The Impact of Modes of Mediation on the Web Retrieval Process

Pannu, Mandeep Kaur

Award date:
2011

Awarding institution:
Coventry University

The Impact of Modes of Mediation on the Web Retrieval Process

Mandeep Pannu

A thesis submitted in partial fulfilment of the University's requirements for the Degree of Doctor of Philosophy

August 2011

Faculty of Engineering and Computing

COVENTRY UNIVERSITY

Abstract

This research is an integral part of the effort aimed at overcoming the limitations of the classic search engines. This thesis is concerned with the investigation of the impact of different modes of mediation on the web search process. Conceptually, it is divided into three main parts. The first part details the investigation of methods and mechanisms in user profile generation and in filtering search results. The second part deals with the presentation of an approach and its application in the development of a mediation framework between the user and the classic Web Search engines. This involved the integration of the explicit, implicit and hybrid modes of mediation within a content-based method, and was facilitated by the adoption of the Vector Space Model. The third part presents an extensive comparative evaluation of the impact of the different types of mediation systems on web search, in terms of precision, recall and F-measure. The thesis concludes by identifying the contribution of the research programme and the satisfaction of the stated objectives.

Acknowledgement

I would like to take the opportunity to express my gratitude to everyone who supported me in completing this research.

First I would like to acknowledge the enormous support and guidance offered by my Director of Studies Dr. Rachid Anane. His advice, constant support and encouragement were invaluable and have helped me acquire new research skills and have also shaped my research ideas.

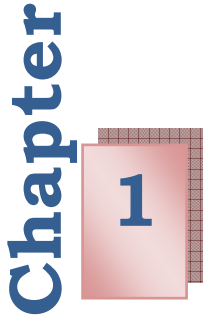
I am also thankful to my Supervisors Dr. Michael Odetayo and Prof. Anne James for their support and feedback.

I would like to express my sincere appreciation to my colleagues at the DSM group for creating a great atmosphere to work in. Thanks to them the time spent on my research was not only fruitful but also a pleasant experience.

Finally I would like to dedicate this achievement to my family and friends. Special thanks to my Dad Prof. Kartar Singh Pannu and to my husband Mr. Harroop Singh Pannu. My Dad has always had faith in me, a priceless motivation which I truly appreciate. My husband's care and love made it possible for me to complete this work.

Table of Contents

Figures



Introduction

1.1 Introduction

The Web has become an integral part of many social, business and scientific activities. Its ability to act a repository for a vast amount of information and as a medium for a variety of transactions, have contributed significantly to its phenomenal growth. Some of the key factors that underline its ubiquity as a foundational system include availability from anywhere and anytime, simultaneous access to up to date information, and support for dynamic and interactive modes of operation as well as access through familiar interfaces. The reliance of the interactions on widely accepted protocols is a further enhancement to the transparent identification and retrieval of resources.

The ad hoc and arbitrary nature of user intervention promotes a dual role for users as both consumers and producers of information. These two perspectives have a direct impact of the interaction with Web content. The background, the context and the aims of the producers or authors lead, in particular, to the creation and publication of documents of varied content, description and quality. As consumers, users are potentially exposed to a large number of documents whose relevance is now considered an important criterion in assessing the usefulness of the Web.

1.2 Web search

As a hypermedia system, the Web links billions of web pages and the role of the search engines is to harness and marshal these resources and mediate between the Web and the users. The mapping of a large portion of the Web into the indexes of the search engine is designed to capture as much of the web as possible. The narrowing and formulation of the search information is achieved mainly through keyword specification.

Finding the required information on the Web can be difficult and time consuming, and the results are often described by users as less accurate than desired. Users may spend a lot of time and effort scanning through a large amount of documents in order to find the relevant information. The reliance mostly on keywords and its linguistic implications is one of the major reasons for the low accuracy in information retrieval (Brusilovsky and Tasso 2004). The retrieval process of most search engines is also influenced by link popularity and page ranking algorithm. Web search engines are designed to serve a generic user irrespective of individual needs and interests. This raises the fundamental issue of how to identify and select the information that is relevant to a specific user. This concern over the lack of differentiation and precision has provided the foundation for the research into Web Search personalisation. The current consensus is that the retrieval process can be improved through the personalisation of the search process and the filtering of documents according to specific needs and interests.

1.3 Personalisation

In personalisation the focus is on the needs of the individual users and their queries. Personalisation can be automatic or customised (Pazzani and Billsus 2007). With personalised systems the results become useful when the user provides sufficient feedback on previously received results or relevant profile information. The personalised filtering process starts with individual users, their preferences and the generation of their profiles.

A user profile is not confined to a list of keywords only; it may contain information regarding user behaviour, context and other preferences (Ghosh and Dekhil 2009). Gils et al. (2003) have defined a user profile as a whole set of preferences that can affect the behaviour of a search engine, including constraints put on the search results. Two approaches can be used for user profiling: implicitly generated and explicitly generated. In implicit user profiling, the behaviour of the users and their activities are observed from different perspectives and the information is collected as the user interacts with the system. Explicit profile generation, on the other hand, requires the users to directly provide specific information in order to create an individual user profile.

These two approaches raise some important issues. In many cases in the explicit approach the users may not be fully aware of their current and future needs. This approach may require pre-defined categorization of user interests. Furthermore, it is intrusive and can be time consuming and awkward for the user. However it affords the user with some direct control over the search process. The other approach - the implicit profile generation is transparent from the user point of view, but it is not trivial for an automated system to determine the relevance of a page that the user is viewing. The underlying assumption is that a user is expected to spend more time on relevant pages, and may wish to print or save them instead of merely reading them on-line. It entails that sole reliance on the gathering of behavioural data during a browsing session may not be adequate and may be open to interpretation. This method may not reflect accurately the current interests of the user or their changes. Its main advantage however is that it is not intrusive.

1.4 Document filtering

The personalisation process can either be focused on individuals and their interaction with documents, or on the identification of shared patterns of behaviour and the segmentation of the user population into groups of common interests. In the first case, the content-based approach, the coupling between user and content is an important part of the filtering

process. In the second case, it is the nature of the generic behaviour of a group that is the focal point of activity. Recommendation systems represent one form of implicit collaboration between users and rely on historical behaviour.

One issue in the personalisation and the filtering processes is the selection of an appropriate model for the efficient representation and manipulation of user profiles and documents. It should be capable of facilitating the determination of relevant documents in terms of similarity between users and documents.

1.5 Research aims and objectives

The primary goal of this research is to introduce a novel method of user profiling that combines explicit and implicit profiles, and to investigate if and how this integration can enhance the effectiveness of the retrieval process in comparison with traditional search engines (Google and Yahoo!), in terms of recall and precision. The following tasks have been identified as necessary to achieve the objectives of this research:

- To identify and investigate issues related to the web and search engines.
- To investigate the role of different personalisation techniques and retrieval models in the enhancement of the quality of retrieval process.
- To propose a novel approach for enhancing the filtering of search results by combining selectively different methods.
- To design and implement a mediation framework that enables the deployment of three different user profiling methods.
- To perform a quantitative evaluation of the mediation framework in terms of precision, recall and F-measure as well as a qualitative evaluation.

1.6 Research programme

This research is concerned with the investigation of personalisation in Web search and the presentation of a searching approach based on user profiling. The approach is applied in the design and implementation of a mediation framework, which incorporates variants of explicit and implicit user profile generation. The research work is supported by an evaluation of the proposed approach.

This thesis details the different stages of the research work in conformance with the stated aims and objectives. Conceptually, the thesis is divided into three main parts: Web and personalisation, design and implementation of a mediation framework and evaluation.

1.6.1 Web and personalisation

This part defines the context of the research programme, and identifies issues related to the Web and the search engines and their limitations. It provides the rationale for the investigation of the role of personalisation and relevant techniques in the enhancement of the quality of the retrieval process. Approaches to user profile generation and models for information representation and filtering are investigated and detailed.

1.6.2 Design and implementation of a mediation framework

The proposed approach is aimed at investigating the impact of different modes of mediation on the Web search process within a content-based framework. Three types of mediation are considered; they all involve profile generation, document representation and information filtering. In the first type of mediation the users are required to *explicitly* specify their interests. In the second type the system plays an active role in generating a profile for the user *implicitly*, through the monitoring and the recording of specific features of the interaction of the user with documents. In the third type of mediation, the explicit and implicit methods are combined into a hybrid

system to improve the filtering process. The three types of user profiling are incorporated into the design and implementation of a mediation framework.

1.6.3 Evaluation

An extensive quantitative and qualitative evaluation of the framework is presented. In the quantitative evaluation the performance of the three mediation systems is measured in terms of two metrics: precision and recall. Experimental results are presented and analysed as part of a comparative evaluation with Google and Yahoo. The mediation framework is also put into a wider research context through a qualitative evaluation against other systems.

1.7 Contribution

The main contributions of this research are detailed as follows:

- The proposal of a novel approach which seamlessly combines explicit and implicit user profiling.
- The design and implementation of a mediation framework that follows the proposed approach.
- The implementation of three different mediation systems, explicit, implicit and hybrid.
- The experimental evaluation of the three systems in terms of precision and recall, and the statistical validation of the results.
- The validation of the view that personalisation can offer an effective way of dealing with information overload.

Chapter 2

The Web and Search Engines

2.1 Introduction

The growth of the Web and the increase in the number of users owe much to the important part that search engines have played in facilitating access to a vast repository of information (Lawrence 2000). The increasing amount of information and services available on the Web has a significant impact on users. Finding the relevant information on the Web can be incrementally difficult, time consuming, confusing and frustrating for most web users. The quality of the Web Search is often due to the fact that the design of Web Searching systems lacks any awareness of the needs of users (Bernard and Spink 2006). Search methods and algorithms need to be adapted to help find relevant results faster by improving recall and precision. In order to retrieve and provide the information a user is searching for there is a critical need to understand how people use the Web, how they search for information and what tools and techniques they use to find documents that are relevant to them**.** This chapter is concerned with the presentation of the salient features of the Web and the search engines, and ways of evaluating their performance.

