Ranking by calibrated AdaBoost

Róbert Busa-Fekete BUSAROBI@GMAIL.COM
Linear Accelerator Laboratory (LAL), University of Paris-Sud, CNRS
Orsay, 91898, France
Research Group on Artificial Intelligence of the
Hungarian Academy of Sciences and University of Szeged
Aradi vértanúk tere 1., H-6720 Szeged, Hungary

Balázs Kégl BALAZS.KEGL@GMAIL.COM
LAL/LRI, University of Paris-Sud, CNRS
Orsay, 91898, France

Tamás Éltető ELTETO@LRI.FR
Computer Science Laboratory (LRI),
University of Paris-Sud, CNRS and INRIA-Saclay, 91405 Orsay, France

György Szarvas SZARVAS@TK.INFORMATIK.TU-DARMSTADT.DE
Ubiquitous Knowledge Processing (UKP) Lab,
Computer Science Department Technische Universität Darmstadt, D-64289 Darmstadt, Germany

Editor: Olivier Chapelle, Yi Chang, Tie-Yan Liu

Abstract

This paper describes the ideas and methodologies that we used in the Yahoo learning-to-rank challenge[1]. Our technique is essentially pointwise with a listwise touch at the last combination step. The main ingredients of our approach are 1) preprocessing (querywise normalization) 2) multi-class AdaBoost.MH 3) regression calibration, and 4) an exponentially weighted forecaster for model combination. In post-challenge analysis we found that preprocessing and training AdaBoost with a wide variety of hyperparameters improved individual models significantly, the final listwise ensemble step was crucial, whereas calibration helped only in creating diversity.

Keywords: AdaBoost, AdaBoost.MH, ranking, regression calibration, exponentially weighted forecaster

1. Introduction

Ranking systems are traditionally classified into three distinct categories. In the simplest *pointwise* approach, the instances are first assigned a relevance score using classical regression or classification techniques, and then ranked by posterior scores obtained using the trained model. In the *pairwise* approach, the order of pairs of instances is treated as a binary label and learned by a classification method (Freund et al., 2003). Finally, in the most complex *listwise* approach, the fully ranked lists are learned by a tailor-made learning method which aims to optimize a ranking-specific evaluation metric (such as Normalized Discounted Cumulative Gain (NDCG) or the Expected Reciprocal Rank (ERR)) during the

1. Our team name was LAL.

learning process (Valizadegan et al., 2009). Although it seems to be the general consensus in the recent learning-to-rank literature (Cao et al., 2007; Valizadegan et al., 2009) that pairwise and listwise techniques outperform the pointwise approach, the results of the contest (Table 1) paint a different picture. The winners of the two tracks did use pairwise and listwise techniques, but all the other contestants (who disseminated their methods), including us, followed the pointwise approach and achieved competitive results. We did try some pairwise-flavored ideas in some of the steps (e.g., in the calibration step) with no success. On the other hand, a computationally cheap listwise model-combination post-processing did improve our results significantly. So, it seems to us that the jury is out. One possible narrative is that if you have enough computational power and you master your listwise or pairwise techniques, they may be the best choice, but simple, robust, and computationally cheaper pointwise approaches work practically as well, especially on large data.

Rank	**Track 1**		**Track 2**	
1	Ca3Si2O7	0	MN-U	0
2	catonakeyboardinspace	748	arizona	307
3	MLG	1651	Joker	363
4	Joker	1828	ULG-PG	1790
5	AG	2448	VeryGoodSignal	1844
6	**LAL**	2856	ya	1983
7	HotStepper	3946	WashU	2292
8	WashU	4307	catonakeyboardinspace	2329
9	ya	4346	CLTeam	2578
10	ULG-PG	4373	yareg	2956
11	yareg	5562	**LAL**	2991
12	MN-U	5857	lily	3510

Table 1: For the score we calculated the ERR difference from the winning team's score in each track and multiplied it by 10^6. Teams with red, green, and blue backgrounds used pointwise, pairwise, and listwise approaches, respectively. We have no information on uncolored teams. The name of our team was LAL.

