An Analysis of Factors Used in Search Engine Ranking

Albert Bifet
Technical University of Catalonia
abifet@lsi.upc.es

Carlos Castillo
University of Chile
ccastillo@dcc.uchile.cl

Paul-Alexandru Chirita
L3S Research Center
chirita@l3s.de

Ingmar Weber
Max-Planck-Institute for Computer Science
iweber@mpi-sb.mpg.de

April 8, 2005

Abstract

This paper investigates the influence of different page features on the ranking of search engine results. We use Google (via its API) as our testbed and analyze the result rankings for several queries of different categories using statistical methods. We reformulate the problem of learning the underlying, hidden scores as a binary classification problem. To this problem we then apply both linear and non-linear methods. In all cases, we split the data into a training set and a test set to obtain a meaningful, unbiased estimator for the quality of our predictor. Although our results clearly show that the scoring function cannot be approximated well using only the observed features, we do obtain many interesting insights along the way and discuss ways of obtaining a better estimate and main limitations in trying to do so.

1 Introduction

In the age of digitalized information the world is relying more and more on web search engines when it comes to satisfying an information need. When given a new task, 88% of the time internet users start at such a search engine [11]. Part of the search engines' success might be due to their simplicity: you enter some words and the results are then output in form of a ranked list, in which the search engine estimates the relevance of each indexed website to your query.

Today Google claims to have indexed 8,058,044,651 web pages[1]. Even with this sheer enormous volume of information it is relatively "easy" to find a list of pages containing given query terms. The difficult part is then to select, from the possible myriad of matching pages, the "best" 10 or 20 according to some computable quality measure which, ideally, closely resembles the user's notion of relevance. The ability of any search engine to closely match this human notion has a major impact on its success.

Users are usually satisfied when the presented ranking leads to an answer to their queries. However, they normally do not question *why* they were presented with exactly this ranking, and how the search engine inferred which website is most relevant to their particular request. This situation is usually different when querying for their own name(s), as one might suddenly wonder why her personal homepage is (not) ranked highest.

E-commerce websites, on the other hand, have a much more tangible interest in the exact ranking process, as a high ranking in a popular search engine for a certain product query leads to more web traffic and ultimately translates into higher sales. Thus, "Search Engine Optimization" (SEO) has long been recognized as a lucrative business on its own.

Although the exact details are not publicly known, it is generally assumed that each search engine assigns a score to each result that satisfies a Boolean search criteria and then sorts the results according to this score. Its

[1] Interestingly, this number used to remain just below 2^{32} for at least 9 months, indicating a 32-bit hardware threshold of the previous implementation. Just on November 11^{th}, 2004 the index size jumped to about 8 thousand million pages, possibly in response to the launch of the MSN search engine.

value depends on the value of certain *features* of each webpage in the result set, e.g. its PageRank score, the text similarity between the query and the document, etc.

In this paper we approximate the underlying ranking function by analyzing query results. First, we obtain for each query result numerical values for a large number of observable features, thus converting each document into a vector. We then train our models on the difference vectors between documents at different ranks. By labeling these vectors with "+" for the direction towards the top and "–" otherwise, we reformulate our problem as a binary classification one: given a pair of documents, we try to predict which one is ranked above the other. This gives a partial order of the vectors, i.e., documents. Therefore, the binary classification trees we used do not give a complete order, i.e., a full ranking. On the other hand, for the models with linear decision boundaries, namely logistic regression and support vector machines, the normal vector to this linear boundary gives the trendline of the direction of an improvement in the underlying scores and the scalar product with this vector gives a full ranking. Although our methodology can be applied to any search engine, we chose Google as it is nowadays the most widely used [3]. More specifically, we used the Google API [5] to retrieve search results.

In section 2 we discuss previous work on this subject. We then describe our statistical methodology for estimating the ranking function in section 3. Section 4 briefly presents the components of the system. Section 5 gives our experimental results. Possible improvements and shortcomings of our approach are discussed in section 6.