2.2 World Wide Web

The Web is a collection of interlinked documents accessible via the Internet. Initially, the Web was designed to help a changing society with communicating and sharing ideas (Hendler and Berners-Lee 2010). In general, users try to acquire information by entering keywords or known URLs. However, the way to express requests in terms of keywords remains a significant challenge.

Conceptually, the Web is divided into two parts, the visible web and the invisible web. The visible web allows crawling and indexing of information by search engines; as every page within it can be reached from other pages through hyperlinks (Berners-Lee et al. 1994).

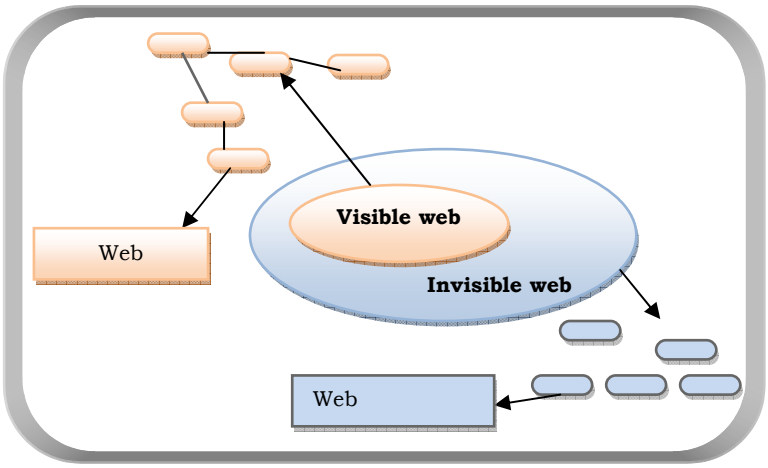


Figure 2.1: Web Structure

Figure 2.1 describes the structure of the Web. In the visible web, documents are linked to each other either directly or indirectly. Documents which are part of the visible Web can be indexed and retrieved by general search engines like Yahoo!, Google, or AltaVista. In the invisible web, however, the pages are disconnected from the visible graph and thus they cannot be reached by any indexer. Once a page from a disconnected cluster becomes visible, either by being linked from a visible page or being added directly to a crawler database, the web crawlers will be able to index that page. However, even if a page is connected to the visible web, it may not be useful to the

search engine if the information on the page requires user authentication. Although search engines allow users to perform quick search on millions of web pages, they are still unable retrieve information from the invisible Web due to limited access.

2.3 Search Engines

A search engine is a tool for retrieving information from the Web (Bernard and Spink 2006). The term search engine is often defined for both directories manually created by humans as well as crawler-based search engines (Holmes, 2006). Some search engines such as Yahoo! and Google also include "Yellow pages" – directories that a user can browse to find the web pages offering a variety of content. Search engines retrieve results based on similarity of documents to a user query, and retrieve everything that has high similarity, irrespective of whether it is relevant or not. Most of the documents returned by the search engines contain or are related to the keywords entered by users during query formation. Very often the results do not match the interests or preferences of users. Search engines that rely on keywords only return many low quality search results (Brin and Page 1998). This issue can be better explored by considering the structure and behaviour of a search engine.

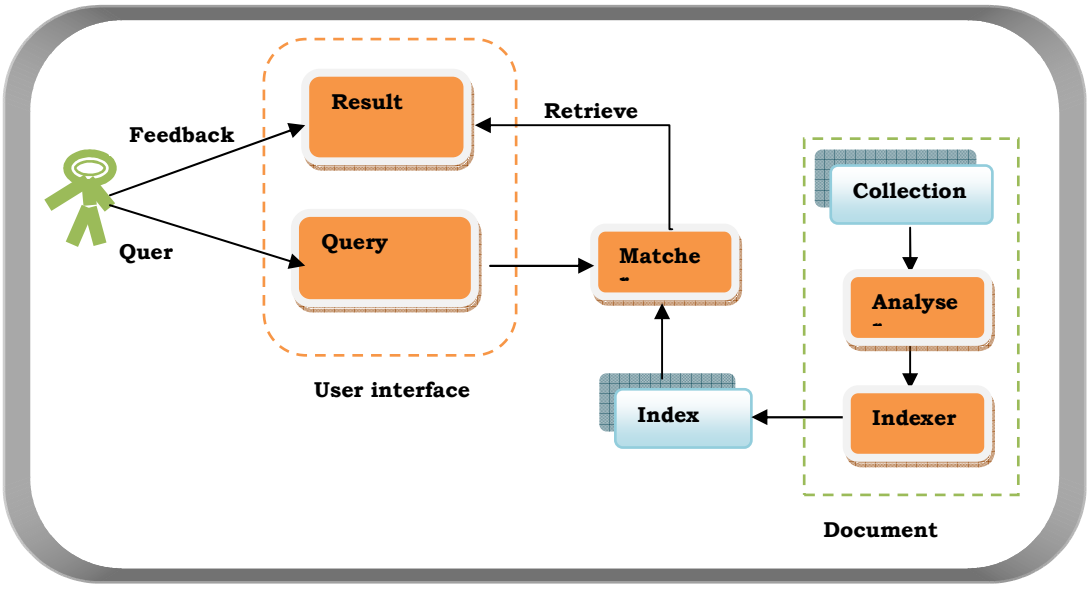


Figure 2.2: General web search engine architecture

Figure 2.2 presents the overall architecture of a search engine. In general a search engine includes two main parts. The first part is concerned with the creation of a repository of documents, and the second part with query processing. In document gathering, the document manager collects the documents, analyses them and sends them to the indexer. The Indexer creates a large database containing information about the content of web pages without actually storing the entire pages. The stored information is updated on a regular basis to keep the current versions of frequently changing pages and to discover new documents.

In the part responsible for the query processing the query is received from the user through the search engine interface. The interface usually allows users to express their information needs in keywords. The query processor analyses the tokenized terms, deletes stop words, applies word stemming, creates a query representation and computes the weight by matching the similarity the query and the content of individual documents in the database. The returned results are ordered and presented to the user based on a ranking algorithm specific to each search engine. Google makes use of the PageRank algorithm for ordering the web pages based on their popularity, a feature which is different from the factors that underpin the main retrieval models; the Boolean model, the vector space model and the probabilistic model (Yip and Quiroga 2008). Other search engines such as Yahoo! and AltaVista utilize similar algorithms. A search engine scoring algorithm can be based on Boolean logic (present or absent query terms), term frequency, and query term weight (Liddy, 2005).

When a user enters the keywords of interest into the search engine, the engine scans its own database for the web pages with contents that match the entered query and returns their URLs (Busby, 2003). The results retrieved by the search engines may not satisfy the needs of the users, especially when the documents are long (Li and He 2010).

2.3.1 Popular search engines

This section presents a review of two popular search engines, Google and Yahoo!. Some of the techniques and searching algorithms used by search engines to retrieve documents from the Web are also described.

2.3.1.1 Google

Google was founded by Stanford university students Larry Page and Sergey Brin in 1998. Google uses the automatic Boolean operations between the terms in a query – users can specify which keywords have to be present in the retrieved documents, which are wanted but not required and which should not be present in retrieved documents (Burright 2006). A spell checking mechanism is built into the search engine which can positively increase the experience of the user by displaying suggestions as to how an entered query can be rewritten. One of the most important factors in the success of Google is that the relevance ranking is based not only on indexed page content but also on hyperlinks analysis.

The Google search engine uses its own algorithm for ordering the search results (Brin and Page 1998). The PageRank algorithm is used to estimate the quality of a document; it calculates the score depending on how many other documents are referring to it (the more the higher the rank), on how the referring pages are rated themselves (a higher ranked page has higher influence), and on how many pages the evaluated document has links to - the more outgoing links the worse the value for that page will be (Grossman and Frieder 2004). These criteria tend to increase the calculated ranking for popular documents (presumably better quality documents); these documents will be retrieved more often even if their similarity to the entered query is not very high.

Figure 2.3: Example of PageRank algorithm (Yip and Quiroga 2008)

Figure 2.3 presents a simplified idea of how a PageRank value can be calculated for a website. Each rectangle represents a web document, and the number inside represents the PageRank value of that document. The value assigned to each document depends on the value for each of the pages that reference that document, and on the number of web pages that are referenced by that document. Each web page is assigned a score that represents how important this page is for other documents. In the example document A is referencing two documents, its score is 0.4 and therefore the score for each of the referenced documents is increased by 0.2; documents B and C both have 0.2 sore. Document B is referencing the document C; therefore the score of document C is increased by a further 0.2. Finally, document C is referencing document A which raises its score to 0.4 (Yip and Quiroga 2008).

Another design feature of Google aimed at improving the search result is the indexing method. In a basic indexing scheme the importance of a keyword for a document depends on how many times it appears in the document. In the Google indexing mechanism the keywords are considered more important if they appear in the headings or at the top of the document. The PageRank algorithm and the indexing method are the key to the success of Google over other search engines (Brin and Page 1998). The PageRank

algorithm represents one method of improving the quality of search results in terms of precision rather than recall (Yip and Quiroga 2008).

In order to manage the extremely large data sets generated by the indexer Google introduced the MapReduce framework in 2004. A map function is executed on distributed machines in clusters to generate a list of key/value pairs from each machine; the lists are later merged by the *reduce* function to create the final result. For example, when a search query is being executed, for every machine in a cluster the map function could analyse the documents stored on that machine to produce a list of documents relevant to the query. This smaller set of data produced by each host in the cluster can then be further condensed by combining lists from each host into one final list. This approach allows the scaling up of the system by adding more machines to a cluster or by connecting distributed clusters (Dean and Ghemawat 2008). To further improve scalability, clusters can be duplicated. When a user submits a query, the system balances the load by forwarding the query to a cluster with a low load. The query is then processed by the search engine with the use of the copy of the database that is available in that cluster.

According to Google over one hundred factors such as popularity of the page (PageRank), the position of keywords in a page (e.g. whether they are in the header or not), and the distance between the keywords are considered when generating the list of documents that match the query. The detailed algorithm used by Google is protected in order to minimise the possibility of creating spamming pages designed especially to be ranked highly by the search engine without having any useful content (Blachman and Peek 2007).

