A Fast Filter Feature Selection Algorithm for Ranking (Technical Report)

Miguel Costa [1,2]
miguel.costa@fccn.pt

Mário J. Silva [2]
mjs@di.fc.ul.pt

[1] Foundation for National Scientific Computing, Portugal
[2] University of Lisbon, Faculty of Sciences, LaSIGE, Portugal

ABSTRACT

Learning to rank (L2R) algorithms optimize search relevance by tuning the weights among large pools of ranking features. However, including more features in the ranking model also implies a larger implementation effort, more processing time to rank documents, and in some cases, more index space to store the features. In addition, as shown in this paper, more features do not necessarily lead to better relevance. We analyzed feature selection heuristics to find the best relation between the number of features and relevance. As a result, we propose a new heuristic for predicting the upper bound improvement that an additional feature can add to a feature set. We show that it is more precise, robust and faster than previous heuristics. Its application selects at most 15% of the ranking features used in three LETOR datasets, and even so, the MAP and NDCG@10 metrics are equally good.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Selection Process; D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*

General Terms

Algorithms, Experimentation

Keywords

Information retrieval, learning to rank, feature selection

1. INTRODUCTION

Information retrieval (IR) researchers have proposed many ranking algorithms that estimate information relevance. Previous evaluations showed that combinations of ranking functions tend to provide better results than any single function [1, 2, 13]. An individual function is also more susceptible to influences caused by the lack or excess of data (e.g. spam). Therefore, it is advantageous to use different aspects of the data to build a more precise and robust ranking model. By robust, we mean a model capable of coping well with variations in data. For instance, a document can receive a low relevance score due to a small query term frequency, but a high number of inlinks can identify the document as important. All these factors must be properly balanced by the model.

A ranking model estimates the query relevance of a document. Documents matching the query are then sorted in descending order by their relevance score, which enables users to find information efficiently. Thus, the generation of such model is a fundamental problem in IR. We decompose this generation in a pipeline of four steps: (1) extraction of low-level ranking features, such as term frequency or document length, which are then (2) assembled in high-level ranking features (a.k.a. ranking functions), such as BM25 [21]; (3) the most suitable features for a retrieval task are selected and (4) combined in a way to maximize the results' relevance. For simplicity, this combination is normally linear, i.e. for a document d with a vector of low-level ranking features associated, $\vec{d}$, the values produced by the n selected ranking features are added after each feature f_i is weighted by a coefficient λ_i and adjusted with a value b_i:

$$\text{rankingModel}(\vec{d}) = \sum_{i=1}^{n} \lambda_i f_i(\vec{d}) + b_i \qquad (1)$$

The first two steps are well studied and some ranking features, such as BM25, are good ranking models alone [14]. Combining them manually is not trivial, specially when the retrieval task is new or new types of features are considered (e.g. semantically annotated content). Hence, in the last few years the fourth step has been concentrating attention. Supervised learning algorithms have been employed to tune the weights between combined ranking features, resulting in significant improvements [12]. However, using more ranking features does not necessarily lead to higher relevance. In many cases, features are redundant or irrelevant.

Redundant features overvalue documents, weighting them multiple times, while irrelevant features produce random distributions of document relevance scores, confusing machine learning algorithms and degrading results [7]. Most learning algorithms are susceptible to this randomness. By removing redundant and irrelevant features, the ranking model is simplified, which tends to avoid overfitting (fits training data closely, but fails to generalize to unseen test data), improves relevance, reduces data and speedup learning algorithms. By using fewer ranking features, search engines

Technical Report January 2010

can respond faster to users' queries. Search engines can also reduce their indexes if these contain the data of the features that are to be excluded. Space and speed are two of our main concerns, since we are developing a web archive with full-text search over collections with hundreds of millions of documents [6]. The other concern is reducing the effort for creating a precise and robust ranking model. Implementing dozens of ranking features for the model is time consuming and laborious. Thus, we are willing to lose some relevance in exchange of a large reduction in the number of features.

Our study focuses on feature selection for ranking, the third step of the pipeline. The goal is to find an algorithm that minimizes the number of features, while keeping the results' relevance. Our main contribution is a new, fast and robust feature selection algorithm for ranking that achieves this goal better than previous known algorithms.

This paper is organized as follows. In Section 2, we cover the related work. In Section 3, we describe our algorithm. The conducted experiments are explained in Section 4 and results are detailed in Section 5. Ranking model variations are analyzed in Section 6 and Section 7 finalizes with the conclusions.

2. RELATED WORK

Feature selection is a common preprocessing step in learning algorithms for searching an optimal subset of features, filtering out as much of the irrelevant and redundant information as possible. It is a well known problem in classification. In ranking, the problem is similar. Both classification and ranking select the features most suitable to produce a highly correlated model with the expected output, yet with features uncorrelated among them. In classification, the learning algorithms produce a classification model that predicts the class of each document, while in ranking, the learning algorithms tune the weights between the combined ranking features to produce a model that maximizes the relevance of ranked lists of documents.