2 Related Work

Pringle, Allison and Dowe [13] addressed both the problem of determining when certain pages are retrieved at all, and that of explaining how a given ranking was obtained. For the first problem they used decision trees, while the second was tackled using linear regression, exploiting the explicit scores returned by InfoSeek [8]. Both of these approaches are different from ours as is the way they obtained their data, namely by creating artificial websites and not using real web pages as we are. Furthermore, they did not use a separate test data set to obtain an unbiased estimator of the quality of their predictor.

Sedigh and Roudaki [15] presented a simple linear regression model that approximates the dynamics governing the behavior of Google where, given some observable features, they try to predict the *absolute* rank of a webpage. This has, among other things, the disadvantage that once documents are added such a prediction can no longer hold, as you trained on absolute ranks, which depend on the knowledge of all documents in the set. Our approach has the advantage that the universe of documents does not need to be known. Especially for their methodology it would have been interesting to see the performance on a separate test data set that was not used for training.

Joachims [9] presented an approach to automatically optimize the retrieval quality of search engines using clickthrough data. He rephrased the problem of learning a linear ranking as one of binary classification with linear decision boundaries and then applied Support Vector Machines (SVM) to this inference problem. We use the same reformulation, though we do not restrict ourselves to SVMs.

There is also a substantial amount of literature on various theoretical aspects of learning a ranking function, see, e.g., [2, 4]. As our focus was on obtaining an *interpretable* model for the ranking function, we limited ourselves to using logistic regression, SVMs and binary classification trees.

3 Estimating a Ranking Function

Our goal is to obtain an estimation function f for the scoring function of a search engine, and then to compare our predicted rankings with the actual rankings of this search engine.

3.1 Data set

As it is unlikely that a large search engine would employ the same ranking criterion for all types of queries, we used several sets of homogeneous queries, all dealing with a certain topic, assuming that they are ranked with the same function.

Table 1 shows our queries dataset, which is divided into four categories: Art, States, Spam and Multiple. Arts is a list of artists and States is a list of US States. For both of these categories the queries consisted of a single term. The Spam category contains phrases for which one can expect many "search engine optimized" web pages. If a search engine wants to retrieve any high quality pages for such queries it must use an elaborate

Table 1: Table of training, validation and test data classified by the query data set. Validation data is used for feature selection and tree pruning. The test data is only used to get an unbiased estimate of the generalization error.

Dataset	Type	Query terms
Arts	Training	Albertinelli, Bacchiacca, Botticelli, Botticini, Foschi, Franciabigio, Leonardo
	Validation	Michelangelo, Picasso
	Test	Pordenone, Rosselli, Verrocchio
States	Training	Arizona, Arkansas, Connecticut, Idaho, Illinois, Iowa, Kansas
	Validation	Michigan, Nevada
	Test	Ohio, Oregon, Utah
Spam	Training	buy cds, buy dvds, cheap software, download movies, download music, dvd player, free movies
	Validation	free mp3, free music
	Test	free ringtones, music videos, software downloads
Multiple	Training	anova bootstrap feature missing principal squared, analysis frequent likelihood misclassification pruning statistical, adaptive classification gating linear model proximity subset, association generalization local naive regularization test, analysis cubic gradient margin optimal risk support, automatic decision validation overfitting smoother adaptive, activation discriminant hyperplane loss single validation
	Validation	basis eigenvalue margin maximum local support, basis early information adaboost margin soft
	Test	logistic bias entropy markov piecewise loss, feature complexity gaussian logistic normal ridge training, discriminant gaussian link multiple radial supervised

ranking function. Finally, the Multiple category consists of queries with 6 terms from the domain of statistical inference.

One of our main contributions lies in the statistical rigor with which we approach the problem. Thus, for example, to obtain statistically meaningful results we partition the 12 queries for a given category into three disjoint sets:

Training Set (7 queries): We use this set of queries to learn a linear scoring function or a decision tree. The data that we actually trained on consists of difference vectors corresponding to different feature vectors. See section 3.2 for a detailed explanation.

