Robust Ranking Models using Noisy Feedback

Christian Pölitz and Ralf Schenkel

Saarland University, Germany

Abstract. Direct feedback of users of search engines by click information is naturally noisy. Ranking models that integrate such feedback in their training process must cope with this noise. In worst case such noise can lead to large variance among the results for different queries in the resulting rankings. We propose to integrate model averaging like bagging and random forest methods to reduce the variance in the ranking models. We perform an experimental study on different noise levels using a state of the art ranking model.

1 Introduction

Modern search engines collect large amounts of user feedback from the interactions with the search interface. Such feedback is being more and more used for the generation and the maintainance of ranking models [6]. There have been different approaches to integrate the feedback directly into a ranking model. Joachims for instance showed in [8] how to extract preference information from query logs. Such preferences represent for a given query that a certain url, and hence the corresponding content, is preferred over another url. Important aspects are the ranking quality and the robustness of the ranking models when trained merely or mostly on user feedback data. W.r.t. these aspects we investigate ranking models that are purely trained on noisy user feedback.

Due to the very nature of click data it highly consists of noise. Such noise comes from many sources, for instance from the unpredictability of human behavior or different usage of search results. It is worth mentioning that we do not treat biases of the interaction with search engine as noise, like that the user almost surely clicks on the first results regardless of the relevance, as described by Joachims in [9]. We consider noise as a deviation from the distribution of the relevance information from the click information from users. Considering the preferences used in our approach, this means some preferences result only from the noise in the feedback. Hence such noise will introduce more variance in the training data for the ranking model and hence introduce variance in the ranking results.

Conceptionally there are at least two ways to cope with noise. First we can try to eliminate or reduce the noise before presented to the ranking model for training. Second, we can directly handle the noise in the ranking model while training. This paper is about the second approach. We investigate machine learning methods that inherently can reduce such noise.

More concrete, we integrate bagging methods [2] and random forest methods [3] in state of the art ranking models. This is potentially helpful since such methods reduce the variance and hence reduce noise by aggregating and averaging over many models. We use GBrank [11] as ranking model, since it directly uses preference information as described above to find a good ranking.

The main contribution of this paper is to study how noise in feedback influences ranking models that use them for training. We especially investigate how noise can degenerate the prediction capability of ranking models over different queries. We analyze how the quality of rankings is influenced by different levels of noise. Finally we show how bagging and random forest methods can reduce the effects of the noise.

In contrast to our approach, Carvalho et al. explain in [4] a method to reduce noise of a linear ranking model. In a two step optimization they propose to use the output of a linear model to train a second linear ranking model that uses a more stable loss function to reduce the influence of noise. Unfortunately this methods works only for linear models, but the most successful ranking models are non-linear [5]. Another drawback is that this approach can not distinguish between noise from the training data and noise that comes from the first ranking model. This model can be badly trained or be inadequate for the task. Concerning this, it is not clear if the results they name are really from data noise reduction or from model refinement.

2 Feedback data

The user interactions with the results of search queries provide valuable information of how satisfied the users are with the results. As investigated by Joachims [8] such information can by derived from the set of interactions with the users. By this, it is possible to properly integrate the feedback in the training of a ranking model.

We can use the user feedback for the training of the ranking model either indirect, direct but delayed or direct. Indirectly integrating the feedback in the training means we simulate the necessary training data and present it to the ranking model. This simulation can be based on heuristics or machine learning using the feedback from the users for training as in [6]. We can also collect a certain amount of user feedback, aggregate and average it as in [1]. Here the feedback is used directly but with a delay. This can be used to reduce noise before presented to the training of the ranking model. Finally, we can directly use the preference information for the training without previous aggregation, smoothing, other inputs from other machine learning models or any heuristic. The only preparation is to derive the preference information as proposed by Joachims [8] from the feedback. This can be easily done on the fly.

We use the following preference information directly derived from the implicit user feedback. Given a query q_j there are features vectors $f_{i,j}$ for each url u_i summarizing statistics of the query, the url and similarity of the query and the url. Possible features include tf-idf scores, BM25 scores or the page rank [10].

These features are usually generated from the search engine and then fed into the ranking model to find an ordering of the urls with the most likely relevant urls at the beginning. Using the approach from Joachims we can use the click data to directly generate preference information in the following format: $(\{f_{i,j}\}, \{f_{i',j}\})$, for a given query q_j and a url i that is preferred over url i'.

3 Ranking Models

We concentrate on ranking models that can directly use the feedback from users, with GBrank [11] as a concrete example. This ranking model uses preference data as explained above.

