# CUSTOMIZATION OF A GENERIC SEARCH ENGINE BY ADDING USER CATEGORIES

by

AJAY MADKAIKER

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON
May 2006

To my parents, Mohandas and Shanta Madkaiker

## ACKNOWLEDGEMENTS

I am grateful to Dr. Manfred Huber for accepting me as a student under his supervision. I thank Dr. Alp Aslandogan and Dr. Lynn Peterson for serving as committee members. I thank my family for their support for all these years and years to come. Last but not the least I thank friends Vasanth Ranganathan, Gokuldas Naik, Deepak Jayanna, colleagues Ashokkumaar P Loganathan and Anup Siddarth Sabbi and well wishers for their understanding and compassion.

December 6, 2005

## ABSTRACT

CUSTOMIZATION OF A GENERIC SEARCH ENGINE
BY ADDING USER CATEGORIES

Publication No. ______

Ajay Madkaiker, M.S.
The University of Texas at Arlington, 2006

Supervising Professor: Manfred Huber

The current search engines available on the Net are generic in nature. They do not consider user preferences and treat all user's information needs in the same way. As a result they frequently return a large number of links, that do not meet the user's information need. This requires more searching to find what the user is looking for. For example, if a user is interested in a particular game, e.g. *cricket*, and enters the query *world cup*, a generic search engine would return links of all the sports that hold a *world cup*. The user has to browse through a considerable number of non-relevant pages before he is able to get to links he is looking for. This is also because search engines don't have the ability to ask a few questions and they also can not rely on judgment and past experience to rank web pages, in the way humans can. This raises the issue of customizing a generic search engine to consider user preferences.

There have been a number of attempts in the past to personalize the search for information on the Net. These systems are based on relevance feedback methods, similarity measures, or storing a user profile explicitly or implicitly. Some of them have shown impressive results in query expansion and providing pages similar to the user's interest.

Here we propose a novel system for personalizing web search. Our method is based on creating a user profile as he performs his routine searches in a given user category. In this customization, the user is allowed to create personal user categories within which he could search for information on the Net without getting too many irrelevant links in his search results. The application enhances the query by adding words that are generated from the user profile stored for a particular user category. It uses information about the probabilistic co-occurrence of words in the user profile with other words in the query as a measure for adding words. The snippets returned from the generic search engine are then classified on the basis of the user profile and are re-ordered according to a measure representing the interest of the user.

With our method the user not only gets personalized query expansion but also receives re-ordered search results from the search engine. In this system the query expansion is made to optimize the estimated returns of the search engine, taking into account the classification accuracy and re-ranking of results. The system adds words that would give the user the best possible returns according to his user profile.

# TABLE OF CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

# INTRODUCTION

Searching for Web documents you are interested in can be easy or seem impossibly difficult. This is in part due to the sheer size of the World Wide Web (WWW), currently estimated to contain 10 billion documents. In Web searching for documents using keyword-based search engines, the user is always guessing what words will be in the pages he/she wants to find. This issue can be addressed by personalizing web search engines.

By personalizing the search engine we can search for information from the user's perspective. For this we build a user profile on the client machine. Using this user profile we expand a query in the user's perspective, i.e. in a way the user "looks" at particular information which would help the search engine return results relevant to the user rather than what is popular or "important" on the Internet. We also classify the search results returned from a generic search as relevant or not relevant to the user category. This saves valuable user time while searching for information on the Internet. Therefore a personalization of search engines currently on the web is very useful.

When one is "searching" for information on the Web one is not searching for it directly. Through the computer one accesses one or more of many intermediate search tools currently available. One searches that search tool's database or collection of sites which is a subset of the entire WWW. The search tool provides the hypertext links with URLs to other pages. The following are the three basic types of search tools:

i. Search engines: The databases of search engines are built by computer programs ("spiders") and not by human selection. The pages stored in these databases are sometimes organized by subject categories and are ranked by a computer algorithm. They contain every word of the web pages they link to in the form of an index, often only the word frequencies. To search for information through them one finds the pages by specific keywords that match words in the pages browsed by the search engine. They are huge and very often retrieve a lot of information. The results from them are unevaluated, they contain both good and irrelevant documents, and one must evaluate everything one finds.

ii. Subject directories: Subject directories are mostly built by human selection not by computers or spider programs and are organized into subject categories. The subjects in them are not standardized and vary according to the scope of each directory. They could contain the full text of the web pages they link to. One can see titles, descriptors, subject categories, etc. of the subjects in them. They use broad or general terms for describing the pages present in the directories. They are specialized and smaller than most search engines. They have a huge range in size and are often carefully evaluated and annotated.

iii. Searchable database contents or the "Invisible Web": The "visible web" is what one sees in the results from general web search engines. It is also what one sees in almost all subject directories. The "invisible web" is what one cannot retrieve ("see") in the search results and other links contained in these types of tools. Most of the invisible web is made up of the contents of thousands of specialized searchable databases that one can search via the web. The search results from many of these databases are delivered in web pages that are generated in response to the search.

Such pages very often are not stored since it is easier and cheaper to dynamically generate the answer to the search query than to store all possible pages containing all possible answers to all possible queries people could make to the database. The "Invisible web" is estimated to offer two or three times as many pages as the visible web. There are many specialized searchable databases that the World Wide Web allows you to access through a search box in a web page. The terms you use in your search are sent to that specialized database, and are returned to you in another web page that is dynamically generated as a response. It is not retained anywhere after your search.

## 1.1 Popular Search Engines

Search engine technology has had to scale dramatically to keep up with the growth of the web. Ten years ago one of the first web search engines, the World Wide Web Worm (WWWW) had an index of 110,000 web pages and web accessible documents. By 2000, the web had surpassed the billion pages mark. Currently it is estimated that there are around 10 billion pages on the Internet.

At the same time, the number of queries search engines handle has grown incredibly. WWWW received an average of about 1500 queries per day in 1994; as of last year *Google* is reported to have received 250 million queries per day. In order to scale to the Web, a search engine needs to have fast crawling technology to gather web documents and keep them up to date. Storage space must be used efficiently to store indexes and, optionally, the documents themselves. The indexing system must process hundreds of gigabytes of data efficiently. Queries must be handled quickly, at a rate of hundreds to thousands per second.

Many techniques are used in modern search engines to provide more context for user queries. *Yahoo!*, *ODP* and *Google* return both categories and documents. *Northern Light* and *WiseNut* cluster their results into categories, and *Vivismo* groups results dynamically into clusters. *Teoma* clusters its results and provides query refinements.

In this report we attempt to personalize a generic search engine. We do this by building a user profile for each of the user categories the user creates. By building a user profile we can add words collected from the profile to enhance queries to match the user requirements for information. We also re-rank the search results returned so that the relevant links are provided in the top few positions. This way the user saves time needed to locate the links that are useful for him.

## 1.2 Contribution of this Report

Generally a user in search for information on his topic of choice would enter his query on the search site and get the results from the search engine. The search engines currently on the Net provide only the popular results back to the user. In order to personalize a user's search for information we customize the returns of the search engine by developing a client side application. This application on the client machine is used to query the generic search engine. In the application we build a user model which captures the user's need for information according to his tastes and interest. Here we use a collection of web pages that the users deems relevant for his category and also the pages that he considers not relevant. This gives a fair idea about the information the user is interested in. When the search engine is queried, the application modifies the query according to the pages collected in the client's interest and returns the results from the search engine on the modified query. These results are then re-ranked on the basis of snippets returned. Classifying the snippets for relevant search results is better than downloading an entire web page from the Internet for classification as it saves bandwidth and computation time on the client side. All search results classified as relevant are placed in the top positions saving the user time to browse through irrelevant popular search results which would be returned otherwise. Thus the system presented here provides a personalization effect and can save the user valuable time.

Google which is currently the most popular search engine is used as a generic search engine in this report. It provides a Web Interface for developing programs. Using this web interface we can access Google's resources for searching information on the Net.

## 1.3 Organization of the Report

The further report is divided into the following chapters: Chapter 2 discusses the current methods available to ease and personalize the search for information on the Internet. Chapter 3 presents the work done for the report, including building the model and the expression used for enhancing the search results. Chapter 4 shows the experimental analysis of the proposed model and its improvement over finding information on a generic search engine. Chapter 5 concludes the report and discusses future work.

# CHAPTER 2

# RELATED WORK

The automated categorization of text into predefined categories has witnessed a booming interest in the past 10 years, due to the increased availability of documents in digital form and the ensuing need to organize them. In the last few years, content-based document management tasks have gained a prominent status in the information systems field. Text categorization (TC a.k.a. text classification or topic spotting), the activity of labeling natural language text with thematic categories from a predefined set, is one such task. TC is now being applied in many contexts ranging from document indexing based on a controlled vocabulary, to document filtering, automated metadata generation, word sense disambiguation, population of hierarchical catalogues of web resources, and in general any application requiring document organization or selective and adaptive document dispatching [33].

Until the late '80s the most popular approach to TC, was a knowledge engineering (KE) one, consisting of manually defining a set of rules encoding expert knowledge on how to classify documents in the given categories. In the '90s this approach has increasingly lost popularity in favor of the machine learning (ML) paradigm, according to which a general inductive process automatically builds a text classifier by learning, from a set of pre-classified documents, the characteristics of the categories of interest. The advantages of this approach are accuracy comparable to that achieved by human experts and a considerable savings in terms of expert labor, since no intervention from either knowledge engineers or domain experts is needed for the construction of the classifier or for its porting to a different set of categories [16, 25].

There are various related works published on personalization of web search in information filtering and intelligent agents. Following are examples of such agents: *Syskill and Webert* is a software agent that learns to rate pages on the World Wide Web (WWW), deciding what pages might interest a user [26]. *WebWatcher* [36] assists users in browsing the World Wide Web. It guides the user along an appropriate path through the collection of web pages based on its knowledge of the user's interests, of the location and relevance of various items in the collection, and of the way in which others have interacted with the collection in the past. *Letizia* [20] is a user interface agent that assists a user browsing the World Wide Web. As the user operates a conventional Web browser such as Netscape, the agent tracks user behavior and attempts to anticipate items of interest by doing concurrent, autonomous exploration of the links from the user's current position. *CiteCeer* [14] and *Liza* [21] are other such systems that provide assistance in retrieving scientific literature and user assistance in web browsing by explicit user feedback on the web pages.

## 2.1 Single-Label versus Multi-Label text categorization

Different constraints may be enforced in the TC task, depending on the application. For instance, we might require that for a given integer k, exactly k( or$\leq k, or \geq k$) elements of C, where C=$c_1, \ldots, c_{|C|}$ is a set of predefined categories, be assigned to each $d_j \in D$ where $D = d_1, \ldots, d_n$ is the domain of documents and $\Phi : D * C \rightarrow T, F$ is a function giving out a value T if $d_j$ belongs to category $c_i$ and value F otherwise. The case in which exactly one category must be assigned to each $d_j \in D$ is often called the *single-label* case, while the case in which any number of categories from 0 to $|C|$ may be assigned to the same $d_j \in D$ is dubbed the *multi-label* case.

A special case of *single-label* TC is binary TC, in which each $d_j \in D$ must be assigned either to category $c_i$ or to its complement $\sim c_i$. From a theoretical point of view, the binary case is more general than the *multi-label* case since an algorithm for binary classification can also be used for *multi-label* classification: one needs only to transform the problem of multi-label classification under $c_1, \ldots, c_{|C|}$ into $|C|$ independent problems of binary classification under $c_i, \sim c_i$ for $i = 1, \ldots, |C|$. However, this requires that the categories be stochastically independent of each other, that is, for any c', c" the value of $\Phi(d_j, c')$ does not depend on the value of $\Phi(d_j, c'')$ and vice versa. This is usually assumed to be the case but the converse is not true: an algorithm for multi-label classification cannot be used for either binary or single-label classification. To classify a given document $d_j$ (i) the classifier might attribute k>1 categories to $d_j$ , and it might not be obvious how to choose a "most appropriate" category from them; or (ii) the classifier might attribute to $d_j$ no category at all, and it might not be obvious how to choose a "least inappropriate" category from C. [33]

## 2.2 Category Pivoted vs Document Pivoted text categorization

There are two different ways of using a text classifier. Given $d_j \in D$, we might want to find all the $c_i \in C$ under which it should be filed (*document-pivoted categorization*- DPC). Alternatively, given $c_i \in C$, we might want to find all the $d_j \in D$ that should be filed under it (*category-pivoted categorization*- CPC). This distinction is more pragmatic than conceptual, but is important since the sets $C$ and $D$ might be available in their entirety right form the start. It is also relevant to the choice of the classifier building method, as some of these methods allow the construction of classifiers with a definite slant toward one or the other style i.e. DPC or CPC.

DPC is suitable when documents become available at different moments in time, e.g. in filtering e-mail. CPC is instead suitable when (i) a new category $c_{|C|+1}$ may be added to an existing set C= $c_1, \ldots, c_{|C|}$after a number of documents have been classified under C and (ii) these documents need to be reconsidered for classification under $c_{|C|+1}$.

DPC is used more often than CPC, as the former situation is more common than the latter. [33]

## 2.3 The Machine Learning approach to Text Categorization.

In the 80's the most popular approach for the creation of automatic document classifiers consisted in manually building, by means of knowledge engineering techniques, an expert system capable of taking TC decisions. It would typically consist of a set of manually defined logical rules, one per category, of type

$$\textbf{If} < DNFformula > \textbf{then} < category > \tag{2.1}$$

A DNF ("disjunctive normal form") formula is a disjunction of conjunctive clauses; the document is classified under *category* if and only if it satisfies the formula, that is, if and only if it satisfies at least one of the clauses. A drawback of this approach is the knowledge acquisition bottleneck well known from the expert system literature. That is, the rules must be manually defined by a knowledge engineer with the aid of a domain expert: if the set of categories is updated, then these two professionals must intervene again, and if the classifier is ported to a completely different domain (i.e. set of categories), a different domain expert needs to intervene and the work has to be repeated from scratch.