Feature selection algorithms are usually organized in three categories, depending on how they interact with the learning algorithms [7]:

filter algorithms perform a preprocessing step independent of the learning algorithm. By analyzing only the general characteristics of the feature data (e.g. statistical dependencies), they compute a goodness measure of the feature subset, which is fast and enables them to easily scale to high-dimensional datasets. However, they optimize the subset toward maximizing a goodness measure, instead of maximizing the measure used to evaluate the final results.

wrapper algorithms use the learning algorithm as a black box, generating a model that is used to score the feature subsets. Thus, the search is guided to maximize the final results, instead of a goodness measure of the subset. These algorithms tend to produce superior results, but the training of the learning algorithm is computationally intensive, being impractical for large scale problems. Another problem is that these algorithms have a higher risk of overfitting.

embedded algorithms implement the search for the best subset inside the learning algorithm. Like wrapper algorithms, the search is aimed at maximizing final results due to the interaction with the learning algorithm, but have the advantage of being less computationally intensive. As disadvantage, they are complex to implement and are intrinsic to the learning algorithm.

In this work, we are mainly concerned with algorithms that can produce ranking models in feasible time for the number of features we are currently studying, 64. There are search engines using a larger number, such as Google that uses more than 200 features (see `http://www.google.com/corporate/tech.html`). Most filter algorithms satisfy this speed requirement, so we will focus on this category. A greedy search strategy for wrapping or embedded algorithms can also be used. In greedy search, each of the non-selected features is combined with the previously selected features and the one that improves the results the most is chosen. The search is iteratively performed until no more features remain or a stop condition is reached.

Filter algorithms evaluate features either independently or in a dependent manner. We unfold both approaches.

2.1 Independent Feature Selection

In this approach, the features are independently ranked by their correlation with the target class (relevance judgment class) or by some importance measure. Then, those under a threshold or over a subset size are excluded. The feature independence assumption makes this the faster approach.

Several measures are used for this purpose, for instance, statistical measures, such as Chi-squared and ReliefF [22], or entropy based measures [17], such as Information Gain (IG) and Symmetrical Uncertainty (SU) [26].

Independent feature selection presents however a major drawback: it ignores that a feature that is useless on its own may still provide a significant improvement when combined with others. For instance, query-independent ranking functions produce a static importance score per document, resulting in a bad correlation with the target class. As result, these functions are not selected. However, some query-independent ranking functions, such as PageRank [18], are among the preferred algorithms to include in a web ranking model [2].

2.2 Dependent Feature Selection

The search for the best subset of features can follow different strategies, such as exhaustive or greedy searches. A common heuristic is using a goodness measure of the subset, based on a tradeoff between features' relevance and correlation. Hall developed the Correlation-based Feature Selection (CFS) algorithm [8]. It calculates for a feature subset S of size k, the correlation average of all its features with the target class, $\overline{\tau_{tf}}$, and the correlation average between all its features, $\overline{\tau_{ff}}$. Correlations are measured with SU and both averages are used in the $Merit$ function:

$$Merit(S) = \frac{k\overline{\tau_{tf}}}{\sqrt{k + k(k-1)\overline{\tau_{ff}}}}$$

The goal is to maximize the numerator, while minimizing the denominator. The algorithm executes a best-first search, where a new feature f_i is iteratively added to S, if for any other feature f_j not selected, $Merit(S + \{f_i\}) > Merit(S + \{f_j\})$ and $Merit(S + \{f_i\}) > Merit(S)$. The best-first search is a breadth-first search that follows the

best paths. The algorithm stops after the consecutive paths result in no improvement over S.

Yu and Liu developed the Fast Correlation Based Filter (FCBF) [25]. A feature f_i is considered irrelevant if $SU(f_i, t) < \delta$, being δ a threshold and t the target class. In an analogous way, two features f_i and f_j are redundant if $SU(f_i, f_j) > \delta$. The algorithm avoids all pairwise comparisons by ordering all the features by their $SU(f_i, t)$. Then it iteratively selects the top feature f_i and removes the next features f_j, while $SU(f_i, f_j) > SU(f_i, t)$.

Dash et al. developed the Consistency filter algorithm to evaluate feature subsets based on an inconsistency measure [3]. An inconsistency occurs when two instances of the dataset have the same values, but different target class values. These cases create randomness, reducing the discriminating power of the selected features. The ratios of inconsistency are the selection criteria to choose the best subset. Inconsistency, denoted U, has an interesting characteristic, monotonicity, which guarantees that if $S_i \supseteq S_j \Rightarrow U(S_i) \leq U(S_j)$, for any subsets S_i and S_j. Hence, the less inconsistent subset will contain all the features and the search method will try to find the smallest subset with similar inconsistencies, removing one feature at a time (backward elimination).

The INTERACT algorithm is an extension of the Consistency algorithm, which first sorts all the features by their $SU(f_i, t)$ and then iteratively tests the feature with the smallest $SU(f_i, t)$ [28]. The feature is maintained if removing it increases the inconsistency above a threshold. The algorithm focuses on the selection of the features that most contribute to the consistency of the dataset.

