Improving Category Specific Web Search by Learning Query Modifications

Eric J. Glover[1,2], Gary W. Flake[1], Steve Lawrence[1], William P. Birmingham[2], Andries Kruger[1], C. Lee Giles[1,3], David M. Pennock[1]

{compuman,flake,lawrence,akruger,giles,dpennock}@research.nj.nec.com[1]
{compuman,wpb}@eecs.umich.edu[2]
giles@ist.psu.edu[3]

NEC Research Institute[1]
4 Independence Way
Princeton, NJ 08540

EECS Department[2]
University of Michigan
Ann Arbor, MI 48109

Information Sciences and Technology[3]
Pennsylvania State University
University Park, PA 16801

Abstract

Users looking for documents within specific categories may have a difficult time locating valuable documents using general purpose search engines. We present an automated method for learning query modifications that can dramatically improve precision for locating pages within specified categories using web search engines. We also present a classification procedure that can recognize pages in a specific category with high precision, using textual content, text location, and HTML structure. Evaluation shows that the approach is highly effective for locating personal homepages and calls for papers. These algorithms are used to improve category specific search in the Inquirus 2 search engine.

1: Introduction

Typical web search engines index millions of pages across a variety of categories, and return results ranked by *expected topical relevance*. Only a small percentage of these pages may be of a specific category, for example, personal homepages, or calls for papers. A user may examine large numbers of pages about the right topic, but not of the desired category. In this paper, we describe a methodology for category-specific web search. We use a classifier to recognize web pages of a specific category and learn modifications to queries that bias results toward documents in that category. Using this approach, we have developed metasearch tools to effectively retrieve documents in several categories, including personal homepages, calls for papers, research papers, product reviews, and guide or FAQ documents.

For a specific category, our first step is to train a support vector machine (SVM) [16] to classify pages by membership in the desired category. Performance is improved by considering, in addition to words and phrases, the documents' HTML structure and simple word location information (e.g., whether a word appears near the top of the document).

Second, we learn a set of query modifications. For this experiment, a *query modification* is a set of extra words or phrases added to a user query to increase the likelihood that results of the desired category are ranked near the top.[1] Since not all search engines respond the same way to modifications, we use our classifier to automatically evaluate the results from each search engine, and produce a ranking of search engine and query modification pairs. This approach compensates for differences between performance on the training set and the search engine, which has a larger database and unknown ordering policy.

2: Background

The primary tools used for locating materials on the web are search engines that strive to be comprehensive by indexing a large subset of the web (the most comprehensive is estimated to cover about 16%) [14]. In addition to general-purpose search engines, there are special-purpose search engines, metasearch engines, focused crawlers [4, 5], and advanced tools designed to help users find materials on the web.

[1]Our system supports field based modifications, such as a constraint on the URL or anchortext.

0-7695-0942-8/01 $10.00 © 2001 IEEE

2.1: Web Search

A typical search engine takes as input a user's query and returns results believed to be *topically relevant*. An alternate approach allows the user to browse a subject hierarchy. Subject hierarchies are typically created by humans and often have much lower coverage than major general-purpose search engines.

The search engine Northern Light has an approach called "custom folders" that organizes search results into categories. Although results may be organized into clusters, if the desired category is not one of the fixed choices, the user must still manually filter results. Northern Light currently allows users to specify 31 different categories prior to searching. Northern Light does not distribute its algorithm for clustering, so a user is unable to evaluate results from other search engines using the same rules.

One limitation of a general-purpose search engine is the relatively low coverage of the entire web. One approach for improving coverage is to combine results from multiple search engines in a metasearch engine. A metasearch engine could increase coverage to as much as 42% of the web in February 1999 [14]. Some popular metasearch engines include Ask Jeeves, DogPile, SavvySearch [10], MetaCrawler [18], and ProFusion [7].

A typical metasearch engine considers only the titles, summaries and URLs of search results, limiting the ability to assess relevance or predict the category of a result. A *content-based metasearch engine*, such as Inquirus [13], downloads all results and considers the full text and HTML of documents when making relevance judgments (this approach can easily be extended to non-textual information).

A second improvement to metasearch engines is source selection, based on the user's desired category. Some metasearch engines such as SavvySearch [10], and ProFusion [7] consider, among other factors, the user's subject or category when choosing which search engines to use. Choosing specific sources may improve precision, but may exclude general-purpose search engines that contain valuable results.

