Feature Selection for Document Ranking using Best First Search and Coordinate Ascent

Van Dang and W. Bruce Croft
Center for Intelligent Information Retrieval
Department of Computer Science
University of Massachusetts
Amherst, MA 01003
{vdang, croft}@cs.umass.edu

ABSTRACT

Feature selection is an important problem in machine learning since it helps reduce the number of features a learner has to examine and reduce errors from irrelevant features. Even though feature selection is well studied in the area of classification, this is not the case for ranking algorithms. In this paper, we propose a feature selection technique for ranking based on the wrapper approach used in classification. Our method uses the best first search strategy incrementally to partition the feature set into subsets. Features in each subset are then combined into a single feature using coordinate ascent in such a way that it maximizes any defined retrieval measure on a training set. Our experiments with many state-of-the-art ranking algorithms, namely RankNet, RankBoost, AdaRank and Coordinate Ascent, have shown that the proposed method can reduce the original set of features to a much more compact set while at least retaining the ranking effectiveness regardless of the ranking method in use.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Selection process

General Terms

Algorithms, Measurement, Performance, Experimentation.

Keywords

Learning to rank, feature selection

1. INTRODUCTION

An effective ranking framework is certainly the core component of any information retrieval (IR) system. Many ranking models have been proposed, the most popular of which are *BM25* [13] and the language modeling framework [6]. These models make use of a small number of features such as term frequency, inverse document frequency and document length. They have the advantage of being fast and produce reasonably good resutls. When more features become available, however, incorporating them into these models is usually difficult since it requires a significant change in the underlying model. For example, the BM25 model was modified to include PageRank as a prior [5] or to incorporate term proximity information [3].

Supervised learning to rank algorithms [10, 8, 1, 14, 7] can help overcome that limitation. They treat query-document pairs as objects to rank, each of which is represented using any set of features. Existing work has shown, by incorporating many features, they produce better results than classical models mentioned above [14, 7].

In the area of machine learning, feature selection is the task of selecting a subset of features to be considered by the learner. This is important since learning with too many features is wasteful and even worse, learning from the wrong features will make the resulting learner less effective. In the classification problem, feature selection approaches can be divided into three categories: the *filter* approach which selects features based on some criterion that is independent of the metric being optimized, the *wrapper* approach which picks features that, with the learning technique in use, produces the best result with respect to the metric being considered, and the *embedding* approach which embeds the feature selection procedure into the learning process.

Even though feature selection for classification is well studied, there has been less research on this topic for ranking. The most recent technique that we are aware of is the work by Geng et al. [9] which follows the *filter* approach. Instead, we prefer the *wrapper* method since the selection of features is based on the effective metric that will be optimized by the learning procedure. Our approach uses *best first search* to come up with subsets of features and uses *coordinate ascent* to learn the weights for those features. Once a best subset is obtained, a new feature is defined based on the learned combination of features in this set and these features are removed from the feature pool. The process is repeated until all features are considered. The set of new features will be used to train the ranker instead of the original features. Our experiments with four well-known ranking algorithms – RankNet [1], RankBoost [8], AdaRank [14] and Coordinate Ascent [7] – show that the set of new features, while being much more compact, is at least as effective as the original set in terms of *NDCG@5*.

2. PROPOSED METHOD

Our proposed method is a simple modification of the standard *wrapper* approach that can be found in [11].

WOODSTOCK '97 El Paso, Texas USA
ACM X-XXXXX-XX-X/XX/XX.

Table 1: A Best-First-Search procedure. $R = 5$ is used in our experiments.

$P = \emptyset$
$best = null$
Randomly pick a node v
Train $\mathcal{A}$ on $\mathcal{T}$ using v to maximize $\Lambda(\mathcal{T}, \mathcal{A}_v)$
Add v to P
while $|P| > 0$
  $v \leftarrow \arg\max_{u \in P} \Lambda(\mathcal{T}, \mathcal{A}_u)$
  Remove v from P
  if $\Lambda(\mathcal{T}, \mathcal{A}_v) > \Lambda(\mathcal{T}, \mathcal{A}_{best})$ **then** $best \leftarrow v$
  if *best* did not change in the last R rounds
    then STOP and RETURN *best*
  for each of v's neighbors u
    Train $\mathcal{A}$ on $\mathcal{T}$ using u to maximize $\Lambda(\mathcal{T}, \mathcal{A}_u)$
    add u to P
  end for
end while
RETURN *best*

