Exploiting Web Features for Relevance Feedback

Completed Research Full Papers

Frederico Araújo Durão
Federal University of Bahia
fdurao@ufba.br
Carsten Schjønning
Aalborg University
carsten.schjonning@cs.aau.dk
René Bech Rasmussen
Aalborg University
rene.bech@cs.aau.dk
João Paulo Dias de Almeida
Federal University of Bahia
joao.dias@ufba.br
Paulo Roberto de Souza
Federal University of Bahia
paulo.prsdesouza@ufba.br
Daniel Santos Peixoto
Federal University of Bahia
danielpeixoto@dcc.ufba.br

Abstract

Keyword-search is the prevalent method for finding information on the Web today. However more expressive methods to acknowledge the intriguing characteristics of today's user/content combination on the Web are required. Based on this premise, this paper sets forth to investigate a very expressive method called relevance feedback, on which the user judges the relevance of pages and continuously redirect the search toward the assumed user-preferred pages. This proposal incorporates relevance feedback into a completely functioning search engine to improve the functioning keyword-searching mechanism. We investigate a number of web page features, which could be pursued for ranking pages according to relevance feedback. By applying supervised learning algorithms, we aim at determining how those features should be weighed for the best outcome in ranking and thus engineered for relevance feedback.

Keywords

Information Retrieval, Relevance Feedback, Learning to Rank, Web Features.

Introduction

The Web today can be considered the largest resource for finding information. Information retrieval (IR) methods used on the Web are crucial for the retrieval efficiency experienced by the user. The IR method called *relevance feedback* allows users to judge the relevance of search results (documents, images, videos, etc.) and use this information to improve the final result set. The intuition behind is that users find difficult to formulate an effective query, but it is easy to judge particular search results, resulting in an iterative query refinement.

Relevance feedback can be categorized as explicit or implicit. It is explicit when the user manually provides the feedback to the system (e.g. by marking results). Otherwise, it is implicit when the system obtains this information automatically (e.g. by user clicks). In the literature, several methods for gathering user's feedback have been investigated including the analysis of click-through behavior (Liu Y et al. 2017), query expansion (Dalton J et al. 2014), and re-ranking (Iscen et al. 2018). Despite much effort have been put toward the development of expressive methods, there exist still research opportunities to address the use of learning technologies for relevance feedback.

In this paper we propose a learning to rank method that investigates a number of web page features, which could be pursued for ranking pages according to relevance feedback. The hypothesis is that web page features may serve as key quality elements which can determine whether a page will be better judged or not. The contributions of this paper include i) investigation on how additional features can be engineered for

relevance feedback, ii) incorporate these features into a novel approach combining relevance feedback and learning to rank method, and iii) a proof-of-concept system called Rate and Seek. The following sections present the background of Relevance Feedback, related works, the proposal, evaluation of the approach and conclusion remarks.

Background

In the literature *Relevance Feedback* is historically is divided into three different categories, namely i) explicit, ii) implicit and iii) pseudo feedback (Scholer F et al. 2008). The three categories describe how the feedback received by the system is obtained, and the underlying implementation, leveraging the relevance feedback, will operate based on which category is chosen. Such information will be applied on the relevant documents to either adjust the weights of terms in the original query or to add new terms to the original query. Relevance feedback is often implemented using the Rocchio Algorithm (Salton G and Buckley C. 1988). Besides *Relevance Feedback*, there exist in the literature other indicators such as PageRank (Jilani TA et al. 2015), HITS, document length, number of outgoing/ingoing links (Kumar A and Singh RK. 2017). The problem that arises is how these indicators of relevance, or so-called *features*, should be weighted to give the best overall score by which the documents should be sorted. *Learning to Rank* then arises as a new approach to build a ranking model that is then used to rank documents on top of machine learning techniques (Liu TY. 2009). In a typical learning to rank scenario the ranking model will be a linear weighting of features such that the scoring function is usually expressed as: $score(d) = \sum_{i=0}^{k} f_i . w_i$, where $\vec{f}$ is a *feature vector* representing a document, $\vec{w}$ is a vector containing the weights of all features such that w_i is the weight of feature *i* and *k* is the total number of features. The weight vector is considered universal and will therefore only be learned once and then applied every time a ranking must be produced. This requires two tasks: i) creating a feature vector for each document and ii) finding a learning algorithm that can produce a weight vector. In general, the goal is to find the weight vector that produces the best possible ranking through supervised learning algorithms where labeled training instances are investigated and used to infer a weight vector that optimizes the goal or minimizes the error on the training instances. Two underlying learning to rank algorithms are: i) *Binary Classification* which produces a weighting vector, on which the relevance label can only be "relevant" or "irrelevant", and *Linear Regression* an approach where the relevance labels are taken as real number values, and a scale "very irrelevant", "irrelevant", "relevant", "very relevant" can be applied as -2, -1, 1 and 2.