To further improve the user's ability to find relevant documents in a specific category, Inquirus has been extended to a preference-based metasearch engine, Inquirus 2 [8]. Inquirus 2 adds the ability to perform both source selection and query modification, as shown in Figure 1. The category-specific knowledge used by Inquirus 2 (sources, query modifications, and the classifiers) was learned using the procedures described in this paper. Our procedure automates the process of choosing sources and query modifications that are likely to yield results both topically relevant and of the desired category. In addition, the classifier can be used to better predict the *value* to the user.

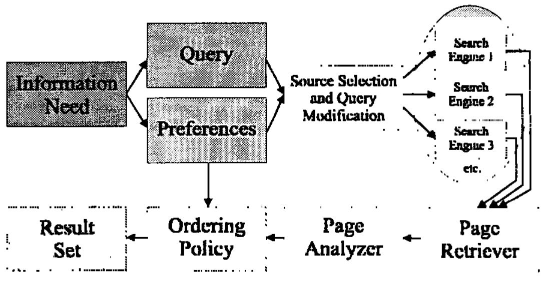

Figure 1. The Inquirus 2 metasearch engine improves web search by considering more than just the query when making search decisions.

2.2: Query Modification

Query modification is not a new concept. For years, a process called *query reformulation* or *relevance feedback* has been used to enhance the precision of search systems. In query modification the query used internally is different from the one submitted by the user. Modifications include changing terms (or making phrases), removing terms, or adding extra terms. The goal is an internal query that is more representative of the user's intent, given knowledge about the contents of the database. A simple example is a user typing in `Michael Jordan`. If the user is looking for sports-related results, a better query might be `Michael Jordan and basketball`, helping to reduce the chance of a document being returned about the country of Jordan, or a different Michael Jordan.

Mitra et al. [15] describe an automatic approach to discover extra query terms that can improve search precision. Their basic algorithm, like other *relevance feedback* algorithms, retrieves an initial set of possibly relevant documents, and discovers correlated features to be used to expand the query. Unlike other algorithms, they attempt to focus on the "most relevant" results, as opposed to using the entire set. By considering results more consistent with the user's original query, a more effective query modification can be generated. Their work assumes that the user is concerned only with topical relevance, and does not have a specific category need (that is not present in the query).

Other related work includes the Watson project [2], an integrated metasearch tool that modifies queries to general purpose search engines with the goal of returning results related to a document that the user is viewing or editing.

2.3: SVMs and Web Page Classification

Categorizing web pages is a well researched problem. We choose to use an SVM classifier [20] because it is resistant to overfitting, can handle large dimensionality, and has been shown to be highly effective when compared to other methods for text classification [11, 12]. A brief description of SVMs follows.

Consider a set of data points, $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_N, y_N)\}$, such that $\boldsymbol{x}_i$ is an input and y_i is a target output. An SVM is calculated as a weighted sum of kernel function outputs. The kernel function of an SVM is written as $K(\boldsymbol{x}_a, \boldsymbol{x}_b)$ and it can be an inner product, Gaussian, polynomial, or any other function that obeys Mercer's condition.

In the case of classification, the output of an SVM is defined as:

$$f(\boldsymbol{x}, \boldsymbol{\lambda}) = \sum_{i=1}^{N} y_i \lambda_i K(\boldsymbol{x}_i, \boldsymbol{x}) + \lambda_0. \tag{1}$$

The objective function (which should be minimized) is:

$$E(\boldsymbol{\lambda}) = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \lambda_i \lambda_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) - \sum_{i=1}^{N} \lambda_i, \tag{2}$$

subject to the box constraint $0 \leq \lambda_i \leq C, \forall_i$ and the linear constraint $\sum_{i=1}^{N} y_i \lambda_i = 0$. C is a user-defined constant that represents a balance between the model complexity and the approximation error. Equation 2 will always have a single minimum with respect to the Lagrange multipliers, $\boldsymbol{\lambda}$. The minimum to Equation 2 can be found with any of a family of algorithms, all of which are based on constrained quadratic programming. We used a variation of Platt's Sequential Minimal Optimization algorithm [16, 17] in all of our experiments.

When Equation 2 is minimal, Equation 1 will have a classification margin that is maximized for the training set. For the case of a linear kernel function ($K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$), an SVM finds a decision boundary that is balanced between the class boundaries of the two classes. In the nonlinear case, the margin of the classifier is maximized in the kernel function space, which results in a nonlinear classification boundary.