We also shared another important feature of using an *ensemble method* with most of the other contestants. In fact, we followed a *double* ensemble approach in the sense that our core technique was based on AdaBoost (Freund and Schapire, 1997), but at the end we used another ensemble technique to combine models trained with different hyper-parameters. In Section 2 we present our four steps of preprocessing, training, calibration, and model selection/combination. Since most of the contestants used similar techniques, here we highlight some of the elements of our approach that seem to be unique.

- Although AdaBoost has a pairwise version (Freund et al., 2003), we used it in a classical pointwise manner. Moreover, even though the five-valued relevance score would have suggested to use a regression technique, we trained AdaBoost.MH (Schapire

and Singer, 1999) in a *multi-class classification* setup, and obtained a real valued score by *regression-calibration*.

- We *did not validate* any of the hyperparameters. We trained a large number of models and used a simple exponential weighting technique to combine them into a final meta-ensemble.
- We used *decision products* (Kégl and Busa-Fekete, 2009) together with the more traditional decision trees.
- We did *querywise* standardization and whitening to normalize the raw features (similarly to the catonakeyboardinspace team).

Among the plethora of ranking algorithms, our approach is the closest to the McRank algorithm (Li et al., 2007). We both use a multi-class learning algorithm at the core (they use gradient boosting whereas we apply AdaBoost.MH). The novelty in our approach is that we use product base classifiers beside tree base classifiers and apply several different calibration approaches. Both elements add more diversity to our models that we exploit by a final meta-ensemble technique.

The outline of the paper is as follows. In Section 2 we describe the main ingredients of our approach. In Section 3 we present post-contest analysis. To gain more insight on what worked and what did not, we made an attempt to uncouple some of the elements of the algorithm and analyzed them individually in terms of their effect on the final performance.

2. Calibrated multi-class AdaBoost.MH with exponentially weighted model combination

2.1. Preprocessing

Feature transformations. The contest organizers used their in-house feature extractor to provide real-valued features for the query-document pairs. We augmented this feature set in two different ways. First, we standardized each feature to have mean zero and standard deviation one, and second, we carried out principal component analysis and whitening to get rid of possible correlations between the features. It is important to note that both transformations were done *querywise*.[2] The intuition behind querywise normalization is that features used in a learning-to-rank task often represent some kind of count[3]. Some of these counts are not comparable in an absolute way: for example, the number of times a query term occurs in a document is not comparable between common query terms (e.g., "dog") and rare query terms (e.g., "Adaboost"). Our bandit-based boosting technique (Busa-Fekete and Kégl, 2010) is able to handle a large set of possibly noisy features, so there was no computational reason not to use all features. Eliminating the original features would have also had a detrimental effect since probably not all the features were count-based. At the same time, we were curious about which normalization would be the best in terms

2. In fact, global standardization would make no sense since we used tree-like predictors, insensitive to any monotonic transformation of the features.
3. For instance, the number of occurrences of the term in the abstract of the document; see the feature list of the LETOR project at `http://research.microsoft.com/en-us/projects/mslr/feature.aspx`

of performance, so we created three distinct data sets: 1) only the original features, 2) the original features augmented with standardized features, and 3) the original features augmented with whitened features.[4]

Label groupings. Each query-document pair was assigned an integer relevance label between 0 and 4. The most natural pointwise approach is to predict this label in a *regression setup*, however, we decided to cast the problem as *multi-class classification* – mainly since our boosting implementation is designed for this task – and applied regression calibration as a post-processing step (Section 2.3).

Annotating relevance is an arduous and sometimes subjective task, so the relevance labels provided by humans can have considerable label noise. Deciding whether a document is relevant or not is a relatively easy task, so the label noise mainly affects the integer-valued *degree* of relevance. Inconsistency in the labeling usually materializes as a random upward or downward shift in the relevance levels. As a result, the borders between adjacent relevance levels might be blurred. In practice, this kind of noise can be estimated and resolved using inter-annotator agreement tests. Since we had no access to multiple annotations or to the actual annotation protocol, we decided to use multiple label groupings (Table 2) in the multi-class classification setup. To control random shift noise, we decided to group neighboring relevance labels in various ways, hoping that a more accurate model can be learned in this way. An additional benefit of this setup was that it increased the diversity of the models which we could later exploit by our model combination approach (Section 2.4).