Related Work

This section discusses a list of important related works in the literature. The Dynamic Feedback Optimization approach (Buckley C and Salton G. 1995) try to improve weights in a dynamic fashion by testing possible changes of query weights on the learning set documents. The resulting optimized query performs up to 15% better than the original when evaluated on the test set. Scholer F et al. (2008) claim that click-through data has shown that only 52% of all clicked pages are considered relevant by the user. Furthermore, the user cannot control the search process completely as the system may infer wrong preferences - leaving the user helpless and feeling stuck. Ayadi MG et al. (2016) propose the integration of a medical Content-based image retrieval (CBIR) system with different stages of the relevance feedback process through an efficient fusion of color, shape, and texture features. Sejal D. et al. (2018) propose an image recommendation framework with user relevance feedback session and visual features to extract relevant images based on user inputs. User feedback is retrieved from image search history with clicked and un-clicked images. Image features are computed off-line and later used to find relevance between images. Boteanu B. et al. (2017) propose a novel pseudo-relevance feedback perspective to social search results diversification. They used an automatic generation of user feedback in an unsupervised scenario based on a hierarchical clustering algorithm and a re-ranking scheme. In this unsupervised scenario, positive and negative examples are used to generate better representations of visual classes of data. Suditu N and Fleuret F. (2016) investigate an innovative query-free retrieval approach. Starting from a heuristic sampling of the collection, this approach relies solely on an iterative relevance feedback mechanism driven by the user's subjective judgments of image similarities. Petronette DCG et al. (2015) use a semi-supervised approach for implementing Relevance Feedback (RF) based search services. It is semi-supervised in the sense that it learns from both labeled and unlabeled data. Basically, the method combines labeled data available along with RF iterations with contextual information provided by unlabeled data.

Combining Relevance Feedback and Learning to Rank

This section depicts all steps from building feature vectors until the live system where the user judges pages causing pages to be re-ranked. In comparison to existing methodologies in the literature, this approach takes into account a number web page features as a measure for the page's overall quality rather than the customary click through analysis Scholer F et al. (2008), Sejal D. et al. (2018). In addition to the document similarity analysis, this approach pays special attention to distinct and relevant features including URL content, format and the size of the page as a pointer for higher quality search results.

Learning Weights from Marking Feedback

With supervised learning, labeled instances are needed for training a model and calculating a weight vector. The labeled instances are simply pages or more specifically feature vectors which relevance to the query (rating) is known. In this way we are able to apply learning to rank on these labeled instances, trying to acquire a weight vector which ranks them as ideal as possible according to their ratings. However, this requires us to acquire some training data. Thus, we have produced queries with matching user judgments for this purpose. Each query with matching judgments is called a *qrel*, and simply consists of a search query and the matching set of search results with approximately 30 of the pages judged. The goal is to imitate a real feedback scenario where some of the pages are given as feedback and other pages are re-ranked according to this feedback, different combinations of judged pages can be marked as a relevance feedback.

Building Feature Vectors

As in the live system, when relevance feedback is given, the feature vectors are constructed for the pages to re-rank. However, in the learning process only pages which are judged can be used since supervised learning requires labeled instances both for training and evaluation. The pages chosen for relevance feedback are discarded as they could skew the training and evaluation. This results in a list of rated/labeled feature vectors for the pages to re-rank, which, with each *qrel*, can work as a set of training instances or test set for the learning of weights for the best ranking. There exist however two types of features, which can be added to a feature vector for comparison: *dependent* and the *independent* ones.