2.3.1.2 Yahoo!

Yahoo! was developed by David Filo and Jerry Yang in 1994. Yahoo! has a large search engine database that also includes the Yahoo! directory. It supports full Boolean search features like AND, OR and NOT. Search terms are connected by AND operator by default (Notess 2008). Besides the Web

Search the MyYahoo! portal has customised features like stock prices, weather, news and sports and the portal interface can be personalised by the users (My Yahoo! 2011).

Similarly to the Google Search the Yahoo! search process involves two steps. The first step involves building and maintaining the database, and the second one is concerned with finding a list of documents in response to a query entered by a user.

In the first phase, Yahoo! uses a web crawler that follows all static links (with exception if a link leads to a directory or file that is marked as excluded). Dynamic pages are not indexed and Yahoo! recommendation to developers is not to use dynamic links for pages that should be indexed. Not all hyperlinks retrieved from the indexed document are valid. For some of the links a server can return HTTP error 404 stating that the document with the URL stored in that link is invalid. The crawler would normally ignore such URLs, however some servers instead of returning the error present a custom document that informs a user about the error. To avoid indexing such error information documents as standard content, the Yahoo! crawler tries to create URLs that will be invalid, by appending several random strings to the URL of one of the documents from the domain that is being indexed.

A phrase-based indexing is used to represent each visited document. The importance of each term depends on the number of occurrences and on the position of the terms within the document (Slawski 2008). The Yahoo! search engine also analyzes document attributes such as title, meta-tags and associated links (Yahoo! Advertising Blog 2010).

The second part of searching is information retrieval – documents in the database are compared to the query, and the most similar documents are returned. The search results are sorted based on the similarity of the query to the document (Notess 2008). In addition the ranking algorithm makes use of Click Popularity – a value describing how often a link is chosen by the

users from the search results. If more people click the URL of web site, then it is considered important and is ranked higher (My Yahoo! 2011).

It is expected that from early 2012 Yahoo! will stop using its own search engine and will provide results from Microsoft Bing. In the United States this transition has already occurred in 2010 (Yahoo! Advertising Blog 2010).

2.3.2 Search engines issues

This section is concerned with the identification of the major limitations of search engines.

2.3.2.1 Keywords are expressed in Natural Language

One of the main issues in Web Information Retrieval is that the domain of discourse of humans is often not taken into account by the search systems. The natural languages used by humans are not being interpreted appropriately by machines. Instead, keywords are being simply compared with words in documents without analyzing their meaning. If any keyword is missing in the text or if it is spelt incorrectly or a different variant of the same word is used its interpretation by the search engine may be incorrect and may yield inconsistent or irrelevant results. This potential mismatch between the search engine and the interests of the user may have an adverse effect on the user experience. This reliance on keywords only can result in low quality of matches (Brin and Page 1998), and is a major reason for the low retrieval accuracy (Brusilovsky and Tasso 2004). One keyword can have different meanings. For example, two different users enter a query for “Orange” as the search query, motivated by different needs. If the first user is interested in Orange – as a mobile phone company, and the second user is interested in a kind of fruit, then irrespective of the meaning of the keyword the query is same for the search engine.

Sometimes users are unsure about the terms or keywords that they have typed in the search text box, and even when a query is very specific, the user may still not be able to find desired documents. Search engines have a very

limited mechanism for expressing the information according to the needs of the users (Brusilovsky and Tasso 2004).

Search engines like Google deal with this problem by providing spellchecking and generating suggestions of different keywords that are often used in conjunction with keywords entered by the user (Google Help 2011). Search engines are programmed to produce results based on what most users are looking for when using particular keywords. For example, sometimes one word may refer to multiple items, such as 'science' may refer to computer Science, science games or science museum. Search engines results are based on average trends rather than the needs of a single user as they are often not able to track the behaviour of individual users.

2.3.2.2 Search engines retrieval results are based on link popularity

In the link popularity scheme, popular pages become more popular and new pages or unlinked pages are extremely hard to find. Sometimes it is impossible to access high quality information through search engines (Lawrence, 2000). For example, Google search technology ranks the pages according to link popularity rather than users interest (Busby, 2003); it does not consider the intentions of the user in ranking relevant pages to the user (Grimmelmann, 2007). If two different users - with different interests - submit the same query with different intentions the same result can be returned (Sugiyama, Hatano and Yoshikawa 2004). Many results do not reflect the intention of the user (liu, Yu and Meng 2006). Almost half of the documents returned may not be relevant to the user because the search engines do not often filter the pages to satisfy the preferences of the user (Tanudjala and Mui 2002).

Google addresses the problem of low rank for new pages by continuously updating its index. How often a web portal is revisited depends on how high is its rank and how often it has changed in the past. With its algorithm the Google search engine allows access to pages updated on a daily basis, like documents published on news portals (Google Webmaster Tools Help 2011).

2.3.2.3 Search engines are vulnerable to spamming

Another problem that search engines have to face is that web developers who are aware of the algorithms used by search engines can design web pages that appear higher in search results – without increasing the quality of documents content. Yahoo! defines spam as pages that have been created using these techniques to promote results that are inappropriate, redundant or poor-quality. These techniques includes inserting keywords that are unrelated to site (often by inserting text that is invisible to the user or presenting different versions of a page to the search engine). It can also be done by creating farms of websites designed only to increase rankings of other pages (Hunt, 2005) by e.g. providing links to these pages, surrounded with keywords that are not related to the page, but are often used in search queries.

2.4 Search engine performance

Web search results depend on three important aspects: the size of the Web, how frequently the information is updated and the ranking algorithm used by search engines. These factors and the arbitrariness of some results have called into question the usefulness of search engines, and led to the introduction of ways of evaluating the quality of the retrieval process.

To achieve a high quality of search results the system needs to match the results with the queries of the users and their information needs. In a perfect situation the information retrieval system retrieves only relevant documents and all relevant documents are retrieved. However, in many situations users will very often be presented with relevant and non-relevant documents in response to a query, and some relevant documents will not be included in the results.

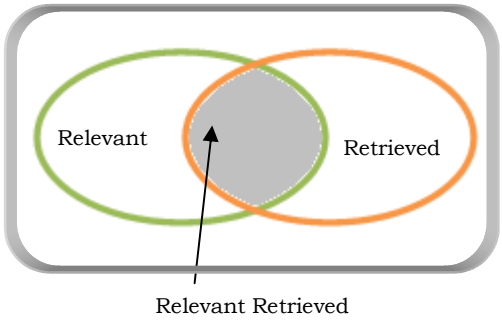


Figure 2.4 Relevant and retrieved documents sets

Precision and recall are widely used in information retrieval as a measure of the effectiveness of a system. Figure 2.4 gives a graphical representation of the documents space and documents classification.

2.4.1 Precision and Recall

Precision and recall were introduced as measures for evaluating the effectiveness of search engines (Mowshowitz and Kawaguchi 2005). In information retrieval they are expressed in terms of retrieved documents, those returned by a search engine in response to a query, and relevant documents, those related to the search topic.

Precision is the percentage of relevant documents within the list of retrieved documents, while recall is the percentage of relevant retrieved documents within the list of the all relevant documents. The need to measure the effectiveness of the system is to determine whether it provides a better ranking of results compared to traditional searching methods. Both precision and recall values are considered crucial for measuring the effectiveness of the system. It is worth noting in particular that precision is not binary but continuous.

Precision is the proportion of the number of relevant documents retrieved to the total number of retrieved documents.

$$Precision = \frac{number\ of\ relevant\ documents\ retrieved}{total\ retrieved\ documents} \qquad \text{[Equation 2.1]}$$

Precision measures the correctness or exactness of the results – in the perfect situation when all relevant documents are returned, the precision value would be one. When users are searching the Web via search engines, they only interact with the top N of the retrieved results. The top N results are considered the most important (Polyvyanyy and Kuropka 2007, Beza-Yates and Ribeiro-Neto 1999).

A system could have a good precision record when retrieving 15 documents but only 13 of them are relevant to the needs of the user. There are situations however where many documents – that also would be found relevant by the user – are not retrieved by search engines; this results in low recall.

Recall is a measure of the completeness or sensitivity of the retrieval process. Recall is the proportion of the number of relevant documents retrieved to the total number of relevant documents based on user query (Polyvyanyy and Kuropka 2007).

$$Recall = \frac{number\ of\ relevant\ documents\ retrieved}{total\ number\ of\ relevant\ documents} \qquad \text{[Equation 2.2]}$$

Recall measures the comprehensiveness of the result and consequently high values are desired. The problem at this point is that estimating the number of relevant documents is a non-trivial task (Grossman and Frieder 2004).

Precision and recall are often combined with equal weight into a single measure, $F_{\beta,}$ for positive real values of β. This measure was derived by van Rijsbergen and has the additional advantage of assigning different weights to precision and to recall (Van Rijsbergen, 1979).

$$F_B = (1 + \beta^2).\frac{precision \cdot recall}{\beta^2 \cdot precision + recall} \qquad \text{[Equation 2.3]}$$

By setting β to a value bigger than 1, more weight is given to recall, whereas a value lower than 1 means that precision is weighted higher than recall (Van Rijsbergen 1979 and (Manning, Raghavan and Schutze 2008).

The most common value for β is 1 which yields the harmonic mean of precision and recall (Beza-Yates and Ribeiro-Neto 1999, (Manning, Raghavan and Schutze 2008). The F-measure or balanced F-score is represented by the following formula:

$$F = 2\frac{Precision \cdot Recall}{Precision + Recall} \qquad \text{[Equation 2.4]}$$

The F-measure is a measure of the accuracy of the retrieval process, and has the advantages of generating a single value for comparative evaluations.