Geng et al. developed the GAS algorithm [5]. As far as we know, it is the only filter feature selection algorithm developed for ranking. Contrary to previous algorithms, GAS uses relevance measures to identify relevant features: the Mean Average Precision (MAP) [14] and the Normalized Discount Cumulative Gain (NDCG) [9]. It also uses loss functions. For redundancy identification it uses the Kendall τ correlation coefficient. The redundancy of the features is parameterized as a proportion of the features' relevance. Using a greedy algorithm and satisfying the condition that at each iteration a new feature is added to the previous subset, i.e. $S_{t+1} \supset S_t$, they find an optimal solution for their goodness measure of the subset.

2.3 Feature Selection Limitations on Ranking

Almost all feature selection algorithms have been developed for classification. Using them for ranking straightforwardly penalizes results. The more symptomatic differences are:

1. The relevance of a document is query-dependent. Hence, feature selection algorithms for ranking must perform a per query analysis, while the algorithms for classification use the query identifier as any other feature.

2. The metrics to evaluate classification and ranking algorithms are different. While in classification precision and recall are both important (e.g. F-measure), precision is usually more important for ranking (e.g. MAP).

3. Usually, a goodness measure designed to evaluate feature subsets is a tradeoff between the relevance of the features (their sum) and their redundancy (their pairwise sum). Correlation takes a crucial importance in the evaluation, since redundancy is measured as the correlation between features values, and relevance, in some cases, as the correlation of feature values with the target class. The problem is that a correlation between two features, measures in the same way, but erroneously, the discrepancies at the top and bottom of the ranking. New correlation measures for information retrieval, such as the proposed by Yilmaz et al., address this ranking order problem [24].

3. FEATURE SELECTION ALGORITHM

We created a supervised filter feature selection algorithm for ranking that overcomes the limitations pointed in Section 2.3: the features are separately analyzed for each query and the implemented heuristic is driven by a relevance measure, such as MAP, instead of a goodness measure based on ranking correlations.

We will start by explaining the heuristic to estimate the ranking that would result from optimally combining two feature sets in a ranking model and then we will detail the proposed algorithm.

3.1 Estimating the Optimum Combination

Our heuristic is based on the following assumptions:

1. L2R algorithms maximize search relevance and studies demonstrated that they produce good results [12]. Assuming a linear combination of features as exhibited in Equation 1, L2R algorithms will tune the λ_i and b_i values associated to each feature f_i to produce a ranking with all relevant documents ahead of the nonrelevant or the closest to this ranking.

2. Ranking features are monotonic functions, meaning that the more relevant a document is with respect to a query, the higher is its feature value. Thus, for each feature f_i the λ_i must be positive to guarantee the monotonicity.

3. If a document d_y occurs ahead of a document d_x in all the ranking features, i.e. $\forall f_i \in F, f_i(d_y) > f_i(d_x)$, then this precedence is maintained in the ranking of F. Equation 1 shows that it is necessary at least one feature f_i ranking results in the inverse order, to multiply a positive λ_i with a value large enough to invert the order of the two documents ranked by F. We also assume that if two documents occur only in one ranking, then this precedence must be maintained because their values on the other ranking are too small to enable a λ_i to invert their order.

Summarizing, we assume that in an optimum case, L2R algorithms will rank all relevant documents ahead of the nonrelevant, except on the cases of the third assumption. Based on this, we estimate the ranking produced by the linear combination of the rankings of two feature sets. Iteratively, we select from the two rankings, the minimum length sequence of nonrelevant documents followed by a sequence of relevant documents. Both sequences are selected. This way, we guarantee two things: (1) the relevant documents will be ranked as close to the top as possible; and as a consequence, (2) the number of relevant documents ahead of the nonrelevant is maximized. Naturally, the linear combination of ranking features given by the λ coefficients does not

always allow arbitrarily switching between both rankings to pick the best sequence of documents. However, as we do not know the λ coefficients, we assume the best case that a linear combination would possibly achieve. Our heuristic is not too far from reality. An experiment with 1000 random pairs of features showed an average Kendall τ correlation of 0.6 between the rankings produced by our heuristic and the rankings produced from the L2R algorithm SVM-MAP [27].

Algorithm 1 BestCombinationRankings(R_{best},R_{f_j})

1: $p_{best} \Leftarrow -1$
2: $p_{f_j} \Leftarrow -1$
3: $R_{comb} \Leftarrow \varnothing$
4: **while** $p_{best} < size(R_{best})$ OR $pf_j < size(R_{f_j})$ **do**
5: $\quad d_{best} \Leftarrow distanceToNextRelevant(p_{best}, R_{best})$
6: $\quad d_{f_j} \Leftarrow distanceToNextRelevant(p_{f_j}, R_{f_j})$
7: $\quad$ **if** $d_{best} <= d_{f_j}$ **then**
8: $\quad\quad R_{comb} \Leftarrow R_{comb} \bigcup R_{best}[p_{best}+1, p_{best}+d_{best}]$
9: $\quad\quad p_{best} \Leftarrow p_{best} + d_{best}$
10: $\quad$ **else**
11: $\quad\quad R_{comb} \Leftarrow R_{comb} \bigcup R_{f_j}[p_{f_j}+1, p_{f_j}+d_{f_j}]$
12: $\quad\quad p_{f_j} \Leftarrow p_{f_j} + d_{f_j}$
13: $\quad$ **end if**
14: **end while**
15: $R_{comb} \Leftarrow R_{comb} \bigcup R_{best}[p_{best}+1, size(R_{best})-1]$
16: $R_{comb} \Leftarrow R_{comb} \bigcup R_{f_j}[p_{f_j}+1, size(R_{f_j})-1]$
17: **return** R_{comb}