GBrank uses a gradient boosting method to minimize wrong preference prediction. Gradient boosting [7] is an optimization method to minimize (or maximize) a loss function L by gradient descent. A gradient descent is an iterative method that estimates a sequence of gradients for the loss function to successively approximate a minimum (or maximum). Concretely, we walk through the function space by the following iteration:

$$F_{t+1} = F_t - \gamma_t \cdot \nabla L(F) \tag{1}$$

The gradient $\nabla L(F)$ can be a simple model like a regression, we will use regression trees. To integrate the preference data, GBrank minimizes the number of wrong preference predictions. This means for preference data $(f_{i,j}, f_{i',j})$ we want that $F(f_{i,j}) > F(f_{i',j})$. To achieve this goal, GBrank minimizes the following loss function that accounts for wrong predictions:

$$L(F) = \frac{1}{2} \cdot \sum_i \left(\max\left\{0, F(f_{i',j}) - F(f_{i,j})\right\}\right)^2 \tag{2}$$

We extend this ranking model with bagging and random forest to reduce the variance and hence the noise. Bagging trains multiple models over different bootstrap samples, drawn with replacement, of the training data. For prediction the results of all models are averaged. Random forest trains multiple trees over different samples of the training data and further samples the features to be used for the tree induction. Both models show a reduction in the variance in many experiments. Conceptionally we simply use multiple regression trees for each stage of the gradient boosting. This means the gradients are averaged gradients resulting from multiple trees.

4 Evaluation

For the evaluation we use LETOR a standard learning to rank data set [10]. The data set contains a set of queries and resulting urls with gradual preference information by labels. These labels indicate how relevant certain urls are w.r.t. a query. We use this to simulate preference data. This is done by building pairs of urls (i, i') for each query j s.t. the label for url i is larger than the label for url i'.

The pair of urls is transformed as pair of features $(f_{i,j}, f_{i',j})$ for the training of the ranking model. Further we integrate different levels of noise in the preference information. For this we simulate noise by varying the preference information with a certain probability . Hence, $(f_{i',j}, f_{i,j})$ instead of $(f_{i,j}, f_{i',j})$. We simulate different levels of noise with different probabilities. We use probabilities from 0 to 40 percent. To show the benefit of bagging and random forest we train models with 1 to 50 trees per iteration. Further we use GBrank as ranking model. The standard GBrank model uses only one tree in each iteration to model the gradient. A model with n trees at each iteration uses n different trees over different sample. Hence, in each iteration the gradient is modeled by an average of these trees.

Since we optimize for a minimum of wrongly predicted preferences we use the following quality metric. The quality of a ranking model using an independent test set $S = \{(f_{i,j}, f_{i',j})\}$ is the ratio of the number of pairs in S with correct predicted order, hence $F(f_{i,j}) < F(f_{i',j})$, to the total number of pairs in the test set S. Shortly we denote this metric as PR for prediction rate.

The first experiment investigates only bagging methods in the GBrank model. We want to explore the influence of the number of trees for bagging on the variance in PR over different queries.

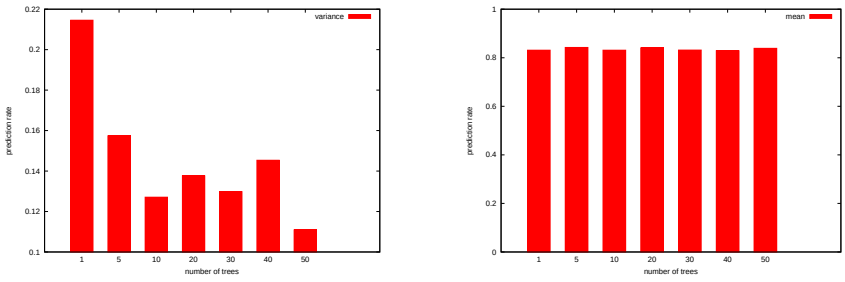

Fig. 1. Variance and mean of correct prediction rate for data set NP2003.

In Figure 1 we display for different numbers of trees the mean PR (right) and the variance (left) on the PR. The plots show the mean results over all tested noise levels over different queries. Obviously the standard GBrank, using one tree per iteration, has the largest variance. The more trees used per iteration, the lower the variances gets, except some small jitter. For this data set the mean PRs show that with more trees at each iteration only the variance reduces, the mean PR itself does not change much. The improvements in the model influence only the variance.

In Table 1 we show the variances (top) and the means (bottom) of the PR for six different data sets from the used data collection. We see that, expect for data set TD2003, the variance decreases with increasing number of trees for bagging. Further the mean in PR increases with more trees. This means more trees reduces the variances in the ranking results over different queries.