The determination of precision and recall and the combination of their relative values yields four possible cases (Mowshowitz and Kawaguchi 2005):

- Case 1: recall is high and precision low – when most of relevant documents have been retrieved, but the number of irrelevant retrieved documents is very high.
- Case 2: recall is low and precision is high – when most of the relevant documents are not retrieved, but the number of irrelevant documents is lower – Kumar and Prakash (2009) point out that in their study this case applies to simple one word queries in Yahoo!.
- Case 3: recall is low and precision is low – when retrieved documents are mostly irrelevant and majority of relevant documents is not retrieved.
- Case 4: both precision and recall are high – when most retrieved documents are relevant and only some irrelevant are included.

In the extreme cases, a value 1 for the precision indicates that all the returned documents were relevant, but offers no suggestion on whether all the relevant documents were retrieved. A value of for the recall is a clears statement that all relevant documents were retrieved, but is silent on the number of irrelevant documents. It is generally agreed that in the retrieval process, most search engines display an inverse relationship between precision and recall. The recall can be improved by retrieved more

documents at the cost of precision by also retrieving more irrelevant documents (Kumar and Prakash 2009).

2.4.2 Information retrieval system evaluation

A framework for the evaluation of an information retrieval system includes the following:

1. A document test collection.
2. A set of information needs expressed as queries often in terms of keywords.
3. A set of relevance judgements for the documents retrieved. The documents are manually assessed as relevant or irrelevant on the basis of individual query-document pair.

Many document collections that can be used for information retrieval systems exist. GOV and REUTERS RCV1 are good examples. Traditionally the TREC (Text Retrieval Conference) collection has been used for testing the performance of various retrieval systems (Voorhees and Harnam 2005). TREC and GOV2 are collections maintained by the US National Institute of Standards and Technology (NIST).

As the processing and judgment of the whole web is infeasible, competing retrieval systems are evaluated by applying them to one or more of the documents collections. A pool of documents can be created as the union of several collections, where duplicates are removed. The pool represents all the documents, and all relevant documents are assumed to be in the resulting pool.

As the collection in the database of most search engines holds millions of documents, the pooling approach pioneered in TREC was applied successfully to the search engines by various researchers (Clarke and Willett 1997, Kumar and Prakash 2009, Shafi and Rather 2005). The use of many search engines is motivated by the need to access a larger pool of documents

and to overcome the inherent bias of the search engines (Mowshowitz and Kawaguchi 2005, Mowshowitz and Kawaguchi 2002).

The precision and recall can now defined in terms of the pool of documents retrieved by all the systems:

$$Precision = \frac{number\ of\ relevant\ documents\ retrieved\ by\ system}{number\ of\ documents\ retrieved\ by\ system} \quad \text{[Equation 2.5]}$$

The recall is relative and is defined as:

$$Relative\ recall = \frac{number\ of\ relevant\ documents\ retrieved\ by\ system}{total\ number\ of\ relevant\ documents\ in\ the\ pool} \quad \text{[Equation 2.6]}$$

The definition of these measures will be further refined in the evaluation in Chapter 6 in order to take into account the manual judgment of the documents by using a fixed scale.

2.5 Summary

In the search for relevant documents through search engines, users have to go through several queries in order to find the results that match their interests. A query may be interpreted as encapsulating all the interests of a user, which may produce irrelevant documents. One of the reasons for the lack of precision is the fact that users enter short and specific queries. Search engines retrieve all the results based on a single user query and often do not take into account the information needs of the users. This concern over the lack of differentiation and precision has provided the focus for the research into personalisation systems, where search results are filtered according to the profiles of users. The aim of these systems is to improve the precision and the recall of the retrieval system, by adapting the web search process to specific information needs.

Chapter 3

Web Personalisation

3.1 Introduction

This chapter is concerned with web search personalisation. Web search personalisation involves creating systems that can take into account the preferences of users to filter the search results according to their information needs. As the sources of information on the web and the number of web users are increasing every day, it is crucial to improve the quality of search results. The techniques used by search engines and personalised systems tend to retrieve both relevant and irrelevant information. This demands advanced solutions for acquiring the information that meets the needs for users (Klusch 2001). This chapter involves the examination of the techniques used for filtering web documents. The focus of this chapter is on explicit and implicit user profiling. It also reviews some content-based and collaborative filtering systems as well as hybrid systems.

3.2 User Profiling

User profiles can contain a set of preferences regarding system behaviour and constraints on the search results (Gils et al. 2003). In general, user profiles are not defined by a simple list of keywords which represents the user interests; they may contain user information regarding behaviour and

context (Ghosh and Dekhil 2009). A system can collect information for the user profile from the browsing history and documents ratings (provided by users) to improve the search results (Gasparetti and Micarelli 2007). Users can mark the information on different pages as interesting and choose the most relevant according to their interests (Grcar, Mladenic and Grobelnik 2005). Previous queries can also be recorded and reused to improve the search effectiveness.

In user profile creation two kinds of approach are considered particularly useful for information retrieval: explicit user profiling and implicit user profiling. In the explicit profile a user creates the profile or provides feedback on the basis of their needs, while in implicit approach the system creates profiles based on search histories and browsing behaviour.

3.2.1 Explicit Profile

Explicit profile creation involves asking users for specific information in order to create an individual user profile. To learn about specific users needs, a large amount of information is required from users. The information regarding the interests of the user is usually gathered by specifying keywords or giving feedback on visited documents (Salton et al. 1997). Salton et al. (1997) have considered user involvement as a powerful way of improving the relevance of the search results, and new system based on the information explicitly provided by users are constantly being developed (Rastegari and Shamsuddin 2010). Relevance feedback relies on explicit assessments provided by users.

Stegmann (2005) presented an approach to explicit user profiling that compiles personal interests by means of an adaptive natural language dialogue. The system captures the information provided by users during a dialogue session and stores it in an explicit user profile.

The explicit profile creation can help specify the result and user preferences over time (Smyth and Wilson 2003). However, the lack of user understanding in terms of keyword search can complicate the process of finding relevant

results. When users get search results prepared for an average user, they have to go through many returned documents to find the relevant ones. In the explicit user profile generation users can build their own profile according to their specific interest and needs. However to reduce the cognitive burden on the user, implicit feedback can be used with the same effectiveness as the explicit feedback (Hopfgarter et. al 2008).

3.2.2 Implicit Profile

The process of creating explicit profile increases the cognitive burden on the users. In general, users are very reluctant to provide feedback (White, Jose and Ruthven 2003) and for that reason different techniques are proposed to implicitly estimate the feedback that would be given by a user (Hussein and Elsayed 2008).

A number of methods have been used for implicit profile generation to improve the search results on the web. Implicit generation requires observing user behaviour and capturing their search histories (Shen, Tan and Zhai 2006, Gasparetti and Micarelli 2007). User actions that needs to be observed includes time spent on reading a web page, saving, printing, clicking, selecting, and bookmarking (Claypool, Waseda and Brown 2001). Aoidh, Bertolotto and Wilson (2007) proposed an implicit profiling that involves capturing user mouse movements as well – e.g. by storing text under the mouse pointer, as user may be using the mouse pointer for reading. A more recent approach used by Hussein and Elsayed (2008) has involved capturing users' facial expression to implicitly estimate the users' interest in a document being displayed.

Gasparetti and Micarelli (2007) proposed a technique for building implicit user profiles with the help of browsing history. The algorithm relies heavily on the textual context of the links followed by users during browsing. The disadvantage of this or similar algorithms is that these algorithms rely only on a positive feedback (Gemechu, Yu and Ting 2010). One advantage of this technique is that it does not require user involvement. Changes in the

interests or search area of the users may not be reflected immediately in the results returned by the search engines.

3.3 Personalised Systems

The relation between a user query and web pages is problematical and is driving the research in the field of information retrieval. Users have a variety of needs and the retrieval systems are often unable to offer the solution to fulfil individual user requirements (Zigoris and Zhang 2006). The retrieval system or search engines retain large, fast growing indexes can cause the performance of searching techniques to decrease, which is one of the main causes for the low quality of search results (Sankaradass and Arputharaj 2011). Researchers have classified and introduced various schemes for web personalisation (Pazzani and Billsus 2007). Personalised systems help users overcome the limitations of web search by extracting keywords based on individual preferences (Rastegari and Shamsuddin 2010). Personalisation can be automatic (implicitly) or customised (explicitly). The customisation may be able to help filter out the irrelevant document according to an individual user preference (Gauch, Chaffee and Alaxander 2003, Sieg, Mobasher and Burke 2004). However, the personalised search engines results focus on the users rather than only on their submitted queries (Ferragina and Gulli 2005). Instead of focusing on the query alone, a personalised system can use the information stored in a user profile, created either implicitly or explicitly, to present more relevant documents in search results by filtering and reordering the results of a query (Rastegari and Shamsuddin 2010).

There are three main kinds of personalised systems that are considered effective for filtering and retrieving the information on the web: content-based filtering, collaborative filtering and the combination of both called hybrid filtering.

3.3.1 Content-based filtering system

A content-based system makes recommendations based on a description of a web page (or an item in a shop inventory) that has been created during indexing, and on the interests of the users. The system first collects the explicit preferences of the user and then evaluates the relevance of web pages in terms of its content and similarity to user preferences. It scrutinises the description of the items to identify items that are of interest to a particular user (Pazzani and Billsus 2007). The information about the preferences of the users is gathered from requested web pages (or items descriptions) in the form of feedback. For that reason, the system can only suggest items in the same category of items that have been previously explored by the user.

According to Ichikawa et al. (2008), a content-based system makes the recommendation of items that are similar to items used previously in conjunction with currently visited items, or the item with the highest similarity to the ones preferred in the past by the user. The system will add suggestions which the user might find interesting or useful based on previous history of the user and contents similarity.

Syskill & Webert (Pazzani, Muramatsu and Billsus 1996) is a content-based system that makes recommendations of web pages based on explicit user feedback. A user can rate a page on a three points scale. If the user rates some links in a web page, the system recommends to users other related pages that might be of interest. Once a page is ranked as high, the system analyses the page content to learn about the information the user is interested in. The system can be used to make an item recommendation that is based either on the user interest profile or the user's query (Garden and Dudek 2006). This method is based on accurate item data and neighbourhood structure. Naming the current browsing subject to create a separate profile depending on what the user is searching for. The system does not help the user to explore new topics because it can only make the recommendations based on similarity to previously visited pages. If the user

wants to change the area of interest then a new profile has to be created for the new area (Pazzani and Billsus 2007).

