Report: Ranking Under Temporal Constraints

Andreas Frische, Amalia Radoiu

Saarland University

Abstract. Learning To Rank techniques have been tailored to produce the most effective Ranking Functions, albeit at the price of neglecting efficiency. This report presents the work by Wang et al. [6] , that proposed a method for building query-dependent linear ranking functions, such that time constraints are met.

1 INTRODUCTION

The quality of search results depends to a great part on the order they are presented in; e.g. few people will look past the first page of a search engine's result. *Ranking* induces a weak order on the results on a query thus allowing sorting the results. We also know that users have different expectancies concerning the time it takes an action in a website or application to return; for instance they want search results to complete instantly but may be more tolerant when checking out an online store. So obviously the time a query needs is important; yet the computational cost of Ranking will depend on the query itself.

Conventionally either a fast ranking function, which completes within the time constraint even for a reasonable bad case, is chosen, thus sacrificing quality of results or alternatively a set of differently fast ranking functions are implement and chosen from appropriately.[6] now proposes a different approach, abstracting from individual ranking functions to choosing individual features such that the time constraint is still met.

Since this approach is very recent, a "Related Work" section is omitted; the used references however may of course provide further insight. Section 2 describes the approach itself, Section 3 experimental results and finally Section 4 will comment on the strengths and weaknesses of the discussed paper.

2 Constrained Linear Ranking

2.1 Features And Linear Ranking Functions

Linear Ranking functions, a a class of simple and commonly used ranking functions, is based on the weighted linear combination of *features,* functions that capture certain document properties, formally .

$$Rank(Query, Document) = \sum_i \lambda_i f_i(Query, Document)$$

Features can be based for example on *Term Occurrence* or *Term Proximity, User Behavior (cmp. [1])* or *linguistic* properties of a document(cmp. [6] table 1).

Yet, it should be clear that the computational cost of a feature is query dependent; for instance a *Unigram Term Occurrence* Feature , such as counting the times a term occurs, will depend on the number of terms; as another example consider the case of linguistic features which may take a shortcut, thus being almost free, if they detect the query is not a sentence. Formally, feature weights weights Λ will take a parametric form: $\lambda(Query, Document) = \sum_i w_i g_i(Query)$; where W are free parameters, resp. *meta feature weights,* and G are *meta features,* such as the time the query itself or different bigrams[1] from the query occur in a Wikipedia Title.(cmp. [6] table 2)

2.2 Feature Selection

Offline Feature Selection can not incorporate this. Formally, a constrained linear ranking function will have the following form:

$$Score(Query, Document) = \sum_{enabled\, features} \lambda_i f_i(Query, Document)$$

[6] proposes two algorithms for feature selection: one selecting independently and one heuristically exploiting interdependencies between features.

Intuitively, given a *time limit* T, only the most important features should be evaluated. To that end, a feature cost metric is needed, which is defined relatively against a baseline ranking function, as the sum of document frequencies $DF(t)$of each term t required to compute a feature f_i.

[1] an n-gram is a sub-sequence of n items from a given sequence

> "For features defined over unigrams, this sum only has a single component, while for features defined over bigrams, it has two components. Intuitively, this analytical model captures the fact that evaluating more features and evaluating each feature over longer postings lists (i.e., large DF values) will both result in greater time complexity."[6]

Additionally the weight of a feature has to be computed from the meta-features. However, in larger feature sets to choose from, some features may be redundant; therefore similar features, i.e. features with the same *concept*[2] should be penalized. In the presented paper, a feature is penalized, if a similar feature is already part of the feature selection and it's weight is below a certain threshold. Note that in this way, only "unimportant" features will be penalized.

2.3 Algorithms

2.3 and 2.3 show the algorithms for independent and joint feature selection, respectively. It should be easy to see, that these are variations of a greedy solution to the *knapsack problem*, where the *cost* of an feature corresponds to the size of an item to put into the sack.

Independent Feature Selection .

```
Algorithm 1: Independent Ranking
Input: Query execution time requirement T(q);
       ranking features $f_i(q, \cdot)$; meta-features $g_j(q)$
       and meta-features weights $w_j$
Output: Temporal constrained ranking function R(q)
Compute feature weights: $\lambda_i(q) = \sum_j w_j \; g_j(q)$;
Compute feature profit density: $p_i = \frac{\lambda_i(q)}{C(f_i)}$;
Queue $\mathcal{F}$: features sorted by decreasing profit density;
Initialize $R(q) = \{\}$;
Initialize $totalCost = 0$;
while $size(\mathcal{F}) > 0$ do
    Remove $f_i$ from head of $\mathcal{F}$ ;
    if $totalCost$ + cost($f_i$) $< T(q)$ then
        add $<f_i, \lambda_i>$ to ranking function $R(q)$;
        $totalCost = totalCost$ + cost($f_i$);
end
return $R(q)$;
```