New labels:	1	2	3	4	5
Original labels:	$\{0\}$	$\{1\}$	$\{2\}$	$\{3\}$	$\{4\}$
Binary labels:	$\{0\}$	$\{1,2,3,4\}$			
Three classes (1):	$\{0\}$	$\{1,2\}$	$\{3,4\}$		
Three classes (2):	$\{0\}$	$\{1,2,3\}$	$\{4\}$		
Four classes:	$\{0\}$	$\{1,2\}$	$\{3\}$	$\{4\}$	

Table 2: The five different label groupings we used.

The classical way to select the best training set and the best label grouping would be to handle these selection variables as hyperparameters, and validate them on hold-out validation sets. There is increasing evidence[5] that the paradigm of selecting one set of hyperparameters is suboptimal to keeping all models and combining them in an ensemble setup, so we trained models using all fifteen combinations of training sets and label groupings, and use an ensemble technique (Section 2.4) to find the best way to weight these models to obtain a final ranking.

2.2. Training AdaBoost.MH

Hyperparameters. Our basic modeling tool was multi-class AdaBoost.MH of Schapire and Singer (1999). We used our open source implementation available at `multiboost.org`.[6]

4. If the eigenvalue calculation failed, we used regularization with a value of 0.001.
5. See for example this volume or entries in last year's KDD-Cup (Dror et al., 2009).
6. For the details of the implementation and for the undefined notations see (Kégl and Busa-Fekete, 2009) and (Busa-Fekete and Kégl, 2010).

We used *decision trees* and *decision products* (Kégl and Busa-Fekete, 2009) as base learners. The number of tree leaves, the number of product terms, and the number of boosting iterations ranged from 8 to 25, from 2 to 6, and from 150000 to 10^6, respectively. Instead of choosing the best hyperparameters, we only did a rough pre-validation to have an idea about the range of the optimal hyperparameters, then we let the meta-ensemble technique (Section 2.4) to find the best way to weight the different models.

Initial weights. Motivated by a one-against-all scheme for multi-class classification, the standard way to weight the ℓth label of the ith instance in AdaBoost.MH is

$$w_{i,\ell}^{(1)} = \begin{cases} 1/(2n) & \text{if } \ell_i = \ell, \\ 1/(2n(K-1)) & \text{otherwise,} \end{cases}$$

where ℓ_i is the correct (grouped) label of the ith query-document pair, n is the number of training instances, and K is the number of classes (grouped labels).[7] Instead of this setup, we further up-weighted relevant instances exponentially proportionally to their relevance, so, for example, an instance $\mathbf{x}_i$ with relevance $\ell_i = 4$ was twice as important in the global training cost than an instance with relevance $\ell_i = 3$, and four times as important than an instance with relevance $\ell_i = 2$. Formally, the initial (unnormalized) weight of ℓth label of the ith instance is

$$w_{i,\ell}^{(1)} = \begin{cases} 2^{\ell_i} & \text{if } \ell_i = \ell, \\ 2^{\ell_i}/(K-1) & \text{otherwise.} \end{cases}$$

The weights are then normalized to sum to 1. This weighting scheme was motivated by the evaluation metric: the weight of an instance in the ERR score is exponentially proportional to the relevance label of the instance itself.

Bandit boosting. Since the training datasets were prohibitively large to enable full feature search in each boosting iteration, we used bandit boosting (Busa-Fekete and Kégl, 2010), an accelerated version of AdaBoost.MH.

2.3. Calibration of the output of AdaBoost.MH.

AdaBoost.MH outputs a strong classifier $\mathbf{f}^{(T)}(\mathbf{x}) = \sum_{t=1}^{T} \alpha^{(t)} \mathbf{h}^{(t)}(\mathbf{x})$, where $\mathbf{h}^{(t)}(\mathbf{x})$ is a $\{-1,+1\}^K$-valued base classifier (K is number of classes), and $\alpha^{(t)}$ is its weight. In classical multi-class classification, the elements of $\mathbf{f}^{(T)}(\mathbf{x}) = \left(f_1^{(T)}(\mathbf{x}), \dots, f_K^{(T)}(\mathbf{x})\right)$ are treated as posterior scores corresponding to the labels, and the predicted label is

$$\widehat{\ell}(\mathbf{x}) = \operatorname*{arg\,max}_{\ell=1,\dots,K} f_\ell^{(T)}(\mathbf{x}).$$