Letizia (Lieberman 1995) was a content-based system that was designed to help explore the web by implicitly learning user interest by analysing the individual user browsing behaviour. It was assuming that the user is interested in a document if the document was saved or bookmarked, and that user was not interested if the document was left without following links inside it. The system then analysed the documents linked to currently displayed web page and suggested linked documents that the user was likely to find interesting or useful according to the system. WebWatcher was a similar system that was designed to discretely retrieve information from the web pages available through the links from a visited web page (Mladenic 1996). Both systems were learning without interacting with the users, and did not ask users for keywords or rankings. The WebWatcher system was also suggesting links to the users based on their similarity to individual user's choice of web pages (Mladenic 1996). Although the focus of many researchers has been on the methods of implicit learning, the reliability of these methods is still an issue (Jung, Herlocker and Webster 2007).

Budzik and Hammond (1999) proposed a similar method that automatically retrieves documents from links on the currently opened web page and proposes the URLs that lead to documents that conform to the previous behaviour of the user. The main advantage of this technique is that it does not require specific feedback. The data can be gathered at no extra cost from the user perspective (Kelly and Belkin 2001). However, as the user profile is created in an implicit way, the observed information does not necessarily reflect the user's intention.

Personalized Recommended System (PRES) makes recommendations based on the comparison of the user profile with each document in the collection of indexed documents (Meteren and Someren 2000). PRES collects articles about home improvements and creates dynamic hyperlinks to make searching easier (Meteren and Someren 2000). First, the system removes all

HTML tags and stop words and then removes the prefixes and suffixes. For example, the word “Computer” and “Computing” is reduced to “Comput”. The users explicitly set their preferences to improve the effectiveness of the search result (Swapna and Ravindran 2008).

Fig 3.1 has been removed due to third party copyright. The unabridged version of the thesis can be viewed at the Lanchester Library, Coventry University

Figure 3.1 PRES Architecture (Meteren and Someren 2000)

Figure 3.1 shows the PRES architecture. In this system, the user requests web pages and provides the feedback to the user profile. The system learns the user profile from the user’s feedback. It compares the user profile with the documents of collection in the basis of relevance and similarity (Meteren and Someren 2000). To improve the performance a user can provide feedback based on received content (Swapna and Ravindran 2008).

3.3.2 Collaborative filtering system

Two approaches to collaborative filtering are considered: user based and item based. A collaborative filtering system can recommend content to a user by learning from similar users, or by detecting groups of similar items (Khribi, Jemni and Nasraoui 2009).

In a user based collaborative system, the method assumes that similar users prefer similar things. The system compares a user rating with the ratings given by other users to find similar users (Rashid et al. 2002). A collaborative system can make recommendations to a user based on the items that were chosen by similar users. The system uses the feedback from a set of people concerning a set of items for recommendation but ignores the content of the items. It does not make any recommendation for new users until it finds a group of similar users (Klusch 2001).

Figure 3.2 User based Collaborative filtering (Kamishima and Akaho 2006)

Figure 3.2 illustrates the idea of the user-based Collaborative filtering. The recommendation is based on items selected by other users grouped together with the targeted user. The content of each item is ignored – the recommendation is only based on users' recommendations.

The system scans set of items to find items that are similar to the items bought or visited before by the targeted user (Sarwar, Konstan and Riedl 2005). The system takes items that were, for example, bought by different users together with the visited item. The similarity between the users (e.g. similarity resulting from the previous buying history) is ignored. The system compares items based on the shared appreciation of users, in order to create neighbourhoods of similar items (Sarwar, Konstan and Riedl 2005).

Siteseer was one of the first collaborative web page recommendation systems targeted for scientific and academic papers. It was based on bookmarks created by users to identify papers of interest. The system was comparing the sets of bookmarks generated by individual users to make recommendations for them (Rucker and Polanco 1997).

Lemire has proposed an algorithm based on predicting the rating for an investigated item based on the difference between the ratings provided by a user to a set of items and the rating provided for the investigated item by another user. Both users are assumed to have also ranked the same items (Lemire and Maclachlan 2005).

Different algorithms can be used for collaborative filtering, but the common part is finding the similarity between two users either directly (in user-based version) or by looking at the items bought/ranked by users (in item-based version). The advantage of the collaborative model is that it can provide recommendation based on multiple users to provide accurate results. However new items will not be recommended until some a user takes an interest in them (Kagie, Loos and Wezel 2009). An additional method has to be used for introducing new items into the recommendations.

Kamishima has proposed an extension of Collaborative filtering system called Nantonac. This system measures preferences of the user by a ranking method. The preference patterns of the users are represented by orders and are sorted according to the degree of user's preferences (Kamishima and Akaho 2006).

The recommendation made by ordering similar items by user preferences, without giving exact values to rate each item. The system first collects the information about user preferences by asking the users to decide which of the displayed items is preferred. After receiving feedback for a series of e.g. pairs of items the system can search for users with similar preferences. Finally, the system recommends the items based on the preferences entered by the similar users (Kamishima and Akaho 2006).

3.3.3 Hybrid systems

In hybrid systems, content-based and collaborative filtering are used together to recommend pages to the users. Different kinds of approaches such as RAAP (Delgado, Ishii and Ura 1998) and Fab (Balabanovic and Shoham 1997) and P-Tango (Claypool et al. 1999) were considered as hybrid systems. Today Amazon Webstore and eBay are among the best examples of users of a hybrid system for generating suggestions (Parkes and Seuken 2011).

The Fab system uses content-based techniques for collaborative recommendation (Balabanovic and Shoham 1997). The system gathers user profiles based on visited pages content, and then compares profiles of other users to create clusters of similar users.

Figure 3.3 Overview of the Fab System (Balabanovic and Shoham 1997)

Figure 3.3 describes the Fab system architecture overview. The system includes two kinds of agents called selection agent and collection agent. The collection agents collect the recommended pages from the web and the selection agents redirect those pages to the users according to individual interests (Balabanovic and Shoham 1997).

P-Tango is a hybrid system used for online newspaper domain. In this system the user rates the items explicitly. The system calculates predicted ratings for items based on content similarity to already ranked items and based on sets of items ranked highly by other users that are similar (in terms of preferences) to the targeted user. The system combines the two predictions using an adaptive weighted average. It is not apparent however how the weights of content-based and collaborative predictions are decided (Claypool et al. 1999).

Amazon uses a hybrid system for generating suggestions (Parkes and Seuken 2011). The users explicitly provide the items ratings to the system and its purchase history is kept for future use. The algorithm used – named Item-to-Item Collaborative Filtering is designed to return as accurate results as possible in a short time, even if the recommendation is based on short shopping history (e.g. for new users). When a user views an item, the system suggest other items that are often bought together with the selected one, rather than clustering customers according to demographic or shopping history. If this extra data is available then the system can favour items that were bought together with the viewed item by users with similar interests to the targeted users.

3.3.4 Limitations of web personalisation

A content-based recommendation system calculates the similarity between the content of items while collaborative filtering determines information relevance based on the similarity between users or items.

A content-based system addresses the issue of how to construct a profile that accurately represents user interests. It is however hard to determine what information is more or less interesting to a user (Claypool et al. 1999). For example, if the user is interested in one category then the content based system will add the category to the list of preferences. As the number of categories increases, the system starts to lose its effectiveness. The system does not help to discover new items because it only recommends the items

that are similar to already visited items; it can only recommend items from a narrow topic range. If a user wants to change his area of interest then the system is not useful at all (Paulson and Tzanavari 2003). Some systems request explicitly the input of the preferences of the user such as ratings. The lack of feedback can also affect the performance quality as the systems that learn from user feedback and rating of items (Swapna and Ravindran 2008).

Collaborative filtering overcomes some of the limitations of content-based approach. A collaborative system can suggest some items based on rating given by other users, instead of the contents of the items (Li and Kim 2003). However, the system also raises some issues. First, the lack of available ratings (e.g. for new items) can affect the results. A collaborative system cannot suggest the items that do not have any user's recommendations. In addition the system is unable to recommend items of interest to new users because of the lack of the information about them. If the targeted users have different preferences from the group they may be assigned to, because of their short history, the system will provide recommendations of poor quality (Huang, Chen and Zeng 2004). In a collaborative system, the past shopping history of a user is considered in order to make recommendations. When a new item is added it will not be recommended until a significant number of users have rated it.

The system can provide incorrect recommendation in situation of limited user's feedback and with no similarity between users' interest (Huang, Chen and Zeng 2004). Kamishima system, Nantonac, is based on a ranking that asks users to sort many items (based on users preferences) before it could provide valid recommendations. Many users unfortunately tended to give up before completing the learning process (Kamishima and Akaho 2006).

There are different limitations for all types of web personalisation. The content-based methods are over specialised – only items similar to already known by the user are presented, it is also unable to provide recommendation to new users. In the user-based collaborative filtering the

scalability becomes a problem. With very large number of users finding groups of similar users is demanding long computation time. In addition, as typical user is only rating a small subset of items available, finding a group is not always possible for all users. The hybrid approach is trying to address these limitations (Parkes and Seuken 2011).

3.4 Summary

Profile generation can be performed explicitly or implicitly. The explicit approach requires the active participation of the user and the implicit approach attempts to gather information in the background. While the collaborative approach appear to have more affinity with the implicit method, in particular in determining user and item similarity, the content-based method tends to be accurate, does not suffer from the cold start and provides more focus.

Chapter 4

Information Retrieval Models

4.1 Introduction

The aim of this chapter is to introduce three models for representing documents and profiles in the search process, and to examine their computational processes. The volume of document databases, the large number of users and their different interests creates the need for precise and efficient filtering techniques (Grossman and Frieder 2004). This chapter investigates different information retrieval models, which can be used to determine the similarity between documents and user profiles. It will focus on three models: the Boolean Model, the Vector Space Model, and the Probabilistic Model. These models are significant because they are representative of three different mathematical models, with their own methods for representing documents and calculating similarity between documents and users' profiles.