Validation Set (2 queries): The validation set is used for greedy stepwise forward feature selection and for pruning the decision tree. In cases where features were selected directly, this set was merged with the training one. Tree pruning is explained in section 3.5.

Test Set (3 queries): To estimate the generalization error of our ranking function, we compare our predicted rankings with the observed rankings for these queries. Simply testing the performance on the training set itself leads to an overly optimistic estimate of the quality of the model. Without the use of such a set, we could have obtained wrong, but seemingly better, results due to overfitting.

3.2 Reformulation as binary classification problem

Let q be a query, and $\mathbf{u}, \mathbf{v}$ be feature vector of pages, where each entry in the vector corresponds to a particular feature, e.g., how often a query term appears on the page or whether or not it appears in the URL (the list of features we considered is provided in section 4.2).

Let $\mathbf{u} <_q \mathbf{v}$ represent the ordering returned by the ranking function of a search engine for the given query, so $\mathbf{u} <_q \mathbf{v}$ if and only if the page with feature vector $\mathbf{u}$ is ranked below the one with feature vector $\mathbf{v}$ in the search engine's results to query q. In the following, we drop the subscript q.

Ultimately, we want to find a real-valued function f such that $f(\mathbf{u}) < f(\mathbf{v})$ whenever $\mathbf{u} < \mathbf{v}$. If we assume that f is linear, then there exists a vector $\mathbf{w}$ such that $f(\mathbf{u}) = \mathbf{w} \cdot \mathbf{u}$, using $\cdot$ to denote the scalar product.

Then,

$$\begin{aligned} & f(\mathbf{u}) < f(\mathbf{v}) \\ \Leftrightarrow \quad & \mathbf{w} \cdot \mathbf{u} < \mathbf{w} \cdot \mathbf{v} \\ \Leftrightarrow \quad & \mathbf{w} \cdot (\mathbf{v} - \mathbf{u}) > 0 \ . \end{aligned}$$

So the problem of finding a linear function f is equivalent to finding a vector $\mathbf{w}$ such that $(\mathbf{v}-\mathbf{u})\cdot\mathbf{w} > 0$ if and only if $\mathbf{v}$ is ranked above $\mathbf{u}$. Geometrically speaking, $\mathbf{w}$ points in the direction of the increase in score value and the vectors are sorted according to the length of their projection onto this direction. If we label the difference vector $(\mathbf{v}-\mathbf{u})$ "x" if $\mathbf{v}$ is ranked above $\mathbf{u}$ and "–" otherwise we see that our problem is equivalent to finding a hyperplane with normal $\mathbf{w}$ which perfectly separates the "+" and the "–" training points.

Conversely, if we have a binary classification scheme which uses a linear decision boundary through the origin, we can use the normal vector to this plane as our $\mathbf{w}$ which defines the function f. As our data is symmetric about the origin by construction –because each comparison between two different ranks leads to two vectors pointing in opposite directions– many schemes will satisfy the property. Note that the real data will not perfectly follow a linear function in the observed features as we might not know all the underlying features which are actually used and because the function is most probably non-linear. Thus, our classification scheme needs to cope with misclassified points.

In the following we briefly introduce two classification schemes with linear decision boundaries, logistic regression models and support vector machines, and also discuss the use of binary classification trees, which are highly non-linear. See [6] for a good introduction to the area of statistical learning algorithms. We will write $\mathbf{x}$ for an either labeled or unlabeled difference vector $\mathbf{v}-\mathbf{u}$ and consider the task of classifying unlabeled points into one of the two classes "+" or "–".

It is worth noting that we not only train these linear models on the original features but we also experimented with including quadratic terms of features which, in the original feature space, leading to non-linear decision boundaries which can capture more subtle relations but are also more prone to overfitting. See section 3.7 for details on the issue of feature selection and transformations.