Some research has focused on using hyperlinks, in addition to text and HTML, as a means of clustering or classifying web pages [3, 6]. Our work assumes the need to determine the class of a page based solely on its raw contents, without access to the inbound link information.

```
QUIP($\mathcal{A}$, $\mathcal{B}$, $SE$, $Q$, $n$)
  INPUT:   Training examples $\mathcal{A}$(pos) and $\mathcal{B}$(neg)
           Set of search engines $SE$, test queries $Q$
           The number of results to consider $n$
  OUTPUT:  Ranked list of search engine,
             query modification tuples $\{se, qm\}$

  1. Generate set of features $F$ from $\mathcal{A}$ and $\mathcal{B}$
  2. Using $F$ train SVM classifier
  3. $F^*$ is the top 100 features from $F$
  4. Select set of possible query modifications,
       $QM = \{F^*\} \cup \{F^* \times F^*\}$
  5. Remove duplicate or redundant modifications
  6. $QM'$ = PRE-PROCESS-QMOD($QM$,$\mathcal{A}$,$\mathcal{B}$)
  7. The set of tested modifications $QM'' \subseteq QM'$
  8. return SCORE-TUPLES ($QM''$,$SE$,$Q$,$n$)
```

Table 1. QUery modification Inference Procedure.

3: Procedure

Table 1 shows our main algorithm, QUERY MODIFICATION INFERENCE PROCEDURE (QUIP). This algorithm first trains an SVM classifier on labeled data. The algorithm then automatically generates a set of good query modifications, ranked by expected recall. Finally, using the learned classifier to evaluate the query modifications on real search engines, a rank ordering of query modification, search engine tuples is produced. The classifier and the tuples are incorporated into Inquirus 2 to improve category-specific web search.

3.1: Training the Classifier

First we train a binary classifier to accurately recognize positive examples of a category with a low false-positive rate. To train the classifier, it is necessary to convert training documents into binary feature vectors, which requires choosing a set of reasonable features. Even though an SVM classifier may be able to handle thousands of features, adding features of low value could reduce the generalizability of the classifier. Thus, dimensionality reduction is performed on the initial feature set.

Unlike typical text classifiers, we consider words, phrases and underlying HTML structure, as well as limited text location information. A document that says "home page" in bold is different from one that mentions it in anchor text, or in the last sentence of the document. We also added special features to capture non-textual concepts, such as a URL corresponding to a personal directory. Table 2 describes the representation of document features.

3.1.1 Initial Dimensionality Reduction

Rare words and very frequent words are less likely to be useful for a classifier. We perform a two step

Code	Description
T	Title word or phrase
TS	Occurs in first 75 terms of the document
F	Occurs anywhere in full-text (except title)
E	Occurs in a heading, or is emphasized
UP	Word or special character (tilde) occurs in the URL path
UF	Word or special character occurs in the file name portion of the URL
A	Occurs in the anchortext
S	Special symbol – Captures non-textual concepts, such as personal directory, top of tree, name in title

Table 2. Document vector types used

process to reduce the number of features to a user specified level.

First, we remove all features that do not occur in a specified percentage of documents. A feature f is removed if it occurs in less than the required percentage (threshold) of both the positive and negative sets, i.e.,

$$\left(|\mathcal{A}_f|/|\mathcal{A}| < \mathcal{T}^+\right) \text{ and } \left(|\mathcal{B}_f|/|\mathcal{B}| < \mathcal{T}^-\right)$$

Where:

- $\mathcal{A}$: the set of positive examples.
- $\mathcal{B}$: the set of negative examples.
- $\mathcal{A}_f$: documents in $\mathcal{A}$ that contain feature f.
- $\mathcal{B}_f$: documents in $\mathcal{B}$ that contain feature f.
- $\mathcal{T}^+$: threshold for positive features.
- $\mathcal{T}^-$: threshold for negative features.

Second, we rank the remaining features based on entropy loss. No stop word lists are used.