Algorithm 1 details our heuristic for estimating the optimum ranking that one could obtain when combining a feature set with a new feature, given their ranked result lists. The algorithm starts by assigning indexes, p_{best} and p_{f_j}, to the head of each ranking list, R_{best} and Rf_j (lines 1-2). Then, iteratively, while at least one of the indexes p does not reach the end of its corresponding ranking R, the minimum distance is calculated from p to the position of the next relevant document in R (lines 4-6). The ranking of R with the minimum distance d is selected, its next d elements are added to R_{comb} ($d-1$ nonrelevant and 1 relevant) and p is incremented by d (lines 7-13). Note that, if no more relevant documents exist in a ranking, then the minimum distance will be set to a number larger than the ranking sizes. Thus, only when there are no more relevant documents in the two rankings, the remaining nonrelevant documents are added to R_{comb} (lines 15-16). Note also that the documents already picked from one ranking are ignored for any further processing on the other ranking.

3.2 Greedy Algorithm

The estimate of the ranking that would result from the optimum combination between two rankings, as described in the previous section, enables us to calculate the maximum gain that a feature f_i can give to a feature subset S_{best}. The gain is calculated as the difference between the relevance values (e.g. MAP) of the estimated rankings from subsets $S_{best}+\{f_i\}$ and S_{best}. A small gain is a clear indication that adding this feature will not improve the results' relevancy. Despite our heuristic not providing a very close estimate, it enables us to safely discard the features that have an upper bound gain smaller than our lower threshold.

The main idea of our algorithm, called BestGain, is to select in each iteration, the feature f_i that leads to the highest gain when combined with the previously selected features, S_{best}. In this way, we quantify how much a feature f_i can complement S_{best}, by shifting up or bringing in new relevant documents for the top positions. For instance, a duplicated feature will not produce any gain, which implies its exclusion. The algorithm will stop selecting features when the gain offered by the remaining features is too small. This works exactly like a greedy wrapper algorithm, with the difference that instead of computing the optimized ranking from a L2R algorithm, we compute a best case estimate, which is much faster to compute. We try to combine the best of two worlds: the fastness of filter algorithms with the superior results of the wrapper algorithms.

Algorithm 2 BestGain

1: $f_i \Leftarrow argmax_{f_i} Relevance(ranking(f_i))$
2: $F \Leftarrow F \setminus \{f_i\}$
3: $S_{best} \Leftarrow \{f_i\}$
4: $R_{best} \Leftarrow ranking(f_i)$
5: **repeat**
6: $\quad$ **for** $j = 1$ to $|F|$ **do**
7: $\quad\quad$ **for** $q = 1$ to $|Q|$ **do**
8: $\quad\quad\quad R_j \Leftarrow BestCombinationRankings(R_{best}, ranking(f_j))$
9: $\quad\quad\quad gain_j \Leftarrow gain_j + \frac{Relevance(R_j) - Relevance(R_{best})}{|Q|}$
10: $\quad\quad$ **end for**
11: $\quad$ **end for**
12: $\quad f_k \Leftarrow argmax_{f_k} gain_k$
13: $\quad F \Leftarrow F \setminus \{f_k\}$
14: $\quad S_{best} \Leftarrow S_{best} \bigcup \{f_k\}$
15: $\quad R_{best} \Leftarrow R_k$
16: **until** $gain_k < \delta$
17: **return** S_{best}

Algorithm 2 details all the steps. It starts by selecting the feature f_i yielding the highest *Relevance* value (e.g. MAP) and moving it from the subset F containing all the features to the final subset S_{best} (lines 1-3). Then, the ranking of f_i is stored. For each of the remaining features f_j in F, it combines for each query q, the ranking of f_j with the best ranking achieved until that moment, R_{best} (lines 6-8). The *gain* is calculated as the average difference between the *Relevance* values of the rankings from subsets $S_{best}+\{f_i\}$ and S_{best} (line 9). The feature f_k that produces the highest *gain* is added to S_{best} and its combined ranking assigned to R_{best} (lines 12-15). This ranking is iteratively improved with new features, until the *gain* is smaller than a threshold δ.

Having one ranking model implies that the features must be combined in the same manner for all queries. However, we applied the best linear combination for each query, which is not realistic, but guarantees a valid upper bound gain.