3.3 Logistic Regression

Logistic regression models the posterior probabilities of the classes, i.e., the probabilities of belonging to a certain class given the coordinates $\mathbf{x}$, via linear functions in the coordinate vector. In the case of only two classes "–" and "+" the model has the form

$$\log \frac{P(\text{class = "+"}|X = x)}{P(\text{class = "–"}|X = x)} = \beta_0 + \mathbf{w}\cdot\mathbf{x} \qquad (1)$$

As it can be seen easily, thisgives rise to linear decision boundaries, on which $P(\text{class = "+"}|X = x) = P(\text{class = "–"}|X = x)$. Due to the symmetry in our data the offset β_0 will be zero and the vector $\mathbf{w}$ gives the desired weights of the factors involved, indicating the direction from low ranks to high ranks. We refer the reader to [6] for a full description of the training algorithm. For our experiments we used Matlab's implementation of `glmfit` (generalized linear model) with a binomial distribution to compute the logistic regression models.

3.4 Support Vector Machines

Support Vector Machines typically use linear decision boundaries in a *transformed* feature space. The idea is that often in this higher, or even infinitely, dimensional space the data becomes separable by hyperplanes. As the mapping does not need to be computed implicitly, but only inner products defining the kernel need to be known, this approach becomes feasible. However, as we want to be able to ultimately use the related normal vector for ranking in the original feature space we only use support vector machines with linear kernels which correspond to hyperplanes in this space. For our experiments we used the `SMV light` [10] implementation with the default setting for the parameter C which is the average of $(\mathbf{x}\cdot\mathbf{x})^{-1}$. This parameter allows trading-off margin size against training error.

3.5 Binary Classification Trees

The function f that is used in practice by a real search engine might not be linear for a variety of reasons. One is that, for efficiency reasons, a search engine might use several layers of indices, where the first layer is able to answer most queries and only if this level fails the query is send to subsequent levels. Such a behavior and other simple non-linear behaviors, as using thresholds for certain features, can at least partly be captured by decision trees.

A classification tree is built through a process known as binary recursive partitioning. The algorithm iteratively breaks up the records into two parts, examining one variable at a time and splitting the records on the basis of a dividing line in that variable. The process continues until no more useful splits can be found. Toward the end, idiosyncrasies of training records at a particular node display patterns that are peculiar only to those records. These patterns can become meaningless and often harmful for predicting unknown labels. Pruning the tree is the process of removing such leaves and branches to shrink the tree to a smaller set of crucial splits which leads to a better performance on general data. For our experiments we used Matlab's implementation of `treefit` and `treeprune`.

3.6 Selecting Difference Vectors

In most cases we trained on *all* possible pairs, which for n results are $n(n-1)$ pairs, each pair resulting in two difference vectors with opposite labels. This selection of pairs was used to train both the logistic regression models and the SVM classifiers. To reduce the computation time for the classification trees we only trained on "shifted pairs" of the form (1,$\lfloor n/2 \rfloor + 1$), (2, $\lfloor n/2 \rfloor + 2$), ... ($\lfloor n/2 \rfloor$,n), where again each such pair gives rise to two difference vectors. For this subset of pairs there are thus $2 \times \lceil n/2 \rceil$ points to classify. Due to the large constant difference between considered ranks these pairs are expected to be rather robust with respect to noise due to only minor differences between consecutive ranks. However, training the linear models with this subset made hardly any difference compared to training them on all pairs. So we conjecture that more pairs would not give significantly different classification trees.

3.7 Feature Selection and Transformations

The importance of the individual features for each of the four categories is given in Table 2. This table gives the precision which can be obtained on the entire data set (for each category, downloading the top 100 results per query) if we only ranked according to a single feature.

We experimented with various strategies of selecting subsets and functions of the original features as input values for the data mining algorithms. Subsets of features were selected in two ways. Firstly, in a greedy stepwise forward manner, always adding the feature which gave the best improvement on the validation data set. This strategy did not give better results than the other approaches, and thus we omit its results. Secondly, we used the p strongest features for $p \in 1, 3, 5, 10$.

Table 2: Table of the precision values obtained using only individual features to predict the ranking –feature abbreviations are explained in section 4.2. The top three features for each query set are shown in bold, and a (-) indicates a negative correlation.