3.1.2: Expected Entropy Loss

Entropy is computed independently for each feature. Let C be the event indicating whether the document is a member of the specified category (e.g., whether the document is a personal homepage). Let f denote the event that the document contains the specified feature (e.g., contains "my" in the title). The prior entropy of the class distribution is $e \equiv -\Pr(C)\lg\Pr(C) - \Pr(\overline{C})\lg\Pr(\overline{C})$. The posterior entropy of the class when the feature is present is $e_f \equiv -\Pr(C|f)\lg\Pr(C|f) - \Pr(\overline{C}|f)\lg\Pr(\overline{C}|f)$; likewise, the posterior entropy of the class when the feature is absent is $e_{\overline{f}} \equiv -\Pr(C|\overline{f})\lg\Pr(C|\overline{f}) - \Pr(\overline{C}|\overline{f})\lg\Pr(\overline{C}|\overline{f})$. Thus, the expected posterior entropy is $e_f\Pr(f) + e_{\overline{f}}\Pr(\overline{f})$, and the *expected entropy loss* is

$$e - \left(e_f\Pr(f) + e_{\overline{f}}\Pr(\overline{f})\right).$$

If any of the probabilities are zero, we use a fixed value. Expected entropy loss is synonymous with expected information gain, and is *always* nonnegative [1].

All features meeting the threshold are sorted by expected entropy loss to provide an approximation of the usefulness of the individual feature. This approach assigns low scores to features that, although common in both sets, are unlikely to be useful for a binary classifier.

3.2: Choosing Query Modifications

Like the work of Mitra [15], the goal of our query modification is to identify features that could enhance the precision of a query. Unlike their work, we have extra information regarding the user's intent in the form of labelled data. The labelled data defines a category, and the learned modifications can be re-applied for different topical queries that fall in the same category without any re-learning.

Once the training set has been converted to binary feature vectors, we generate a set of query modifications. Our features may be non-textual, or on fields not usable by every search engine, such as anchortext, or the URL. In this paper, we only used features that occurred in the full text or the top 75 words.

To generate the ranked list of possible query modifications, we score all possible query modifications by expected *recall*. We define $\mathcal{QM}$ to be the set of query modifications, or all combinations of one or two features. A user parameter, **P**, is the desired minimum precision. To compute the precision, we must consider the *a priori* probability that a random result from a search engine is in the desired category, as opposed to the probability that a random document from the training set is in the positive set. To compensate for the difference between the *a priori* probability and the distribution in the training set, we add a parameter α, defined below. Table 3 shows our algorithm for ranking the query modifications.

Consider the following definitions:

- $\mathcal{QM}$: Set of all possible query modifications for consideration,
- qm : A single query modification, comprised of a set of one or more features: $qm = \{f_1, f_2, ...f_n\}$,
- $\mathcal{A}_{qm}$: the set of documents from $\mathcal{A}$, the positive set, that contain all the features in qm,
- $\mathcal{B}_{qm}$: the set of documents from $\mathcal{B}$, the negative set, that contain all the features in qm,
- α : factor to compensate for *a priori* probability of the class: $P(A) = \frac{|\mathcal{A}|}{|\mathcal{A}|+\alpha|\mathcal{B}|}$,
- **P**: User provided parameter for minimum desired precision.

```
PRE-PROCESS-QMOD($\mathcal{QM}$, $\mathcal{A}$, $\mathcal{B}$)
  INPUT: $\mathcal{QM}, \mathcal{A}, \mathcal{B}$
  OUTPUT: A ranked list of all $qm \in \mathcal{QM}$

  1. foreach $qm \in \mathcal{QM}$
  2. Compute predicted precision:
       $\mathcal{P}'(qm) = \frac{|\mathcal{A}_{qm}|}{|\mathcal{A}_{qm}|+\alpha|\mathcal{B}_{qm}|}$
  3.   if $\mathcal{P}'(qm) < \mathbf{P}$
  4.       $Score(qm) = 0$
  5.   else
  6.       $Score(qm) = \frac{|\mathcal{A}_{qm}|}{|\mathcal{A}|}$
  7. end foreach
  8. return all $qm \in \mathcal{QM}$ sorted by $Score(qm)$
```

Table 3. Initial ranking of the query modifications.

The algorithm PRE-PROCESS-QMOD returns a ranked list (by expected recall) for each query modification that meets the specified precision level, **P**. We can measure precision on the web, but cannot measure recall without having knowledge of all possible results and their classification. Typical query modification approaches strive to maximize precision; however, a query modification may overly constrain the results causing very high precision, but very low recall. As a result, we feel that ranking query modifications by expected recall is more desirable, as long as they have at least some minimum precision requirement. If a user is searching for a specific page in some category, say an individual's homepage, as opposed to a set of homepages of people who have a particular interest, low recall can make it very likely the desired page is never found [9, 19]. In general, there is an inverse relation between precision and recall. Our approach allows the user to control this balance by choosing the minimum precision level.