An overview of some alternative retrieval models will also be presented.

4.2 Retrieval Models

The first of the information retrieval models presented in this chapter, the Boolean Model, is an example of the set-theoretic models, where documents are represented as sets of words, on which operations are performed in order to determine similarities. The Vector Space Model is an algebraic model in which documents and users' profiles are represented as vectors, and operations, such as the dot product of two vectors, are used to determine similarities as a scalar values. Finally, in the Probabilistic Model the probabilistic inference is used to determinate which documents best suits the information needs of a user. This model relies on probabilistic theorems, such as Bayes' theorem, to compute similarities as probabilities of relevance.

4.3 Document representation and processing

The filtering or retrieval process requires a specific representation of web documents and user profiles. There are three main process of information retrieval system; representation of the content of the documents, representation of the information needs of the users and finally comparison between both representations to retrieve documents so that the returned documents reflects the users' needs (Hiemstra 2009).

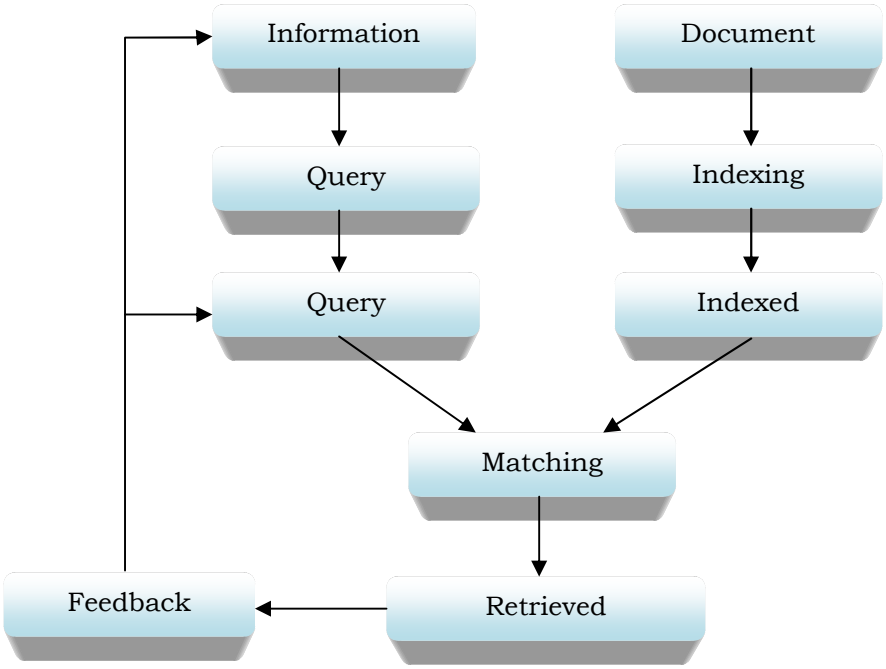


Figure 4.1: Information retrieval processes (Hiemstra 2009)

Figure 4.1 illustrates the basics of an information retrieval process. The matching of the documents and users' needs is based on simplified representation of documents, which were prepared during the indexing process, and on the representation of the targeted user profile. The process of formatting a query can be manual (user enter keywords) or automated (query is generated based on an existing user profile). The representation of document and user profiles can have different forms; for example a list of keywords. The representation depends on the techniques used for matching, e.g. for the Boolean information retrieval both documents and query are represented as simple sets of keywords, while for probabilistic information retrieval and vector space model, the representation includes weights that are assigned to each of the keywords. The analysis algorithm calculates the similarity based on these representations and determines how well each of the documents satisfies the user information requirements (how similar it is to the user profile). As a simplified representation can be less precise and more ambiguous that the original document (or profile), the search results can be less accurate than if a full original document had been compared with full profile, however the computational and storage requirements for such comparison would be higher.

As a web document can be complex, it is required that its content is represented in a form that can be analysed efficiently. The exact representation of the same document can vary from system to system, however in general there is an indexing process that actually converts documents into a simplified form. The basic simplified form of a document can be, for example, a list containing all the distinct keywords used within the document. In a more advanced system, it can be a vector containing keywords-value pairs, where the value can be for instance the number of times a word occurs in the document or the distance between the first occurrence of that word and the start of the document. If the document representation contains some additional rating values (like number of occurrences or position in the document), then a system that is analysing the similarity can be more advanced and has the possibility to provide more accurate results. The indexer used by Google is storing the information

about the position of keywords and the distance between them as well as the kind of HTML tag that is used to enclose it; for example, whether it is H1 tag, which is used for titles or section names or H2 tag which is rather used for subsections and therefore can be considered as less important (Google, Help 2011). In theory a system could use more than one technique for storing the representation of documents, one basic representation for easily filtering out most of the documents and a detailed one, for predicting the relevance of the remaining documents with a higher accuracy.

4.4 Boolean Information Retrieval (BIR)

The Boolean Information Retrieval model is based on classical set theory. Documents are represented as a set of terms it contains (not all words have to be used), while queries are represented as logical expressions. The keywords in the query can be linked together with Boolean operators AND, OR and NOT (Manning, Raghavan and Schutze 2008). Each term can have one of two logic states – it can be either present (logical 1) or absent (logical 0) (Manning, Raghavan and Schutze 2008).

The relevance of a document to the query of a user is calculated by evaluating the logical value of the query as either 1 or 0. A value of 1 is given to every term in the query that exists in the set representing document, and 0 for every term that does not exist in the representation of the document.

4.4.1 Document representation in BIR

For the purpose of Boolean Information Retrieval each document in the database has to be presented as set of terms. In order to limit the size of each representation, not all words have to be stored. Instead a dictionary (set) of interesting words is created. Depending on the purpose of the database the dictionary can be small and contains only words for one specific domain or large, containing e.g. all nouns. During the indexing process, each document is compared to the set of interesting terms to create the vector representation. If the terms dictionary is created as a vector containing words, then each document can be represented by a vector of

ones and zeros. The vector size should be the same as the size of the dictionary vector and for every word in the dictionary; if it is relevant to the document then the document representation vector will contain 1 on the

same position as the word otherwise it will contain 0 for that position.

	Term 1	**Term 2**	**Term 3**	**…**	**Term M**
Dictionary	Coventry	University	Course	…	Library
Document 1	1	0	0	…	1
Document 2	1	1	0	…	0
Document 3	1	1	0	…	1
…	…	…	…	…	…
Document N	0	0	0	…	1

Figure 4.2: Example of documents representation for BRI

In the example in Figure 4.2 the first document can be related to a library in Coventry but it is most likely not the university library because the term 'university' does not occur in it. Document 2 can be related to Coventry University but not to the library while the third document is related to 'Coventry', 'University' and 'Library'.

The exact method of storing the documents representations can vary from system to system, but Boolean Information Retrieval requires a method to verify whether a term is relevant to a document or not (e.g. whether it occurs in the document or – for possible implementation – whether a synonym of the word occurs in the document).

4.4.2 Query representation in BIR

The user query is a logical statement whose value has to be evaluated for each of the documents in the database in order to filter the relevant documents. Each keyword in the query is a single word or conjunctions of words.

Queries are specified as Boolean expressions and terms combined with operators. For example, a query that should return all documents that

contains Term1 and documents that contain Term2 but not Term3 can be expressed as follows:

Query1 = Term1 OR (Term2 AND NOT Term3)

4.4.3 Determination of document relevance in BIR

In order to determinate the relevance of a document to the query, the logical value of the query has to be evaluated. Each term in the query has a logical value 1 if the word exists in the document (or its representation) and logical value 0 if it does not. After all terms in the query are replaced by logical values, the query can be evaluated as any logic sentence. If the sentence is true then the document is considered relevant.

In the example the dictionary has five terms: 'Coventry', 'University', 'Course', 'Cost', and 'Library'. If a user wants to find the cost of the course and information about the university library, the following query can be used:

Dictionary	Coventry	University	Course	Cost	Library
Document 1	1	0	0	0	1
Document 2	1	1	1	1	0
Document 3	1	1	0	0	1
Document 4	1	0	0	1	0
Document 5	0	1	1	0	1

Figure 4.3: Example of documents representation for BIR

Query = Coventry AND University AND ((Course AND Cost) OR Library)

This, after replacing words with values from the terms in each document will produce following sentences:

- *Document 1 = 1 AND 0 AND ((0 AND 0) OR 1) = 0*
- *Document 2 = 1 AND 1 AND ((1 AND 1) OR 0) = 1*
- *Document 3 = 1 AND 1 AND ((0 AND 0) OR 1) = 1*
- *Document 4 = 1 AND 0 AND ((0 AND 1) OR 0) = 0*
- *Document 5 = 0 AND 1 AND ((1 AND 0) OR 1) = 0*

The logical value of query is 1 for Document 2 and Document 3 therefore these two documents would be returned.

4.4.4 Advantages and drawbacks of Boolean Retrieval Model

The Boolean retrieval model enables users to formulate complex logical statements. However, the construction of Boolean queries can be difficult for an average user, and all the terms entered in a query are considered equally important. Due to the binary nature of the results the model does not provide a ranking of retrieved documents, only a set of retrieved document – without any particular order. Also, because an exact matching criterion is used the returned set of documents will be either almost empty (which is a low recall as many relevant document would not be retrieved) or will include many documents (therefore precision would be low as irrelevant document would be also in the set). An example of exact match query is science AND computer. In Boolean terms, the document has to contain both 'science' and 'computer' to satisfy the query. It means if one term is `missing, it will not be considered relevant at all, while if it contains both terms it will be considered fully relevant (Shah 2009). This model has some important limitations. As all terms are equally weighted, this model is more useful for data retrieval than information retrieval (Salton, Fox and Wu 1983). Also, it is often hard to translate an information need into Boolean expression. Finally because of the binary match documents are classified either as relevant or irrelevant, without any intermediate states. As a result the method often returns either very little or too many documents that are not ordered in any particular way (Naik and Rao 2011).