Since the early '90s, the machine learning approach to text categorization has gained popularity and has eventually become the dominant one. In this approach, a general inductive process (also called a learner) automatically builds a classifier for a category $c_i$ by observing the characteristics of a set of documents manually classified under $c_i$ or $\sim c_i$ by a domain expert. From these characteristics, the inductive process gleans the characteristics that a new, unseen document should have in order to be classified under $c_i$. In the machine learning terminology the classification problem is an activity of supervised learning, since the learning process is "supervised" by the knowledge of the categories and of the training instances that belong to them.

The ML approach relies on the availability of an initial corpus $\Omega = \{d_1, \ldots, d_{|\Omega|}\} \subset D$ of documents, pre-classified under $C = \{c_1, \ldots c_{|C|}\}$. That is, the values of the total function $\Phi : D \times C \rightarrow \{T, F\}$ which gives out a value T if the document $d_j$ belongs to the category $c_i$, and otherwise returns F, are known for every pair $< d_j, c_i > \in \Omega \times C$. A document $d_j$ is a positive example of $c_i$ if $\Phi(d_j, c_i) = T$, a negative example of $c_i$ if $\Phi(d_j, c_i) = F$ [33].

## 2.4 Text Classifiers

In this section we will discuss the various text classifiers used in Text Categorization. The inductive construction of a ranking classifier for category $c_i \in C$ usually consists of the definition of a function $CSV_i : D \rightarrow [1, 0]$ that, given a document $d_j$, returns a *categorization status value* for it, that is a number between 0 and 1 which represents the evidence for the fact that $d_j \in c_i$. Documents are then ranked according to their $CSV_i$ value. This works for "document-ranking TC"; "category-ranking TC" is usually tackled by ranking, for a given document $d_j$, its $CSV_i$ scores for the different categories in $C = c_1, \ldots, c_{|C|}$. The $CSV_i$ takes different meanings according to the learning method used. For example in Naive Bayes it is defined in terms of a probability, whereas in the Rocchio method it is a measure of vector closeness in $|n|$-dimensional space.

### 2.4.1 Naive Bayes

Naive Bayes(NB) is a type of probabilistic classifier. The basic idea in the Bayesian approach is to use the joint probabilities of words to estimate the probabilities of a category given a document. The naive part of NB methods is the assumption of word independence [32] , i.e., the conditional probability of a word given a category is assumed to be independent of the other words. This assumption makes the computation of NB classifiers far more efficient than the exponential complexity of the general Bayesian approaches because it does not use word combinations as predictors.

The NB classifier works by predicting the posterior probability of a category $c_j$ among a set of categories $C = c_1, c_2 \ldots, c_n$ given the terms $T = t_1, t_2 \ldots, t_n$ as

$$p(c_j|t_1, t_2, \ldots, t_n) \propto p(t_1, t_2 \ldots, t_n|c_j)p(c_j) \tag{2.2}$$

Making the independence assumption this equation can be simplified using

$$p(T|c_j) \propto \prod_{k=1}^{n} p(t_k|c_j) \tag{2.3}$$

Therefore the posterior probability becomes

$$p(c_j|T) \propto p(c_j) \prod_{k=1}^{n} p(t_k|c_j) \tag{2.4}$$

There have been attempts to relax the *independence assumption.* But relaxing the *independence assumption* produces classifiers of higher computational cost and characterized by harder parameter estimation problems making it a method that might not be as easy to follow [17]. Earlier efforts in this direction within probabilistic text search by [28] have not shown the performance improvements that were hoped for.

Recently, the fact that the binary independence assumption seldom harms effectiveness has given some theoretical justification as shown by Domingos and Pazzani [9]. They found that the Naive Bayesian classifier performs quite well in practice even when strong attribute dependence are present when the sample size is small.

In our report we, too, relax the independence assumptions by pairing words together irrespective of the order they are present in the document. We see that by doing this

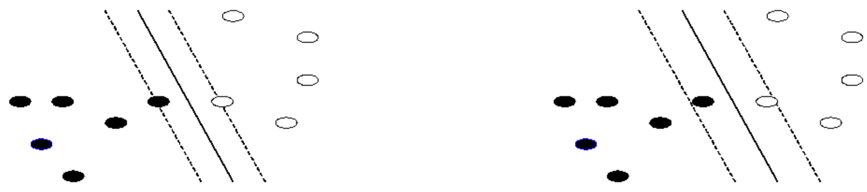

Figure 2.1. The decision line (solid) with the smaller and maximal margin. The data points on the dashed lines are the Support Vectors..

we can capture an important semantic feature of the documents with respect to a query, while adding only limited computational cost.

### 2.4.2 Support Vector Machine

The *Support Vector Machine* (SVM), introduced by Vapnik in 1974, is a learning approach. It was used in text categorization by Joachims in 1998. It is based on the Structural Risk Minimization principle. The method is described over a vector space where the problem is to find a decision surface that "best" separates the data points in two classes. In Figure 2.1 we show a case in a two-dimensional space with linearly separable data points, but the idea can be generalized to a high dimensional space and to data points that are not linearly separable. A decision surface in a linearly separable space is a hyperplane. The solid lines in the graphs in Figure 2.1 show two possible decision surfaces, each of which correctly separates the two groups of data. The dashed lines parallel to the solid ones show how much one can move the decision surface without causing misclassification of data. The distance between each set of parallel lines is referred to as "the margin". The SVM problem is to find the decision surface that maximizes the margin between the data points in a training set. The decision surface learned by a SVM for linearly separable problems is a hyperplane which can be written as

$\vec{w} \cdot \vec{x} - b = 0$

where $\vec{x}$ is an arbitrary data point, and the vector $\vec{w}$ and the constant b are learned from a training set of linearly separable data. When applied to text categorization, a comparison of the performance of SVMs with other classification methods using Reuters-21578 corpus found that a SVM outperformed all other methods tested. [41]

### 2.4.3 k-NN

k-NN stands for k-nearest neighbor classification, a well known statistical approach which has been intensively studied in pattern recognition. The algorithm is as follows:

Given a test document, the system finds the k nearest neighbors among the training documents, and uses the categories of the k neighbors to weight the category candidates. The similarity score of each neighbor document to the test document is used as the weight of the categories of the neighbor document. If several of the k-nearest neighbors share a category, then the surrounding neighbor's weights of that category are added together, and the resulting weighted sum is used as the likelihood score of that category with respect to the test document.

By sorting the scores of the categories, a ranked list is obtained for the test documents. By thresholding on these scores, binary category assignments are obtained. The decision rule in k-NN can be written as:

$$y(\vec{x}, c_j) = \sum_{\vec{d}_{i \in kNN}} sim(\vec{x}, \vec{d_i}) y(\vec{d_i}, c_j) - b_j \tag{2.5}$$

where $y(\vec{d_i}, c_j) \in$ 0,1 is the classification function for document $\vec{d_i}$ with respect to category $c_j$(y=1 for YES, and y=0 for No); $sim(\vec{x}, \vec{d_i})$ is the similarity between the test document $\vec{x}$ and the training document $\vec{d_i}$; and $\vec{b_j}$ is the category specific threshold for the binary decisions. The performance of kNN is among the best for techniques used in text classification. The classifier performs well as it uses the majority decision of k training samples which also reduces the effect of noisy data. The drawback of the approach is the presence of a large feature space, which can become problematic as the size of the training set increases. [41]

### 2.4.4 The Rocchio Method

Linear classifiers which are profile-based have an advantage in terms of interpretability as such a profile is a representation that is more readily understandable by a human than, for example a neural network.The Rocchio method is used for inducing linear, profile-style classifiers. It relies on an adaptation to TC of the well known Rocchio's formula for relevance feedback in the vector space model, and is the only TC method whose roots lie exclusively in the IR tradition rather than in ML [33]. It was proposed by Hull in 1994 and since then it has been an object of research [13]. Rocchio's method computes a classifier $\vec{c_i} = (w_{1i}, \ldots, w_{|\mathcal{T}|i})$ for category $c_i$ by means of the formula

$$w_{ki} = \Big(\frac{\beta}{|\{\bar{d}_j|ca_{ij} = 1\}|} \cdot \sum_{\{\bar{d}_j|ca_{ij}=1\}} w_{kj}\Big) - \Big(\frac{\gamma}{|\{\bar{d}_j|ca_{ij} = 0\}|} \cdot \sum_{\{\bar{d}_j|ca_{ij}=0\}} w_{kj}\Big) \tag{2.6}$$

where $w_{kj}$ is the weight of term $t_k$ in document $d_j$. In this formula, $\beta$ and $\gamma$ are control parameters that allow setting the relative importance of positive and negative examples. Most of the time the negative sets are de-emphasized by setting $\gamma$ to a low value and $\beta$ to a high one.

The Rocchio classifier, like all linear classifiers, has the disadvantage that it divides the space linearly. Hence the issue is how to distinguish documents that are near positives (NPOS) since the documents are linearly classified, i.e. whether the set of negative training instances should be considered in its entirety, or whether a set of well chosen sample of it, like a set of near positives, should be selected. This is due to the fact that

Rocchio takes the average of all positive examples, and as all averages this is only partly representative of the whole set. This can be solved by using the query zoning method [34] which originates from the observation in the original Rocchio formula and is used for relevance feedback in IR. Here near positives tend to be used rather than generic negatives, as the documents on which user judgments are available are among the ones that had scored highest in the previous ranking.

## 2.5 Personalized Information Retrieval

Using the machine learning approach to text categorization discussed in the previous section we can personalize information retrieval (IR). The basic nature of personalized information retrieval is to maintain a history of the user's behavior in his search for information i.e. a user profile. In order to personalize we need to understand the underlying goals of user searches. The user goals can be divided in to the following three categories: Navigational, Informational, and Resource [8].

i. *Navigational* goals are a desire to be taken to the home page of the institution or organization in question. To be considered navigational, the query must have a single authoritative web site the user already has in mind. Most navigational queries consist of names of companies, universities or well known organizations.

ii. *Informational* searches are focused on the user goal of obtaining information about the query topic. This category is used to answer questions that the user has in mind, for example requests for *advice*, and *undirected* requests to simply learn more about a topic. The informational goal also includes the desire to *locate* something in the real world, or simply get a *list* of suggestions for further research.

iii. *Resource* queries represent a goal of obtaining something (other than information). If the resource is something that one plans to use in the offline world, such as song lyrics, recipes, sewing patterns etc. , it is an *obtain* goal. If the resource is something that needs to be installed on the computer or any other electronic device to be useful, the goal is *download*. Finally the *interact* goal occurs when the intended result of the search is a dynamic web service that requires further interaction to achieve the user's task.

With this framework of search goals we can associate goals with queries. There are two ways that the search engine might associate goals with queries at runtime: the user can identify the goal explicitly through a user interface, or the system can attempt to infer the goal automatically. The second approach would involve automatic classification using statistical or machine learning methods.

### 2.5.1 The PageRank Method

Currently the most popular way to search for information is to query a generic search engine available on the Net. In order to gain a better understanding of the search engines of today let us discuss the method by which a popular search engine like Google deals with the situation to provide a highly relevant set of results for the query entered by the user.

The Google search engine has two important features that help it produce high precision (number of relevant documents returned) results. First, it makes use of the link structure of the Web to calculate a quality ranking for each web page. This ranking is called PageRank. It creates a link matrix where each node in the link matrix represents a page in the WWW. Every page in the WWW has some number of forward links (outedges) and backlinks (inedges).

Second, Google utilizes links to improve search results. For most popular subjects, a simple text matching search that is restricted to web page titles performs admirably when PageRank prioritizes the results.

Academic citation literature has been applied to the web, largely by counting citations or backlinks to a given page. This gives some approximation of a page's importance or quality. PageRank extends this idea by not counting links from all pages equally, and by normalizing by the number of links on a page. PageRank is defined as follows:

Assuming a page A has pages $P_1..P_n$ which point to it. The parameter dp is a damping factor which can be set between 0 and 1. Usually the dp is set to 0.85. Also L(A) is defined as the number of links going out of page A. N is the total number of all pages on the Web.

The PageRank of a page A is given as follows:

$$PR(A) = (1 - dp)/N + dp(PR(P_1)/L(P_1) + ... + PR(P_n)/L(P_n)) \tag{2.7}$$

PageRank or PR(A) can be calculated using a simple algorithm and corresponds to the principal eigenvector of the normalized link matrix of the web.

The PageRank system can be thought of as a model of user behavior. Assuming there is a "random surfer" who is given a web page at random and keeps clicking on links, never hitting "back" but eventually gets bored and starts on another random page . The probability that the random surfer visits a pages is its PageRank. It forms a probability distribution over web pages, so the sum of all web pages PageRanks will be one. And the damping factor dp is the probability at each page that the "random surfer" will get bored and request another random page.

Another justification given is that a page can have a high PageRank if there are many pages that point to it, or if there are some pages that point to it that have high PageRank. Intuitively pages that are well cited from many places around the web are worth looking at. Also, pages that have perhaps only one citation from something like the Yahoo! homepage are also generally worth looking at. If a page was not high quality, or was a broken link, it is quite likely that Yahoo's homepage would not link to it. PageRank handles both these cases and everything inbetween by recursively propagating weights through the link structure of the web.

The PageRank method used currently lacks in certain areas. It treats all users of the Internet the same. In reality, however, an Internet user is not limited to a particular age group or class of people; every user is different with different tastes, interest and needs for information. One can not go through all that is popular on the web for the query and decide which of the information provided is useful. One would like to have the search return relevant documents in the first few result positions.

### 2.5.2 Improving User Query and Informational Retrieval

By improving user queries, the results returned by a search engine can be improved dramatically. Google, too provides us services for query expansion and personalized search at *labs.google.com*: *Google Suggest* in its beta stage and *Personalized Search*. *Google Suggest* suggests additions to the search query in real time. This is a useful feature as the user can view suggestions for his/her query before submitting to the search engine. It also mentions the number of results returned on submission. However, the suggestions are based on the popular context not on the category or area of interest of the user.