Feature	Arts	States	Spam	Multiple
NBOD	(-)53.8%	**54.4%**	51.1%	50.9%
FNEN	**(-)59.4%**	53.5%	51.6%	**(-)59.8%**
RFFT	52.1%	(-)54.3%	50.0%	53.9%
ATLE	(-)54.2%	54.0%	50.0%	54.4%
FATT	56.2%	50.3%	53.0%	51.8%
AMQT	55.4%	50.5%	52.1%	56.9%
SIMT (N)	56.5%	(-)52.0%	52.7%	**59.0%**
SIMT	55.4%	(-)50.9%	52.6%	**69.7%**
TMKY (N)	51.4%	51.4%	54.5%	50.0%
TMKY	53.0%	51.4%	**55.1%**	50.0%
ILNK	**58.0%**	**66.3%**	**57.0%**	53.9%
PRNK	**58.7%**	**60.6%**	**55.0%**	57.2%
TDIR	52.3%	53.5%	54.3%	51.9%

Furthermore, we did not only use the raw features but also experimented with non-linear combinations of them. For $p \in 3, 5, 10$ we included all the quadratic terms of the selected features and trained a logistic regression model on this transformed input data. Thus, when working with features x_1, x_2 and x_3 we also include all quadratic terms such as $x_1 \times x_3$ and x_2^2. If we have p basic features then we get $p + p(p-1)/2$ new features, including the linear terms. For no data set this gave better results on the test set than working with only the untransformed linear terms.

Lastly, we tried monotone transformations such as $\log(1+x)$, $1/(1+x)$ and $1-e^{-x}$ on features such as the number of inlinks, the document length or the number of term occurrences which follow a power law. This only lead to slightly worse results which are not included here.

3.8 Normalization

Generally, the outcome of a statistical inference algorithm can depend heavily on the use of data normalization. This is not the case in our setting. Firstly, it should be clear that multiplying the value of any feature *i* in all data points by a constant γ simply rescales the final scoring weight b_i by $1/\gamma$. More interestingly, also rescaling the individual difference vectors $\mathbf{x}$ has almost no influence. The sign of $\mathbf{x} \cdot \mathbf{w}$ does not depend on the length of $\mathbf{x}$, thus any linear scheme working only with the *number* of misclassified training points will yield the same result. The only property that does change is the distance from the separating hyperplane which, for instance, influences the posterior probabilities in logistic regression. However, experiments showed that the overall results were almost identical, also for the classification trees, with very slightly better results obtained by training on the unnormalized difference vectors. All experimental numbers refer to this unnormalized setting.

4 System Architecture

To extract the feature sets used in our analysis we built a system with three components: a downloader, a parser, and an analyzer.

Downloader: software that executes a query and downloads the returned pages using the data set queries

Feature Extractor: software that computes the features of the pages downloaded.

Analyzer: software that analyzes the features of the returned pages and estimates a function using numerical methods.

4.1 Downloader

The downloader receives as an input a data set of queries (see Table 1). For each such query it does the following:

1. Submits the query to the search engine, i.e. to the Google API in our case, downloads all URLs u_i obtained as result set, and checks whether they or their domain are contained in the Open Directory [12]. This is another feature which might influence the ranking function of a search engine, and thus we included it in our study. For simplicity and to avoid bias based on the file type, we limited ourselves to HTML pages.
2. For each u_i, it then sends a query to obtain the number of inlinks, using Google API's ”link:www.abc.om” syntax, as well as the top 5 in-linking pages. We had to limit ourselves to 5 inlinks because of the reduced access offered by the Google API to the search service, i.e., only 1000 queries of 10 answers per day, per user.
3. Finally, it downloads these 5 pages to further analyze their anchor text.

Also, for each individual query term, the “Downloader” submits a single query and outputs the estimate number of results. This will be later used to compute the inverse document frequency value for the tf-idf similarities [14].

4.2 Feature Extractor

The Feature Extractor receives as an input the downloaded pages. Pages are converted to XML using `tidy`, to be able to use the DOM (document object model) API for accessing different parts of the document. This is useful for checking if query terms appear in some special formatting element such as boldface.