4. EXPERIMENT SETUP

4.1 Datasets

Experiments were performed over the LETOR version 3.0, which has been defined as benchmark for ranking [13, 16]. The focus of this benchmark is to normalize tests between L2R algorithms. We found that it can has the same purpose for feature selection applied to ranking. New feature selection algorithms can be compared under the exact same conditions. LETOR aggregates IR test collections, including query sets, relevance judgments, evaluation metrics and evaluation tools. In addition to that, it extracts different low-level and high-level feature values for each <document, query, relevance judgment> triple, eliminating the

dataset	collection	q	f	rel	instances
TD2003	.gov	50	64	2	49058
TD2004	.gov	75	64	2	74146
OHSUMED	ohsumed	106	45	3	16140

q=number of queries; f=number of features; rel=levels of relevance

Table 1: Datasets used in experiments.

usual parsing and indexing difficulties. The results of some state-of-the-art L2R algorithms are also provided for a direct comparison.

On the experiments, we used three datasets included in LETOR and summarized in Table 1. The TD2003 and TD2004 datasets are the topic distillation tasks from TREC 2003 and TREC 2004 web tracks. Both datasets include the .gov collection, composed by a crawl of .gov web sites in early 2002, containing 1,053,110 HTML documents. The datasets also include 50 and 75 queries, respectively. For each query, the framework extracted values from 64 features for all top 1000 documents returned by BM25 [21], with binary judgments (relevant or nonrelevant) associated. The features are for example, the number of inlinks, TFxIDF [14] and BM25 functions over different fields (URL, title, anchor and body) and PageRank [18] (see [16] for a detailed list).

The OHSUMED dataset contains a collection from MEDLINE, a database on medical publications. This dataset contains 348,566 records from 270 medical journals over a five-year period (1987-1991). The fields of the records include title, abstract, MeSH indexing terms, author, source, and publication type. It contains 106 queries with judged documents using a three level scale: definitely, possibly, or not relevant. LETOR extracted values from 45 features for all the judged documents [16].

4.2 Feature selection algorithms

We tuned and tested three feature selection algorithms designed for ranking: our BestGain algorithm using MAP as the relevance measure; the GAS-E algorithm, which is the GAS algorithm using MAP to measure relevance and Kendall τ to measure correlation [5]; and the greedy Wrapper algorithm, exactly like Algorithm 2, but using a L2R algorithm instead of Algorithm 1 in line 8. The first two are filter algorithms, while the last is a wrapper algorithm.

For comparison, we tested four state-of-the art algorithms for feature selection used in classification: FCBF, CFS, Consistency and INTERACT, described in Section 2. They were computed with WEKA, a machine learning framework implemented in JAVA [23]. INTERACT was downloaded from [28]. No tuning was performed over these algorithms. We used WEKA's default parameters.

4.3 Learning to rank algorithms

The relevance of a ranking model is evaluated after learning the weights between the ranking features. We used the SVM-MAP algorithm, a Support Vector Machine (SVM) algorithm extended to ranking with a hinge loss relaxation of MAP loss [27]. The optimal C coefficient for this algorithm, the one that controls the tradeoff between the model complexity and the hinge loss relaxation of MAP loss, was parameterized with the optimal values found to produce the LETOR results (see `http://www.yisongyue.com/results/svmmap_letor3/details.html`). These results are our Baseline, since it contemplates all features.

4.4 Evaluation Methodology and Metrics

We chose a five-fold cross-validation of the algorithms. The datasets were divided in five folders, having each folder three subsets: one for training, one for validation and one for testing. For each folder, the algorithm created a ranking model using the training data. The validation data was ignored, since we used the optimal parameters found for SVM-MAP. The test data was used only on the evaluation of the model, to avoid overfitting and biased estimates of the model [19]. The final results are the averages of the five tests.

The algorithms were evaluated with the Mean Average Precision (MAP) [14] and the Normalized Discount Cumulative Gain at 10 (NDCG@10) [9], two of the most used metrics to evaluate ranking models. Both attribute more weight to the top results. NDCG@10 restricts the evaluation to the top 10 results.

MAP works over binary judgments, while NDCG handles multiple levels of relevance. Due to that, the three levels of relevance for judging OHSUMED were converted into only two when using MAP. The definitely and possibly relevant judgments were both taken as relevant.

5. RESULTS

In this section we experimentally evaluate various aspects of the feature selection algorithms. We also compare it against the Baseline without feature selection (i.e. using all features). All results from the tested algorithms are aggregated in Table 2, divided by metric, dataset, algorithm purpose (classification or ranking) and feature selection taxonomy (filter or wrapper). Each result is the average between the five tests executed due to the five-fold cross validation.

5.1 MAP and NDCG@10 Evaluation

Table 2 shows that the algorithms designed for ranking present better MAP and NDCG@10 than the algorithms designed for classification. The exception is the Consistency algorithm, which works surprisingly well in the three datasets. However, it selects much more features. The three algorithms designed for ranking also exceed, slightly, the Baseline in most of the datasets with much less features. BestGain has the best MAP and NDCG@10 averages over the three datasets. The differences between the BestGain and GAS-E are not significant to identify the best algorithm. On the other hand, the differences to the Wrapper are larger. The Wrapper presents the best MAP and NDCG@10 for the TD2004 and OHSUMED datasets. However, it overfits on the TD2003 dataset, presenting poor results.