In the case of the *Personalized Search* of Google, it maintains a list of searches made by the user. This information is accessible to the user by logging on his free personal Google account. They manage the user's past searches, including the web pages, images, and news headlines. *Personalized Search* orders the user's search results based on his past searches, as well as the search results and news headlines he may have clicked on.

He can view all these items in his search history and remove any items. This forms the basis for delivering the results to the user.

However, Google's personalized search does not let the user build the categories. The service would only refer to the queries made by the user and the documents actually viewed by the user. On the other hand, in our system we maintain a set of categories for the user, with the user mentioning which pages are useful and which are not relevant to the category. Also, maintaining a user profile on the server raises privacy issues, while maintaining the user profile on the client side is a much safer option. The only drawback would be the processing power and the bandwidth to maintain the user profile on a client machine.

In the work of Mitra, Singhal and Buckley [22] they use an approach called *adhoc or blind feedback* for query expansion [29, 10], which takes the form of "pseudo" relevance feedback where the actual involvement of the user is not required. Since casual users seldom provide a system with the relevance judgments needed in relevance feedback. The goal is automatic query expansion via relevance feedback to add more useful words to a query.

The method they follow is that a small set of documents is retrieved using the original query. These document are assumed to be relevant without any intervention by the user. These documents are then used in the relevance feedback process to construct an expanded query which is then run to retrieve the set of documents actually presented to the user. The drawback of this method is that if a large fraction of the documents assumed relevant is actually non-relevant, then the words added to the query are likely to be unrelated to the topic and the quality of the documents returned using the expanded query is likely to be poor. On the other hand, if the initial retrieval quality is reasonably good, i.e. a good proportion of the initially retrieved documents are relevant, the terms added to the query are mostly related to the search topic and the expanded query matches more relevant documents and retrieves them at high ranks. Final results are therefore likely to improve. Thus *adhoc feedback* is capable of both improving and hurting performance for different queries. Therefore, in order to prevent *query drift* - the alteration of the focus of a search topic caused by improper expansion - there should be a large proportion of relevant documents at top ranks. The method they use is to retrieve a small set of (say 20) documents using the original user query; these documents are all assumed to be relevant without any intervention by the user. The assumed relevant documents are then used in a relevance feedback process (the Rocchio method) to construct an expanded query, which is then run to retrieve the set of documents actually presented to the user.

They determine the strong relevance indicator words by using a term correlation equation. This equation uses the inverted document frequency of words (idf) i.e. the presence of words in each of the document collected. When they use this term correlation they show that the relevant documents get a higher rank.

This method does not include what the user has in mind directly, it implicitly assumes that the document returned in the top positions are the ones that specify what the category the user is searching for.

*WebMate* [18] is an agent that helps users to effectively browse and search the Web. It filters the relevant out of the irrelevant articles according to a user profile. *WebMate* collects only examples that are interesting to the user( only positive examples). It utilizes the term frequency-inverse document frequency(TF-IDF) measure in its method. TF-IDF is given as the product of the frequency of a term in a given document and the inverse of the number of documents the term is present in. It represents each document as a vector in a vector space so that documents with similar content have similar vectors. It utilizes the approach of learning a user profile to compile a personal newspaper. It does this in two ways; the first way is to automatically collect a list of URLs that the user wants monitored. *WebMate* parses the HTML pages, extracts the links of each headline, fetches those pages, and calculates the similarity with the current profile. If the similarity is greater than some threshold, it recommends the page to the user and sorts all the recommended pages in decreasing order of similarity to form the personal newspaper. The other option is to construct a query using the top several words in the current profile and send it to popular search engines. It fetches the pages corresponding to each and every URL in the results. It then calculates the similarity of the profile and these web pages and recommends the pages whose similarity is greater than some threshold presenting the results in descending order of relevance. Single query words are usually ambiguous, or too general. Giving additional keywords can refine the search, providing considerable improvement in the retrieval results. *WebMate* uses the Trigger Pair Model [30, 35] to automatically extract keywords to constrain and refine a search for relevant documents. If a word S is significantly correlated with another word T, then (S,T) is considered a "trigger pair", with S being the trigger and T being the triggered word. When S occurs in the document, it triggers T, causing its probability estimate to change. That is, when we see the word S appearing at some point in a text, we expect the word T to appear somewhere after S with some confidence. Trigger pairs are used to expand the search. As shown in the paper the results are better than just applying a single word. This work is similar to the work presented in this report is that we use a concept similar to the Trigger pairs. In the user profile we use pairs of the words(these pairs are unordered though) in the feature set (the words which most represent the user category) to help in query expansion. These pairs of words are selected from the documents collected to denote the category specifications. For the query expansion we add the word that forms the highest product pair with the query entered by the user.

Watson [7] is an Information Management Assistant(IMA). IMAs automatically discover related material on behalf of the user by serving as an intermediary between the user and the information retrieval systems. IMAs observe a user's interaction with every day applications and then anticipate their information needs using a model of the task at hand. IMA's then automatically fulfill these needs using the text of the document the user is manipulating and knowledge of how to form queries to traditional information retrieval systems (e.g. Internet search engines) by adding keywords from the document the user is manipulating. Because IMAs are aware of the user's task, they can augment their explicit query with terms representative of the context of this task. In this way IMAs provide a framework for bringing implicit task context to bear on servicing explicit information requests, significantly reducing ambiguity. Watson observes users in word processing and Web browsing applications and uses a simple model of the user's task, knowledge of term importance, and an understanding of query generation to find relevant documents and service explicit queries. It has several *application adapters* which are used to gain access to an application's internal representation of a document. The adapters produce a document representation, which is sent to the Watson application when deemed necessary. Documents are represented as sequences of words in one of four styles: Normal, emphasized, de-emphasized, or list item. Next, using knowledge of the kind of information need anticipated, Watson transforms the original document representation into a query, constructed based on the content of the document at hand, that will eventually be sent to external information sources in real time. Words are classified into these groups by detecting the appropriate structures in HTML documents(for Internet Explorer), or by using word properties provided by Microsoft Word. Each of Watson's application adapters sends a typed message containing a sequence of words of that type, represented as a string, to the Watson application. Watson then tokenizes the string using a basic lexical analyzer, removes stop words, and sends each term through a weighting algorithm. The preliminary weight of a term varies inversely with the square of its position on a page. Simultaneously, Watson sends tokens to an array of conceptual unit detectors.Each detector is a finite state automaton accepting a sequence of tokens representing a conceptual unit. When a conceptual unit is detected, Watson presents the user with a common action for the item in the form of a button he can press. The query takes the form of an internal query representation which is sent to selected information adapters. Each information adapter translates the query into the source-specific query language and executes a search. Information adapters are also responsible for collecting results, which are gathered and clustered using several heuristic result similarity metrics, effectively eliminating redundant results. In parallel, Watson attempts to detect conceptually atomic, lexically regular structures in the document. Once such objects are detected, Watson presents the user with a common action for the item in the form of a button they can press. For example, if a page contains an address, Watson will present a button allowing the user to access a map for the address. Thus it provides an implicit service to the user for his information search. It monitors the user behavior for information retrieval. This method could be a computational overhead on the client machine. The assistant has to monitor all the actions of the user. While in our method we assist the user only in the search for information in a particular category he mentions.

Letizia [20] is a user interface agent that assists a user browsing the WWW. As the user operates a conventional Web browser, the agent tracks user behavior and attempts to anticipate items of interest by doing concurrent, autonomous exploration of links from the user's current position. The agent automates a browsing strategy consisting of a best-first search augmented by heuristics inferring user interest from browsing behavior. The difference between the user's search and Letizia's is that the user's search has a reliable static evaluation function, but that Letizia can explore search alternatives faster than the user can. Letizia uses the past behavior of the user to anticipate a rough approximation of the user's interests. Letizia's role during user interaction is merely to observe and make inferences from the observation of the users actions that will be relevant to future requests. In parallel with the user's browsing, Letizia conducts a resource-limited search to anticipate the possible future needs of the user. At any time the user may request a set of recommendations from Letizia based on the current state of the user's browsing and Letizia's search. Such recommendations are dynamically recomputed when anything changes or at the user's request. Letizia's interface design goal was to keep the interaction as minimal as possible, and so it does not show the user directly its profile of the user's interest or its analysis of Web pages. It does not provide any direct means to tell the system what the user is interested in.

### 2.5.3 Category Specific Web Search

As mentioned in Chapter 1, current search engines such as *Google* or *Yahoo!* have hierarchies of categories to help users to specify their intentions. Mapping user queries to categories associates one or more categories to his/her query.

Liu,Yu and Meng [12] provide a strategy to model and gather the user's search history, construct a user profile based on the search history and construct a general profile on the ODP(Open Directory Project) category hierarchy, deduce the appropriate categories for each user query based on the user's profile and the general profile. Liu, Yu and Meng provide a small number of categories with respect to the user's query. If none of the categories is desired, the next set of categories is provided. They go on to show that the user usually finds the categories of interest among the first 3 categories obtained by their system. They store the search history of the user, according to them each query has one or two related categories. They state that the search engine can implicitly gain knowledge about the category of the user query. They also store a user profile to represent the user's interests and infer their intentions for new queries. The user profile is a set of categories and for each category, a set of terms (keywords) with weights. Each category represents a user interest in that category. The weight of the terms in a category reflects the significance of the terms in representing the user's interest in that category. For example if the term "apple" has a high weight in the category "cooking", then the occurrence of the word apple in a future query of the user has a tendency to indicate that the category "cooking" is of interest. A user's profile will be learned automatically from the user's search history. In the application they form three matrices in order to represent user search histories and user profiles. The user's search history is represented by two matrices where $DT$ is a document-term matrix, which is constructed from the user queries and the relevant documents.$DC$ is the document-category matrix, which is constructed from the relationships between the categories and the documents. A user profile is represented by a category-term matrix M created by using the DC and the DT matrix. It contains the relationship between the category and the term. The M matrix is created using methods such as Linear Least Squares Fit, k-Nearest neighbor and Rocchio algorithm. They make three comparisons of results of just using the user profile , user and general profile combined, and the general profile. In the results they observed that the user and general profile combined is better than that using one for the two profiles only. This system generates categories for the user to go through with respect to his query. The user has to correctly identify which of the categories presented to him suits him the most otherwise he has to check for other categories in the list. While in our case the category is defined by the user and we provide him the search results based on the category he chooses to search in. There is a little overhead on our part during the data collection of the category but after the data collection is done the category is ready to be used to enhance every query put by the user and classify the results generated from the search engine. In Liu, Yu and Meng's work the user's preferred category may not be always present for all the queries he presents to the application.

The work of Glover et. all [11] shows how to improve Category Specific Web Search by Learning Query Modifications. They use a SVM classifier to recognize web pages of a specific category and learn modifications to queries that bias results toward documents in that category. They train a SVM to classify pages by membership in the desired category. They claim that the performance of web search is improved by considering, in addition to words and phrases, the document's HTML structure and simple word location information (e.g. whether a word appears near the top of the document). They also learn a set of query modifications. A query modification is a set of extra words or phrases added to a user query to increase the likelihood that results of the category are ranked near the top. In query modification the query used internally is different from the one submitted by the user. Modifications include changing terms (or making phrases) and removing terms. Their procedure automates the process of choosing sources and query modifications that are likely to yield results that are both topically relevant and of the desired category. After training the SVM classifier they perform a two step process to reduce the number of features to a user specified level. First they remove all the words that are below an average threshold of both the positive and negative sets. After that they compute the expected entropy loss. All the features meeting the threshold are sorted by expected entropy loss to provide an approximation of the usefulness of the individual feature. Using the user's labeled data they modify the query which would coincide with the user's intent. This work is similar to the work presented in this report. The difference is that they consider the entire document before classifying if it belongs to the category or not. But in our case we save the bandwidth by considering only the snippet returned by the search engine.

## 2.6 Contribution of this Thesis

In the current chapter we discussed some of the methods developed for personalization of information searching over the Internet. Let us discuss in short the way we personalize information searching over the Internet.

We believe an effective information search over the Internet is done by using a search engine. A search engine operates on the keywords supplied to it by the user. But this is not a favorable method as the search engine mostly uses the "popular" content on the Web to return search results. In order to get a good number of relevant search results we have to enhance the search query so as to capture the user's interest in the category created by him. Here, query expansion is performed to improve relevance of results and re-ranking of the results is conducted after the search is made. The query expansion is determined from created word pairs, which are maintained in the user profile of the user category. This method is similar to the adhoc query expansion in the work of Mitra, Singhal and Buckley [22]. Our method differs in the way that the user himself will select the documents relevant to him while in their case the top positioned search results are considered relevant implicitly and the query is improved based on those search results.

Once the search query is made we re-rank the retrieved results based on the user profile stored for the category. The user profile consists of pairs of *features or words* collected from documents which the user thinks are relevant to the categories created by him. The category creation helps the user distinguish his interests. In the work of Liu, Yu and Meng category specific searching is carried out. However in their case they maintain the user profile in terms of weight matrices which could be computationally expensive and require large memory to maintain. While in our case we maintain single probability values for each word pair in the relevant documents.

The user can also define user categories by mentioning the relevant documents while conducting a search on a traditional search engine. The application notes the relevant documents and the irrelevant search results and adds the information to the user profile.

In the next chapter we shall see how we create an user profile for enhancing the user query and classify the retrieved results from the search engine.

# CHAPTER 3

# THE USER MODEL

As discussed in the previous chapter there are various ways in which we can assist the user in looking up information on the Web. The search engine is widely used as a mechanism for finding information on the Internet. However, it does not retain explicit user preferences for personalizing searches made by the user. Search engines present the user with all that is popular on the web related to the search query. This can be too much to handle for any search query posed by user.