We divide the list of the page features that are extracted by the parser into: content, formatting, link and metadata. Where sensible we included both raw and normalized versions of features, e.g. counting both the absolute number of query term occurrences in the title and the percentage of query terms appearing there. Here is the complete list of features we extracted. An (N) after a feature indicates that we also considered a normalized/averaged version of this feature.

Content features, query independent:

- `DIFT` Number of different terms of the document
- `FNEN` Fraction of terms in the documents which can not be found in an English dictionary
- `NBOD` Number of bytes of the original document
- `NBTO` Number of bytes of text in the original document
- `RFFT` Relative frequency of the more frequent term, i.e.: term frequency. If a document has 3 words and the most frequent word repeats 2 times, then this is 2/3
- `ATLE` Average term length

Content features, query dependent:

- TFQT Term frequency of query term = Number of occurrences of the query term (averaged over the different query terms) (N)

- SIMT Similarity of the term to the document, in terms of vector space model. We compute it using the frequency of terms in documents and the inverse document frequency of each term. (N)

- APQT Average position of the query terms in the document = 1 at the beginning, 0 if at the end and in-between in the middle.

- AMQT Average matches of the query terms

- CTQW Closeness of terms in the query in the webpage (distance in number of terms , smallest windows containing all of them)

- FATT Anchor text term frequency

Formatting features, query-dependent. We used the number of "special" occurrences divided by the total number of occurrences of query terms:

- THTM Term in a special document zone including HTML tags: B, I, U, FONT, BIG, H1-H6, A, LI and TITLE (N)

- TATV Term as an attribute value (element/@attribute): IMG/@ALT, IMG/@TITLE (N)

- TMKY Term in the meta keywords or description (N)

- TCLP Term in capitals in the page (N)

Link features, query-independent:

- ILNK Number of pages linking to a page, in-degree approximated using Google API link: queries

- PRNK PageRank of the page, or the approximation of the PageRank in a 0-10 scale obtained from Google's toolbar.

- OLNK Number of out-links in the page

- FOLN Fraction of out-links to external Web sites

Metadata features:

- TURL Term is in the page's URL or not.

- TDIR Term is listed in a web directory or not.

We computed text similarity SIMT of the query and the returned document. SIMT is computed using a TF-IDF weighting scheme similar to the one used by Salton [14], where the similarity is defined using the weight of each query term. This weight is computed using the normalized frequency of terms in documents and the idf inverse document frequency of each term.

4.3 Analyzer

The analyzer is the component is the last stage of our system and obtains the estimate of the ranking function. The statistical methods used to obtain this estimate were discussed in section 3. To evaluate the performance we used the following quality measures, each having its own justification.

1. **Precision on all pairs:** This measure simply looks at *all* possible pairs and the corresponding difference vectors. The precision is the percentage of correctly classified vectors, thus corresponding to a correct "**u** is ranked above **v**" decision. For the cases where we also have a total ordering, namely for logistic regression and the SVM model, this measure can be computed from the Kendall's τ measure as: 50%+ 50 * Kendall's τ %.
2. **Precision on "shifted" pairs:** See 3.6 for a description of the "shifted" pairs. Here we only look at the percentage of correctly classified "shifted" pairs, which are further apart in the original ranking and their relative order is thus easier to predict.
3. **Precision on "top" pairs:** Here we only consider for each query the top result and all its difference vectors. This number thus gives the percentage of web pages which are (correctly) predicted to be ranked below the highest ranking document.

5 Experimental Results

To give a quantitative estimate of the importance of each individual feature Table 2 gives the precision values which can be obtained for each category if one ranked *only* according to this individual feature. However, only features which were among the top 5 strongest for at least one category are included in the table. A (-) indicates that the feature is negatively correlated with the score, i.e., an increase in the feature value is an indicator for a worse ranking. Note that using only the strongest feature gives a precision between

Table 3: Ranking for the query ”discriminant gaussian link multiple radial supervised” from the Multiple query set when using *only* the `SIMT` feature. There were 33 results for this query.