5.2 Number of features

The algorithms designed for ranking select less features than the algorithms designed for classification (see Table 2). The exception is the FCBF algorithm, but it presents the worst MAP and NDCG@10 from all the tested algorithms. The BestGain and the Wrapper algorithms always select at most 15% of the features. It is noteworthy that the BestGain presents better NDCG@10 for the OHSUMED than the Baseline, selecting just one feature. The GAS-E fails to select few features on the TD2003 dataset. It selects 49.2 features from the 64, which is much more than the selected

Metric	Dataset	classification				ranking			Baseline
		filter						wrapper	
		FCBF	CFS	Consistency	INTERACT	GAS-E	BestGain	Greedy Wrapper	
MAP	TD2003	9.51%	17.08%	**25.20%***	20.64%	**24.86%***	**25.74%***	20.71%	24.45%
	TD2004	15.77%	18.16%	20.32%	17.13%	20.04%	20.33%	**23.00%***	20.49%
	OHSUMED	42.24%	43.34%	44.09%	44.16%	**44.72%***	44.00%	**45.08%***	44.53%
	avg.	22.51%	26.19%	**29.87%***	27.31%	**29.87%***	**30.03%***	29.60%	29.82%
NDCG@10	TD2003	14.27%	23.32%	32.25%	26.47%	32.07%	**33.42%***	24.65%	32.82%
	TD2004	22.49%	25.42%	28.79%	24.03%	**29.39%***	**30.62%***	**32.42%***	29.07%
	OHSUMED	38.82%	40.40%	42.18%	42.41%	**45.30%***	**44.32%***	**45.40%***	43.19%
	avg.	25.19%	29.71%	34.41%	30.97%	**35.59%***	**36.12%***	34.15%	35.02%
# features	TD2003	3.8	15	30.6	20.6	49.2	9.8	4.2	64
	TD2004	3	12.8	27.8	18	2.4	2.2	6	64
	OHSUMED	1	4.2	35	34.8	2.6	1	2	45
	avg.	2.6	10.7	31.1	24.5	18.1	4.3	4.1	57.7
time (min)	TD2003	0.6	1.1	21.9	1.2	0.18 (+238)	1	241.2	-
	TD2004	1	1.6	60	3.9	0.16 (+271)	1.6	215.8	-
	OHSUMED	0.1	0.1	1.9	0.3	0.08 (+40)	0.4	53.8	-
	avg.	0.6	0.9	27.9	1.8	0.14 (+183)	1	170.3	-

* indicates that results are higher than the baseline

Table 2: Results for feature selection algorithms.

by the other two algorithms. We saw that GAS-E degrades the ranking results when it subtracts the correlation between features, so the tuned algorithm mostly ignores the correlation.

The number of features selected can be compared to the reported by Metzler in experimenting his feature selection wrapper algorithm for ranking [15]. His best results have ranking models with at most five features. However, the datasets and evaluation methodology used in this evaluation are different.

5.3 Computational Time

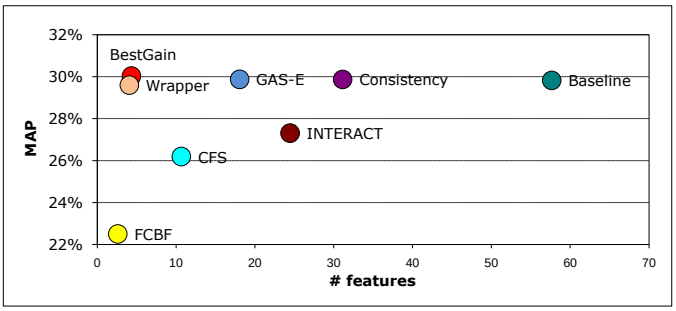
The computation times of each algorithm are detailed in Table 2. Algorithms run on an Intel quad-core machine at 2.33GHz and 8GB of RAM. All algorithms are fast, running in average in less than two minutes. The exception for the classification-based algorithms is the Consistency algorithm, which requires around half an hour to select the features. The exception for the ranking-based algorithms is the Wrapper algorithm, which takes more than two hours. The complexity of the greedy Wrapper algorithm is the same complexity of the BestGain. It will require $O(n^2)$ steps for n features. However, as the results show, the learning in each step of the Wrapper increases extremely the computing time. Optimizing the greedy Wrapper took us around four days in comparison with less than an hour for the BestGain. For a larger set of features and a consequent larger selection, the computing time for the Wrapper will become prohibitive.

The GAS-E algorithm is the fastest, taking just a few seconds to select the features. However, it is necessary to first create a Kendall τ correlation matrix with the correlation between all pairs of features. This matrix took in average 183 minutes to create. Additionally, the GAS-E has two parameters to tune, the proportion between the relevance and redundancy of the features and the threshold of the stop condition. The first parameter requires extra tests to tune, not necessary on the BestGain or the Wrapper that have only the second parameter to tune. If we count the overall computing time, the BestGain algorithm is the fastest of the three.