As a solution to this problem we have developed a system that uses stored personalized user preferences. It helps to simplify looking up information on the Web. In this system we maintain a user profile for improving searching for information over the Internet. From the stored user profile we add words to the search query that are related to the information the user is looking for. We therefore can get an enhanced search query. This query contains more information from the user profile which represents what the user would be actually looking for. This helps the user to get returns from the search engine that are more related to the user profile i.e. more relevant to his interests. The search results thus obtained are re-ordered and presented to the intended user. This gets the relevant search results into the top positions so the user does not have to browse through irrelevant search results before reaching the relevant search results.

An addition of a user profile on the client machine can be done either explicitly or implicitly, explicitly meaning that the user is aware of such an application in between him and the search engine while implicitly means that the user is not aware of such an application between him and the search engine. The system developed works implicitly.

In the user profile, the user can create a number of categories. The category is a label for a particular preferred user interest. Each category is an independent entity which retains the user's preferences in a different area of his interest. Each category uses a set of documents for profiling user interest. This user profile is later used for retrieving search results that are found appropriate for the intended category.

A generic search engine Google is used for testing the new system, it returns a two line piece retracted from the document around the used keywords with each of its search result called *snippet*. These snippets are used for classification of search result.

The whole procedure can be summed up in the following steps

i. The user selects a user defined category to search for information over the Internet.

ii. He then enters a search query into the system and the application modifies this query by adding relevant word from the stored user profile. It dispatches the modified query to the search engine.

iii. The search engine returns the results based on the modified query.

iv. These returned results are then resorted by the application on the basis of the probability values of snippets words returned with the search results in the order of relevance with reference to the user preference.

The application thus helps the user to modify his query with reference to the category he is searching in. Once the search is made it classifies the returns and presents him the most important ones at the top of the list. This saves the user valuable time, otherwise spent on reformulating his query and browsing through the search results returned for the reformulated query.

## 3.1 Query Modification

Generally a user finds it tough to express his information needs in just two or three words. This is again complicated by the fact that many words have different meanings in different contexts. Therefore we need to modify the query entered by the user in the context of the category created.

In order to modify the user's query we collect the user's preference in the form of URLs that match his area of interest and pre-process the data collected. Pre-processing deals with removal of punctuations and HTML tags from the document collected for the creation of the category.

For the additional word selection we use a function based on Bayes theory [19] [41] that helps us decide if a particular word is a good choice to be added to the search query.

The probability equation used for adding a new word to the search query is as follows:

Let G be the event that a particular document is good or relevant, and $W_1, W_2, W_3, ..., W_n$ be the search query words with the exception of $W_n$ which is added by the application. The other query words are provided by the user denoting his information request in a given category. The probability that the document conforms to a particular category is given by $P(G|W_1, W_2, W_3 \dots W_n)$. The order of search query words is assumed to be independent of other words. This means that the position of words in the search query is not considered.

$$\begin{aligned} &P(G|W_1, W_2, W_3, \dots W_n) \\ &= \frac{P(G, W_1, W_2, \dots W_n)}{P(W_1, W_2, \dots W_n)} \\ &= \frac{P(G)P(W_1|G)P(W_2|G, W_1)P(W_3|G, W_1, W_2)..P(W_n|G, W_1, \dots, W_{n-1})}{P(W_1)P(W_2|W_1)P(W_3|W_1, W_2)..P(W_n|W_1, \dots, W_{n-1})} \end{aligned} \tag{3.1}$$

The above equation signifies that probabilities of all the words are conditionally dependent on $W_1$, i.e. the probability of the words being present while the word $W_1$ is present.

Since the first word $W_1$ entered by the user can not be taken into consideration, we need to re-arrange the construction of $W_1, W_2, \ldots W_n$ to $W_n, W_1, \ldots W_{n-1}$ such that $W_n$ would be the word on which every search query word would be conditionally *dependent*.

This way we can add a word which is the best possible extension for the user query from the *good or relevant* set of the category. The above equation then transforms into the following form

$$= \frac{P(G)P(W_n|G)P(W_1|G, W_n) \ldots P(W_n - 1|G, W_n, W_1 \ldots W_{n-2})}{P(W_n)P(W_1|W_n)P(W_2|W_n, W_1) \ldots P(W_{n-1}|W_n, W_2 \ldots W_{n-2})} \tag{3.2}$$

Equation 3.2 as it computes interdependencies of the words, is computationally very expensive. Therefore we suggest a modification of the equation based on the Bigram Model [39] which states as follows:

For a sentence S, containing words $W_1, W_2 ... W_n$ without loss of generality we can express it as a probability

$P(S) = P(W_1)P(W_2|W_1)P(W_3|W_1, W_2) \ldots P(W_i|W_1 \ldots W_{n-1})$

In the bigram model we make an approximation that the probability of a word being present depends on the presence of the immediately preceding word. Therefore we approximate P(S) as

$P(S) = \prod_i^n P(W_i|W_{i-1})$

This equation assumes a word is dependent only on the previous word in the sentence or string this helps in reducing probability dependencies.

By using a concept similar to the bigram model we can evaluate the terms in Equation 3.2 terms in the form of $P(W_i|G, W_n)$, which is similar to the *feature or word* selection method employed in Equation 3.2. It shows a word $W_i$'s probability of being in a good document with a new word $W_n$ entered by the system. Since we consider the dependence of one word only we assume the new word $W_n$ to be the considered for dependence. This helps the system to estimate if the word $W_n$ is a good choice or not. Equation 3.2 then can be transformed into

$$\equiv \frac{P(G)P(W_n|G)P(W_1|W_n,G)P(W_2|W_n,G)\ldots P(W_{n-1}|W_n,G)}{P(W_n)P(W_1|W_n)P(W_2|W_n)\ldots P(W_{n-1}|W_n)} \tag{3.3}$$

Therefore the next word added is predominantly present in the relevant documents with other query words in the user profile. Hence the selection of a word that increases the value of the equation should enhance the return of search results. The method described above relaxes the independence assumption used by the Naive Bayes method. It performs better as the semantics of a language does not support an independence assumption: every word is dependent on other words in a sentence. Hence an independence assumption of Naive Bayes would not work effectively with two line snippets used for classification of documents. But in our method we use word pairs which helps in classification of a two line snippet. Therefore adding a new word $W_n$ conforms with not only the user category but also with the user search query. This would help the search engine return results in the notion of the search category.

In this section we saw how a word is considered from the user profile to be used for search query modification. But just modifying a query is not enough. We still would have to check how the search engine would perform in case of a modified query. We therefore estimate the returns if match the user created category. In the next section we shall see how we estimate the behavior of a generic search engine before a search query is dispatched. This is important as we can then predict if an addition of a word was helpful or not.

## 3.2 Estimating the Return

A generic search engine does not return all of the search query words presented to it in the search results. It generally pulls out a page that has a good number of hits and with one or all of search query words present in its snippet. This is a very generalized way for presenting search results with which not many users could relate to. We observe the following average returns of search query words for a generic search engine.

Table 3.1. Probability of the words returned in the search results for query

| Query Length | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 1.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 0.0 | 1.0 | 0.0 | 0.0 | 0.0 |
| 3 | 0.0 | 0.15 | 0.85 | 0.0 | 0.0 |
| 4 | 0.0 | 0.1 | 0.6 | 0.3 | 0.0 |
| 5 | 0.0 | 0.2 | 0.4 | 0.35 | 0.05 |

The values in Table 3.1 show us the average probability of returning query words in search results. Using these values as *probability of number of words returned* for the expected search results we develop a *utility function* for estimating the quality of search results returned by the search engine. This function will help us estimate the ordered search results returned on addition of the new word added.

The probability values in the Table 3.1 denote the estimated variations of query words present in the search result returned for a particular query length. For example, consider the third row in Table 3.1. 85% of the search result contain three words and 15% contain two words. Therefore 85% of the search results would be expected to contain all three words in the query, including the new word added by the application to the user search query, and the remaining 15% would contain two of the words of the three word search query.

For every search query we make, we try to estimate the possible outcomes of all possible combinations of the search query words that could be used by the search engine. For example, for a two word query the possible combinations would be $\{W_1\}$,$\{W_2\}$,$\{W_1, W_2\}$. We compute each combination's probability value. Using Equation 3.3 we then multiply it with the corresponding average words returned from Table 3.1, i.e. the number of search query words in the returns of the search results. This is done for the fact mentioned earlier that not all the words presented as search query are returned by the search engine. The average number of words returned in the snippets per search query length is an important factor to be considered as presence of relevant words in the snippet received would help the relevant search result to get a better probability value.

When multiplied by a *metric* used by the user for judging the order of the search results returned, this would help us to know if the addition of the word would help the prospective search. This equation is used as the *utility* equation.

The utility equation is developed as follows

$queryWords = \{W_1, W_2, \dots, W_n\}$

This is a set of words composed of the words that the user uses for searching information on the Internet.

$queryWords_i \in \{queryWords_i | 1 \leq i \leq 2^{|queryWords|}\} = 2^{queryWords}$

These are the possible subsets of the queryWords. We use this set to determine which of the combination would give us the best ordered search results on submitting to the generic search engine.

Let N be the number of combinations of the words in the query and $N_s$ the number of combinations of queryWords, which have a probability to indicate a good document higher than 0.5.

$N_s = |\{queryWords_l | P(G|queryWords_l) \geq 0.5\}|$

$\forall\ 0 \leq i \leq N\ queryWords_i \in queryWords$

$\forall\ 0 \leq i \leq N\ P(G|queryWords_i) \geq P(G|queryWords_{i+1})$

Within the set of returned word combinations, combinations are sorted by their likelihood to indicate a relevant document. This is used in re-ordering of the search results. The highest values represents the best possible returns.

Let M be number of slots or positions in which the search results are returned. In our experiments we have chosen it as 20 slots. This is a generic number of results a user would be able to get in one page. Therefore the utility of a feature or a word is calculated as follows:

$$\begin{aligned} Utility(W_n) = \sum_{k=1}^{N_s} P(G|queryWords_k) * \Bigg( \Big( \sum_{m=\lceil 1+\sum_{l=1}^{k-1} M_l \rceil}^{\lfloor \sum_{l=1}^{k} M_l \rfloor} Metric_m \Big) \\ + \big( \lceil 1 + \sum_{l=1}^{k-1} M_l \rceil - \big(1 + \sum_{l=1}^{k-1} M_l\big)\big) * Metric_{\lfloor 1+\sum_{l=1}^{k-1} M_l \rfloor} \\ + \big( \sum_{l=1}^{k} M_l - \lfloor \sum_{l=1}^{k} M_l \rfloor \big) * Metric_{\lfloor 1+\sum_{l=1}^{k} M_l \rfloor} \Bigg) \end{aligned} \tag{3.4}$$

where $M_k$ is the expected number of search results that contain word combinations $queryWords_k$. Metric is the value assigned to a relevant result in slot.

The utility equation Equation 3.4 represents the real world scenario. The search engine returns only some of the search query words in the search results for a query. We consider all combinations of query words as a set of $\{queryWords_k\}$. This is done to ensure that the combination features or query words which could contrast well between the relevant and the non relevant documents is taken into account. The $P(G|queryWords_k)$ signifies the probability of “k” number of query words returned by a search query in relevant documents. We compute the probability values for each of the query words combination and sort them in their descending values. This order tells us the best possible order of relevant search results. Each probability values of the combination is assigned to a particular number of slots. This slot value computation is done by the following equation :

$$M_k = P(|queryWords_k|) * \frac{1}{^{|queryWords|}\mathrm{C}_{|queryWords_k|}} * M$$

The notation $^n\mathrm{C}_c$ , represents the number of combinations of length c in a query of length n. $M_k$ refers to the number of slots assigned to a query word combination $queryWord_k$ in case they are relevant i.e if the $P(G|queryWords_k)$ is above 0.5.

The later part of Equation 3.4 deals with the fact that the slot assigned for each probability values need not be a whole number. In case the number of slots assigned to a particular combination is not a whole number we consider partial slots and compute a score according to the *metric* for the partial slot rather than a whole slot shown in Figure 3.1.

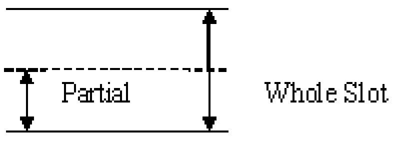

Figure 3.1. Slot Allotment.

This equation will therefore assure that the word being added is enhancing the query based on the category selected for searching. Therefore the *utility equation* considers the number of words returned , the positions the user likes his relevant search results in and the query word enhancement based on the returns of the search engine.

A *metric* followed by the user allows him to indicate the relevant order of the search results. The search results on the top of the list are assigned a positive reward. The *metric* can be selected by the user and capture the user's way of looking at the ordering of search results and the relevance of search results to the user category. For example, consider a search result ordering *metric* which assigns weights of 200,190,180, . . . , 20,10 for the 20 results in descending order returned from the search engine and a weight of -1 for every irrelevant search result. This would be a monotonic *metric*. So if there were two search result sets, say in which the first 10 results are relevant and the rest of the results are irrelevant. The *metric* assigns a weight of 1550-10 =1540. In another case if only the alternate search results were relevant the weight assigned would be 1100-10=1090. Therefore the search result set which has a continuous relevant search results in the top few position would get a higher score, assuring that the relevant search results are always placed on the top locations so that the user does not have to browse through irrelevant search and find relevant search results down the order.

The *metric* allows the user to assign a value to the ordering of relevant search results done by the system. That is, if the user likes most of his relevant search results in the top three positions only he could assign a higher positive value for those positions and negative values for the remaining positions. This will help the system select words that would maximize the value of the metric. The word with the highest value of the metric would be selected for addition.

As we see in the Equation 3.4 the word with the highest utility would be the ideal choice for adding to the search query. With this equation we have also estimated the returns of the search engine which generally bases its returns on the popularity of the document and the number of search query words it contains. It tells us how likely the returns would match the user category. The word selection procedure is based on partial independence assumption. The words in the query are dependent only on one word i.e. the next word to be added by the application.

