An Ensemble Ranking Solution for the Yahoo! Learning to Rank Challenge

Ming-Feng Tsai D92003@CSIE.NTU.EDU.TW
Department of Computer Science
University of Singapore
13 Computing Drive, Singapore

Shang-Tse Chen B95100@CSIE.NTU.EDU.TW
Yao-Nan Chen B95049@CSIE.NTU.EDU.TW
Chun-Sung Ferng B95018@CSIE.NTU.EDU.TW
Chia-Hsuan Wang B94B02009@CSIE.NTU.EDU.TW
Tzay-Yeu Wen B95106@CSIE.NTU.EDU.TW
Hsuan-Tien Lin HTLIN@CSIE.NTU.EDU.TW
Department of Computer Science and Information Engineering
National Taiwan University
Taipei 106, Taiwan

Editor:

Abstract

This paper describes our proposed solution for the Yahoo! Learning to Rank challenge. The solution consists of an ensemble of three point-wise, two pair-wise and one list-wise approaches. In our experiments, the point-wise approaches are observed to outperform pair-wise and list-wise ones in general, and the final ensemble is capable of further improving the performance over any single approach. In terms of the online validation performance, our proposed solution achieves an ERR of 0.4565 (NDCG 0.7870) for set 1.

Keywords: ranking, ensemble learning

1. Introduction

We explore six approaches to learn from set 1 of the Yahoo! Learning to Rank challenge. To train with the huge set effectively and efficiently, we adopt three point-wise ranking approaches: ORSVM, Poly-ORSVM, and ORBoost; to capture the essence of the ranking problem, we take two pair-wise ranking approaches: Linear RankSVM and Rank Logistic Regression; to take the evaluation criteria of the competition into account, we include one list-wise ranking approach: BoltzRank. We systematically study several settings of each approach and compare their strength and weakness. After the studies, we combine some selected settings of each approach and form an ensemble for the final submission. Next, we will first describe our studies on the six approaches and the corresponding results in Sections 2 to 7. Then, we report our final ensemble in Section 8 and summarize in Section 9.

Below we describe the notations that will be used throughout this paper: $\mathbf{x}$ represents an instance vector in $\mathbb{R}^d$; y represents its given rank (label) in the range of $\{0, 1, \ldots, K-1\}$. The total number of queries is Q, and the given data set of the competition is of the

form $\bigcup_{q=1}^{Q}\{(q, \mathbf{x}_{qn}, y_{qn})\}_{n=1}^{N_q}$, where N_q is the number of examples $(q, \mathbf{x}_{qn}, y_{qn})$ in query q. In the training part of set 1 in the competition, $d = 519$, $K = 5$, $Q = 19944$ and $\sum_{q=1}^{Q} N_q = 473134$. We will generally use N to denote the number of training examples that are given to a learning algorithm, and T to denote the number of iterations that an iterative learning algorithm uses for training.

2. Ordinal Ranking SVM

Ordinal Ranking SVM (ORSVM; Li and Lin, 2007) is an approach for solving ordinal ranking (regression). The approach reduces ordinal ranking to SVM-based binary classification by thresholding an underlying raw score for each instance. We apply the approach in this competition as follows. First, we combine all examples from all queries q to a big training set with ordinal ranking examples $\{(\mathbf{x}_{qn}, y_{qn})\}$. Then, we transform each example to $K-1$ binary classification examples of the form

$$\left\{\Big((\mathbf{x}_{qn}, \mathbf{e}_k), \quad 2[\![k < y_{qn}]\!] - 1\Big)\right\}_{k=1}^{K-1},$$

where $\mathbf{e}_k$ is a vector that contains a single 1 at the k-th component and 0 otherwise. The transformed examples are then fed to an SVM solver using a composite kernel defined by the summation of a nonlinear kernel for the $\mathbf{x}_{qn}$ part and a linear kernel for the $\mathbf{e}_k$ part. We use the learned decision function from the solver to predict a relevance score for any test instance $\mathbf{x}$ by plugging $(\mathbf{x}, 0, 0, \ldots, 0)$ into the function. Finally, we order the instances in the test queries by their predicted relevance scores to form the ranked list.