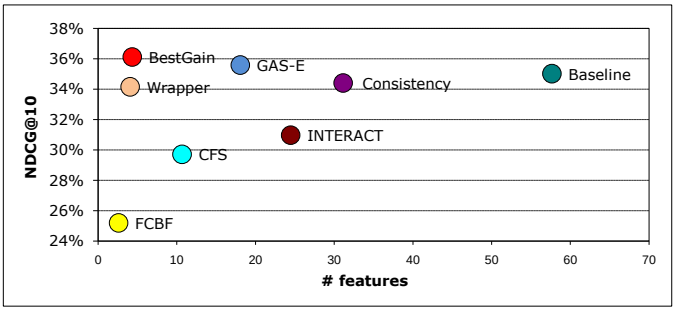
5.4 Discussions

For an easier comparison between the algorithms and the Baseline, Figures 1(a) and 1(b) combine the relevance values in the y-axis, MAP and NDCG@10, with the number of features in the x-axis. These values are the averages between all three datasets. Our goal is to find the algorithm closest to the upper left corner of the figures, i.e. the one creating ranking models with the best relevance and the minimum number of features. The results show that:

1. More features do not lead to better results. At most 15% of the features showed to be sufficient to present as good results as with all the features, for a statistical significance level of 0.05 using a paired Student's t-test.
2. The three tested algorithms designed for ranking present the best relation between the number of features reduced and MAP or NDCG@10. The BestGain algorithm achieves the best relation between all algorithms.
3. The GAS-E algorithm selects too many features on the TD2003 dataset, while the Wrapper algorithm overfits on the same dataset. Both algorithms have a robustness inferior than the offered by the BestGain.
4. As expected, the Wrapper algorithm is much slower than the other algorithms due to the learning phase. The filter algorithms will be the only viable choice when the number of features grows significantly.



(a)



(b)

Figure 1: Comparison of feature selection algorithms evaluated with (a) MAP and (b) NDCG@10.

5. Improving the baselines by removing features is a difficult task, due to the good optimization performed by the L2R algorithms.

BestGain estimates rankings independently of the evaluation metric, but only handles binary relevance judgments. We were expecting that this would have a negative effect on the creation of ranking models for the OHSUMED dataset, which contains ternary judgments. Surprisingly, we achieved better results with NDCG@10 than with MAP. One explanation is that the BestGain heuristic selects the features that add more relevant documents to the top positions. Thus, measures such as NDCG@10 and P@10 that only account for the top positions are more likely to achieve accurate estimates of the combined rankings.

6. FURTHER EXPERIMENTS

At the end of the previous experiment, we expected to find the more relevant and discriminating features occurring in all the ranking models of a dataset and between datasets. Five ranking models were created for each dataset given the five-fold cross validation adopted in the evaluation methodology. We also expected to find the typical number of features of a good ranking model. This knowledge would enable us to have a better understanding of the datasets and the features that have influence on them. As result, we could engineer more precise ranking models. Results derived from the BestGain algorithm confirmed only part of our expectations.

Table 3 shows the number of features selected for the five ranking models created from each dataset. For instance, two close datasets sharing the same collection, TD2003 and TD2004, have ranking models sizes from two to eleven features. The variance across models of the same dataset is only of two features.

dataset	ranking model				
	1	2	3	4	5
TD2003	9	11	9	10	10
TD2004	3	2	2	2	2
OHSUMED	1	1	1	1	1

Table 3: Number of features selected.

dataset	# of repetitions				
	5x	4x	3x	2x	1x
TD2003	2	1	5	8	4
TD2004	1	0	1	0	3
OHSUMED	0	0	1	0	2

Table 4: Number of features repeated across the models of the same dataset.

We analyzed the regularity of features selected across ranking models created for the same dataset. Table 4 details that two features are repeated in five ranking models for the TD2003 dataset and one in four models. There is some overlap of features between models, nevertheless the majority only occurs once or two.

Analyzing the overlap between the features across ranking models of different datasets showed us a weak repetition (see Table 5). Only one feature occurs on the ten ranking models. We only analyzed the intersection between TD2003 and TD2004, because OHSUMED has different features and fields, which are not comparable. The overlap between both datasets identifies the most relevant and robust features, detailed in Table 6 (see description in [16]). A shallow analysis over the top ten most repeated features, indicates that propagation algorithms and term weighting functions, such as BM25 applied over the title and anchors, are prevalent. Features based on the URLs are not present. This result is in conformity with Kang and Kim, which conclude that URL information is poor for topic distillation despite being relevant for homepage finding tasks [10].

The overall results show that there is a high variance in the ranking models automatically produced, which suggests that differences in data and query sets can strongly bias the choice of the best ranking model. Thus, a unique ranking model can hardly give good results for all type of queries in generic web search engines. One alternative is to pick a ranking model according to the query type [4, 10]. However, it is often hard to classify queries due to their small number of terms.