2.1 Notation

Let $\mathcal{F} = \{f_1, f_2, ..., f_n\}$ be the set of original features. Let $\mathcal{T} = \{r_1, r_2, ..., r_m\}$ be the set of training samples where r_i is the list of documents for the query q_i. Let d_i^j be the j-th document in the list r_i. Each d_i^j is represented as a feature vector $\{f_1^{(i)(j)}, f_2^{(i)(j)}, ..., f_n^{(i)(j)}\}$

Let $\mathcal{A}_\mathcal{F}$ be any ranking algorithm that utilizes the set of features $\mathcal{F}$. To rank a list of documents r_i using $\mathcal{A}_\mathcal{F}$, we reorder all d_i^j based on $\mathcal{A}_\mathcal{F}(d_i^j)$, which is the score the ranker assigns to this document.

Let Λ be the metric that the learner tries to optimize. It can be any effectiveness metric such as *NDCG* or *MAP*. We define $\Lambda(\mathcal{T}, \mathcal{A}_\mathcal{F})$ to be the metric score that the ranking algorithm $\mathcal{A}$ using the set of features $\mathcal{F}$ achieves on the dataset $\mathcal{T}$. Note that the goal of the training process is to learn $\mathcal{A}$ such that it maximizes $\Lambda(\mathcal{T}, \mathcal{A}_\mathcal{F})$.

2.2 Method

The goal of our technique is to partition $\mathcal{F}$ in a greedy way into k non-overlapping subsets $\{\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_k\}$ – each of which has size of at most s – together with a set of learned rankers $\{\mathcal{A}_{\mathcal{F}_1}, \mathcal{A}_{\mathcal{F}_2}, ..., \mathcal{A}_{\mathcal{F}_k}\}$ where each $\mathcal{A}_{\mathcal{F}_t}$ is trained to maximize $\Lambda(\mathcal{T}, \mathcal{A}_{\mathcal{F}_t})$ using the set of features $\mathcal{F}_t$.

We do that in a slightly different way than the method described in [11]. We first put all the features into a pool. We build an undirected graph where each node represents a subset of features in that pool that has size at most s. An edge is created for any pairs of nodes where one of them can be obtained from the other by adding *exactly one* feature. Then we apply the best first search procedure described in Table 1 to come up with $\mathcal{F}_1$ and $\mathcal{A}_{\mathcal{F}_1}$ ($R = 5$ is used in all of our experiments). All features in $\mathcal{F}_1$ are then removed from the pool. We rebuild the graph from the remaining features and repeat the same procedure until the pool is empty.

Once the feature selection procedure is done, we define a new feature f'_t corresponding to each $\mathcal{F}_t$ such that $f'^{(i)(j)}_t = \mathcal{A}_{\mathcal{F}_t}(d_i^j)$. Therefore, each original feature vector $\{f_1^{(i)(j)}, f_2^{(i)(j)}, ..., f_n^{(i)(j)}\}$ becomes $\{f'^{(i)(j)}_1, f'^{(i)(j)}_2, ..., f'^{(i)(j)}_k\}$ where $k < n$. The learning process then proceeds with this new feature vector.

3. RANKING METHODS

To evaluate our feature selection technique, we test it with four popular ranking algorithms: RankNet [1], RankBoost [8], AdaRank [14] and Coordinate Ascent [7].

3.1 RankNet

RankNet [1] is a probabilistic pair-wise ranking framework based on neural networks. For every pair of correctly ranked documents, each document is propagated through the net separately. The difference between the two outputs are mapped to a probability by the logistic function. The cross entropy loss is then computed from that probability and the true label for that pair. Next, all weights in the network are updated using the error back propagation and the gradient descent method.

3.2 RankBoost

RankBoost [8] is a pair-wise boosting technique for ranking. Training proceeds in rounds. It starts with all document pairs being assigned with an equal weight. At each round, the learner selects the weak ranker that achieves the smallest pair-wise loss on the training data with respect to the current weight distribution. Pairs that are correctly ranked have their weight decreased and those that are incorrectly ranked have their weight increased so that the learner will focus more on the hard samples in the next round. The final model is essentially a linear combination of weak rankers. Weak rankers theoretically can be of any type but they are most commonly chosen as a binary function with a single feature and a threshold. This function assigns a score of 1 to a document of which feature value exceeds the threshold and 0 otherwise.