3.3: Scoring Query Modifications and Engines

Not all search engines respond the same way to a query modification, or contain the same distribution of documents. One user may be looking for homepages of persons who work for a company, while another might be looking for homepages of people who own a Ford truck; in each case, the *best* query modification and search engine may be different. To rank the search engine, query modification tuples, we use representative test queries and apply the classifier to the results. Table 4 shows a simple algorithm that can rank a set of search engines and query modifications starting with a set of sample queries. The ranking is based on the number of valid documents returned that are classified as true by the learned classifier.[2] In addition, a specified parameter n controls how many results are considered for each search engine query. Considering too many results may harm performance, while too few may not accurately capture the effectiveness of a given tuple.

```
SCORE-TUPLES($QM, SE, Q, n$)
  INPUT: List of search engines and query
         modifications to test, test queries, $Q$,
         a parameter $n$
  OUTPUT: A ranked list of all $< qm, se >$
  1. foreach $qm \in QM$
  2.   foreach $se \in SE$
  3.     $Score_{qm,se}$ = EVAL-TUPLE($qm, se, Q, n$)
  4.   end foreach
  5. end foreach
  6. return all $< qm, se >$ in $QM, SE$,
       sorted by $Score_{qm,se}$

EVAL-TUPLE($qm, se, Q, n$)
  INPUT: Query modification $qm$, search engine $se$,
         test queries $Q, n$
  OUTPUT: Relative score for the tuple $< qm, se >$
  1. INITALIZE $score = 0$
  2. foreach $q \in Q$
  3.   $R$ = first $n$ result URLs from search engine $se$
       with query CONCAT($q, qm$)
  4.   foreach url $u \in R$
  5.     $p = DOWNLOAD(u)$
  7.     if $p \neq nil$ AND $CLASSIFY(p, u) = TRUE$
  8.       $score = score + 1$
  9.     end if
  10.  end foreach
  11. end foreach
  12. return $score$
```

Table 4. Functions to score each $< qm, se >$ pair for a given classifier and test queries.

Thus, we define for training and testing:

- QM'' : Set of query modifications for testing, $QM'' \subset \mathcal{QM}$,
- p : A single web page $< url, html >$,
- se : A search engine, $se \in SE$, all search engines,
- q : A query to be submitted to a search engine, Q is the set of test queries,
- $p = DOWNLOAD(u)$: Downloads page p corresponding to URL u, if there is an error then p is defined as nil,
- $CLASSIFY(p, u)$: Function which returns true if page p at URL u is of the desired category.

Although the algorithm could compute scores for hundreds of test queries and tens of thousands of pos-

[2]The learned classifier considers many features, in addition to words and phrases, reducing the chance that a single feature causes a page to always classify as true. In addition, SVMs are designed not to overfit, further reducing the chance of a single or small set of features dominating.

#	Feature	#	QMOD	R
1	S.personalDIR	1	my + welcome	25%
2	TS.my	2	my + welcome to	20%
3	UF./	3	my + i am	14%
4	T.s	4	my + m	14%
5	F.my	5	my + home page	14%
6	T.page	6	my + am	13%

Table 5. (left) Top six features for personal homepages. (right) Top eight query modifications, before evaluation, with predicted recall.

sible query modifications, we caution against running it on a large set, so as to avoid sending large numbers of queries to the search engines. To further reduce the burden on each search engine, we ran the algorithm serially, alternating search engines between each request. For our experiments, we choose QM'' to be a relatively small subset of the set of all ranked query modifications, plus the query with no modification, and at least one "naive" modification for comparison.

4: Results

To test our approach, we chose two categories: personal homepages and calls for papers. In each, we assumed that the user started with a subject and wished to find a set of pages in the desired category, as opposed to a signle specific page. For each case we started with a dataset of positive and negative examples, 2,618 and 2,703 pages for personal homepages and calls for papers, respectively. The initial dataset was split into a training and test set, an SVM was trained, and evaluated. There was no overlap between the URLs in the training and test sets. We then generated a ranked list of query modifications, of which several were evaluated using the algorithm in Table 4.

The classifiers performed well: the false-positive rate was low, less than 2% for both personal homepages and calls for papers. In addition, for each category, several highly effective query modifications that performed better than the naive modifications were discovered.