From the reduction view of the approach, the time complexity of ORSVM is similar to that of a usual binary SVM on $N(K-1)$ examples, with a space complexity of $O(N^2)$ for storing the nonlinear kernel part. Following the analysis of the LIBSVM solver that we adopt (Chang and Lin, 2001), the time complexity of our ORSVM implementation[1] is $O(TNKd)$ when considering a nonlinear kernel on $\mathbf{x}_{qn}$ that can be evaluated within $O(d)$ and caching most of the kernel evaluations.

An important feature of ORSVM is that we can assign a cost vector per example to indicate the penalties for different kinds of erroneous predictions (Li and Lin, 2007). This setting enables us to encode the relative importance of each instance and each prediction. To match the evaluation criteria of the competition, we adopt an ERR-oriented cost setting. In particular, the k-th component of the cost vector for an instance $\mathbf{x}_{qn}$ is defined as the difference to the optimal ERR. That is,

$$\begin{aligned} & \textit{the } k\textit{-th cost component for instance } \mathbf{x}_{qn} \\ = \; & (\textit{optimal ERR of query } q) - (\textit{ERR when } (q, \mathbf{x}_{qn}, y_{qn}) \textit{ is mis-predicted as rank } k). \end{aligned}$$

2.1 Experiments

We adopt the perceptron kernel (Lin and Li, 2008) as the nonlinear kernel for $\mathbf{x}$ in ORSVM because the kernel allows for faster parameter selection. Since the given data set is large, we select the C parameter in ORSVM with 100 smaller data sets, each of which consists of

1. downloadable from `http://www.work.caltech.edu/~htlin/program/libsvm/`

Table 1: ORSVM without ERR-oriented Cost

chunk size	400	40000	100000
training time (min)	≈ 100	≈ 500	≈ 4000
online ERR	0.4447	0.4500	0.4519

Table 2: ORSVM with ERR-oriented Cost

chunk size	40000	100000
training time (min)	≈ 1000	≈ 6000
online ERR	0.4517	0.4527

4000 examples. The best parameter is then decided by a majority vote from cross-validation on the small data sets.

The training set that we use for ORSVM (before doing ensemble learning in Section 8) consists about 400000 examples, which is time-consuming to train with a single-CPU machine. To decrease the training time, we try dividing the set into smaller chunks, and average the decision functions learned from those chunks as the final decision function. In Table 1, we report the results of general ORSVM with respect to different chunk sizes. We can see that when the chunk size increases, ORSVM can achieve better results. Nevertheless, the corresponding training time also increases rapidly.

Table 2 shows the results of ORSVM with ERR-oriented cost on different chunk sizes. Comparing Table 2 with Table 1, we see that the results with ERR-oriented cost are better than those without the cost, which means that the proposed ERR-oriented cost can improve ranking quality effectively.

3. Polynomial ORSVM

Polynomial ORSVM (Poly-ORSVM; Lin, 2008) shares the same root with ORSVM, except that the solver adopted is LIBLINEAR (Fan et al., 2008) instead of LIBSVM. The main reason for considering Poly-ORSVM with the LIBLINEAR solver is to decrease the training time needed. In particular, the dual coordinate descent implementation in LIBLINEAR is of time complexity $O(Td)$ for training a binary SVM or ORSVM. That is, there is no dependence on N and K within each iteration.[2]

As reported by Chang et al. (2010), a low-dimensional polynomial feature mapping can improve the performance of linear SVM. We adopt a similar idea in Poly-ORSVM. In particular, we append the products of original features as new feature terms, which provide nonlinear information during training. The simplest idea is to use products of two original feature components, which would be called degree-2 expansions. Nevertheless, for the original d features, using all degree-2 expansions is computational infeasible. Thus, we study two ideas: randomly keeping only d terms from the degree-2 terms, or keeping only the non-cross terms.