Since it is expensive to download the entire search result document we make classification of the search result if it belongs to the category or not by classifying its snippet

```
Let D be the set of documents specifying the user's preference for the category, containing both positive and negative documents.
Let MAX and the MIN be the maximum and minimum of the probability values.
      $D = D_1, D_2, \ldots, D_n$
Let F be the set of features found from the documents, in order to get to the candidate features which are category specific
Check for the MAX and the MIN of the probability values of each feature
    **For** every feature
        **If** (maxOfFeature > MAX)
            MAX = maxOfNextFeature
        **ElseIf**(minOfFeature > minOfPreviousFeature)
            MIN = minOfNextFeature.
            **ElseIf**Remove the feature,did not satisfy category condition between MIN and MAX
    **End For**.
Let C be the set of candidate features,which satisfy the MIN and the MAX condition set above.
      $F = F_1, F_2, \ldots, F_n$
    **if**matching word is found in the query
            Remove the word from the candidate features
    **For** every search query
            Calculate the utility value U(i) for each of the candidate features
    **End For**
The candidate feature with the maximum utility would be the next word to be added.
```

Figure 3.2. Procedure for selecting the next word for the search query.

```
Let S be the set of words contained in the snippet.
    $W_1, W_2, \ldots W_n$
Find the words in the feature set that occur in the snippet.
These words would form the classification criterion
**For** each snippet
    Evaluate the probability value P(G $|W_1 \ldots W_n$) Equation 3.3
**End For**
Present the probability values in descending order
```

Figure 3.3. Re-Ordering Algorithm.

i.e. a two line piece retracted from the document around the used keywords. This snippet represents the document with reference to the query terms. We then use the *Re-Ordering Algorithm* in Figure 3.3 to re-order the results returned from the search engine. This algorithm is based on Equation 3.3. The re-ordering procedure is done on the probability values generated using Equation 3.3. These values represent the relevance of the search results with respect to the category. We re-sort these values in descending order. This will put the relevant search results on the top positions and irrelevant ones on the lower positions.

The application then returns the re-ordered search results. This ascertains that the search results has the best possible match to the category selected and the relevant search results are placed in the top positions of returned search results.

## 3.3 Storing User Preferences

To make the application efficient, we need to store the user preferences on the client machine. We first collect a list of URLs which the user finds appropriate for a given category he is searching in, this forms the *positive set.* In order for search results to be classified accurately we also collect the URL's which are inappropriate for the category this forms the *negative set.*

### 3.3.1 Storing User Categories

We download and store the documents which are classified as *positive* or *negative* in a particular user profile for a given category in the corpus. This forms the basis for maintaining the user preference for the category. By processing these documents we generate a set of words, the *feature set*, that are directly related to the category. These are the words that are then used for enhancing queries the user enters in a generic search engine. Figure 3.4 depicts how the application maintains the user category.

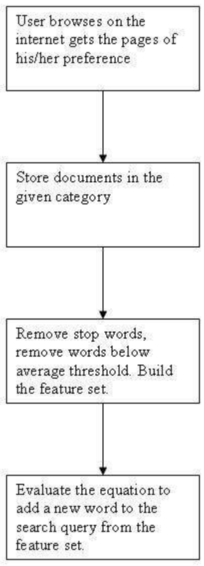
User browses on the internet gets the pages of his/her preference
Store documents in the given category
Remove stop words, remove words below average threshold. Build the feature set.
Evaluate the equation to add a new word to the search query from the feature set.

Figure 3.4. Collecting and Preprocessing of User Category Data.

We begin by first removing stop words and punctuation marks and form a *feature set* consisting of the words in the documents, of which one is added to the search query.

Since each category has a large number of feature words, it would be computationally expensive to compute Equation 3.3 for every search the user makes. On evaluation of Equation 3.3 we observe that the equation could be broken down in to the following terms

$$\frac{P(W_2|W_1, G)}{P(W_2|W_1)} \tag{3.5}$$

This term is a ratio of the *probability* for two words $W_1$ and $W_2$ , the words in the feature set. This term signifies the presence of $W_2$ in a *good* or *relevant* document with the presence of $W_1$ in the document.

We compute the probability values for each of the words depending on each and every other word in the *good* and in the *general* set. We store these values for future searches in the given category.

Equation 3.5 represents a ratio of both $W_1$ and $W_2$ being together in a *good* document set of the category. This value tells us about the relation between these two words. A higher value indicates that these words are mostly occurring together in the *good* document set of the category. A low value would indicate that these words do not occur together frequently in the *good* document set of the category.

It would be computationally expensive to check for each and every word for expansion of query. We use the method discussed in Algorithm 3.5. Here we generate a set of words to be considered for addition, for the enhancement of the query. These words are the best possible addition for the query. This is a pre-selection procedure for addition of words to the search query. The words selected in this method will be tested for their utility values and then the word with the highest utility from Equation 3.4 will be added to the query.

### 3.3.2 Pre-Selection Procedure

For extracting the best possible word for extension of a search query so that the search engine returns the most relevant search results for a given category, we have devised a novel method. We look for pairs of words that lie between the range of the highest *value* of a pair formed by words in Equation 3.5 entered by user and words in the feature set and the lowest *value* of pair formed by a query word with other words in feature set. The word pair forming the highest and the lowest value of the probability equation need not be the same. These are the prior conditions for words to be selected for consideration to be added to the search query. The words which satisfies the upper and the lower limit set by words in the search query and the feature set are the best possible words that Equation 3.3 would select for addition. This priority condition helps us save computation time of trying each and every word in the feature set. Algorithm 3.5 shows how the pre-selection condition for selecting words is implemented.

```
Let Max and Min be the maximum and minimum *probability value* for a pair. And
addWord be the list of words
    **For** number of pairs of words
        If probabilityValue(nextPair) $\geq$ Max)
            Max=probabilityValue(nextPair)
            Add the word pair to addWord
        ElseIf probabilityValue(nextPair) $\geq$ Min
            Min = probabilityValue(nextPair)
            Add the word pair to addWord
        EndIf
    **End For**
```

Figure 3.5. Procedure for finding the best possible word for addition.

### 3.3.3 Entropy Calculation

After calculating the utility according to Equation 3.4 for the pre-selected words for addition to the search query, it can occur that two words result in the same utility value. We use entropy or information gain to break the tie.

*Entropy* is a measure for information gain. Given a collection S, containing *positive* and *negative* examples of some target concept, the entropy of S relative to this boolean classification is

$$Entropy(S) \equiv -p_{\oplus}log_2 - p_{\ominus}log_2 p_{\ominus} \tag{3.6}$$

where $p_{\oplus}$ is the proportion of positive examples in S and $p_{\ominus}$ is the proportion of negative examples in S. In all the calculations involving entropy we define 0 log 0 to be 0. *Entropy* is a measure of the impurity in a collection of training examples. We use this to measure the effectiveness of an attribute in classifying the training data. The measure used is called *information gain*, and is simply the expected reduction in entropy caused by partitioning the examples according to this attribute. Information Gain, $Gain(S, A)$ of an attribute A, relative to a collection of examples $S$, is defined as

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{3.7}$$

where $Values(A)$ is the set of all possible values for attribute A, and $S_v$ is the subset of S for which attribute A has value $v$ [23].

## 3.4 Summary

In this chapter we saw how a search query is modified by our application. We first evaluate the probability equation in Equation 3.2 to check for additional word to be added to the search query. Based on the Bigram model we assume that the word depends only one word i.e. the word being added to the system, saving expensive computation time. We also observe that the search engine does not return all the words in the search query in its search results. Therefore in the utility equation, Equation 3.4, we have a term considering the average number of words returned by the search engine in each of its search results for a given query length. In the utility function we also consider the users way of re-ordering search results by having a *metric* defined by the user in the utility function Equation 3.4. The word with the highest utility value would be then added to the search query for performing the search.

Finally after we execute the search we re-order the search results using the re-order routine (Algorithm 3.3) to help the user find the relevant search results on the top positions of the search result set.

The application as a whole is intended to save a user valuable time over the Internet while searching for information. With the user category, we know the user's preference and it helps in reformulating his query according to the category selected. It adds a personalization effect and delivers relevant search results in the top locations.

In the next chapter we shall see the experimental analysis of the method discussed in this chapter and compare it with other methods used in enhancing web search.

# CHAPTER 4

# EXPERIMENTS AND RESULTS

In this chapter we have evaluated our application in two user categories. The first category is that of sports where the user is interested in a particular sport: *Cricket*. The second category is that of documents containing information regarding *K12* (Kindergarten to Twelfth grade) subjects and concepts students learn, and career choices they could make after graduation. When the user selects a category for searching information we first enhance the search query according to a feature set developed for that particular user category. Then we re-rank search results generated from a generic search engine. This would provide search results relevant to the category in the topmost positions in the list of search results returned by a generic search engine.

## 4.1 Collection of Data

The data is provided in a list of URLs. These are the web pages that a user is interested in for a particular user category. These URLs form the data set for the users preference. This dataset contains both the *positive* and *negative* set. *Positive* web pages are those pages that are relevant to a user category and *negative* web pages are not relevant to the user category. The experiments were conducted on a varying number of web pages, for example 60 for the first user category and 200 for the second user category: this number of pages collected need not be a specific number. A user could collect any number of documents. The system starts building a category with the number of documents available to it. It is also important to note here that while the documents were provided a priori for these experiments, they could also be collected on-line during the user's normal search activity. The documents pointed to by these URLs were then downloaded and stored on the client machine. We pre-process these documents to remove stop words and punctuation marks. These documents are then added to a corpus specifying the user category.

## 4.2 Implementation of the Application

We have developed this application in Java, J2SE 1.4.2. Other tools used were

1. WEKA ( Waikato Environment for Knowledge Analysis) A Machine Learning tool from the University of Waikato, New Zealand. This tool helped us in *filtering* stopwords and machine learning algorithms. Of these machine learning algorithms we use Naive Bayes for comparison to the system developed by us. It also has data visualization procedures that help in scrutinizing data for information regarding the words and the dependencies they share among each other.
2. HTMLParser (http://sourceforge.net/projects/htmlparser) is an open source tool that helped us in the extraction of text from HTML pages. It parses HTML pages and removes HTML tags for text extraction. This text is used as a string which is added to a corpus. This string is then preprocessed to extract information. In the form of words which we use to define features for each category.
3. JUDGE (Java Utility for Document Genre Eduction)
   http://sourceforge.net/projects/judgeproject is an open source framework for classification of documents. It uses WEKA for machine learning algorithms. The documents collected after HTML parsing are added to a user category in the definition, if the given document is relevant to the user category or not, as mentioned by the user. Using this utility we extract features (words) out of the collected documents.

4. GoogleAPI: Google, a generic search engine. It provides a Web API service which connects a local machine to Google. Using this API we query Google and get search results which we would see on the Web.

## 4.3 The Search Return

Using the feature selection method as discussed in Chapter 3, the feature selection algorithm (Algorithm 3.2) modifies the query and collects the search results given by the search engine. In this report we use Google as a generic search engine. This search engine currently seems to be the most popular for retrieving quality results. It also provides an API through which we can query it from a local machine.

### 4.3.1 Search Response

Each time we issue a search request to Google, a response is returned in a particular format. The meaning of these values is described as follows

**estimatedTotalResultsCount**- The estimated total number of results that exist for the query.

**resultElements** - An array of resultElement items. This corresponds to the actual list of search results.

**searchQuery** - This is the value of <q> query terms for the search request.

**startIndex** - Indicates the index (1-based) of the first search result in resultElements.

**endIndex** - Indicates the index (1-based) of the last search result in resultElements.

### 4.3.2 Result Element

The following are the fields that are present in the resultElement data structure returned from Google

**URL**- The URL of the search result, returned as text, with an absolute URL path.

**snippet** - A text excerpt from the results page that shows the query in context as it appears on the matching results page. This is formatted HTML and usually includes <b> bold tags within it. Query terms will be highlighted in bold in the results, and line breaks will be included for proper text wrapping.The query term does not always appear in the snippet.

**title** - The title of the search result, returned as HTML.

## 4.4 Category 1: Sports

Here we test our application in the category of sports. In this category the user is interested in *cricket*, a popular sport in my home country India. We have provided the system with 60 (relevant :35 and not relevant: 25) documents as a preference of the user. Using these 60 pages the application generates a feature set which it uses to modify the search query and classify the search results returned from the search engine.

In the results shown later we discuss efficiency of the system on two parameters the number of relevant results obtained, and the utility value got on submitting the modified search query.

The *metric* is used to judge if the order of relevant documents in the search results is better than the one given by the original query order. It is used in the definition of the utility function, in Chapter 3, Equation 3.4. The one we use is as follows: for the first position in the search query results the score assigned is 200, for the second 190 and so on until the twentieth for which it is 10. This way we can measure if the ordering of relevant documents improves or not.

Table 4.1 shows the various search queries tried out on the system. Each of these search queries were modified and the results was tabulated accordingly. Table 4.1 shows the number of relevant documents from among the first 20 search results returned by

Table 4.1. Category 1: Query Improvement using our classifier

| Query | Word Added | Relevant Before | Relevant After | Utility |
|---|---|---|---|---|
| Bangladesh batting | oval | 7.0 | 12.0 | 735.0 |
| England cricket | plays | 8.0 | 12.0 | 735.0 |
| Sachin Tendulkar | plays | 15.0 | 14.0 | 315.0 |
| Cricket batsmen | plays | 11.0 | 11.0 | 735.0 |
| New Zealand cricket team | pak | 9.0 | 12.0 | 315.0 |

the search engine. Normally a user is incapable of browsing through all of the results returned by the search engine as the number of results runs into millions for some queries.