Query-Independent Features

Query-independent features represent a page's overall quality or relevance to any given query, and hence have the same independent values for every query. The considered query-independent features are:

- **StaticRating**. It represents the value for the feature *StaticRating* in the feature vector for page p while I is the position of a page in the initial ranking returned by *Rate and Seek* (see Equation 1). The initial ranking of search results is likely to be a relevant descriptor of a page in the process of re-ranking. By transforming the initial ranking to a feature for a page, it should be possible to weigh the initial position of the page and hence cause the re-ranking of pages to be based upon the initial ranking. Consequently, increasing the weighing of this feature decreases the effect the relevance has on the initial ranking returned by the search engine.

$$\vec{f}_{p,staticRating} = \frac{max_i - I_p}{max_i} \quad (1)$$

- **Age.** The age of a page means the date it was modified. This feature can be relevant for finding an appropriate page, as shown in Equation 2.

$$\vec{f}_{p,Age} = RetrievalDate_p - AlterationDate_p \quad (2)$$

- **PageRank.** The PageRank for each page reflects the overall popularity of a page seen by its link structure.
- **DomainPageRank.** By only considering the domain part of a page's URL, the PageRank is fetched, representing the link popularity of the domain.

- **AverageTF.** The average number of each term in the visible text on the page. This represents how diversified the use of terms is, as shown in Equation 3.

$$\vec{f}_{p,AverageTF} = \frac{NonuniqueTerms_p}{UniqueTerms_p} \tag{3}$$

- **MaxTF.** The maximum use of a single term in the visible text on the page.
- **UniqueTerms.** The number of different terms on the page.
- **Size.** The size of the page in bytes, including the HTML and text but not images.
- **SlashCount.** The number of "/" in the URL can be considered an indication whether the page is a subpage of a domain potentially located in several levels of subfolders.
- **URLLength.** The length of the URL that can be considered a value somewhat similar to *SlashCount* which is likely to increase with the length of the URL and vice versa.
- **NoIngoingURLs.** The number of ingoing URLs. Before such a measure was also used as a query-independent feature for acknowledging that the more pages linking a page, the more popular the page.
- **NoOutgoingURLs.** The number of outgoing URLs. Extensive use of outgoing links can be considered obscuring the user experience and a weighing negatively against such pages, could be considered.

Query-Dependent Features

Query-dependent features are features which values represent how close the appertaining page is to the query-typically by means of similarity.

The Text Feature

In a typical document scoring scenario, a query-vector containing the searched terms is compared (performing the dot-product) against each page to rank; scoring the pages according to their relevance to the query. In this approach, the query-vector incorporates the relevance feedback and the calculation relies on a modification to the Rocchio-algorithm, called Ide-Regular, which removes the normalization (Rocchio JJ. 1971):

$$\vec{q}_{opt} = \sum_{\vec{d} \in R} \vec{d} - \sum_{\vec{d} \in I} \vec{d} \tag{4}$$

Without the normalization, the influence of relevant and irrelevant pages differs with the number of relevant pages compared to irrelevant pages. In Equation 4 there are no scales of relevance feedback, i.e. "relevant" vs. "irrelevant". Hence, to acknowledge the 4-scaled relevance feedback, the rating, λ, is added as a factor for scaling the term importance according to the rating (Equation 5):

$$\vec{q} = \sum_{\vec{d} \in R} \vec{d}\,\lambda_d + \sum_{\vec{d} \in I} \vec{d}\,\lambda_d \tag{5}$$

The ratings for the 4-scaled relevance feedback are represented as -2, -1, 1, 2 for "very irrelevant", "irrelevant", "relevant" and "very relevant" accordingly.

Link Structure

With relevance feedback, we are again interested in the similarity between the pages given as relevance feedback and a certain page, and hence we focus on the similarity between two pages which can be derived from their common link structure. For this purpose, we have the Amsler similarity measure (Amsler R. 1972), which counts the link relations that two pages have in common, as follows:

$$ams(a,b) = \frac{((I_a \cup O_a) \cap (I_a \cap O_a))}{((I_a \cup O_a) \cup (I_a \cap O_a))} \quad (6)$$

Where a and b are distinct pages, I_x is the set of ingoing links for a page x and O_x is the set of outgoing links for a page x. According to Equation 6, the count of the 4 relations are calculated in the numerator and then normalized by the number of in- and out-going links in a and b, to ensure that more links are not favored in giving a potential increased hit in relations.