Predicted rank	1	2	3	4	5	6
Google rank	3	5	6	4	13	2

57% for the Spam category and 69.7% for the Multiple category. Table 3 gives the top 6 results of a query ranking obtained when only using the text similarity (`SIMT`) feature. For the Arts category the strongest indicator of a high ranking was a low fraction of non-English terms (`FNEN`), followed by the PageRank (`PRNK`). For the States queries the by far strongest such indicator was the number of in-links (`ILNK`). For the Spam query this was also the strongest feature but far less significantly, emphasizing that for these queries it would be fatal to put too much weight on any individual feature. Out of all four categories it was also this category where the directory information (`TDIR`) was most closely linked to the ranking. Lastly, for the long queries from the Multiple category the `SIMT` was most closely correlated with the ranking.

The best performances of any model for each category are listed in Table 4. Unfortunately, these are only marginally better than the baseline values for the strongest feature in Table 2. Table 4 also lists the details of the corresponding model. The best pruned trees consisted in almost all cases of a single node corresponding to the strongest individual feature. The fact that the SVMs did not give an improvement over the baseline for any model might be due to an inappropriate (default) choice of the regularization parameter C with which the authors did not experiment.

It is worthing pointing out that the precision is significantly higher for the “shifted” pairs, as can be seen from the second column of Table 4. These pairs are further apart and are thus easier to classify as, in general, the differences in the relevant features will be more striking.

6 Shortcomings and Room for Improvement

As across all data sets and for all methods and feature transformations considered the best test precision was only about 65%, as shown in Table 4 the question arises, how this could be improved?.

Working with only our collection of features we can, given the number of different methods and transformations we tried, safely claim that the answer is a negative one: it cannot be improved substantially. More features, such as the domaing ending (.edu, .com, etc.), which also could have an influence on the ranking, could be included in the analysis but are unlikely to give a dramatic boost. Similarly, much larger training data sets would probably exhibit only a minor influence; in our case we were strongly hindered by the query limitations of the Google API. The real problem seems to lie in the fact that many crucial features are hidden and cannot be observed from the outside.

These features which are certainly relevant may include:

- The query logs, which Google obtains through its toolbar.
- The age of the incoming links and other information related to web link dynamics.
- The rate of change at which a website changes, obtained by repeated web crawls.
- The “true” number of ingoing links, as Google’s `link:www.abc.com` only gives a lower bound.
- The “true” PageRank used by Google, as the one displayed in its toolbar is only an approximation, and furthermore, seems to be too strongly correlated to the number of in-links [16].

Some of these could, however, theoretically be obtained by a web search engine with a large enough set of indexed web pages. It might also be worth including a category with random, artificial terms or numbers, assuming that there are still a few hits for these terms. For such a category at least the use of query logs could be largely ruled out.

More fundamentally, one can only speculate about the algorithmic details. It is, e.g., possible that Google uses (variants of) topic-sensitive PageRank [7] or the Hilltop algorithm [1], both of which try to overcome the

Table 4: Best precision achieved on all, “shifted” and “top” pairs. We include the performance on the test data as well as on the whole data set, including training, validation and test sets.

	% all pairs correct		% “shifted” pairs correct		% “top” pairs correct		
Dataset	Test	All	Test	All	Test	All	Best model
Arts	63.7%	61.8%	69.1%	66.4%	47.6%	48.0%	Log. regr., strongest 3 features
States	64.6%	66.3%	73.2%	73.8%	97.6%	98.5%	Class. tree, only ILINK feature
Spam	62.5%	59.5%	70.5%	62.1%	98.2%	74.8%	Log. regr., strongest 10 features
Multiple	67.5%	70.9%	78.1%	81.3%	81.0%	87.0%	Log. reg., strongest 3 features

notion of a global, topic-independent measure of quality, which is inherent to PageRank. For these algorithms the ranking would no longer be a function of simple features and a much more elaborate analysis would be needed.

One should also not forget that any web search engine always has the option of “manually” re-ranking the results for certain queries. It is known that in certain countries search engines voluntarily cooperate with the authorities to exclude certain web pages for legal reasons from their results. Likewise, it is imaginable that for certain queries pages are pushed up or down because of financial or other agreements.