7. CONCLUSIONS

We used LETOR to study feature selection algorithms for ranking, with the goal of minimizing the number of features, while keeping the results' relevance. We studied seven algorithms, three of which are designed for ranking and four for classification. The results achieved with the algorithms designed for ranking are generally superior. The results of our supervised algorithm, called BestGain, are superior on average to all the others, measured with MAP and NDCG@10, over three different datasets. The results also show that with

dataset	# of repetitions									
	10x	9x	8x	7x	6x	5x	4x	3x	2x	1x
TD2003 ∩ TD2004	1	0	0	0	1	0	1	7	8	3

Table 5: Number of features repeated across the models of the TD2003 and TD2004 datasets.

ranking feature	# of repetitions
HostRank	10
Sitemap based term propagation	6
LMIR.JM of extracted title	4

Table 6: Most repeated features.

a small set of criteriously selected features, 15% at most, we can achieve equally good relevance than when using all the features. This emphasizes the major drawback of L2R algorithms: they unnecessarily produce large dimensional models.

We studied the variations between the resulting ranking models produced over all datasets. We discovered that models vary in size and on the features selected, even when the datasets are closely related. There is a high sensitivity to query set changes, which indicates that a single ranking model for generic web search engines tends to perform badly. This also confirms that a ranking model tuned with a test collection, such as the ones from TREC, will probably present poor results in a real searching system. Future L2R algorithms should contemplate in their fitness or loss functions the robustness of models, using measures such as query-level stability [11] or Geometric Mean Average Precision (GMAP) [20].

8. REFERENCES

[1] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[2] N. Craswell, S. Robertson, H. Zaragoza, and M. Taylor. Relevance weighting for query independent evidence. In *Proc. of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 416–423, 2005.

[3] M. Dash, H. Liu, and H. Motoda. Consistency Based Feature Selection. In *Proc. of the 4th Pacific-Asia Conference on Knowledge Discovery and Data Mining, Current Issues and New Applications*, pages 98–109, 2000.

[4] X. Geng, T. Liu, T. Qin, A. Arnold, H. Li, and H. Shum. Query dependent ranking using k-nearest neighbor. In *Proc. of the 31st International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–122, 2008.

[5] X. Geng, T. Liu, T. Qin, and H. Li. Feature selection for ranking. In *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 407–414, 2007.

[6] D. Gomes, A. Nogueira, J. Miranda, and M. Costa. Introducing the Portuguese web archive initiative. In *Proc. of the 8th International Web Archiving Workshop*, 2008.

[7] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Machine Learning Research*, 3:1157–1182, 2003.

[8] M. Hall. Correlation-based feature selection for discrete and numeric class machine learning. In *Proc. of the 17th International Conference on Machine Learning*, pages 359–366, 2000.

[9] K. Järvelin and J. Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems*, 20(4):422–446, 2002.

[10] I. Kang and G. Kim. Query type classification for web document retrieval. In *Proc. of the 26th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 64–71, 2003.

[11] Y. Lan, T. Liu, T. Qin, Z. Ma, and H. Li. Query-level stability and generalization in learning to rank. In *Proc. of the 25th International Conference on Machine Learning*, pages 512–519, 2008.

[12] T. Liu. *Learning to rank for information retrieval*, volume 3 of *Foundations and Trends in Information Retrieval*. Now Publishers Inc., 2009.

[13] T. Liu, J. Xu, T. Qin, W. Xiong, and H. Li. Letor: Benchmark dataset for research on learning to rank for information retrieval. In *Proc. of SIGIR 2007 Workshop on Learning to Rank for Information Retrieval*, 2007.

[14] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[15] D. Metzler. Automatic feature selection in the Markov random field model for information retrieval. In *Proc. of the 16th ACM Conference on Information and Knowledge Management*, pages 253–262, 2007.

[16] Microsoft Research Asia. Letor: A benchmark collection for learning to rank for information retrieval (an incomplete draft), March 2009.

[17] T. M. Mitchell. *Machine Learning*. McGraw-Hill Co., 1997.

[18] L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: bringing order to the web. Technical report, Stanford Digital Library Technologies Project, 1998.

[19] J. Reunanen. Overfitting in making comparisons between variable selection methods. *The Journal of Machine Learning Research*, 3:1371–1382, 2003.

[20] S. Robertson. On GMAP: and other transformations. In *Proc. of the 15th ACM International Conference on Information and Knowledge Management*, pages 78–83, 2006.

[21] S. E. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford. Okapi at TREC-3. In *Proc. of the 3rd Text REtrieval Conference*, pages 109–126, 1995.

[22] M. Robnik-Šikonja and I. Kononenko. Theoretical and empirical analysis of ReliefF and RReliefF. *Machine Learning*, 53(1):23–69, 2003.

[23] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., 2005.

[24] E. Yilmaz, J. Aslam, and S. Robertson. A new rank correlation coefficient for information retrieval. In *Proc. of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 587–594, 2008.

[25] L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proc. of the 20th International Conference on Machine Learning*, pages 856–863, 2003.

[26] L. Yu and H. Liu. Efficient feature selection via analysis of relevance and redundancy. *The Journal of Machine Learning Research*, 5:1205–1224, 2004.

[27] Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In *Proc. of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 271–278, 2007.

[28] Z. Zhao and H. Liu. Searching for interacting features. In *Proc. of the 20th International Joint Conference on AI (IJCAI-07)*, 2007. `http://www.public.asu.edu/~huanliu/INTERACT/INTERACTsoftware.html`.