3.3 AdaRank

The idea of AdaRank [14] is similar to that of RankBoost except that it is a list-wise approach. Hence, it directly maximizes any desired IR metric such as *NDCG* and *MAP* whereas RankBoost's objective is to minimize the pair-wise loss.

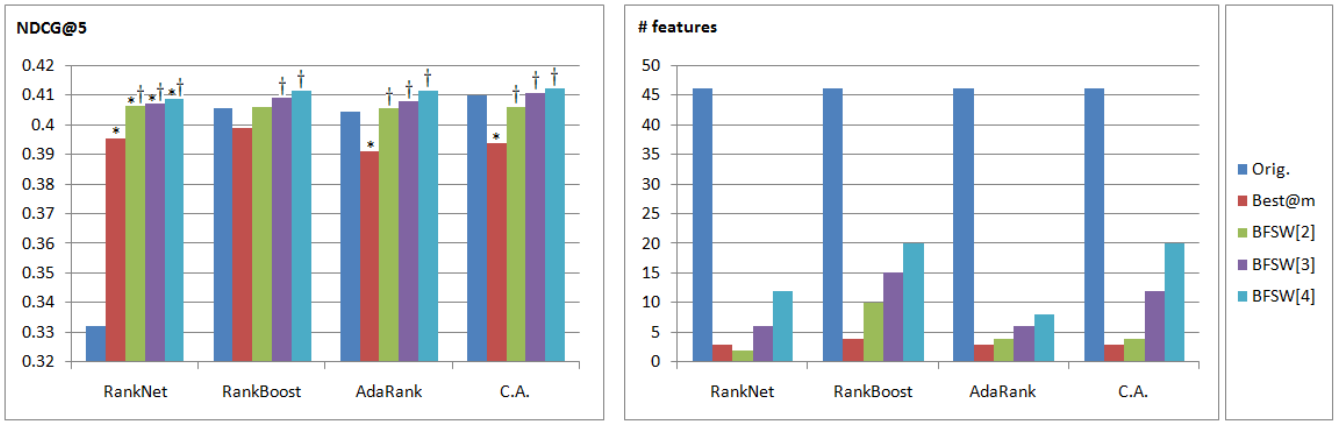
3.4 Coordinate Ascent

Metzler and Croft [7] have proposed a list-wise linear model for information retrieval which uses coordinate ascent to optimize the model's parameters. Coordinate ascent is a well-known technique for unconstrained optimization. It optimizes multivariate objective functions by sequentially doing optimization in one dimension at a time. It cycles through each parameter and optimizes over it while fixing all the others. Note that in the context of this paper, we use the term *Coordinate Ascent* to refer to this particular ranking technique other than the general optimization method.

4. EXPERMENTS

4.1 Datasets

We conduct our experiments on the LETOR 4.0 dataset. It was created from the Gov-2 document collection which contains roughly 25 million web pages. Two query sets from Million Query TREC 2007 and 2008 are included in this dataset, referred to as *MQ2007* and *MQ2008* respectively. *MQ2007* contains roughly 1700 queries and *MQ2008* has about 800. Each query-document pair is represented by a 46-feature vector.



(a) NDCG@5 (b) # Features used

Figure 1: Results on the *MQ2007* dataset: (a) Performance of rankers using different set of features and (b) the number of features they use. The first and the second bar in each group correspond to the system using the set of all original features and the set produced by `best@m` respectively. The last three bars are for systems using features generated by our approach with s equals 2, 3, and 4 respectively. * and † indicate significant difference to `orig.` and `best@m` respectively.

4.2 Experimental Design

For our feature selection procedure, we experiment with different values for s (the *size* of the feature subset $\mathcal{F}_i$) varying from 2 to 4. For each value of s, we consider using top-k subsets $\{\mathcal{F}_1, \mathcal{F}_2, ..., \mathcal{F}_k\}$ where $k \in \{1..5\}$ to train the ranker and use a validation set to select k.

Holding the ranker fixed to a particular technique, we compare systems with different s values to the baseline which uses all original features. We will refer to the baseline as `orig.` and systems using features generated by our method as `BFSW`.