2. The trade-off is that LIBLINEAR may use more iterations. But in general LIBLINEAR can still benefit from its faster training per iteration.

Table 3: Poly-ORSVM with Different Terms

	random terms	random terms	non-cross terms
C parameter	0.1	1	0.01
internal validation ERR	0.4467	0.4485	0.4486

Table 4: Poly-ORSVM with Different Degrees of Expansions

	degree-1 (linear)	degree-2	degree-3	degree-4
best C selected from internal validation	0.1	0.01	0.001	0.001
training time (min)	≈ 19	≈ 8	≈ 6	≈ 13
online ERR	0.4413	0.4432	0.4451	0.4456
online NDCG	0.7566	0.7587	0.7625	0.7643

Table 3 reports the results on our internal validation data set, which is a random 20% of the original training set. We keep the same number of random terms to the non-cross terms in the experiments. From Table 3, when the numbers of appended terms are the same, Poly-ORSVM with non-cross terms is better than or similar to Poly-ORSVM with random terms. Thus, expanding only the non-cross terms, namely, the n^{th} powers of each features, is a promising way for Poly-ORSVM.

To improve the degree-2 Poly-ORSVM, we also append the 3^{rd}, and 4^{th} powers as new feature terms. The results are listed in Table 4 and demonstrate that Poly-ORSVM can efficiently achieve promising results with those new feature terms. In particular, using the 3^{rd}, and 4^{th} powers of each feature shows better results compared with the ones with only linear or the 2^{nd} power of feature. Although the ranking performance of Poly-ORSVM is not as good as that of ORSVM (online ERR 0.4527), the time cost is reduced dramatically.

4. Ordinal Regression Boosting

Ordinal Regression Boosting (ORBoost; Lin and Li, 2006) is a boosting-like approach for ordinal regression with large-margin thresholded ensembles. The is similar to ORSVM, but adjusts the underlying scores and thresholds adaptively with the help of a base learner. In our experiments, we study two variants of ORBoost: ORBoost with all margins (ORBoost-All) and ORBoost with left-right margins (ORBoost-LR). ORBoost-All attempts to minimize the absolute cost between the predicted and actual ordinal ranks, whereas ORBoost-LR attempts to minimize the classification cost.

As done by (Lin and Li, 2006), we take the decision stumps and perceptrons as base learners of ORBoost. The time complexity of ORBoost is $O(T \cdot dN \log N)$ for decision stumps and $O(T \cdot T_P N \log N)$ for perceptrons, where T is the number of boosting iterations and T_P is the number of internal perceptron learning iterations.

In addition to the ORBoost variants and the base learners, we also explore some instance-level weighting schemes to better match the evaluation criteria of the competition. In particular, for an instance $\mathbf{x}_{qn}$, we design the following schemes to emphasize high-rank instances and balance the influence of queries with excessive instances.

- weight by rank: $w_1(\mathbf{x}_{qn}) = y_{qn} + 1$

Table 5: ORBoost with Decision Stumps

T	1000		5000		10000	
internal validation	ERR	NDCG	ERR	NDCG	ERR	NDCG
ORBoost-All	0.4536	0.7579	0.4555	0.7625	0.4549	0.7625
ORBoost-LR	0.4510	0.7554	0.4518	0.7596	0.4525	0.7614
online	ERR	NDCG	ERR	NDCG	ERR	NDCG
ORBoost-All	0.4454	0.7639	0.4469	0.7676	0.4467	0.7682
ORBoost-LR	0.4447	0.7655	0.4457	0.7680	0.4457	0.7678

Table 6: ORBoost-All with Decision Stumps and Perceptrons

	internal validation		online		
	ERR	NDCG	ERR	NDCG	training time (min)
decision stump	0.4536	0.7579	0.4454	0.7639	424
perceptron	0.4463	0.7479	0.4447	0.7655	6805