4.1: Personal Homepages

A personal homepage is a difficult concept to define objectively. The definition we used is a page made by an individual (or family) in an individual role, with the intent of being the entry point for information about the person (or family). It is possible a person may have more than one personal homepage, and it is also acceptable for a person to dedicate their homepage to an interest and or hobby, but not to corporate endeavors. Pages that were manual redirects

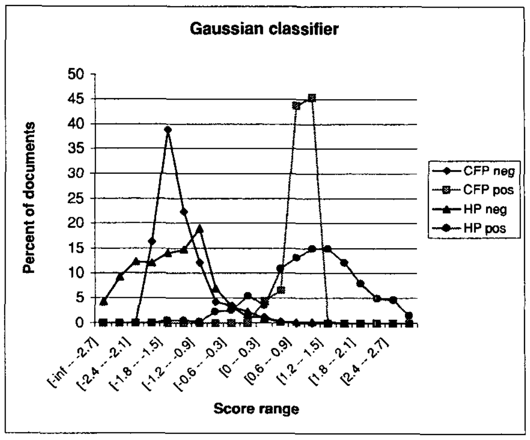

Figure 2. The distribution of scores from the SVM for test data for personal homepages and calls for papers.

or entry pages (pages that had only an image and a small amount of text) were removed.

We started with a set of 718 positive and 1,900 negative examples. The positive list was created by classifying pages from many sources, including the Personal Home Page list on Yahoo,[3] several University and research sites, and several ISP sites that offered personal homepages, as well as a few that were gathered from friends and colleagues. The negative examples were from pages selected from the logs of Inquirus 2. A first pass of the algorithm was run to generate a classifier that was applied to the negative set to identify possible false-negatives, which were manually checked and removed if positive.

The training set was created by choosing about 350 positive URLs at random, removing several pages from GeoCities to prevent biasing the classifier on the GeoCities URL, resulting in 327 positive examples. 1,500 negative pages were randomly selected for the training set. The remaining 391 positive and 400 negative examples were used in the test set.

4.1.1: Classifier

The parameters for the dimensionality reduction were a threshold of 7.5% for both the positive and negative features, and of the 1,061 features meeting the threshold the top 400 were kept, as ranked by expected entropy loss (α was set to 25). Table 5 shows the top six features as ranked by entropy loss and the top six query modifications as ranked by the algo-

[3]It should be noted that the majority of pages on this list resulted in either server or network errors, or were not personal homepages.

Rank	Query Mod	Search Eng	Prec
1	my "home page"	FAST	64.5%
2	+"my name is"	AV	57.5%
3	"s home"	Google	54.5%
4	+my +"i am"	AV	52%
5	+"s home"	AV	52%
10	"my name is"	FAST	39.5%
11	+my +"home page"	AV	38.5%
27	"my name +is"	Google	6.5%
-	no mod	AV	8%
-	no mod	Google	6%
-	no mod	FAST	3.5%

Table 6. Summarized results for the query modifications as applied to several search engines.

rithm in Table 3 with a minimum precision requirement of 25%.

To test the classifier, we applied it to the test set, and to several other sources of positive and negative examples. On four different categories in the open directory, the classifier had over 96% accuracy with no personal homepages in thetest set. On several lists of personal homepages from Universities, only two out of 127 were misclassified. The overall accuracy for the test set, using the Gaussian kernel, was 88.4% and 98% for the positive and negative sets, respectively. The classifier was also tested on the calls-for-papers *training* set with 99.2% overall accuracy.

4.1.2: Query Modifications

We generated and tested several query modifications as described by the algorithms in Tables 3 and 4. Table 5 (right) lists the top six query modifications and their predicted recall. To test the query modifications, we chose three search engines: AltaVista, FAST (AllTheWeb), and Google. We chose four test queries: `chess`, `"ballroom dancing,"` `beagles`, and `cat`. We tested nine query modifications: `my welcome`, `my "welcome to"`, `my "i am"`, `my "home page"`, `"s home"`, `"home page"`, `my name is`, `i resume`. Table 6 shows the precision of the 200 possible results (50 for each query).

Table 6 shows clearly the wide variation among search engines for the same queries and query modifications. The query modification[4] `"my name is"` worked very well for AltaVista, very poorly for Google, and in the middle for FAST. However, the query modification `my & "home page"` worked well for FAST, but performed significantly worse for AltaVista.