We have tried the following five queries in category 1 on our system to judge its performance

1. *Bangladesh batting*: The user is trying to find information for the batting of the country Bangladesh in cricket. The generic search engine returns this in the context of *batting* because it is a more popular word. Our application modifies this particular query into *Bangladesh batting oval.* This query gets us the search results that the country Bangladesh has played in cricket grounds that have a common name as *oval.*
2. *England cricket*: This query signifies involvement of the country England in cricket. The word added by the application is *plays.*
3. *Sachin Tendulkar*: This is the name of the most popular Indian cricket player. Hence by querying we get a huge number of results about his achievements but also down the order we get results of person who are similarly named. The application adds a word *plays* this word adds a cricket sense to it. But while re-ordering we get a lower score than before. This shows that there is also a possibility of getting a lower score on re-ordering. This is partly due to the fact the original query might

have had a decent number of relevant documents hence re-arranging causes lowering of score.

4 *Cricket Batsmen* : By this query one tries to find out more about the batsmen in the sport. By the presence of the word *cricket* in the query, the query is definitely directed towards cricket. But the results returned by the generic search engine include certain unnecessary results from news sites, and biology sites which are eliminated by adding the word *plays* which emphasizes the query's intention of collecting information about batsmen in cricket only.

5 New Zealand cricket team: This query is directed towards knowing the performance or composition of the team. The application adds the word *pak* which is a short form for the country *Pakistan.* This word gets us the results that signify the games played by the two countries.

Table 4.1 shows how the application modifies the query and improves the returned results. The Utility value as discussed in the previous chapter is a score for re-ordered sets of the returned search results. The table shows the expected utility value for the modified query. This utility value signifies the order and the relevance of the search results that could be expected on submission of the modified query to a generic search engine.

As we see from the table the relevant results improve after addition of a word from the feature set. The number of relevant results improve in most cases considered as shown in the Figure 4.1.

To evaluate the ordering of results, we will compare three types of orderings:

1. Original query score: This is the original ordering score as given by the search engine. We have not modified neither the query or the ordering of the search results. This method is not very useful as a generic search engine normally treats

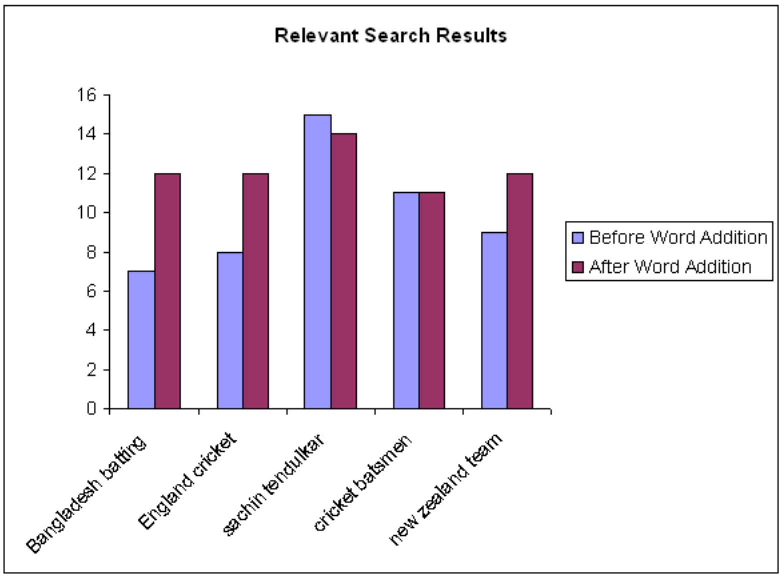

Figure 4.1. Category 1: The no. of relevant documents in search results out of first 20.

all the users similar. Therefore the relevant search results for some users may be provided in the lower positions in the list of the search results.

2. Modified query score: This is the score of the order of search results when the additional word is added and the search results are provided by the generic search engine as is, without trying to re-order them. This score suggests the improvement, if any, in getting more relevant search results. The additional word adds a *category* sense to the query. Therefore we believe it would add more relevant search results to the list returned by the search engine.
3. Modified query and Re-ordered score: After we have added our word and re-ordered it with the classification probability based on the returned snippets. This is the score of the final order of the search results. It shows if an improved result order of the relevant search results is obtained. This is because the relevant search results which were in the lower ranks are pulled up in the top positions by the application, thus increasing the overall ordering score.

Table 4.2. Category 1: Result set ordering comparison

| Query | Word Added | original order | order score | re-order score |
|---|---|---|---|---|
| Bangladesh batting | oval | 1080 | 1120 | 1300 |
| England cricket | plays | 990 | 1440 | 1550 |
| Sachin Tendulkar | plays | 1420 | 1810 | 1620 |
| Cricket batsmen | plays | 1250 | 1240 | 1700 |
| New Zealand cricket team | pak | 1200 | 1290 | 1720 |

Table 4.2 shows the score of relevant documents for three cases: the original query, modified query, and the re-ordered search results on modified query.

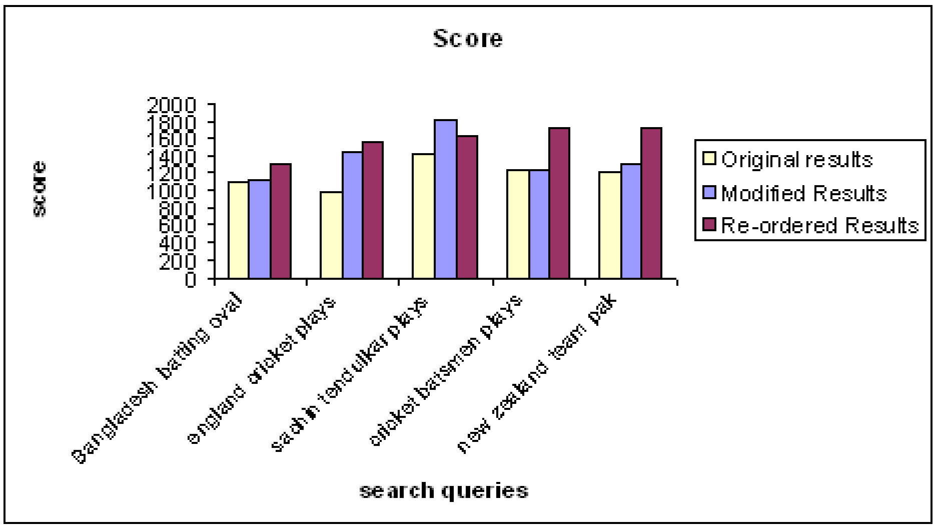

Figure 4.2. Category 1: The Ordering of relevant documents in search results.

As shown in Figure 4.2, the re-ordering score gives more relevant search results except in the one case in the third search query. This is an outlier as shown in Figure 4.2. In the third query, *sachin tendulkar*, an additional word does not necessarily increase relevant search results as shown in Table 4.1. As a result, re-ordering does not increase the score. Also in the case of the fourth query *cricket batsmen* an additional word in this case does not add more “cricket sense” to it. In this case we find that number of relevant search results is the same as before adding a word from category created by user. But by re-ordering we are able improve the score i.e. we are able to provide relevant search results in the top positions.

Overall, the results show that the method presented here in all cases improved the quality of the search results compared to the original search.

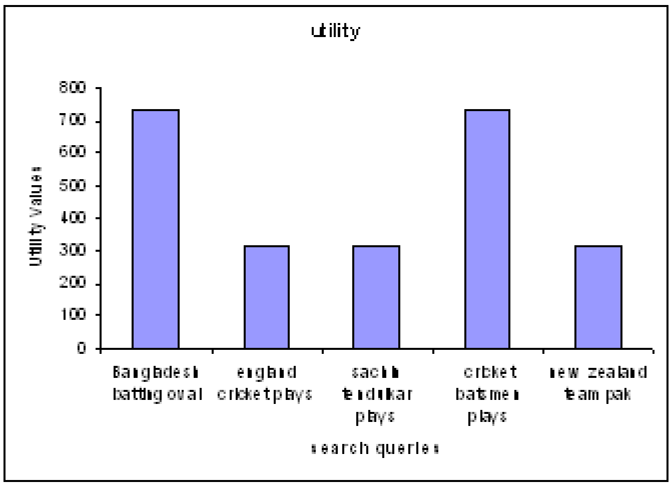

Figure 4.3. Category 1: Estimate of relevant search results.

### 4.4.1 Comparison with other methods

#### 4.4.1.1 Naive Bayes

Naive Bayes is one of the best methods for classification. But its usage in the domain of text categorization is not very high. It assumes that all the features of a document are independent of each other. It builds an automatic classifier based on Bayesian theory discussed in Chapter 2. While comparing the number of relevant documents generated by the NB classifier to our classifier we find that our classifier out performs the NB classifier.

As shown in Figure 4.4 we see that our method always improves the count of the relevant document versus the NB classifier. The NB classifier does well in one case only. In the fourth search query the Naive Bayes classifier is able to add a relevant word to the query "cricket batsman" which improves its number of relevant search results count.

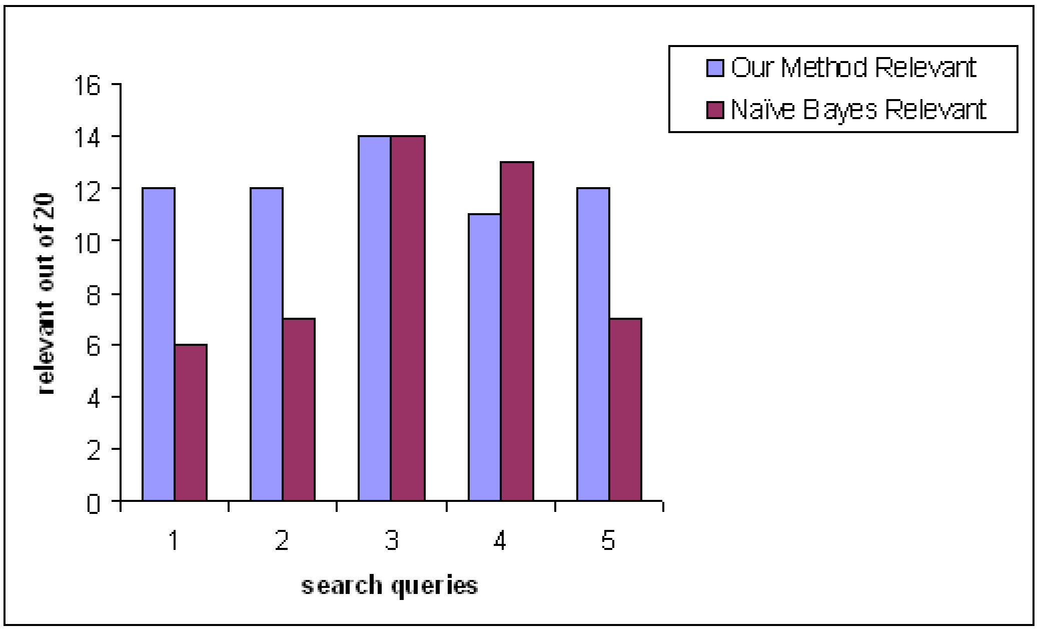

Figure 4.4. Category 1:Comparison of number of relevant documents with Naive Bayes.

On comparing the ordering of the search result by both the classifiers, Figure 4.6 shows us the various schemes in which the ordering is done. We see that our method outperforms the Naive Bayes classifier, as the independence assumption does not allow it to consider other relevant words. It considers words only with a high probability in conjunction with the search query. While in our case we consider pairs of words that gives us more words to consider for addition to the search query as shown in Figure 4.6.

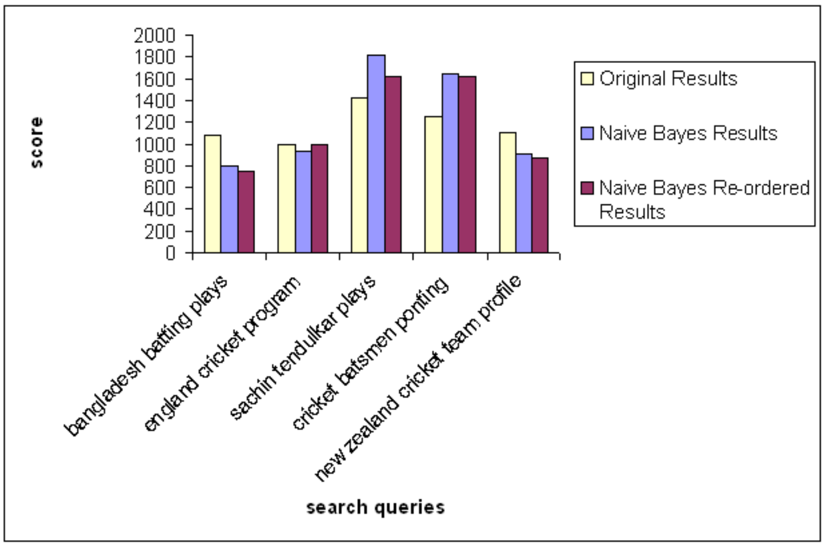

Figure 4.5. Category 1: Using Naive Bayes ordering of relevant documents in search results.

Table 4.3. Category 1: Ordering Search Results using Naive Bayes Classifier

| Query | Word Added | original order | NB method | re-order score |
|---|---|---|---|---|
| Bangladesh batting | oval | 1080 | 800 | 750 |
| England cricket | program | 990 | 930 | 990 |
| Sachin Tendulkar | plays | 1420 | 1810 | 1620 |
| Cricket batsmen | ponting | 1250 | 1640 | 1620 |
| New Zealand cricket team | profile | 910 | 1720 | 870 |

The re-ordering of the NB classifier is of limited quality as it considers each word independent. The snippet gives it really small data to test on. But in our method the small dataset is not a problem as we have already established relations between each and every word in the dataset. This gives our method an advantage in the case of small test cases i.e. snippets to classify.