- weight by normalizing per query: $w_2(\mathbf{x}_{qn}) = \frac{1}{N_q}$
- combine the above two schemes: $w_3(\mathbf{x}_{qn}) = \frac{y_{qn}+1}{\sum_m (y_{qm}+1)}$
- weight by pair-wise ranking:

$$w_4(\mathbf{x}_{qn}) \propto \sum_m \Big| y_{qn} - y_{qm} \Big|^{\max(y_{qn}, y_{qm})}$$

  with $\sum_i w_4(\mathbf{x}_{qi}) = 1$ for each query q (like w_2).

4.1 Experiments

First we conduct a comparison between ORBoost-All and ORBoost-LR. We randomly keep 10% of the original data set as internal validation, and use the other 90% for training. In these experiments, decision stumps are used as base learners. As shown in Table 5, ORBoost-All is better than ORBoost-LR in terms of both ERR and NDCG.

Next, we compare the decision stump and perceptron base learners with ORBoost-All. The results are shown in Table 6, which indicates that decision stumps are superior to perceptrons in terms of both effectiveness and efficiency.

Table 7 compares the different weighting schemes proposed. In general, all the weighting schemes improve the ERR score, although the improvements are small sometimes. The highest online ERR obtained from ORBoost is 0.4473 with the w_3 weighting scheme.

5. Linear RankSVM

RankSVM (Herbrich et al., 1999) reduces the ranking problem to binary classification by pair-wise comparisons. In particular, it works on pairs of examples $(\mathbf{x}_{qn}, \mathbf{x}_{pm})$ and their corresponding ranks (y_{qn}, y_{pm}) with $y_{qn} \neq y_{pm}$. For each pair, linear RankSVM forms a

Table 7: ORBoost-All with Decision Stump under Different Weighting Schemes

$T = 1000$	training		validation		online		training time (min)
	ERR	NDCG	ERR	NDCG	ERR	NDCG	
w_1	0.4502	0.7511	0.4539	0.7526	0.4464	0.7636	431
w_2	0.4497	0.7534	0.4562	0.7556	0.4467	0.7651	487
w_3	0.4505	0.7503	0.4548	0.7498	0.4462	0.7618	465
w_4	0.4516	0.7502	0.4550	0.7500	0.4463	0.7594	493
$T = 5000$	training		validation		online		training time (min)
	ERR	NDCG	ERR	NDCG	ERR	NDCG	
w_1	0.4526	0.7577	0.4538	0.7530	N/A	N/A	2375
w_2	0.4533	0.7610	0.4560	0.7560	0.4471	0.7664	2481
w_3	0.4536	0.7573	0.4531	0.7524	0.4473	0.7634	2205
w_4	0.4556	0.7590	0.4551	0.7510	0.4459	0.7602	2238

Table 8: RankSVM with Different Settings

	internal validation		online	
settings	ERR	NDCG	ERR	NDCG
overall normalization	0.4253	0.699	N/A	N/A
query-level normalization	0.4387	0.7213	0.4290	0.7226
query-level normalization with weight scheme	0.4479	0.7449	0.4428	0.7550

binary example of the form

$$\Big(\mathbf{x}_{qn} - \mathbf{x}_{pm}, \quad 2[\![y_{qn} < y_{pm}]\!] - 1\Big)$$

Then, the binary examples are sent to a linear SVM solver (without the bias term) to learn a decision function, which can be directly used to compute the relevance score for any test instance.

For this competition, in view of efficiency, we construct only pairs within the same query. That is, only pairs with $q = p$ are considered. The off-line construction and loading the data into memory takes $O(\sum_q N_q^2)$. Then, similar to Poly-ORSVM, we use LIBLINEAR (Fan et al., 2008) to solve the binary classification problem with $O(Td)$ time complexity. To improve the performance with respect to the ERR criteria, we also carry out a weighting scheme to emphasize important pairs. In particular, for a pair with rank (y_{qn}, y_{pm}), its weight is set to $\max(y_{qn}, y_{pm})^{|y_{qn}-y_{pm}|}$.