[4]The correct syntax for Google, i.e. adding pluses to stop words, was always used when generating query modifications.

With no modification, the best performer was AltaVista returning only 8% of the results as personal home pages. For the the naive modification of `"home page,"` Google performed best with 28.5% precision, FAST had 15.5% precision and AltaVista had 8.5% precision. All three were worse than the highly ranked learned modifications, `my "home page"`, `"my name is"`, and `"s home"`, each with over 50% precision.

We note that ranking of query modifications is dependant on the choice of test queries, i.e., different sets of test queries may produce different rankings of query modifications and search engines.

4.2: Calls For Papers

A good call for papers typically contains a title describing the event, a list of topics, a program committee, deadlines and submission information. We obtained a list of possible CFP's by combining URLs from lists of many CFPs and results from multiple search engines for a variety of queries likely to yield conference-related pages.

We started with a set of 432 manually classified calls for papers, and 2,269 negative examples, consisting of several "random" URLs from the Inquirus 2 logs, and about 850 conference related pages. The training set consisted of 249 positive and 1,250 negative pages (randomly selected, and with a limit of 20 pages from any one domain). The remaining 183 positive and 1,019 negative examples formed the test set.

4.2.1: Classifier

The parameters for dimensionality reduction were thresholds of 7.5% for both positive and negative features. Of the 2,868 features meeting the threshold, the top 750 were kept by expected entropy loss (α was set to 25). The top-ranked features, all occuring in the full-text were: `"for papers"`, `"call for papers"`, `"papers"`, and `"submission"`. The feature `"call for papers"` occuring in the top seventy five terms was also in the top five.

To test the SVM classifier, we applied it to the test set, to the set of known personal homepages, and to several other negative sets. For the test set, with the Gaussian kernel the accuracy was 100% and 98.6% for positive and negative respectively. For the personal homepages training set, and for the Open Directory category of AI, accuracy was also 100%. We also created a second set of 160 positive calls for papers from several conference sites with an accuracy of 91.3%. The lower accuracy on the second test set was likely due to the large number of foreign pages that appeared to have a different basic structure than

#	Query Mod	Search Eng	% Prec
1	"call +for papers" "+will +be"	Google	88%
2	"call for papers" "will be"	FAST	85%
3	"call for" authors	FAST	83%
4	"call +for papers"	Google	83%
5	"call +for" authors	Google	82%
6	"for papers" "will be"	FAST	79%
11	+notification +"important dates"	AV	75%
13	+"call for papers" +"will be"	AV	64%
-	no mod	Google	2%
-	no mod	FAST	.7%
-	no mod	AV	0%

Table 7. Results for query modifications for calls for papers as applied to several search engines.

those in the training set (although all pages contained some English).

4.2.2: Query Modifications

Just as for personal homepages, after learning the classifier, we generated and tested several query modifications as described by the algorithms in Tables 3 and 4.

The top-ranked query modifications were: `"for papers"` & `"will be"`, `"call for papers"` & `"will be"`, `call` & `authors`, `"call for"` & `authors`, and `"call for papers"` & `not`. The top five query modifications each had roughly 50% predicted recall. We used a minimum precision requirement of 40%. To test the query modifications, we chose three search engines: AltaVista, FAST (AllTheWeb), and Google. We chose three test queries: `databases`, `"natural language processing"` and `algorithms`. We tested six query modifications, and the the query without modification. The query modifications were: `"for papers"` & `"will be"`, `"call for papers"` & `"will be"`, `call authors`, `"call for"` & `authors`, `"call for papers"`, `notification` & `"important dates"`. Table 7 shows the precision of results classified as calls for papers, out of a possible 150 (50 for each query were retrieved).

The results for query modifications for calls for papers were more consistent among the three search engines than for personal homepages, with FAST and Google having nearly identical scores for each query modification. As expected, without modification 2% or fewer of the results were calls for papers. In this case, however, the naive modification of `"call for papers"` performed quite well. Google had the highest precision for the naive modification of 83%, FAST had 76% precision, while AltaVista had only 49% precision. In general, the learned modifications performed better than the highly ranked naive ones. Of course, if the search engines change their ordering policy or their databases, these results could change.