In case the labels are ordered, the simplest way to obtain a real-valued prediction is by first normalizing the output vector into $[0,1]^K$ using

$$\mathbf{f}'^{(T)}(\mathbf{x}) = \frac{1}{2}\left[1 + \frac{\mathbf{f}^{(T)}(\mathbf{x})}{\sum_{t=1}^{T} \alpha^{(t)}}\right],$$

7. For a thorough explanation, see (2) in (Kégl and Busa-Fekete, 2009) or Section 7.2 in (Schapire and Singer, 1999).

computing posterior probabilities

$$\widetilde{\mathbf{f}}^{(T)}(\mathbf{x}) = \frac{\mathbf{f}'^{(T)}(\mathbf{x})}{\sum_{\ell=1}^{K} f'^{(T)}_{\ell}(\mathbf{x})},$$

and averaging the labels under the posterior to obtain a relevance estimate

$$\widehat{r}(\mathbf{x}) = \sum_{\ell=1}^{K} \ell \widetilde{f}^{(T)}_{\ell}(\mathbf{x}). \tag{1}$$

It is known that output calibration can improve the performance of AdaBoost, especially when posterior scores need to be interpreted as probabilities (Niculescu-Mizil and Caruana, 2005). Thus, we decided to learn $\widehat{r} : \mathbb{R}^K \to \mathbb{R}$ using the raw K-dimensional output to obtain

$$\widehat{r}(\mathbf{x}) = g\left(\mathbf{f}^{(T)}(\mathbf{x})\right).$$

The output of the calibration function g is the *scoring function* using the ranking terminology (Clémençon and Vayatis, 2009), thus the ranking task is solved by ordering the objects of interest based on $\widehat{r}(\mathbf{x})$.

The calibration was carried out by minimizing the classical squared error on a small validation set taken from the training data. We used five different regression methods: Gaussian process regression, logistic regression, linear regression, neural network regression, and polynomial regression of degree between 2 and 5. As for other model parameters, we let the final meta-ensemble technique (Section 2.4) to find the best way to weight the obtained models in the final ranking.

2.4. Ensemble of ensembles.

The output of the first three steps is a set of relevance predictions $\{\widehat{r}_j(\mathbf{x})\}_{j=1}^{N}$ where N is in the order of thousands. Each relevance prediction can be used as a scoring function to rank the query-document pairs $\mathbf{x}_i$. Until this point it is a pure pointwise approach. To fine-tune the algorithm and to make use of the diversity of our models, we decided to combine them using an exponentially weighted forecaster (Cesa-Bianchi and Lugosi, 2006). The reason of using this particular weighting scheme is twofold. First, it is simple and computationally efficient to tune which is important when we have a large number of models. Second, theoretical guarantees over the cumulative regret of a mixture of experts on individual (model-less) sequences (Cesa-Bianchi and Lugosi, 2006) makes the technique robust against overfitting the validation set.

The weights of the models were tuned on the ERR score of the ranking, giving a slight listwise touch to our approach. Formally, the final scoring function was obtained by

$$\widehat{\widehat{r}}(\mathbf{x}) = \sum_{j:\omega_j > \omega_{\min}} \exp(c\omega_j)\widehat{r}_j(\mathbf{x}), \tag{2}$$

where ω_j is the ERR score of the ranking obtained using $\widehat{r}_j(\mathbf{x})$ as a scoring function. The parameter c controls the dependence of the weights on the ERR values, and the $\omega_{\min}$

parameter determines the number of models participating in the ensemble. Both these parameters were tuned on the official validation set, and set to $\omega_{\min} = 0.4$ and $c = 30$ on Track 1, and $\omega_{\min} = 0.4$ and $c = 10$ on Track 2.

During the contest we also tried out a few commonly used pairwise rank aggregation methods, i.e., Borda's method (Borda, 1781) and Markov Chain Rank Aggregation (MCRA) (Dwork et al., 2001). We calculated the transition matrix in MCRA in many ways. First, we estimated the transitions based on the ratio of Borda counts. Then, we tried to model the transitions using neural networks trained on a dataset where the feature vectors were the pairs of raw output vectors $\mathbf{f}^{(T)}(\mathbf{x})$ of AdaBoost.MH and the labels were the pairwise order of the elements. These techniques worked significantly worse than the combination of pointwise regression calibration and exponential weighting, so we did not use them in the experiments.