[2] i.e. unigram, bigram or n-gram, etc.. Thus they will have the same feature weight.

```
Algorithm 2: Joint Ranking
Input: Query execution time requirement T(q);
       ranking features $f_i(q, \cdot)$; meta-features $g_j(q)$
       and meta-features weights $w_j$
Output: Temporal constrained ranking function R(q)
Compute feature weights: $\lambda_i(q) = \sum_j w_j \; g_j(q)$;
Compute feature profit density: $p_i = \frac{\lambda_i(q)}{C(f_i)}$;
Queue $\mathcal{F}_1$: features sorted by decreasing profit density;
Initialize queue $\mathcal{F}_2$={} ;
Initialize $R(q) = \{\}$;
Initialize $totalCost = 0$;
For each concept $e$, let $G_e$ = set of features defined on $e$;
Let $\lambda_e$ denote the weight of features in $G_e$;
Let $covered[e]$=false for all $e$;
while $size(\mathcal{F}_1) > 0$ or $size(\mathcal{F}_2) > 0$ do
    Remove $f_i$ that has max $p_i$ from head of $\mathcal{F}_1$ or $\mathcal{F}_2$;
    if $totalCost$ + cost($f_i$) < $T(q)$ then
        add $<f_i, \lambda_i>$ to ranking function $R(q)$;
        $totalCost = totalCost$ + cost($f_i$);
        Denote concept covered by $f_i$ by $e\prime$;
        if ($covered[e\prime]$ is false and $\lambda_{e\prime} < \alpha$) then
            $\lambda_{e\prime} = \lambda_{e\prime}$ - $\beta$;
            $G_{e\prime} = G_{e\prime} \setminus f_i$; move $G_{e\prime}$ from $\mathcal{F}_1$ to end of $\mathcal{F}_2$;
            $covered[e\prime]$ = true;
end
return $R(q)$;
```

2.4 Learning To Rank

Some Model parameters, such as the meta-feature weights, the pruning threshold α or the redundancy penalty β need to be *learned.* [4] and [2] give an overview over machine-learned ranking techniques. For the presented paper, a simple line search algorithm is proposed [3], where, given a metric[4], optimal values for the parameters, are iteratively found with a series of line searches[3].

3 Results

The algorithms were evaluated with the three TREC web test collections *Wt10g, Gov2* and *Clue.* First, the model parameters were *learned* on one half of the collections respectively. Then Ranking on the other half, the test set, was performed with

[3] Wang et al. go into more detail in [7]

[4] in this case MAP, Mean Average Precision, is used

All features enabled, feature selection by the *Indept* and the *Joint* algorithm and finally a baseline ranking algorithm *Query-Likelihood.* Consistently *Joint* outperformed the *Indept,* especially at lower time constraints. Also *Joint* had equal or better results than the baseline *QL* in the same time it took to compute QL in 5 out of 6 cases and got close to the all features case after at most 3 times the cost of *QL*.

Time Constraints were always met in over 80% of the cases and 90% in most cases. The authors also note, that a 5% tolerance increases the hit rate near to 1. Of great importance in these results is that some type of queries, those with many frequent terms or longer than average, missed consistently. This suggests that the cost estimate for these queries is wrong.

Additionally the SD model[5] was tested; but was found to be outperformed by *Joint* in all cases.

4 Weakness & Strengths

The presented paper introduces a heuristic approach to feature selection for ranking functions. Feature selection is linear-logarithmic on the query size, i.e. negligible compared to feature evaluation. Machine-based learning research focused on efficiency, but could not handle time constraints. As the evaluation suggests this new approach subsumes many commonly-used models as special cases.

However e.g. the cost estimation and the inclusion of a pruning threshhold seem to be very heuristically. Problematic queries that miss the time constraint stress the importance to tune these parts. Right know features are selected by a *simple,* greedy algorithm; a more sophisticated algorithm may give better results.

References

1. E. Agichtein, E. Brill, and S. Dumais. Improving web search ranking by incorporating user behavior information. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 19–26. ACM, 2006.
2. M. Bendersky, D. Metzler, and W.B. Croft. Learning concept importance using a weighted dependence model. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 31–40. ACM, 2010.
3. Wikipedia Contributors. en.wikipedia: Line search. page oldid 397287089.

4. T.Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.
5. D. Metzler and W.B. Croft. A markov random field model for term dependencies. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 472–479. ACM, 2005.
6. L. Wang, D. Metzler, and J. Lin. Ranking under temporal constraints. 2010.
7. Lidan Wang, Jimmy Lin, and Donald Metzler. Learning to efficiently rank. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '10, pages 138–145, New York, NY, USA, 2010. ACM.