Making Query-Independent Features Query-Dependent

We consider a single query-independent feature from which we want to derive a query-dependent feature using relevance feedback. The query-independent feature for the pages up for re-ranking must be compared to the same feature for each page given as relevance feedback, resulting in the query-dependent feature. Thus, the query-dependent feature for a page can be calculated as:

$$\vec{f}_{i,x} = \frac{\sum_{j \in J} j - (P_{i,y} - P_{i,y})}{(J)} \quad (7)$$

Where $\vec{f}_{i,x}$ is the feature value for a feature x in page i, J is the set of pages given as relevance feedback and λ is the rating -2, 1, 1 or 2 of a page j. The numbers generated for the different features are of very different ranges. Scaling a feature by some value actually influences the learning and hence certain normalization should be performed. The normalization of the query-independent features is performed in different ways, depending on the feature, with the goal of scaling the values so they are between 0 and 1. This undertakes 3 steps: i) The feature values are capped to a max-value, so that very high values in the range do not diminish the influence of smaller values. ii) The values are converted to a logarithmic scale by using $log_{logbase}$ for every value. When using a $logbase > 1$, this will again cause high values to be less dominating compared to lower values, and iii) to normalize, the values are simply divided by max-value used for capping; giving a range between 0 and 1. The resulting equation for pre-processing and normalizing the values is:

$$Process(x, logbase, max) = \frac{\log_{logbase} \min(x, max)}{\log_{logbase} max} \quad (8)$$

As a result, 42 proposed features are candidates to be used for re-ranking with relevance feedback.

Rate and Seek: The Proof-of-Concept System

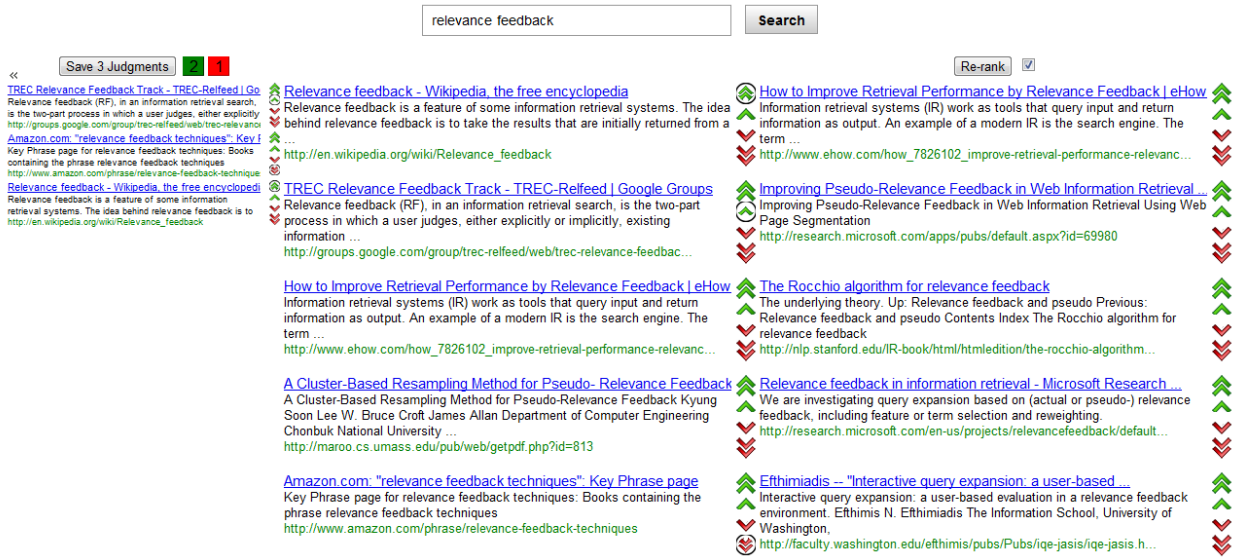

Figure 1: The design and columns in Rate and Seek.

We implemented a proof-of-concept system, called *Rate and Seek* to create a realistic setting for performing relevance feedback by collecting training data from users to support the creating and tuning of algorithms. The system is both used for illustrating how the relevance feedback performs but also to collect training data from users. Hence the system can be seen as a bootstrapping method, where the system is used for creating itself. The implementation used is SVM-light (Joachims T. 2009), which contains both binary classification, regression and a ranking approach. As shown in Figure 1, from left to right, the interface is divided into 3 columns under the search bar: *Judged, Initial*, *Re-ranked*, by which *Judged* represents the pages which have been judged, *Initial* is the initial returned list of search results from the search engine and *Re-ranked* is the re-ranked list of search results. *Judged* keeps a list of judged pages, updating continuously as the user judges page from either column. When a page's rating is updated, every column containing the page is updated with the correct rating and the judged column adds or removes the page if it changes from being unjudged to judged or vice versa. This instantaneous updating of pages causes every judged page to be in the judged column and the same page to be in the same state no matter what column it appears in. While both *Initial* and *Re-ranked* columns are identical in the design, the *Judged* column only contains one page of pages and when it grows it will just expand vertically. The column *Judged* can be seen as unnecessary, since the judged pages will remain in the *Initial* column and hence there is a possibility to collapse the column for additional space. However, *Judged* can be relevant by maintaining an overview of currently judged pages with the possibility of changing the rating.