3. Experiments

Our technique came in 6th on the larger and so computationally more challenging Track 1, and 11th on Track 2. It is surprising that the Track 1 was more popular (Chapelle and Chang, 2011), since the number of teams was higher and there were almost four times more submissions in Track 1 than in Track 2, and thus the results were closer to each other. Our ERR score was within 0.003 to the winner in both tracks (Table 1), although the characteristics of the two tasks were quite different. This proves the robustness of our approach.

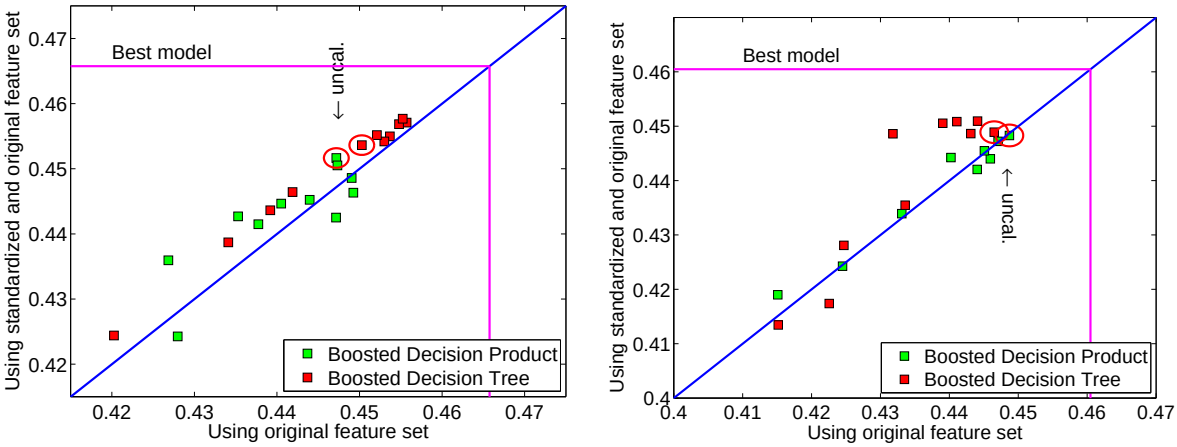
In the rest of this section, we examine what worked and to what extent certain algorithmic choices affected the final results. We use the same analysis methodology in Figures 1–3: each point in the scatterplot is one run of the algorithm with everything unchanged except for the (binary) decision we are examining. By looking at the cloud of points, one can assess qualitatively the effect of the algorithmic decision on the performance of the algorithm.

Figure 1 shows the effects of query-wise standardization. Most of the points lie above the diagonal line indicating that this preprocessing step did improve the individual models. The whitening transformation turned out to be equally effective (results are similar so not shown).

Figures 2 and 3 show the effect of label groupings. The *Three classes (1)* model in Table 2) seems to be definitely better than keeping the original five labels, whereas the binary classification setup did not work. It is also interesting that the different calibration techniques affected the ERR score much more on the five-label sets than on the two- or three-label sets.