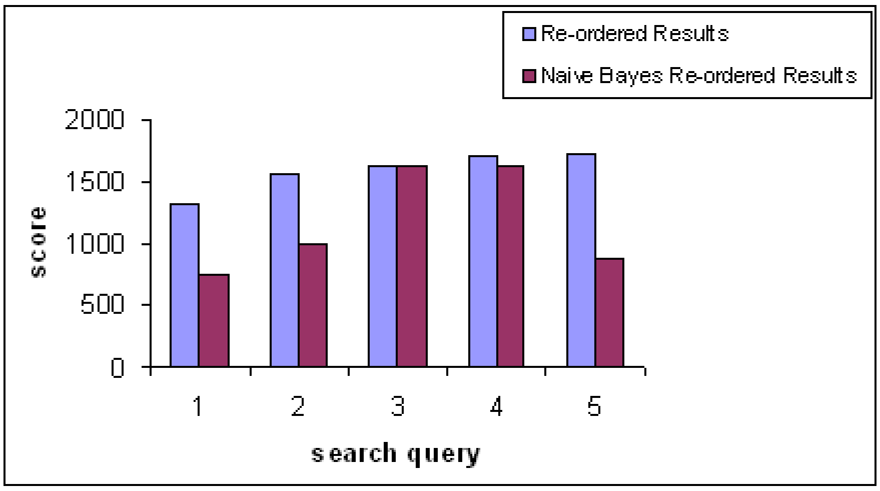

Figure 4.6. Category 1: Comparison of ordering with Naive Bayes.

#### 4.4.1.2 Information Gain

Here we compare our system with that of a system that adds only one word which has the highest information gain value for the given category. Such a system can identify only a single word at a time. Hence irrespective of the query it always adds the same word. This word has the inherent property of classifying the entire document better than any other word in the feature set. Therefore it is selected as the representative of the category to enhance our search query.

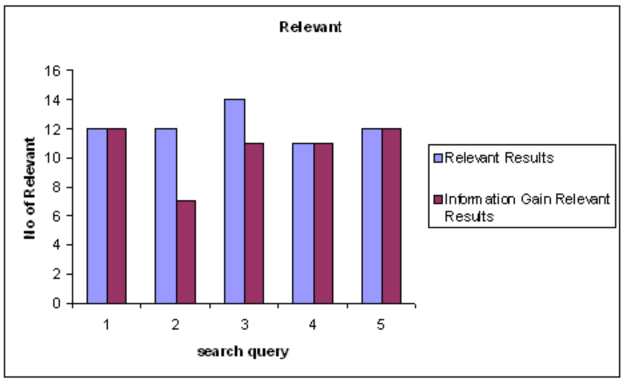

Figure 4.7. Category 1: Relevant documents with Information Gain.

The Figure 4.7 shows the performance of information gain system. In the figure we see that a common one word addition does not identify the category. Therefore the number of relevant search results are not improved on addition of this word.

The word added in this case is “zone” this word has the highest information gain of 0.64 in the category. The number of relevant documents added by such a system would not be significant as it does not consider the nature of the query before adding the word. As shown in Table 4.4 it partially improves the score of the query.

Table 4.4. Category 1: Search Result Ordering using Information Gain

| Query | Word Added | original order score | order score |
|---|---|---|---|
| Bangladesh batting | zone | 1080 | 1490 |
| England cricket | zone | 990 | 640 |
| Sachin Tendulkar | zone | 1420 | 1470 |
| Cricket batsmen | zone | 1250 | 1290 |
| New Zealand cricket team | zone | 1100 | 1350 |

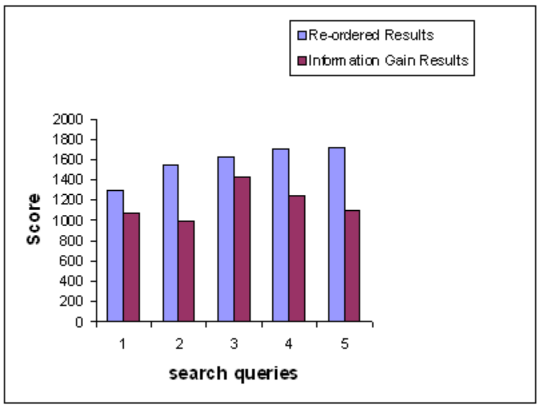

Figure 4.8. Category 1: Comparison of ordering with Information Gain.

Figure 4.8 shows the comparison of Information Gain score with the system we devised, showing that our system outperforms information gain for all queries.

## 4.5 Category 2 : K12 education curriculum

This category is regarding subjects and concepts studied in kindergarten to 12th grade. We have collected pages which would be of interest to such students. This category contains 122 relevant pages and 95 not relevant pages. Over this document collection we build the corpus and generate the feature set of words which identify with this user category.

Table 4.5. Category 2: Query Improvement using our classifier

| Query | Word Added | Relevant Before | Relevant After | Utility |
|---|---|---|---|---|
| K12 education | look | 6.0 | 10.0 | 735.0 |
| kindergarten curriculum | license | 6.0 | 4.0 | 735.0 |
| K12 english teachers | enrollment | 5.0 | 7.0 | 1575.0 |
| employment K12 | progress | 2.0 | 2.0 | 735.0 |
| K12 mathematics | build | 3.0 | 4.0 | 735.0 |

As shown in the Table 4.5 we have tried five queries to test the application in category 2. Figure 4.9 shows us the relevance of search results for the first twenty search results returned by the search engine.

Table 4.6. Category 2: Comparison of result set ordering.

| Query | Word Added | order score | re-order score | original order |
|---|---|---|---|---|
| K12 education | look | 830 | 980 | 1180 |
| kindergarten curriculum | license | 490 | 380 | 580 |
| K12 english teachers | enrollment | 530 | 710 | 780 |
| employment K12 | progress | 200 | 140 | 200 |
| K12 mathematics | build | 400 | 490 | 480 |

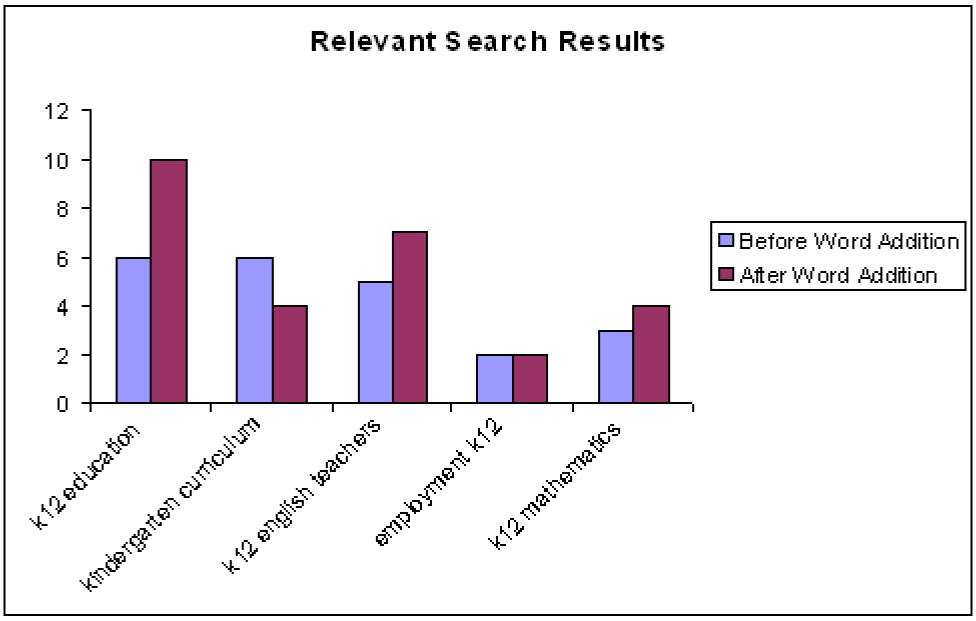

Figure 4.9. Category 2: The number of relevant documents in the search results.

Table 4.6 shows us the information in tabular form of how the application modifies the query for *Category 2* and improves the returned results. As seen in the table, queries are modified by adding a word, most often the number of relevant search results improves.

1. *K12 education*: The user enters this query to learn about the K12 education system. Most of the pages that are returned by the generic search engine are related to computer sellers selling for K12 students. Our application pulls up the pages about the K12 education by adding a relevant word "look" for this query.

2. *kindergarten curriculum*: This query by the user is to learn the topics covered in kindergarten grade. Since a generic search engine caters to users all over the world the search engine ends up showing kindergarten curriculum in other countries, too, which is irrelevant for the user's search for a kindergarten curriculum of a local school.

3. *K12 English teachers*: By this query the user needs to know if there are teachers available for English subjects. Since the query mentions English teachers, it pulls

up certain certification courses available for teaching English. By adding a relevant word from the feature set we are able to get links for English tutoring over the web and for improving English language skills for the user.

4. *employment K12* : By this query the user is trying to find out what kind of employment opportunities are available upon graduating from high school. A generic search engine mostly returns employment opportunities available at a school only. After adding a relevant word, the generic search engine is able to return certain relevant links.

5. K12 mathematics: This query is for knowing more about the K12 mathematics curriculum which could help students improve their math skills but most of the links generated in top 20 links contain lesson plans for teachers teaching math. By adding "build" to the search query we can get links for improving math skills of students.

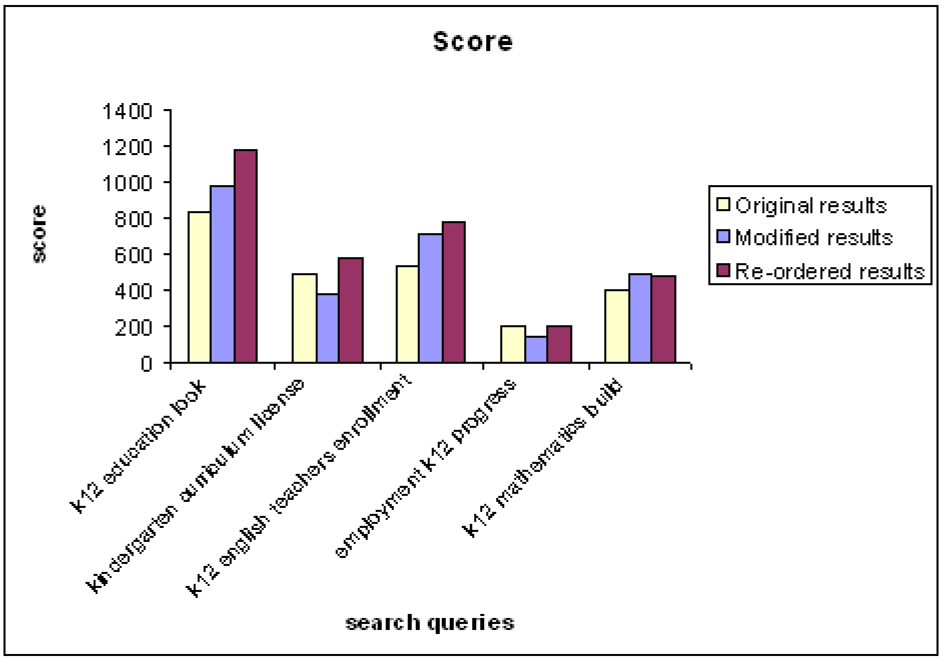

Figure 4.10. Category 2: Ordering of relevant documents in the search results.

As shown in Table 4.6 the re-ordering score gives a better structure for the search results. Also, by adding a relevant word to the search query it improves its count of relevant search results, except in the fourth case where the performance is equal to the original query. In this case the word added by the system did not add to the meaning of the query.

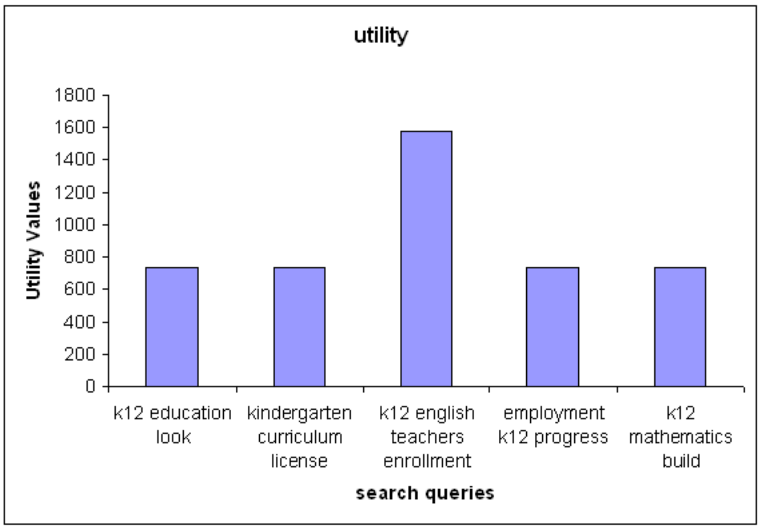

Figure 4.11. Category 2: Utility estimate of the relevant search results.

We see that the utility value is considerably higher for the query "K12 english teachers" as the word added, "enrollment", changes the meaning of the query in the K12 students direction. This change is significant hence the higher utility value.

### 4.5.1 Comparison with other methods

#### 4.5.1.1 Naive Bayes

Table 4.7 shows the number of relevant documents retrieved using the Naive Bayes Classifier. It normally is not able to get a higher number of relevant documents than the original query.

Table 4.7. Category 2: Query Improvement using Naive Bayes Classifier

| Query | Word Added | Relevant Before | Relevant After |
|---|---|---|---|
| K12 education | license | 6.0 | 1.0 |
| kindergarten curriculum | license | 8.0 | 3.0 |
| K12 english teachers | license | 5.0 | 1.0 |
| employment K12 | license | 2.0 | 2.0 |
| K12 mathematics | catalog | 4.0 | 3.0 |

As shown in Figure 4.12 we see that our method always improves the count of the relevant document versus the NB classifier. The NB classifier does well in certain cases only.