The reason for us to choose different query categories and to try to have “homogeneous” queries within one category was that we assume that a query is first categorized and then handled according to the categorization. This categorization could involve the scan for certain keywords indicating a certain topic but it could also involve inferring information about the type of question (homepage finding vs. question answering) and the type of user. A query such as “I’m looking for information about search engines” containing several stopwords, might indicate a user less familiar with using search engines and thus less careful in choosing the query terms. This could imply a boost of query-independent features for such queries. Likewise, a user using advanced query syntax who “knows what he is doing” might be better off with a different ranking scheme. This could be the reason why the queries “adversarial” and “adversarial -lairasrevda” (excluding adversarial spelled backwards) lead to different rankings on Google, although conceptually there is no reason for this.

It is even possible –and sensible– for a web search engine to take information about the query initiator into account. Such information could either be collected in the form of data cookies or, simply, by considering the browser language, connection type or geographical location. A user with a dial-up connection will generally have a different user profile from a user of a high-speed university domain so that different ranking schemes might be appropriate. Similarly, a user in Spain might prefer a different ranking than a user in Germany.

7 Conclusions

Along this study we attempted to produce a complete method for learning search engine ranking function(s). Overall, our experiment was sound and its results were very consistent: ranking only according to the strongest feature for a category gives is able to predict the order in which any pair of pages will appear in the results with a precision of between 57% (for a data set including commercial terms that are used in spam) and 70% (for a data set including long queries from a very specific domain). This was, despite trying various algorithms and feature transformations, only mildly improved by including other features.

If one had not split the data into a training and a test set one could have, given the large number of features and transformations we considered, achieved an almost arbitrarily high precision on the training data, but a worse performance on unseen data. In this article we have also discussed the reasons for these results, which are likely to be related to the lack of many crucial features such as user preferences data and algorithmic details such as the possible use of topic-sensitive PageRank.

Acknowledgment

The authors would like to thank Dr. Jörg Rahnenführer for useful suggestions and clarifications concerning some aspects of the data mining algorithms used.

References

[1] K. Bharat and G. A. Mihaila. When experts agree: using non-affiliated experts to rank popular topics. In *WWW '01: Proceedings of the tenth international conference on World Wide Web*, pages 597–602, 2001.

[2] W. Cohen, R. Schapire, and Y. Singer. Learning to order things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[3] comScore Networks. Press release, 2004.

[4] J. Díez, J. J. del Coz, O. Luaces, F. Goyache, A. M. P. J. Alonso, and A. Bahamonde. Learning to assess from pair-wise comparisons. *LNCS*, 2527, 2002.

[5] Google api. http://api.google.com.

[6] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2003.

[7] T. H. Haveliwala. Topic-sensitive pagerank. In *WWW '02: Proceedings of the eleventh international conference on World Wide Web*, pages 517–526. ACM Press, 2002.

[8] Infoseek search engine. http://www.infoseek.co.uk/.

[9] T. Joachims. Optimizing search engines using clickthrough data. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142. ACM Press, 2002.

[10] T. Joachims. Svm-light support vector machine, 2002. http://svmlight.joachims.org/.

[11] J. Nielsen. When search engines become answer engines. *Jakob Nielsen's Alertbox*, August 2004.

[12] Open directory project. http://dmoz.org/.

[13] G. Pringle, L. Allison, and D. L. Dowe. What is a tall poppy among web pages? *Computer Networks and ISDN Systems*, 30:369–377, 1998.

[14] G. Salton and C. Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5):513–523, 1988.

[15] A. K. Sedigh and M. Roudaki. Identification of the dynamics of the google's ranking algorithm. In *13th IFAC Symposium On System Identification*, 2003.

[16] T. Upstill, N. Craswell, and D. Hawking. Predicting fame and fortune: Pagerank or indegree? In *Proceedings of the Australasian Document Computing Symposium, ADCS2003*, pages 31–40, Canberra, Australia, December 2003.