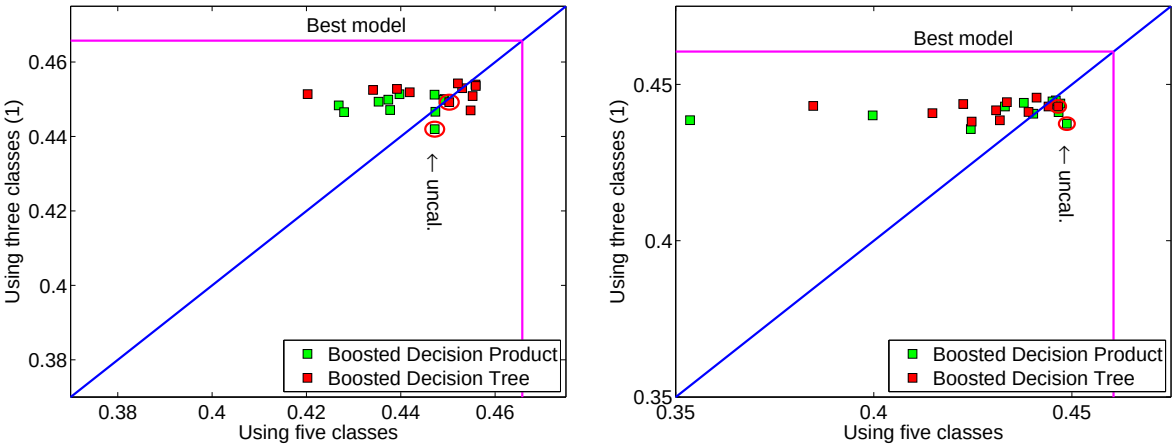
Figures 1–3 all show that individually, models using boosted decision trees are slightly better than models using boosted decision products. On the other hand, when we *combined* only product-based models and only tree-based models, products had a slight edge (see Table 3). Most importantly, the best results came when we combined both types of models, indicating that they capture different aspects of the data.

In Figures 1–3, we encircled the uncalibrated models. In general, calibration did not improve individual models much, especially on Track 2. This is somewhat at odds with the findings of Niculescu-Mizil and Caruana (2005), but note that the error functionals that Niculescu-Mizil and Caruana (2005) used may be more sensitive to miscalibration



(*a*) Track 1 (*b*) Track 2

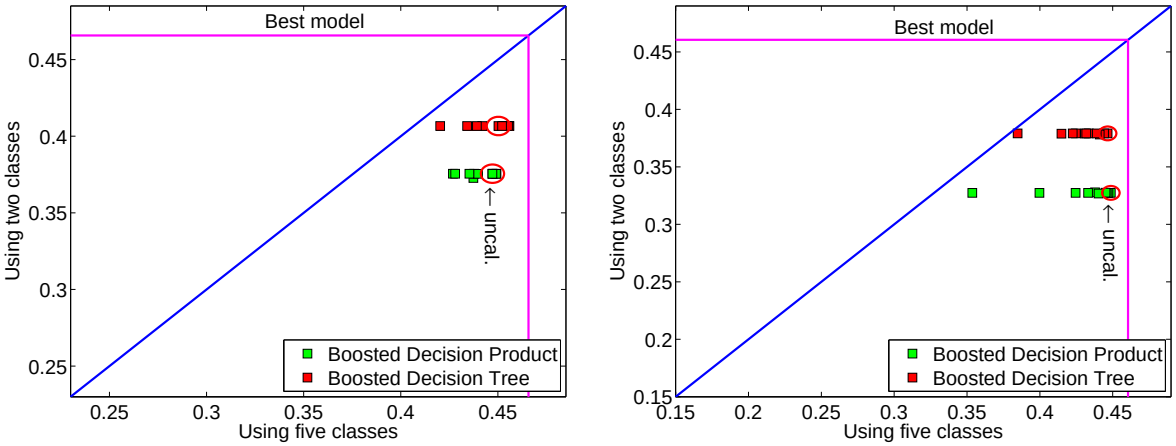
Figure 1: The scatterplot of ERR scores calculated by using only the original feature set (X-axis) versus ERR scores calculated by using the original feature set augmented with standardized features (Y-axis). The rectangles indicate the performance of calibrated models. The horizontal and vertical "Best model" magenta lines denote the performance of our best combined model (official submission). The points corresponding to the uncalibrated scoring function (1) are encircled.



(*a*) Track 1 (*b*) Track 2

Figure 2: Comparison of performances obtained by using label groupings vs. original labels. The notation is the same as in Figure 1

of individual class-posteriors than the estimate of expected relevance (1). Note also that even though the individual models were not much improved by calibration, using different calibration methods did generate diversity, and so implicitly they did improve the final