	Number of trees per iteration						
Data set	1	5	10	20	30	40	50
	Variances in prediction rate						
TD2003	0.1360	0.1334	0.1477	0.1467	0.1477	0.1348	0.1444
TD2004	0.1017	0.0918	0.0787	0.0870	0.0787	0.0776	0.0773
NP2003	0.2145	0.1573	0.1270	0.1375	0.1298	0.1453	0.1109
NP2004	0.2791	0.2160	0.2474	0.1617	0.2024	0.1877	0.1972
HP2003	0.1174	0.0624	0.0438	0.0329	0.0565	0.0538	0.0400
HP2004	0.1508	0.1214	0.1001	0.1041	0.0809	0.0842	0.0885
	Means of prediction rate						
TD2003	0.7648	0.7949	0.7986	0.7863	0.7986	0.7916	0.8034
TD2004	0.7878	0.7964	0.8096	0.8093	0.8062	0.8090	0.8087
NP2003	0.8311	0.8423	0.8310	0.8405	0.8315	0.8288	0.8388
NP2004	0.7730	0.8363	0.7901	0.7856	0.8124	0.7848	0.8214
HP2003	0.9143	0.9786	0.9858	0.9870	0.9822	0.9838	0.9874
HP2004	0.8459	0.8997	0.8812	0.8807	0.9165	0.8807	0.8795

Table 1. Variances and means for all data sets for different number of trees for bagging.

We further integrated the random forest method by performing samplings over the features used in the trees. We tested different sampling rates from using all features to using 50 percent of the features for GBrank with 20 trees per iteration. Unlike the previous experiment we cannot find improvements for variance or mean using different feature sampling rates. Table 2 summarizes the corresponding results. From these experiments we have only evidence that bagging can reduce variance and increase the mean PR.

5 Conclusion

The performed experiments showed that noisy feedback from users can be inherently coped with within the ranking models. We showed that ensemble methods like bagging can indeed reduce the influence of noise, by reducing variance. This is especially important since usually these methods are only used to increase the mean retrieval quality. This means integrating bagging into a ranking models can make it more robust and stable over different queries. In the future we want to further investigate how good state of the art ranking models can integrate all kind of noisy feedback.

References

1. Eugene Agichtein, Eric Brill, Susan Dumais, and Robert Ragno. Learning user interaction models for predicting web search result preferences. In *SIGIR '06*, pages 3–10, New York, NY, USA, 2006. ACM.
2. Leo Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

	Feature sampling rate					
Data set	100%	90%	80%	70%	60%	50%
	Variances in prediction rate					
TD2003	0.1330	0.1366	0.1449	0.1422	0.1368	0.1258
TD2004	0.0768	0.0816	0.0833	0.0879	0.0787	0.0815
NP2003	0.1124	0.1466	0.1160	0.1571	0.1259	0.1343
NP2004	0.2034	0.2129	0.1851	0.2626	0.1824	0.1917
HP2003	0.0918	0.0697	0.0962	0.0950	0.0675	0.0656
HP2004	0.0936	0.0899	0.1087	0.0693	0.0790	0.0997
	Means of prediction rate					
TD2003	0.8443	0.8442	0.8369	0.8394	0.8429	0.8504
TD2004	0.8020	0.8081	0.8206	0.8150	0.8169	0.8124
NP2003	0.8220	0.8254	0.8259	0.8195	0.8389	0.8259
NP2004	0.8911	0.8929	0.8934	0.8610	0.8980	0.8987
HP2003	0.9293	0.9519	0.9203	0.8992	0.9477	0.9348
HP2004	0.9032	0.8722	0.9167	0.9358	0.9409	0.8987

Table 2. Variances and means for all data sets for different feature sampling rates for random forest.

3. Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
4. Vitor R. Carvalho, Jonathan L. Elsas, William W. Cohen, and Jaime G. Carbonell. Suppressing outliers in pairwise preference ranking. In *CIKM*, pages 1487–1488, 2008.
5. Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. *Journal of Machine Learning Research - Proceedings Track*, 14:1–24, 2011.
6. Olivier Chapelle and Ya Zhang. A dynamic bayesian network click model for web search ranking. In *WWW '09*, pages 1–10, New York, NY, USA, 2009. ACM.
7. Jerome H. Friedman. Greedy Function Approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.
8. Thorsten Joachims. Optimizing search engines using clickthrough data. In *KDD '02*, pages 133–142, New York, NY, USA, 2002. ACM.
9. Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05*, pages 154–161, New York, NY, USA, 2005. ACM.
10. Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. Letor: A benchmark collection for research on learning to rank for information retrieval. *Information Retrieval*, 13:346–374, 2010. 10.1007/s10791-009-9123-y.
11. Zhaohui Zheng, Keke Chen, Gordon Sun, and Hongyuan Zha. A regression framework for learning ranking functions using relative relevance judgments. In *SIGIR '07*, pages 287–294, New York, NY, USA, 2007. ACM.