Experimental Evaluation

This section evaluates the proposal in a wide range of aspects, including finding a learning to rank method, feature subset selection and tests exploring if different weight vectors are needed.

Data Collection

The data used for learning weights and in the evaluation was collected using the Rate and Seek system, where more than 13 different users have performed 31 usable queries with associated judgments. The participants were asked to do a query of their own interests, and then judge at least 30 results. Table 1 shows some statistics concerning the remaining data. Note that there is a wide variety in the judgments; ranging from 20 to 235. No *qrel* got less than 20 judgments and have at least 10 negative and 6 positive judgments.

	Min	**Max**	**Average**
Pages	187	477	344.55
• Unjudged	133	451	304.71
• Judged	20	235	39.84
▪ Negative	10	179	26.55
▪ Very Irrelevant	1	146	14.45
▪ Irrelevant	1	33	12.10
◦ Positive	6	56	13.29
▪ Relevant	2	38	7.45
▪ Very Relevant	0	18	5.84

Table 1. Statistics of collected data.

Evaluation Measure

To evaluate rankings, we use DCG equation in a normalized form:

$$Normalized - DCG(x) = \frac{DCG(x) - DCG(worst)}{eval(ideal) - DCG(worst)}$$

, where *x* is the current ranking, *ideal* is the best possible ranking and *worst* is the worst possible ranking. This normalized form makes the DCG values from different *qrels* more comparable.

Marking Feedback

Finding pages that should be used as relevance feedback in the feature vector construction is not obvious. Hence, we choose to take the realistic approach of marking pages from the top of the initial ranking as feedback. For this reason, we use the *MarkEachFirstJudged* algorithm to mark pages as feedback. *MarkEach* refers to the fact that it considers positive and negative pages separately, such that it can be specified that 3 positives and 2 negatives should be marked as feedback. *FirstJudged* means that the pages marked are found from the top of the list. The used feedback scenarios of *MarkEachFirstJudged* performed with 0–4 positive or negative; yielding 24 combinations as the case with 0 positive and 0 negative is ignored.

Learning to Rank Methods and Rating Representation

The learning to rank method used in our experiments, called Ranking SVM (Joachims T. 2009), is strongly competitive with other ranking methods and has proven itself throughout many different uses as a very solid competitor (Liu TY. 2009). However, to fully exploit every learning method to its fullest, giving a fair comparison, we must consider some parameters. One parameter, which is deemed to change using the different methods, is the representation of the class-variable, i.e. the rating of a page. Using binary classification, the only valid values are -1 for pages judged irrelevant and +1 for relevant pages. This will remove some available ranking information; showing the effect of only binary classification and the potential loss in ranking effectiveness. Particularly, the learning to rank method used in our experiments (Ranking SVM) is strongly competitive with other ranking methods and has at the same time proven itself throughout many different uses as a very solid competitor (Liu TY. 2009). The class variable in Ranking SVM is also transformed inside the algorithm to a pairwise relevance ordering of pages. However, the entire scale of ratings is maintained when performing this transformation and causing the initial rating representation to be of less importance, as the algorithm only considers the relative ordering and not the actual rating-values. The support vector regression used is the only method which potentially changes effectiveness with different representation of rating-values. In regression, the algorithm tries to produce weights which approximate the closest to the rating-value as possible. Hence, it is possible to change the interpreted distance between two different rating-values by simply increasing the difference in their numeric values. For the actual test we use two different rating-representations, namely linear and quadratic, which simply is the rating 4-scale represented as {-2, -1, 1, 2} and {-4, -1, 1, 4} respectively. Another parameter is C, which can be considered the trade-off between the training error and the margin maximization performed by a SVM. In the current evaluation, we use the following C values for the different learning methods: i) Classification (Default, 100), ii) Regression (Default, 100) and iii) Ranking (N, 1, 100), where *default* is the C value chosen by SVM-light and *N* is the number of training instances. In order to get an overall view of the learning method's performance, we use several feedback scenarios and average upon their measured ranking quality (given by the assessment-value). The set of feedback scenarios are that of *MarkEachFirstJudged*. For ensuring a solid measure, which is not dependent on the choice between train and test set, we use cross-validation with 8 folds. Last but not least, we use 2 different feature sets for learning, one only weighing *Text* and *StaticRating*, the other weighing every feature we have constructed.