In addition to the usual overall normalization of the features, we also experiment with another setting: query-level normalization. The goal of the setting is to include some query-level information between examples into training.

5.1 Experiments

Table 8 lists the results of RankSVM with different settings. As shown in the table, with query-level feature normalization, RankSVM can get better ranking quality over that with overall normalization. This enhancement may be due to the fact that, within a query,

Table 9: RankLR with Different Settings

	unmodified	step changing	step changing + 2^{nd} order expansion
online ERR	0.4410	0.4442	0.4503
online NDCG	0.7606	0.7596	0.7721
time (min)	1.5	1.5	103

feature will become more discriminative after being normalized. In addition, when the proposed weighting scheme is used, the result is even better than the previous two settings.

6. Rank Logistic Regression

Rank Logistic Regression (RankLR Sculley, 2009) is a pair-wise ranking method similar to RankSVM, but solves the pair-wise ranking task with binary logistic regression rather than SVM. The formulation allows the use of stochastic gradient descent to mitigate the computational difficulties that come from the large number of pairs. Each iteration of RankLR updates the weight vector of the linear function according to the gradient on a randomly-chosen candidate pair. We investigate several ways for the random choice, and find the best one to be uniformly choosing a query q first before drawing draw a pair $(\mathbf{x}_{qn}, \mathbf{x}_{qm})$ that come with $y_{qn} \neq y_{qm}$. Equivalently, it means we view each query of the same importance, similar to the w_2 weighting scheme in ORBoost.

To improve the performance of RankLR, we adopt two variant settings. First, to emphasize the importance of top-ranked instances, we multiply the step size of stochastic gradient descent by $(2^{y_{qn}} - 2^{y_{qm}})$ for the pairs with ranks (y_{qn}, y_{qm}). Second, we add nonlinearity to the formulation by considering an exact 2^{nd} order polynomial expansion with cross-terms.

The time complexity of our final RankLR is simply $O(T \cdot d^2)$, where T is the number of iterations and $O(d^2)$ comes from the weight vector for the 2^{nd} order polynomial expansion.

6.1 Experiments

In our experiments, we set the number of iterations T to 10000000, which is smaller than the total with-in query pairs of 5178545. Nevertheless, we observe that such a choice of T is sufficient for achieving a promising performance.

Table 9 shows the results of RankLR using 80% of the original training examples. We can see that both modifications, step changing and 2^{nd} order expansion, improve the unmodified RankLR significantly.

7. BoltzRank

BoltzRank (Volkovs and Zemel, 2009) is a list-wise ranking algorithm, which evaluates how well an ordered list for a query is ordered with respect to the ground truth of the list. Given N examples, there are $N!$ possible ordered lists, which are the inputs for BoltzRank. Since considering all possible lists is time consuming, BoltzRank uses a special sampling method that reduces the lists being considered while keeping the diversity of the lists.

After sampling, BoltzRank use a neural network as an internal point-wise ranking algorithm. Another variant of BoltzRank uses two neural networks: a point-wise one and

Table 10: BoltzRank with Different Feature Sets

N_1	N_2	training ERR	online ERR	time per iteration (min)
use all 700 features				
15		0.4356	0.4325	150
45		0.4362	0.4339	480
use 6 features selected by AdaRank				
6		0.4409	0.4380	1
6	6	0.4429	0.4394	5
use 10 features selected by AdaRank				
10	10	0.4412	0.4393	8
use 12 features selected by AdaRank				
12	12	0.4429	0.4393	10

a pair-wise one. Then, BoltzRank operates with gradient decent with respect to an error function, for which we take the sum of the ERR criteria and a KL-divergence-based regularization term.

Even with the sampling process, BoltzRank can still be quite slow. Thus, we try conducting feature selection using AdaRank (Xu and Li, 2007) before BoltzRank training. As we shall see next, AdaRank not only speeds up the training time of BoltzRank, but also helps get more accurate results.