We also consider another baseline in which we use each feature from the original set to rank all the list of documents and sort them based on the *NDCG@5* they produce. We then use the top-m of these features to train the ranker. m is also chosen in the range $\{1..5\}$ based on a validation set. The idea is to see whether simply considering only some of the best features from the original set gives better results than using the whole set, and whether our method can do any better than that. This second baseline will be refered to as `best@m`.

All experiments are done using five-fold cross validation. The dataset in each fold is divided into three subsets: the training set, the validation set and the test set. We train all the systems on the training set and select the model that has the best performance on the validation set as the final model which is evaluated on the test set. *NDCG@5* averaged over five folds is used as the performance measure for each system.

4.3 Parameter Settings

We implement our proposed technique as described in section 2.2. Though the ranking algorithm $\mathcal{A}$ used by this procedure can be freely chosen (e.g. to be the same as whatever used in learning process), we fix it to *Coordinate Ascent* for simplicity.

For both *RankNet* and *RankBoost*, we train the ranker for 300 rounds since we observed no performance change after that in our preliminary experiments. In our implementation of *RankNet*, we use the logistic function as its activation function. We set the learning rate to be 0.001 which is halved everytime the cross entropy loss on training data increases as suggested in [1]. For *AdaRank*, we train the ranker until observing a drop in *NDCG@5* between two consecutive rounds that is smaller than a *tolerance* of 0.002. Since *AdaRank* might have the problem of selecting the same feature again and again, we also apply the trick provided in [4]. For *Coordinate Ascent*, we train the ranker with 5 random restarts to avoid local extrema.

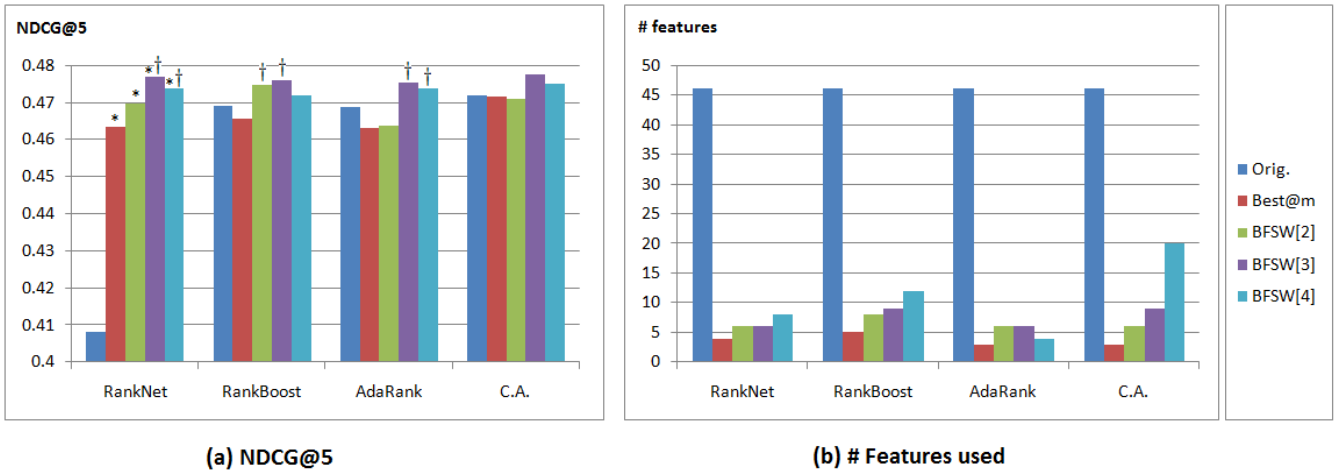
4.4 Results

Fig. 1a demonstrates the results on *MQ2007*. Each group of bars corresponds to one ranking algorithm. Within each group, the first and the second bar show the results obtained using the original set of features and the set of features produced by `best@m` respectively. The last three bars indicates the performance of systems using features generated by our method with s set to 2, 3 and 4 respectively.

It is worth noting from Fig. 1a that with original features, *RankNet* is less effective than other learning to rank algorithms. *RankNet* is based on neural networks which are known to be hard to train. Many important tricks are pointed out in [12]. For this paper, however, we implemented it in a rather straight-forward way. This might be the reason for the bad performance.

The second thing to note from Fig. 1a is that `best@m` is almost always worse than using the original features except for the case of *RankNet*. This suggests that all features are important to some extent and simply using the top-performing features will not help.

