.

The final ranking H is a weighted sum of the weak rankings. In the following theorem we prove a bound on the ranking loss of H. This theorem also provides guidance in choosing at and in designing the weak learner as we discuss below. on the training data. As in standard classification problems, the loss on a separate test set can also be theoretically bounded given appropriate assumptions using uniform-convergence theory.

3.3 Learning to Rank using Gradient Descent

RankProp is also a neural net ranking model. RankProp alternates between two phases: an MSE regression on the current target values, and an adjustment of the target values themselves to reect the current ranking given by the net. The end result is a mapping of the data to a large number of targets which reect the desired ranking, which performs better than just regressing to the original, scaled rank values. Rank prop has the advantage that it is trained on individual patterns rather than pairs; however it is not known under what conditions it converges, and it does not give a probabilistic model. (Herbrich et al., 2000) cast the problem of learning to rank as ordinal regression, that is, learning the mapping of an input vector to a member of an ordered set of numerical ranks. They model ranks as intervals on the real line, and consider loss functions that depend on pairs of examples and their target ranks. The positions of the rank boundaries play a critical role in the final ranking function.

IV.Learning to Rank with Nonsmooth Cost Functions

The ranking task is the task of finding a sort on a set, and as such is related to the task of learning structured outputs. Our approach is very different, however, from recent work on structured outputs, such as the large margin methods of There, structures are also mapped to the reals (through choice of a suitable inner product), but the best output is found by estimating the argmax over all possible outputs. The ranking problem also maps outputs (documents) to the reals, but solves a much simpler problem in that the number of documents to be sorted is tractable. Our focus is on a very different aspect of the problem, namely, finding ways to directly optimize the cost that the user ultimately cares about. As in , we handle cost functions that are multivariate, in the sense that the number of documents returned for a given query can itself vary, but the key challenge we address in this paper is how to work with costs that are everywhere either flat or non-differentiable. However, we emphasize that the method also handles the case of multivariate costs that cannot be represented as a sum of terms, each depending on the output for a single feature vector and its label.

V.Domain Adaptation with Structural Correspondence Learning

Domain adaptation is an important and well studied area in natural language processing. Here we outline a few recent advances. Roark and Bacchiani (2003) use a Dirichlet prior on the multinomial parameters of a generative parsing model to combine a large amount of training data from a source corpus (WSJ), and small amount of training data from a target corpus (Brown). Aside from Florian et al. (2004), several authors have also given techniques for adapting classification to new domains. Chelan and Acero first train a classifier on the source data. Then they use maximum a posteriori estimation of the weights of a maximum entropy target domain classifier. The prior is Gaussian with mean equal to the weights of the source domain classifier. Daum´e

III and Marcu (2006) use an empirical Bayes model to estimate a latent variable model grouping instances into domain-specific or common across both domains.

They also jointly estimate the parameters of the common classification model and the domain specific classification models. Our work focuses on finding a common representation for *features* from different domains, not instances. We believe this is an important distinction, since the same instance can contain some features which are common across domains and some which are domain specific.

VI Conclusions

We have demonstrated a simple and effective method for learning non-smooth target costs. LambdaRank is a general approach: in particular, it can be used to implement RankNet training, and it furnishes a significant training speedup there. We studied LambdaRank in the context of the NDCG target cost for neural network models, but the same ideas apply to any non-smooth target cost, and to any differentiable function class. It would be interesting to investigate using the same method starting with other classifiers such as boosted trees.

VII. References

[1] C. Buckley and E. Voorhees. Evaluating evaluation measure stability. In SIGIR, pages 33–40, 2000.

[2] 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, Bonn, Germany, 2005.

[3] C. Cortes and M. Mohri. Confidence Intervals for the Area Under the ROC Curve. In NIPS 18. MIT Press, 2005.

[4] Y. Freund, R. Iyer, R.E. Schapire, and Y. Singer. An efficient boosting algorithm for combining preferences. Journal of Machine Learning Research, 4:933–969, 2003.

[5] J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: A statistical view of boosting. The Annals of Statistics, 28(2):337–374, 2000.

[6] K. Jarvelin and J. Kekalainen. IR evaluation methods for retrieving highly relevant documents. In SIGIR
23. ACM, 2000.

[7] T. Joachims. A support vector method for multivariate performance measures. In ICML 22, 2005.

[8] I. Matveeva, C. Burges, T. Burkard, A. Lauscius, and L. Wong. High accuracy retrieval with multiple nested rankers. In SIGIR, 2006.

[9] I. Newton. Philosophiae Naturalis Principia Mathematica. The Royal Society, 1687.

[10] S. Robertson and H. Zaragoza. On rank-based effectiveness measures and optimisation. Technical Report MSR-TR-2006-61, Microsoft Research, 2006.

[11] M. Spivak. Calculus on Manifolds. Addison-Wesley, 1965.

[12] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin. Learning structured prediciton models: A large margin approach. In ICML 22, Bonn, Germany, 2005.

[13] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In ICML 24, 2004.

[14] E.M. Voorhees. Overview of the TREC 2001/2002 Question Answering Track. In TREC, 2001,2002.

[15] L. Yan, R. Dodlier, M.C. Mozer, and R. Wolniewicz. Optimizing Classifier Performance via an Approximation to the Wilcoxon-Mann-Whitney Statistic. In ICML 20, 2003.

K.Madhusudan Reddy.M.Tech,

SRI KRISHNA DEVARAYA ENGINEERING COLLEGE,

JNTU Ananthapuram.

S.Md.Ibrahim M.Tech,

Assistant Professor, Department of CSE, SRI KRISHNA DEVARAYA ENGINEERING COLLEGE,

Gooty, Ananthapuram (Dt).