7.1 Experiments

In our experiments with BoltzRank, one hidden layer with N_1 hidden nodes is used for the point-wise neural network, and N_2 hidden nodes for the pair-wise one. We set the learning rate of gradient descent to 0.1. In general, BoltzRank takes 100 to 300 iterations to converge.

Table 10 shows the performance of BoltzRank with different training features. As shown in the table, training on the feature set selected by AdaRank can achieve better results than training on the whole feature set. In addition, using the pair-wise neural networks can improve the performance over using only the point-wise ones.

8. Final Ensemble

After introducing the six individual methods and their respective results, below we describe how we combine these approaches.

We obtain 20 models from the six approaches in the their best settings using a random 80% of set 1. These 20 models includes 12 for ORSVM, 1 for Poly-ORSVM, 3 for ORBoost, 1 for RankSVM, 1 for RankLR, and 2 for BoltzRank. We then use the outcomes of these 20 models as features to learn an ensemble using the remaining 20% of set 1. After carefully studying the advantages and disadvantages of the six approaches for ensemble learning in task 1, we eventually take RankLR with the 2^{nd} order polynomial expansion as the final choice. The best online results of each approach as well as the final ensemble are listed in

Table 11: Online Performance of All Approaches

	ORSVM	Poly-ORSVM	ORBoost	RankSVM	RankLR	BoltzRank	Ensemble
ERR	0.4527	0.4456	0.4473	0.4428	0.4503	0.4394	0.4565
NDCG	0.7820	0.7643	0.7634	0.7550	0.7721	0.7459	0.7870

Table 11. We see that an ensemble is able to achieve significantly better performance than each individual approach.

9. Summary

We have carefully studied six major approaches for the challenge. Our major findings can be summarized as follows. First, point-wise methods, in particular ORSVM with the nonlinear perceptron kernel, yield the best overall performance as individual learners. They are also computationally more efficient than pair-wise or list-wise ranking approaches. Second, cost-sensitive, weighting, and query-based normalization settings that respect the ERR criteria can fine-tune the ranking performance of the basic approaches to get better online ERR results. Third, ensemble learning by stacking can improve the ranking quality over individual approaches.

Acknowledgments

We thank the organizers for running such an interesting competition. We also thank Ken-Yi Lin and Han-Hsing Tu for fruitful discussions. This work was partially supported by the National Science Council of Taiwan via NSC 98-2221-E-002-192.

References

Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

Yin-Wen Chang, Chuo-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *Journal of Machine Learning Research*, 11:1471–1490, 2010.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

Ralf Herbrich, Thore Graepel, and Klaus Obermayer. Large margin rank boundaries for ordinal regression. In *NIPS '99: Advances in Neural Information Processing Systems*, pages 115–132, 1999.

Ling Li and Hsuan-Tien Lin. Ordinal regression by extended binary classification. In *NIPS '07: Advances in Neural Information Processing Systems*, pages 865–872, 2007.

Hsuan-Tien Lin. *From Ordinal Ranking to Binary Classification*. PhD thesis, California Institute of Technology, 2008.

Hsuan-Tien Lin and Ling Li. Support vector machinery for infinite ensemble learning. *Journal of Machine Learning Research*, 9:285–312, 2008.

Hsuan-Tien Lin and Ling Li. Large-margin thresholded ensembles for ordinal regression: Theory and practice. In *ALT '06: Proceedings of Algorithmic Learning Theory*, pages 319–333, 2006.

D. Sculley. Large scale learning to rank. In *NIPS '09 Workshop on Advances in Ranking*, 2009.

Maksims Volkovs and Richard Zemel. Boltzrank: learning to maximize expected ranking gain. In *ICML '09: Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1089–1096, 2009.

Jun Xu and Hang Li. AdaRank: a boosting algorithm for information retrieval. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 391–398, 2007.