To eliminate the problem with different variants of the same words, each word in both dictionary and document can be represented without suffix. Also process of dictionary creation can be altered by representing synonymous as a single word in order to decrease the size of the database, and to eliminate the problem of exact words matching. During the dictionary creation, if a word has already a synonym in the dictionary then it does not have to be added to it. This requires that when a document is being indexed

and a new word is detected in it, then any synonym of that word existing in the dictionary will be considered as existing in the document. This approach can decrease the database size and eliminate the problem with checking for exact match only. However the precision of retrieval can decrease with these optimisations.

4.5 Vector Space Model (VSM)

The Vector Space Model (VSM) is an algebraic model used for information filtering, information retrieval, indexing and relevance ranking (Berry, Drmac and Elizabeth 1999, Polyvyanyy and Kuropka 2007). The Vector Space Model is a way of representing and comparing documents and queries based on words (keywords) with values (Berry, Drmac and Elizabeth 1999). This model can be used to rank the similarity between documents – not just to answer whether document contains required words or not. Each component of a vector represents one term/keyword, and has a value. The value is a real number that indicates how relevant a term is to the document or query being described (Berry, Drmac and Elizabeth 1999). VSM processing can be divided into two stages: Document Indexing with Term Weighting and Documents Relevancy Ranking.

4.5.1 Document Indexing

The first stage of information retrieval is document indexing. Each indexed document is represented as a vector of terms contained by the document and weights of each term. Weight of a term describes how important that term is to the document, e.g. terms from the documents' title will be more important than terms from the footer. The process of creating the vector includes stop words removal and stemming. Stop words like 'of', 'an', 'the', and etc are removed as there are not relevant to the document abstract (Singhal and Salton 1995). Words suffixes – like 'ed', 'ion', 'ing', 'ions' can be removed to avoid recording different variants of a single word.

The indexing process can cover an entire document or only part of a document. Some systems for example only index words from the document title and the abstract, while others index the entire document and then

modify the relevance value of each term depending on the term position in the document.

Every term has to be evaluated to estimate its importance in the document. In the basic implementation the rating can be set according to the number of times that a term occurs in a document. In general, VSM relies on two main factors for term weighting: Term Frequency vector (TF), and Inverse Document Frequency vector (IDF) (Abual-Rub, Abdullah and Rashid 2007). In a term frequency vector created for a document, the rating of a term depends on the number of occurrences of that term in the document. However, some words are very common (e.g. 'a', 'the', 'in') and therefore the rating for these terms would be very high – even if they are not important to the content of the document. To overcome this problem, an Inverse Document Frequency vector (IDF) is created. This vector stores the general importance of every term, in respect to all documents. It is generated by calculating the number of documents that contains each term. The weight for each term in the IDF vector is higher if the term is less popular and lower if it is more popular. The weight value for each term is calculated as the logarithm from the quotient of the total number of indexed documents divided by the number of documents in which the term appears. Once the term frequency vector for each document is created, and one inverse document frequency vector for all documents is ready, then the final document representation is created. The weight for each term in the vector representing each document is calculated by multiplying the weight from the term frequency vector for that document, with the rating of the term in the inverse document frequency vector. If that value would be used to calculate the relevance to a query then long documents would usually be considered more relevant, because each term can occur more times in a longer document. To resolve this issue, the generated is normalised, by dividing weight of each term in the vector by the vector length. The length of a vector is calculated as the square root from sum of squares of all weights in the vector. As the result of normalisation, the length of vectors for all documents is equal to one, and the length of each document does not affect the retrieval process (Singhal and Salton 1995).

4.5.2 Determination of document relevance in VSM

Once the documents are indexed, a search system can rank and order the documents according to the calculated similarity to a query. The query is represented in the same fashion as the documents – by term vector with ratings for each stored term – except that the normalisation of the vector is not essential.

The similarity between a single document and the query is calculated as a cosine similarity between two vectors. If the two vectors are displayed in the N dimensional Cartesian coordinate system (where N is the total number of terms in both vector, and each axis is representing the value of one term) then the cosine similarity would be equal to the cosine of the angle between the two vectors.

To calculate the cosine similarity, the weight of each term from one of the vectors is multiplied with the weight of the same term from other vector (zero weight is assumed if term does not exists), and then all values have to be summarised. Finally that value should be divided by the length of the first vector and by the length of the second vector.

$$Sim\,(D,P) = \frac{D \cdot P}{\|D\|\,\|P\|} = \frac{\sum_{i=1}^{m} d_i p_i}{\sqrt{\sum_{i=1}^{m} d_i^2}\,\sqrt{\sum_{i=1}^{m} p_i^2}} \qquad \text{[Equation 4.1]}$$

In the equation 4.1, D is the term vector for document, P is the term vector for the query, d_i and p_i are components of corresponding vectors.

As term vector for documents is normalised during the indexing, its length can be omitted as it is equal to 1 for all documents. The same applies to the query term vector – it can be normalised once.

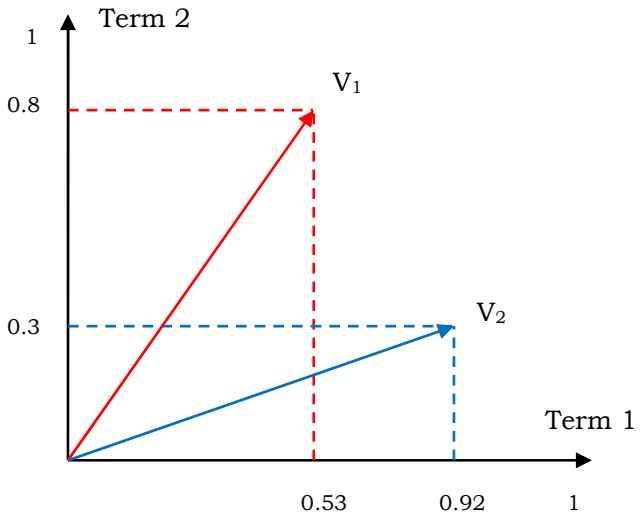


Figure 4.4: An example of two normalised vectors

The figure 4.4 shows an example of two normalised vectors and the cosine similarity between vectors V_1 and V_2 is calculated below.

V_1 = [0.53, 0.85]

V_2 = [0.92, 0.39]

$Sim(V_1, V_2) = \cos(\measuredangle V_1 V_2) = V_1 \cdot V_2 = [0.53, 0.85] \begin{bmatrix} 0.92 \\ 0.39 \end{bmatrix} = [0.53 \cdot 0.92 + 0.85 \cdot 0.39] = 0.4876 + 0.3315 = 0.8191$

The similarity calculated for the two vectors is 0.8191. This value is neither high nor low as the documents are simply sorted by the similarity. A system can present all documents to the user in that order, and the user can decide at which point documents are no longer relevant.

4.5.2.1 Example of VSM application

The example will consider four documents, and one query.

Document	Content
Document 1	Coventry university engineering
Document 2	Warwick university arts department
Document 3	Coventry university Computer Science department
Document 4	Coventry arts department

Figure 4.5: Example of VSM documents

Term:	Coventry	University	Computer	Science
Importance:	1	1	0.5	0.3

Figure 4.6: Example of VSM query

During the indexing, all the terms were extracted from the documents to create representation of each of the document. In the process an Inverse Document Frequency vector has to be generated. To generate IDF vector the indexer first has to create Document Frequency vector (DF) that for every term counts the number of documents that contains the term. Subsequently, the total number of documents is divided by the number of document that contains a specific term, and the logarithm of that value is stored in the Inverse Document Frequency vector (IDF) for that term.

The table in Figure 4.7 presents the importance of each term for each document (D1-D4), general importance of a term to all documents (Document Frequency vector DF), and the inverse document frequency vector (IDF). D/DF_i holds the total number document divided by the number of documents that contains the term.

Terms	**Query**	**D1**	**D2**	**D3**	**D4**	**DF**	**D/DF**	**IDF**
Coventry	1	1	0	1	1	3	1.33	0.12
University	1	1	1	1	0	3	1.33	0.12
Science	1	0	0	1	0	1	4.00	0.60
Engineering	0	1	0	0	0	1	4.00	0.60
Warwick	0	0	1	0	0	1	4.00	0.60
Arts	0	0	1	0	1	2	2.00	0.30
Department	0	0	1	1	1	3	1.33	0.12
Computer	1	0	0	1	0	1	4.00	0.60

Figure 4.7: Example of retrieved results with term frequency

After creating IDF vector, weights for each term in the Document Frequency vector should be multiplied the value for that term stored in the Inverse Document Frequency Vector.

The table in Figure 4.8 presents the values that result from the multiplication of each term frequency with its importance from the IDF vector.

Terms	Query	D1	D2	D3	D4
Coventry	0.12	0.12	0.00	0.12	0.12
University	0.12	0.12	0.12	0.12	0.00
Science	0.60	0.00	0.00	0.60	0.00
Engineering	0.00	0.60	0.00	0.00	0.00
Warwick	0.00	0.00	0.60	0.00	0.00
Arts	0.00	0.00	0.30	0.00	0.30
Department	0.00	0.00	0.12	0.12	0.12
Computer	0.60	0.00	0.00	0.60	0.00
Vector length	0.87	0.63	0.70	0.88	0.35

Figure 4.8: Terms ratings in documents after applying the IDF

All generated vectors should be normalised to eliminate the advantage given to the longer documents, as even if a term is repeated multiple times in longer documents, it should not be considered relevant to that document if it is flooded by other terms. The normalisation of a vector is simply a process of dividing weights of each term stored in that vector by the length of that vector. Vectors after the normalisation are presented on the figure 4.9 below.