(*a*) Track 1 (*b*) Track 2

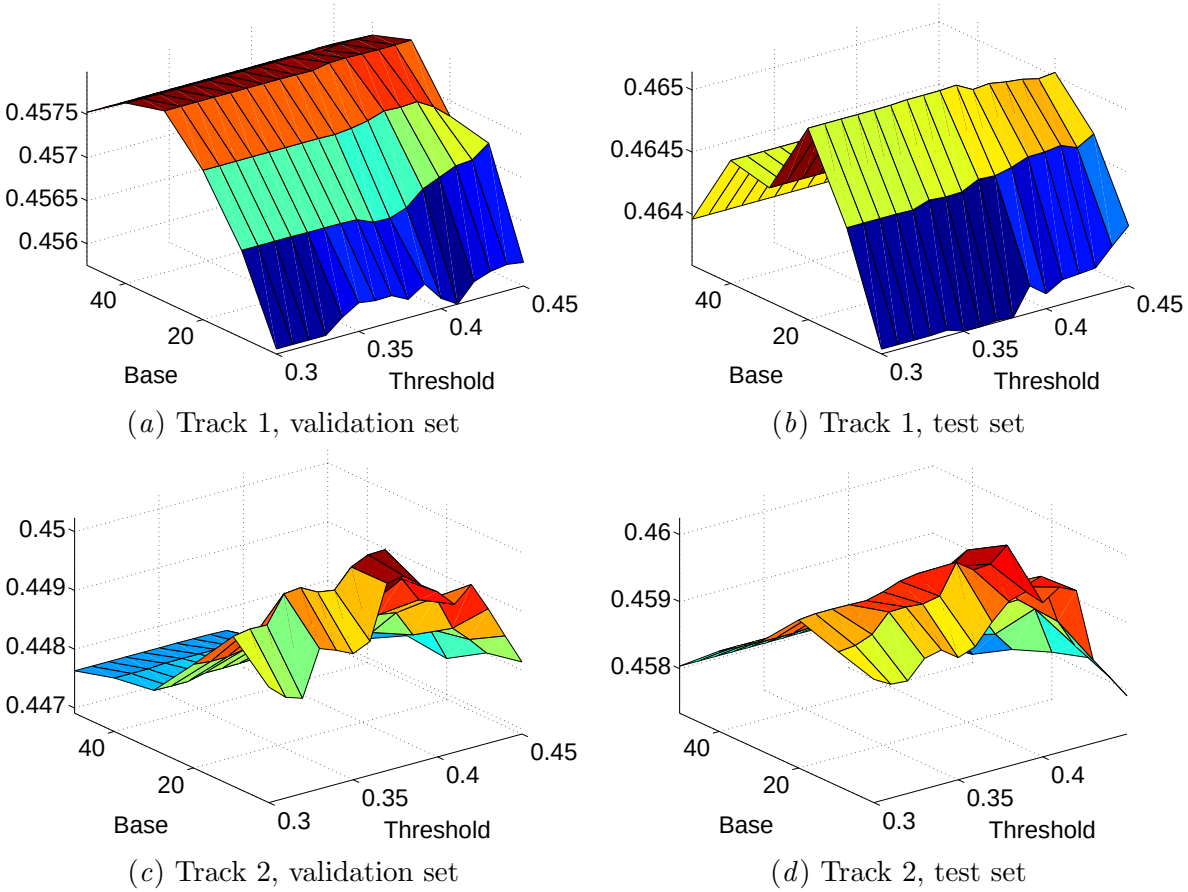
Figure 3: Comparison of performances obtained by using binary labels and original labels. The notation is the same as in Figure 1

Dataset	Uncalibrated		Product		Tree		All	
	Valid	Test	Valid	Test	Valid	Test	Valid	Test
Track 1	0.4482	0.4564	0.4582	0.4651	0.4555	0.4639	0.4580	0.4657
Track 2	0.4398	0.4502	0.4488	0.4599	0.4488	0.4591	0.4501	0.4605

Table 3: The ERR scores of the best combined models using only uncalibrated models, only decision products, only decision trees, and all models. Our best performance is shown on the rightmost column where we combined all boosted models.

model. Using only uncalibrated models in the pool would have generated significantly worse results (Table 3). It is also apparent that there is no effect of the calibration on the binary models (Figure 3). This is in fact what we expected since in the binary case the ordering is strictly determined by the raw output $f_1^{(T)}(\mathbf{x}) = -f_2^{(T)}(\mathbf{x})$, and, unless the calibrating function g is non-monotonic, this order cannot be changed by the calibration.