5: Summary and Future Work

A user with a specific information need category may find it difficult to locate both relevant and on-category results from a general-purpose search engine. Here, we present an automated method for learning search-engine-specific query modifications that can result in very high precision (and reasonable expected recall) for personal homepages and calls for papers. The learned query modifications are shown to have over 50% precision for personal homepages and over 80% precision for calls for papers, compared with the less than 8% and 2% when not using query modifications. In addition, the classifiers have been shown to be able to identify positive examples with about 88% and nearly 100% accuracy for personal homepages and calls for papers respectively. In both cases, naive query modifications did not perform as well as those recommended by our algorithm.

Our classifier is trained on automatically extracted features that consider words and phrases, as well as HTML structure, simple locational information, and other useful features with no textual correspondence. We implement a simple method for dimensionality reduction of the large feature space using expected entropy loss and thresholding. Using the classifier, a simple method is applied to measure the effectiveness of query modifications for individual search engines. Our results indicate that when searching for personal homepages, the precision of individual search engines varied significantly, even for identical queries. For calls for papers, the variation was less significant. This variation is due to either the differences in individual search engine databases, or their individual ordering policies.

One of our goals is to explore methods for automatic training set discovery and expansion, such as boosting, allowing a user to initially provide only three or four positive training URLs. Such approaches will enable us to allow users to more easily generate their own categories for Inquirus 2.

6: Acknowledgements

We would like to thank Frans Coetzee and Bob Krovetz for their assistance in this work. This work was partially funded under NSF contract 9872057, "Congregating Agents."

References

[1] N. Abramson. *Information Theory and Coding*. McGraw-Hill, New York, 1963.

[2] Jay Budzik and Kristian Hammond. Watson: Anticipating and contextualizing information needs. In *62nd Annual Meeting of the American Society for Information Science*, Medford, NJ, 1999.

[3] S. Chakrabarti, B. Dom, and P. Indyk. Enhanced hypertext categorization using hyperlinks. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 27(2):307–318, June 1998.

[4] S. Chakrabarti, M. van der Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. In *WWW8*, Toronto, Canada, 1999.

[5] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using context graphs. In *26th International Conference on Very Large Databases, VLDB 2000*, Cairo, Egypt, 10–14 September 2000.

[6] Gary W. Flake, Steve Lawrence, and C. Lee Giles. Efficient identification of web communities. In *Proceedings of the Sixth International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD-2000)*, pages 150–160, Boston, MA, 2000. ACM Press.

[7] Susan Gauch, Guihun Wang, and Mario Gomez. ProFusion: Intelligent fusion from multiple, distributed search engines. *Journal of Universal Computer Science*, 2(9), 1996.

[8] Eric J. Glover, Steve Lawrence, William P. Birmingham, and C. Lee Giles. Architecture of a metasearch engine that supports user information needs. In *Eighth International Conference on Information and Knowledge Management (CIKM'99)*, pages 210–216, Kansas City, MO, November 1999. ACM Press.

[9] David A. Grossman and Ophir Frieder. *Information Retrieval: Algorithms and Heuristics*. Kluwer Academic Publishers, 1998.

[10] Adele E. Howe and Daniel Dreilinger. SavvySearch: A meta-search engine that learns which search engines to query. *AI Magazine*, 18(2), 1997.

[11] Thorsten Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Tenth European Conference on Machine Learning ECML-98*, pages 137–142, 1999.

[12] James Tin-Yau Kwok. Automated text categorization using support vector machine. In *Proceedings of the International Conference on Neural Information Processing (ICONIP)*, pages 347–351, Kitakyushu, Japan, 1999.

[13] Steve Lawrence and C. Lee Giles. Context and page analysis for improved web search. *IEEE Internet Computing, July-August*, pages 38–46, 1998.

[14] Steve Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400(July 8):107–109, 1999.

[15] Mandar Mitra, Amit Singhal, and Chris Buckley. Improving automatic query expansion. In *ACM SIGIR 98*, Melbourne Australia, 1998. ACM.

[16] J. Platt. Fast training of support vector machines using sequential minimal optimization. In B. Scholkopf, C. Burges, and A. Smola, editors, *Advances in kernel methods - support vector learning*. MIT Press, 1998.

[17] J. Platt. Using sparseness and analytic QP to speed training of support vector machines. In *Advances in Neural Information Processing Systems*, 1999.

[18] E. Selberg and O. Etzioni. The MetaCrawler architecture for resource aggregation on the Web. *IEEE Expert*, (January–February):11–14, 1997.

[19] C. J. van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 2nd edition, 1979.

[20] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer Verlag, New York, 1995.