Features provided by our selection method with $s \geq 3$, on the other hand, are always more effective than the set of original features and $s = 4$ gives the best *NDCG@5* across learning techniques. Meanwhile, in terms of the number of features the learner needs to consider, Fig. 1b shows that the set generated by our method is much more compact



(a) NDCG@5 (b) # Features used

Figure 2: Results on the *MQ2008* dataset: (a) Performance of rankers using different set of features and (b) the number of features they use. The first and second bar in each group correspond to the system using the set of all original features and the set produced by `Best@m` respectively. The last three bars are for systems using features generated by our approach with s equals 2, 3, and 4 respectively. * and † indicate significant difference to `orig.` and `best@m` respectively.

compared to the original set.

We did hypothesis testing using the two-tailed t-test and note that at $p < 0.05$, the improvement our method provide over `orig.` is not significant except in the case of *RankNet*. The performance drop that `best@m` introduces, on the other hand, is significant. Thus, the difference between our method and `best@m` is almost always significant as indicated in Fig. 1a, suggesting that our method is much better than only considering the top-performing features.

One of the goals of any feature selection method is to get the learner to concentrate on important features and neglect those that are not. Our method works by optimizing a small group of features locally then collapsing them into a single features. The learner then only needs to consider a smaller group – compared to the original set – of better features. This is our explanation for the consistency of our results across learning techniques. The results with *RankNet*, in fact, further supports our claim. While its performance using the orginal features is quite bad since we do not apply any tricks, it becomes comparable to other ranking techniques when using feature selection. This indicates that the learning task is easier with fewer features.

Fig. 2 shows results obtained on the *MQ2008* dataset. These reveal similar trends except that the system with $s = 3$ performs better than $s = 4$.

5. CONCLUSIONS

In this paper, we propose a simple wrapper-based method that uses best first search and coordinate ascent to greedily partition a set of features into subsets. Each subset is then replaced by a single new feature. Our results on the LETOR 4.0 dataset have shown that learning from the set of new features, which is much smaller in size, can produce comparable or even better *NDCG@5* performance than learning from the original features.

Future work includes looking into more recent work in feature selection for classification, experimenting with other ranking algorithms such as SVMRank [10] and LambdaRank [2] and comparing our method to other feature selection techniques such as the one proposed in [9].

6. ACKNOWLEDGMENT

This work was supported in part by the Center for Intelligent Information Retrieval and in part by NSF grant #IIS-0711348. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsor.

7. REFERENCES

[1] C.J.C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton and G. Hullender. Learning to rank using gradient descent. In *Proc. of ICML*, pages 89-96, 2005.

[2] C.J.C. Burges, R. Ragno, and Q.V. Le. Learning to rank with nonsmooth cost functions. In *Proc. of NIPS*, 2006.

[3] S. Buttcher, C. L. A. Clarke, and B. Lushman. Term proximity scoring for ad-hoc retrieval on very large text collections. In *Proc. of SIGIR*, pages 621-622, 2006.

[4] M. Cartright, J. Seo and M. Lease. UMass Amherst and UT Austin @ The TREC 2009 Relevance Feedback Track. In *Proc. of TREC*, 2009.

[5] N. Craswell, S. Robertson, H. Zaragoza, and M. Taylor. Relevance weighting for query independent evidence. In *Proc. of SIGIR*, pages 416-423, 2005.

[6] W.B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley, 2009.

[7] D. Metzler and W.B. Croft. Linear feature-based models for information retrieval. *Information Retrieval*, 10(3): 257-274, 2000.

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

[9] X. Geng, T.Y. Liu, T. Qin and H. Li. Feature selection for ranking. In *Proc. of SIGIR*, pages 407-414, 2007.

[10] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of KDD*, pages 133-142, 2002.

[11] R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 1997.

[12] Y. LeCun, L. Bottou, G.B. Orr and K.R. Muller. Efficient Backprop. *Neural Networks: Tricks of the trade*, 1998.

[13] S. E. Robertson and D. A. Hull. The TREC-9 filtering track final report. In *TREC*, pages 25-40, 2000.

[14] J. Xu and H. Li. AdaRank: a boosting algorithm for information retrieval. In *Proc. of SIGIR*, pages 391-398, 2007.