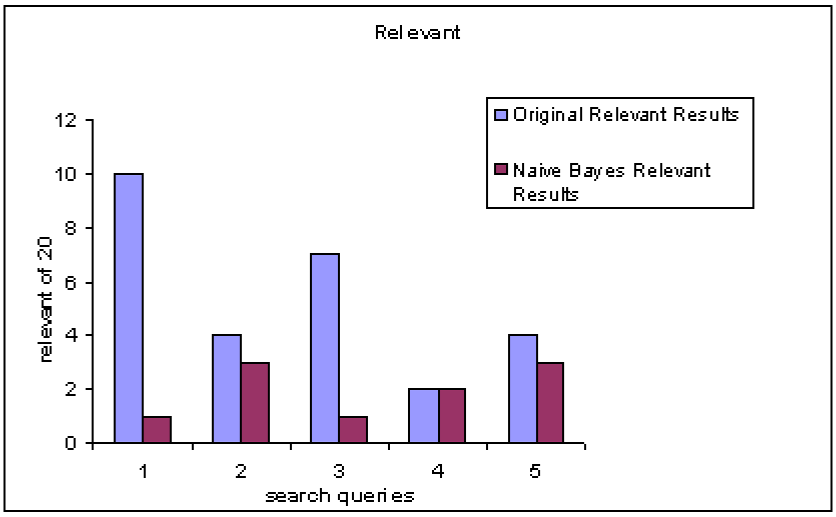

Figure 4.12. Category 2: Relevant documents with Naive Bayes.

Figure 4.13 shows us the ordering of relevant retrieved using the Naive Bayes Classifier.

Comparing the number of relevant search results returned by our classifier and Naive Bayes we see in Figure 4.14 that the ordering of the NB classifier depreciates from the original query since it considers each word as an independent entity.

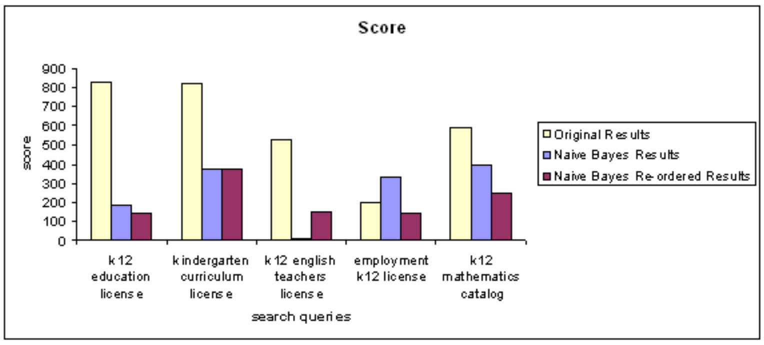

Figure 4.13. Category 2: Ordering of relevant documents with Naive Bayes.

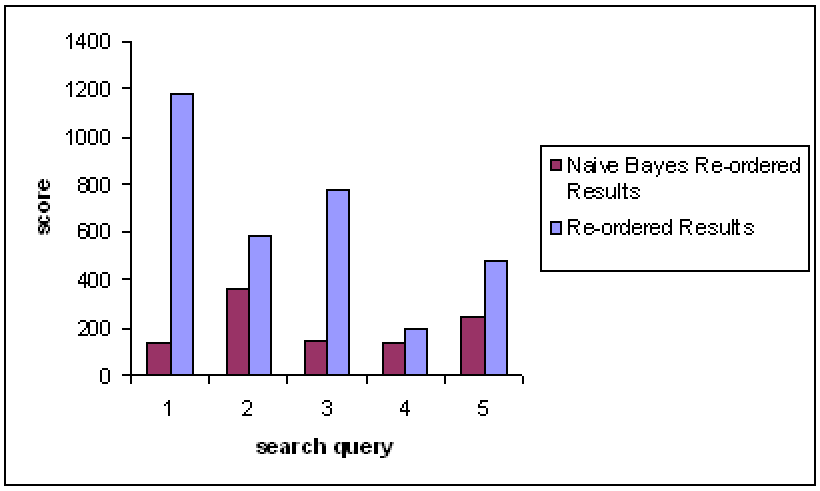

Figure 4.14. Category 2: Comparison of re-ordering with Naive Bayes.

Table 4.8. Category 2: Ordering Search Results using Naive Bayes Classifier

| Query | Word Added | order score | re-order score | original order |
|---|---|---|---|---|
| K12 education | license | 830 | 190 | 140 |
| kindergarten curriculum | license | 820 | 370 | 370 |
| K12 english teachers | license | 530 | 10 | 150 |
| employment K12 | license | 200 | 330 | 140 |
| K12 mathematics | catalog | 590 | 390 | 250 |

Table 4.8 shows metric scores of the Naive Bayes Classifier. We see that the Naive Bayes classifier does not perform well enough to get a higher number of relevant search results. And on re-ordering the score improves only one case.

#### 4.5.1.2 Information Gain

Table 4.9. Category 2: Search Result Ordering using Information Gain

| Query | Word Added | Original order score | Information Gain score |
|---|---|---|---|
| K12 education | workers | 830 | 130 |
| kindergarten curriculum | workers | 820 | 340 |
| K12 english teachers | workers | 530 | 550 |
| employment K12 | workers | 200 | 130 |
| K12 mathematics | workers | 590 | 430 |

As mentioned in the earlier category information gain method selects a word that can best distinguish between the relevant and not relevant documents of a category. The word added in this category is "workers" this word has the highest information gain of 0.96 in the category. As seen in Table 4.9 it improves only in 1 out of 5 times of the queries tried. But in our system it improves significantly for every query.

The Figure 4.15 shows us the relevant documents that a information gain system could get.

Figure 4.16 gives the comparison of the metric scores between our system and the information gain.

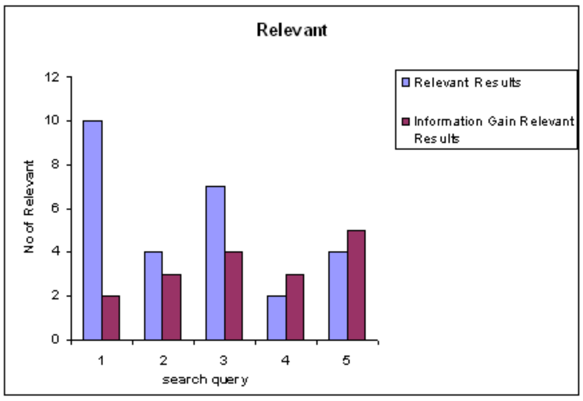

Figure 4.15. Category 2: Relevant documents with Information Gain.

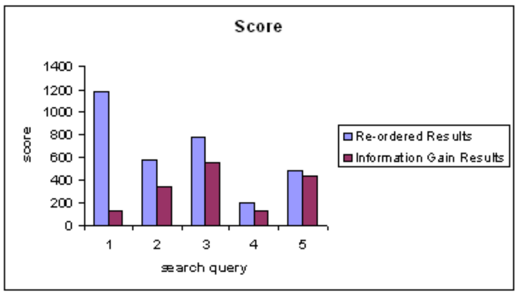

Figure 4.16. Category 2: Comparison of score with Information Gain method.

## 4.6 Evaluation

After conducting the experiments discussed we could infer that our classifier performs better than the other classifiers discussed. The following are the reasons

1. The classifier uses the knowledge of the relationships between the words i.e. their probabilities of occurring with each other in relevant and not relevant documents and hence it is able to classify by associating words even when there is less data available for classification. For example, when choosing the next best word for enhancing the search query our system selects a word that goes well with all the words already present in the query. While a Naive Bayes Method using an independence assumption would select a word which would just have a high probability over the

given features. It misses the semantic nature of documents.

2. It generates a utility value for words that helps the classifier identify a word which it knows would give a good return search results after the search is performed.

3. It re-orders the data retrieved from the search engine, based on a metric which is suitable for the user. For getting the relevant search results on the top few ranks it uses a classification probability parameter and then depending on the metric used calculates a score for the re-ordered search result.

# CHAPTER 5

# CONCLUSION AND FUTURE WORK

The system we developed in this report is an application that assists users to search for information on the Internet. As shown in the previous chapters the user can personalize web search without much overhead on his side. He just has to mention the documents that suit the criterion for the category he is interested in, i.e. he has to define his interest in the category he creates. This gives him the independence to use the application in the way he prefers. The system provides the user a better search experience, a personalized search experience. The user can correct the system by changing the document base on which a particular category is dependent if he/she feels that the application is not meeting his requirements.

The system performs better than traditional forms of web search applications. A "semantic" nature is based on the probabilities of words occurring or not occurring together. It tries to capture a language like feature i.e. a word is always dependent on the occurrence of previous words in a sentence.

## 5.1 Future Work

For the future the system could provide a mechanism so that multiple words can be added to the search query. These words would better capture the user's interest in the category and help classify the returns of the search engine more efficiently.

There could be other features added to the system like an *implicit feedback* mechanism to judge if a page the user is currently viewing is relevant to the category or not. An implicit feedback could, for example identify a page as relevant if the user views the page for a longer period. Other incidents like the number of mouse clicks, number of links used in the page could also be considered.

## REFERENCES

[1] Broder A. A taxonomy of web search. *SIGIR Forum 36(2)*, 2002.

[2] Kjersti Aas and Line Eikvil. Text categorisation: A survey. 1999.

[3] Kamal Nigam Andrew McCallum. A comparision of event models for naive bayes text classification. *15th Ann Int ACM SIGIR Conference on Research and Development in Information Retrieval*, 1998.

[4] R Baeza-Yates. *Modern Information Retrieval.* Addison Wesley, 1999.

[5] Sergey Brin and Lawrence Page. The anatomy of a large-sca.e hypertextual web search engine. 1998.

[6] Sergey Brin, Lawrence Page, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. 1998.

[7] J. Budzik and K. Hammond. Watson: Anticipating and contextualizing information needs, 1999.

[8] Danny Levison Daniel E. Rose. Understanding user goals in web search. *ACM*, 2004.

[9] Pedro Domingos and Michael J. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, 1997.

[10] P. Biron E. Efthimiadis. Query expansion experiments. *Proceedings of the Second Text Retrieval Conference*, 1994.

[11] Glover et all. Improving category specific web search by learning query modifications. *Symposium on Applications and the Internet, SAINT*, 2001.

[12] Weiyi Meng Fang Liu, Clement Yu. Personalized web search by mapping user queries to categories. *ACM*, 2002.

[13] T. Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. *ICML-97 14th International Conference on Machine Learning*, pages 143–151, 1997.

[14] S. Lawrence K. Bollacker and C. Lee Giles. A system for automatic personalized tracking of scientific literature on the web. *ACM DL*, 1999.

[15] Lawrence Shih Yu-Han Chang Jason Rennie David Karger. Not too hot, not too cold: The bundled-svm is just right. *Artificial Intelligence Laboratory; Massachusetts Institute of Technology*, 2002.

[16] K. Knight. Mining online text. *Commum ACM 42*, 11:58–61, 1999.

[17] Daphne Koller and Mehran Sahami. Hierarchically classifying documents using very few words. pages 170–178, 1997.

[18] K. Sycara L. Chen. Webmate: A personal agent for browsing and searching. *Autonomous Agents and Multi Agent Systems*, 1998.

[19] David D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *ECML*, pages 4–15, 1998.

[20] H. Lieberman. Letizia: An agent that assists web browsing. *IJCAI*, 1995.

[21] Y. Shoham M. Balabanovic. Learning information retrieval agents:experiments with automated web browsing. *In On-line Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, 1995.

[22] Chris Buckley Mandar Mitra, Amit Singhal. Improving automatic query expansion. 1998.

[23] T. M. Mitchell. Machine learning. *McGraw Hill International Series*, 1997.

[24] University of Berkeley. http://www.lib.berkeley.edu/teachinglib/guides/internet/, September 2002.

[25] Maria Teresa Pazienza. Information extraction. *Lecture Notes in Computer Science*, 1299, 1997.

[26] M. Pazzani and D. Billsus. Learning and revising user profiles: The identification of interesting web sites. *Maching Learning*, 1997.

[27] Van Rijsbergen. A theoretical basis for the use of co-occurrence data in information. *J. Document*, 33,2:106–119, 1977.

[28] S. E. Robertson, C. J. van Rijsbergen, and M. F. Porter. Probabilistic models of indexing and searching. In *SIGIR '80: Proceedings of the 3rd annual ACM conference on Research and development in information retrieval*, pages 35–56, Kent, UK, UK, 1981. Butterworth & Co.

[29] J.J. Rocchio. Relevance feedback in information retrieval. *In the SMART Retrieval System-Experiments in Automatic Document Processing*, pages 313–323, 1971.

[30] Ronald Rosenfeld. Adaptive statistical language modeling: A maximum entropy approach. *Carnegie Mellon University, Ph.D. Thesis*, 1994.

[31] S M Ruger and S E Gauch. Feature reduction for document clustering and classification. *Imperial College of London*, 2000.

[32] Robertson S.E. and Sparck Jones K. Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27:143–160, 1976.

[33] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[34] M. Singhal, A. Mitra and C. Buckley. Learning routing queries in a query zone. In Proceedings of SIGIR-97, 20th ACM International Conference on Research and Development in Information Retrieval (Philadelphia, US, 1997):2532, 1997.

[35] Robert P. Futrelle Susan Gauch. Experiments in automatic word class and word sense identification for information retrieval. *Proceedings of the Third Annual Symposium on Document Analysis and Information Retrieval*, 1994.

[36] T. Mitchell T. Joachims, D. Freitag. Webwatcher: A tour guide for the world wide web. *IJCAI*, 1997.

[37] Sepandar Kamvar Taher Haveliwala and Glen Jeh. An analytical comparison of approaches to personalizing pagerank. 2002.

[38] V. Vapnic. The nature of statistical learning theory. *Springer*, 1995.

[39] J.Trenkle W.Cavnar. N-gram-based text categorization. *In-Proceedings of SDAIR-94*, 1994.

[40] I. H Witten. *Managing gigabytes : compressing and indexing documents and images.* Morgan Kaufman Publishers, Inc., 1999.

[41] Yiming Yang and Xin Liu. A re-examination of text categorization methods. *SIGIR*, 1, 1999.

## BIOGRAPHICAL STATEMENT

Ajay Madkaiker received his Bachelor of Science degree in computer science from University of Mumbai, Mumbai, India in October of 2001.

He earned his Master of Science in Computer Science from University of Texas at Arlington, Arlington, USA in Fall of 2005. His research interests include text mining, information retrieval and search engines.