The two hyperparameters of the last exponential forecaster step, the base parameter c and the threshold parameter $\omega_{\min}$ were validated manually on the official validation set during the contest. In post-challenge experiments we executed the same grid search on the test set. Figure 4 indicates that the optimal parameters almost coincide on the two sets, so we successfully avoided overfitting. The difference between the tracks is also interesting to note. On Track 1, the ERR score was very insensitive to the threshold $\omega_{\min}$, which means that including bad models did not have a detrimental effect. The base c was relatively large, so these relatively bad models were effectively down-weighted. On the other hand, Track 2 preferred a more uniform weighting among good models, so the threshold had a more important role of eliminating bad models from the pool.



(*a*) Track 1, validation set

(*b*) Track 1, test set

(*c*) Track 2, validation set

(*d*) Track 2, test set

Figure 4: The dependence of the final combined ERR score on the base parameter c and on the threshold parameter $\omega_{\min}$.

3.1. Time and memory requirements

The training time for a single model was about two weeks on Track 1 (300000 iterations) and about ten days for Track 2 (10^6 iterations). The running time of one iteration is not affected by the number of features using bandit-based AdaBoost.MH (Busa-Fekete and Kégl, 2010). The memory requirements were about 6 GB per model. We ran our code parallel on 10 CPUs, so we had about 40 full runs of AdaBoost.MH. The calibration and the combination steps required virtually no time.

4. Conclusions

Our main objective in participating in the contest was to test our recently developed fast boosting algorithm (Busa-Fekete and Kégl, 2010) on a large, real-world problem. Having had little prior experience in ranking, and given that our AdaBoost.MH implementation is designed for multi-class classification, we naturally settled in a pointwise approach with regression calibration. Learning from previous challenges (e.g., (Dror et al., 2009)), we de-

cided to use a model combination method rather than optimizing the individual performance in terms of the hyperparameters. The flexible exponential forecaster approach allowed us to use the official ERR score in this final step, giving a listwise flavor to our technique. In post-challenge analysis we found that this listwise ensemble step was a key to our success. The querywise normalization idea and good individual performance of AdaBoost.MH was also important, whereas regression calibration only helped in creating more diversity.

Acknowledgments

This work was supported by the ANR-07-JCJC-0052 and ANR-2010-COSI-002 grants of the French National Research Agency.

References

J. C. Borda. Mémoire sur les élections au scrutin. *Histoire de l'Académie Royale des Sciences (Académie Royale des Sciences, Paris)*, 1781.

R. Busa-Fekete and B. Kégl. Fast boosting using adversarial bandits. In *International Conference on Machine Learning*, volume 27, pages 143–150, 2010.

Z. Cao, T. Qin, T. Liu, M. Tsai, and H. Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24rd International Conference on Machine Learning*, pages 129–136, 2007.

N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, and Games*. Cambridge University Press, New York, NY, USA, 2006. ISBN 0521841089.

O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *JMLR Workshop and Conference Proceedings*, 14:1–24, 2011.

S. Clémençon and N. Vayatis. Tree-based ranking methods. *IEEE Trans. Inf. Theor.*, 55 (9):4316–4336, 2009.

G. Dror, M. Boullé, I. Guyon, V. Lemaire, and D. Vogel, editors. *Proceedings of KDD-Cup 2009 competition*, volume 7 of *JMLR Workshop and Conference Proceedings*, 2009.

C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW'01: Proceedings of the 10th International Conference on World Wide Web*, pages 613–622, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0.

Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

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.

B. Kégl and R. Busa-Fekete. Boosting products of base classifiers. In *International Conference on Machine Learning*, volume 26, pages 497–504, Montreal, Canada, 2009.

P. Li, C. Burges, and Q. Wu. McRank: Learning to rank using multiple classification and gradient boosting. In *Advances in Neural Information Processing Systems*, volume 19, pages 897–904. The MIT Press, 2007.

A. Niculescu-Mizil and R. Caruana. Obtaining calibrated probabilities from boosting. In *Proceedings of the 21st International Conference on Uncertainty in Artificial Intelligence*, pages 413–420, 2005.

R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

H. Valizadegan, R. Jin, R. Zhang, and J. Mao. Learning to rank by optimizing NDCG measure. In *Advances in Neural Information Processing Systems 22*, pages 1883–1891, 2009.