Feature Subset Selection

It was created and tested on a large set of features to ensure that we capture as many possible properties as possible, which can help determine the similarity between pages and a user's preferences; consequently, causing a better re-ranking of pages. In an ideal world, the learning to rank and hence the learning of weights should include an implicit feature subset selection in weighing expressive features high and non-expressive features low. However, the classification algorithms cannot be expected to completely remove the influence of features which worsen the results. This can be seen in Figure 2 where using a large set of features actually worsen the re-ranking compared to only using two. Thus, it will be beneficial to investigate which features are actually good descriptors for the learning process and subsequently the re-ranking, and

which are not - inherently finding a feature subset which serves as an overall good performer in any possible scenario.

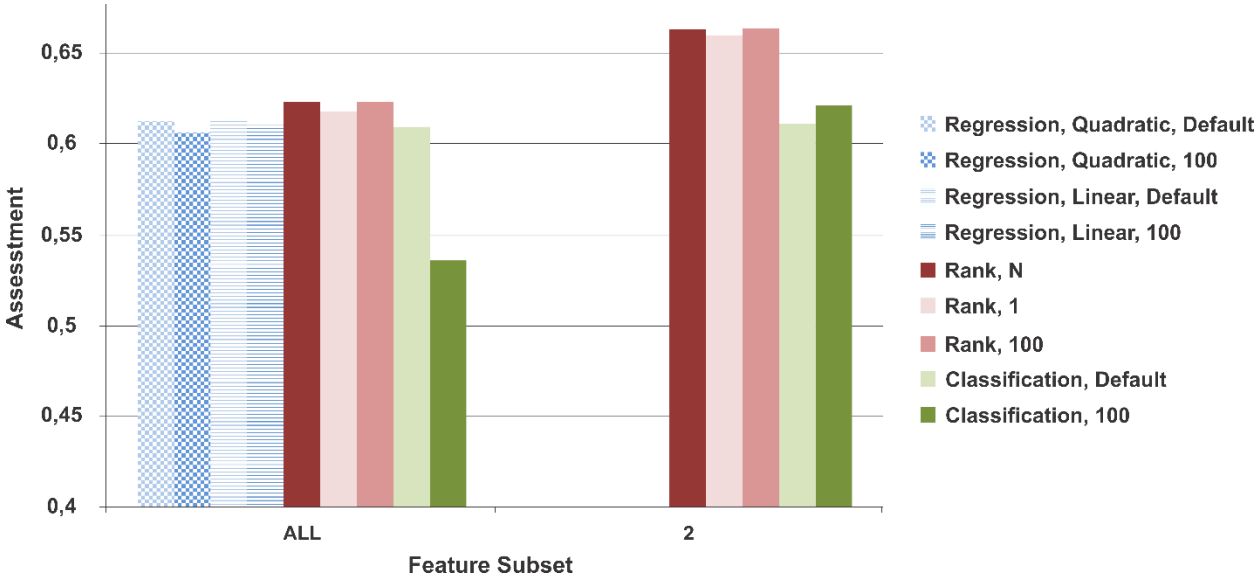

Figure 2: Performance of 3 different learning methods using different parameters.

Finding the Most Popular Features

The possible number of feature subsets is 2^N, with *N* being the number of wanted features. Thus, with 42 possible features it would be infeasible to test every subset's performance.

Feature	Frequency
Text	>60%
StaticRating	50-60%
Title	30-40%
EquallyLinkedDomain, NoOutgoingURLsQD	20-30%
DirectlyLinkedDomain, ViaLinked, AllLinks, StaticBingRatingQD, Snippet, DirectlyLinkingDomain, Description, EquallyLinked, DirectlyLinked,DomainPageRankQD, SlashCount, UniqueTermsQD, AverageTF, DomainPageRank, NoIngoingURLsQD	10-20%
AverageTF, DomainPageRank, NoIngoingURLs, ViaLinkingDomain, MaxTFQD, Keywords, UrlLengthQD, AverageTFQD, EquallyLinkingDomain, ViaLinkedDomain, Age, AgeQD, UrlLength, PageRank, NoOutgoingURLs, Size, ViaLinking, SlashCountQD, SizeQD, DirectlyLinking, MaxTF	<10%

Table 2. All features ranked according to their frequency.

The best feature subset is very likely to change with the training set and hence one should not rely on only a single training set scenario for finding the appropriate feature subset. To help find the best features we use a wrapper method for feature selection. It starts with the empty set and simply uses a greedy approach, selecting the feature which causes the highest assessment, one at a time, until the assessment stops increasing (Kohavi R and John GH. 1997). Consequently, only a small fraction of possible subsets is tested, but with the assumption that the found subset is a good approximation of the optimal. Since we want to find only one feature subset, which performs well on all feedback scenarios, we run the wrapper method on every feedback scenario caused by *MarkEachFirstJudged*, resulting in many different feature subsets. For each feedback scenario, it was used the Holdout-method using 70% as training data and 30% as test data.

Then, we perform each of the 24 feedback scenarios in *MarkEachFirstJudged*, a total of 6 times (each time with a new random training/test data separation) giving us 144 feature subsets. We therefore argue that by ranking the features according to their frequency in the feature subsets, we can derive which features are most likely to be good performers for an overall feedback scenario. Table 2 shows the feature frequencies. As expected, the two most frequent features are *Text* and *StaticRating*, as the most obvious descriptors for a page. However, we have set forth to investigate other features influence and even though of less importance they may still improve the re-ranking quality.

Interestingly questionable features actually seem to give better results than promising ones. For instance, one would think that *DirectlyLinked* would be a very good feature for deriving a page's similarity and hence relevance with the given feedback. However, only very few pages in a training set actually have a value different than 0 in DirectlyLinked, so this feature is not populated enough and hence not represented enough to use for re-ranking. This means that features, like *EquallyLinkedDomain*, which intuitively are less connected with similarity between the pages, actually seem to be a better feature to use. The same goes for *PageRank* and *DomainPageRank*, where the PageRank of too many pages simply is 0. Other less frequent features such as *MaxTF*/*MaxTFQD*, *Size*/*SizeQD,* and *URLLength*/*URLLengthQD*, one would simply argue that they are poor at deriving the similarity and distinguishing relevant from irrelevant pages.

Increasing the Number of Features

We also observe that combining the best features (most frequent in the features subsets), we are most likely to receive a better assessment. We then evaluate the performance of feature subsets containing the *x* most frequent features in the previous test. This means that the feature subset with two features would contain *Text* and *StaticRating* and so forth with a different number of features. The test is performed with 8-fold cross-validation and averaging over the assessments caused by the different feedback scenarios given by *MarkEachFirstJudged*.

Number of Features	**0**	**1**	**2**	**3-5**	**6-9**	**10 - 19**	**20 - 24**	**25 - 29**	**30 - 34**	**35 - 39**	**40 - 44**
Average Assessment	0.55	0.62	0.66	0.66	0.67	0.67	0.66	0.65	0.64	0.63	0.62

Table 3. The assessment with different number of features selected.

Table 3 shows that there is a fine line between *"enough features"* and *"too many"*. As the number of features increases beyond 20, the performance begins to decrease with the number of features; while the increase in features until about 10, the performance increases. Having 0 (zero) features represents the situation of not re-ranking and hence how the initial ranking is assessed. The reason for the incline and declines in performance in the graph can have several explanations. Of course, the fact that we choose the best features first is a logical explanation for the rapid increase in the beginning and the poor performance when bad features are added in the end. However, we would also believe that the first few added features are more likely to increase the performance in general, since they have a larger effect on the re-ranking. Similarly, many features are likely to overlap in their expressiveness, and the feature with the somewhat the same expressiveness, which is added the latest only express the same as the already added feature, and does inherently not increase the re-ranking performance further.

Discussion

By adding new features, we are faced with the challenge of balancing them for the best re-ranking performance according to the relevance feedback. In the search for the best balance, it was used both *feature subset selection* and *learning to rank* for discovering the best combination of features and their weighting. Which learning to rank method to use is absolutely not an easy choice, since there exist many different approaches, all with advantages and disadvantages. Even though we observe that Ranking SVM outperforms the simple methods as linear regression and binary classification, we also realize that the feature's individual and combined expressiveness lead to a better re-ranking. The prevalent features are dominated by the text similarity measures - all being query-dependent except *StaticRating*. Despite the promising results, however, by introducing too many features or bad features the results can diverge away from the optimal ranking instead of converging toward it.

Conclusion and Future Works

This paper investigates the method called *relevance feedback*, on which a user judges the relevance of pages and continuously redirect the search toward the assumed user-preferred pages. Several web page features are observed, as they can be pursued for ranking pages according to relevance feedback. For the experiments, it was created a completely functioning search engine powered by relevance. Results from experiments indicate that there indeed is potential in using relevance feedback and proposing additional measures for optimizing such. While the largest gain is going from no relevance feedback to using the text feature, there is gain using up to seven features. As a future work, we aim at making a thorough investigation of all possible features and settings with the created testing framework and knowledge.

REFERENCES

Amsler, R. 1972. Tech. rep., The University of Texas at Austin, Linguistics Research Center, Austin, TX.

Ayadi, MG., Bouslimi, R., Akaichi, J. 2016. "A medical image retrieval scheme with relevance feedback through a medical social network," *Social Network Analysis and Mining* (6:1) pp. 53

Boteanu, B., Mironica, I., Ionescu, B. 2017. "Pseudo-relevance feedback diversification of social image retrieval results," *Multimedia Tools and Applications* (76:9) pp. 11,889–11,916

Buckley, C., Salton, G. 1995. *Optimization of relevance feedback weights* pp. 351–357

Büttcher, S., Clarke, C., Cormack, GV. 2010 "Information Retrieval: Implementing and Evaluating Search Engines," *The MIT Press*.

Dalton, J., Dietz, L., Allan, J. 2014. "Entity query feature expansion using knowledge base links," in Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval, ACM, pp. 365–374.

Iscen, A., Avrithis, Y., Tolias, G., Furon T., Chum O. 2018. "Fast Spectral Ranking for Similarity Search" in The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7632-7641

Jilani, TA., Fatima, U., Baig, MM., Mahmood, S. 2015. "A survey and comparative study of different page rank algorithms," *International Journal of Computer Applications (*120:24)

Joachims, T. 2009. "Svm-rank: Support vector machine for ranking," Cornell University

Joachims, T., Granka, L., Pan, B., Hembrooke, H., Gay, G. 2017. "Accurately interpreting clickthrough data as implicit feedback," in *ACM SIGIR Forum*, ACM, vol. 51, pp. 4–11

Kohavi, R., John, GH. 1997. "Wrappers for feature subset selection," *Artificial intelligence* (97:1-2) pp. 273–324.

Kumar, A., Singh, RK. 2017. "A study on web structure mining," *International Research Journal of Engineering and Technology* (4:1).

Liu, TY. 2009. *Learning to rank for information retrieval Foundations and Trends in Information Retrieval* (3:3):225–331.

Liu, Y., Nie, JY., Chang, Y. 2017. "Constructing click models for search users," *Information Retrieval Journal* (20:1) pp. 1–3.

Pedronette, DCG., Calumby, RT., Torres, RdS. 2015. "A semi-supervised learning algorithm for relevance feedback and collaborative image retrieval," *EURASIP Journalon Image and Video Processing* 2015 (1:27).

Rocchio, JJ. 1971. "Relevance feedback in information retrieval," *The SMART retrieval system: experiments in automatic document processing,* pp. 313–323.

Salton, G., Buckley, C. 1988. "Term-Weighting Approaches in Automatic Text Retrieval," *IP&M* (24:5) pp. 513–523.

Scholer, F., Shokouhi, M., Billerbeck, B., Turpin, A. 2008. *Using clicks as implicit judgments: expectations versus observations* pp. 28–39.

Sejal, D., Abhishek, D., Venugopal, K., Iyengar, S., Patnaik, L. 2016. "Ir_urfs_vf: image recommendation with user relevance feedback session and visual features in vertical image search," *International Journal of Multimedia Information Retrieval* (5:4) pp. 255–264.

Suditu, N., Fleuret, F. 2016. "Adaptive relevance feedback for large-scale image retrieval," *Multimedia Tools and Applications* (75:12) pp. 6777–6807.