Terms	Query	D1	D2	D3	D4
Coventry	0.14	0.20	0.00	0.14	0.36
University	0.14	0.20	0.18	0.14	0.00
Science	0.69	0.00	0.00	0.69	0.00
Engineering	0.00	0.96	0.00	0.00	0.00
Warwick	0.00	0.00	0.87	0.00	0.00
Arts	0.00	0.00	0.43	0.00	0.86
Department	0.00	0.00	0.18	0.14	0.36
Computer	0.69	0.00	0.00	0.69	0.00
Vector length	1.00	1.00	1.00	1.00	1.00

Figure 4.9: After vectors normalisation

After the indexing process is completed the system is ready to generate responses to queries. In order to retrieve the search results for a specific query, a similarity between the user query and each of the documents has to be calculated.

$$Sim\,(D,P) = \frac{D \cdot P}{\|D\|\,\|P\|} = \frac{\sum_{i=1}^{m} d_i p_i}{\sqrt{\sum_{i=1}^{m} d_i^2}\,\sqrt{\sum_{i=1}^{m} p_i^2}} \qquad \text{[Equation 4.2]}$$

As the document frequency vectors are normalised, its length ||D|| is equal to 1 and can be skipped from the formula. The query can also be normalised once to simplify the similarity computations for each document.

Term:	Coventry	University	Computer	Science
Importance:	1	1	0.5	0.3

Figure 4.10: Example of normalised query vector

The table below presents the normalised query vector (with zeroes for terms that do not exist in the query) and the document vectors representing the documents (D1 to D4). The values in columns Dn*Q for each term are calculated by, multiplying the weight of that term in the document representation with the term weight in the query.

Query Terms	Query	D1	D1*Q	D2	D2*Q	D3	D3*Q	D4	D4*Q
Coventry	0.43	0.32	0.14	0.26	0.11	0.16	0.07	0.00	0.00
University	0.43	0.00	0.00	0.00	0.00	0.78	0.34	0.00	0.00
Science	0.13	1.53	0.20	0.00	0.00	0.00	0.00	0.00	0.00
Engineering	0.00	0.00	0.00	1.24	0.00	0.00	0.00	0.00	0.00
Warwick	0.00	0.00	0.00	0.62	0.00	0.00	0.00	2.47	0.00
Arts	0.00	0.00	0.00	0.26	0.00	0.16	0.00	1.03	0.00
Department	0.00	0.00	0.00	0.00	0.00	0.78	0.00	0.00	0.00
Computing	0.21	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Similarity			0.34		0.11		0.40		0.00

Figure 4.11: Similarity between the document and the query

The similarity is calculated individually for each term and then summarised for the entire document. As both query and document frequency vectors are normalised, there is no need for dividing the calculated similarity by the vectors length. The system will return documents with similarity bigger that some threshold value and sorted by the similarity in descending order.

4.5.3 Advantages and Drawbacks of Vector Space Model

In contrast with the Boolean Retrieval Model, in the VSM a range of values can be applied to each term – both in documents representations and in the user query. Additionally, because of the normalisation of the vectors, long documents are not favoured over short ones and, because of the use of inverse document frequency vector, popular terms are not considered important while rare terms are promoted. In the Boolean Model a page containing a full dictionary would be considered relevant to most of the queries, unless they contain the NOT operator; in the VSM model each term on such page would be considered very irrelevant to the document and as such the document would not be considered highly similar to any short query. That apparent advantage can however be considered a drawback since long document that can contain terms specified in query only in the title and the abstract and yet be still very relevant to the query. The importance of these terms will be low in comparison to a short document that contains the same terms in the footer. More advanced application can calculate the importance of terms differently, for example by preferring terms that appear at the beginning of the document.

Another drawback of the VSM document representation is that the order of the terms is lost and it is not possible to prefer documents that contain query terms that are close to each other, over documents that contain terms separated in different parts of the document.

The VSM is also affected by problem of exact mach and synonymous, however effects of this problem are not as important as in the Boolean model, as the document will not be classified fully irrelevant if one of the words is missing. Also similar techniques that are used to reduce this

problem in the Boolean model can be used in the VSM (e.g. accepting not only extract matching but also synonyms when vectors are being compared). The user can also choose the minimum similarity of retrieved documents to increase the retrieval precision (Kulkarni, Srinivasan and Ramakrishna 1999) however increasing the threshold will also decrease the recall.

4.6 Probabilistic Information Retrieval (PIR)

Both Boolean Model and Vector Space Model provide similarity ratings without considering a level of certainty for the output relevance. There are several models based on probability theory that aim to determine the probability of a document being relevant to a query. (Manning, Raghavan and Schutze 2008).

The Probabilistic retrieval was first proposed by Manor and Kuhns in 1960, (Singhal 2001) and many variants of the Probabilistic Model have been proposed since. Amati et al. (1997) proposed a prototype information system called Profile, in which the user information is represents by a set of, possibly weighted, keywords given by the users or induced by the system. Manavoglu, Pavlov and Giles (2003) describe the behaviour model as a Probabilistic Model that tries to estimate the future actions of users.

4.6.1 Probabilistic Information Retrieval principles

The results retrieved by probabilistic information retrieval systems depend on estimations and probabilities. The first assumption is that terms are dispersed differently between relevant and non-relevant documents (Fuhr 1992). A PIR system ranks documents and sorts them in decreasing order of probability of relevance to the information need once the probability is calculated (Fuhr 1992). The results are as accurate as the calculated probability (Robertson, Rijsbergen and Porter 1981).

The classical probabilistic model returns documents in decreasing order of calculated probability of relevance to the information requirement. After the indexing process every term can have assigned a value that indicates the probability that a document containing this term is relevant to the concept

described by the term. In the retrieval phase the documents have calculated a value which is the sum of probabilities from terms that exists in both a document and in the query. The documents are then retrieved in order according to this value (descending). The document representation for this version of Probabilistic Information Retrieval could be the same as in the Boolean model as it only needs to store information if either document contains a term or not (Robertson, Rijsbergen and Porter 1981).

Similarly to the Inverse Document Vector in the VSM model, a vector has to be created that stores information about how important each term is. If p is the probability that a document which contains a term and it is relevant to the query and q is probability that the document contains the term but it is not relevant, then the weight of the term i is calculated as:

$$w_i = log \frac{p_i(1-q_i)}{q_i(1-p_i)}$$ [Equation 4.3]

Where:

$$p_i = \frac{number\ of\ relevant\ documents\ containing\ term}{total\ number\ relevant\ of\ documents}$$ [Equation 4.4]

$$q_i = \frac{number\ of\ irrerevant\ documents\ containing\ term}{total\ number\ of\ not\ relevant\ documents}$$ [Equation 4.5]

If

- n_i = Number of documents containing term i
- r_i = Number of relevant documents containing term i
- N = Total number of documents
- R = Number of relevant document

Then p_i and q_i can be expressed as

$$p_i = \frac{r_i}{R}$$ [Equation 4.6]

$$q_i = \frac{n_i - r_i}{N-R}$$ [Equation 4.7]

And w_i can be expressed as

$$w_i = log \frac{p_i(1-q_i)}{q_i(1-p_i)} = log \frac{r_i(N-R-n_i+r_i)}{(n_i-r_i)(R-r_i)} \quad \text{[Equation 4.8]}$$

As the number of relevant documents is unknown, some assumptions have to be made. Usually it is assumed that the probability p is constant (e.g. equal to 0.5), and that q can be estimated by the values from Inverse Document Frequency vector, created as in the Vector Space Model (Manning, Raghavan and Schutze 2008).

With the assumption that 50% of the documents containing a term are relevant, the number of relevant document containing the term and the number of irrelevant document containing the term will be equal and their sum will be zero in the denominator. To avoid division by zero when R-r = 0 or n-r=0, a 0.5 can be added to each component of the equation, as follows:

$$w_i = log \frac{(r_i+0.5)(N-R-n_i+r_i+0.5)}{(R-r_i+0.5)(n_i-r_i+0.5)} \quad \text{[Equation 4.9]}$$

The equation indicates how to calculate the probability that a document containing a specific term is relevant to a query with that term. It also considered as the weight of the term (Robertson 2004).

This model is based on uncertain guess of whether a document has relevant content to match the query and the document representation. The Probabilistic Retrieval Model uses the estimation method to retrieve the results based on assumptions that are made explicitly. Relevance feedback can improve the ranking by providing better assumptions.

4.6.2 Probabilistic Retrieval Example

The documents set contain four documents with following contents:

Document	Content
Document 1	Coventry university engineering
Document 2	Warwick university arts department
Document 3	Coventry university Computer Science department
Document 4	Coventry arts department

Figure 4.12: Example of documents

And the user query is: 'Coventry university computing science'.

The indexed documents have been presented in the Figure 4.13. The terms are extracted from the documents. Term weight is applied to each term. The weight of each term is calculated and it can be used in the same way as the *IDF* vector in the Vector Space Model.

Terms	**D1**	**D2**	**D3**	**D4**	**W(term weight)**
Coventry	1	0	1	1	-1.322
University	1	1	1	0	0.26
Science	0	0	1	0	-0.26
Engineering	1	0	0	0	-0.26
Warwick	0	1	0	0	1.32
Arts	0	1	0	1	0.48
Department	0	1	1	1	0.26
Computing	0	0	1	0	-0.26

Figure 4.13: calculating the term weight

Terms	Query	D1*W	D2*W	D3*W	D4*W
Coventry	1	-1.322	0	-1.322	-1.322
University	1	0.26	0.26	0.26	0
Science	1	0	0	-0.26	0
Engineering	0	0	0	0	0
Warwick	0	0	0	0	0
Arts	0	0	0	0	0
Department	0	0	0	0	0
Computing	1	0	0	-0.26	0
Total		-1.062	0.26	-1.582	-1.322

Figure 4.14: Calculating the relevance values

After term weights are calculated, the relevance values can be calculated for each of the documents.

The system returns documents with descending order of calculated relevance.

4.7 Advantages and Drawbacks of Probabilistic Retrieval Model

The Probabilistic retrieval Model is based on assumptions of the number of documents that are relevant and the number of documents that are non-relevant. (Naik and Rao 2011). These assumptions are made explicitly – like assuming that 50% of document containing a term are relevant to that term – however not all assumptions fit the reality which affect retrieval precision and recall (Fuhr, 1992). The total number of relevant documents has to be guessed and p is a constant which is not always true (Jones, Walker and Robertson 2000). To achieve precise results the probabilistic retrieval model requires that terms are independent. The weight calculation ignores the term frequency and position within documents, and therefore longer documents are promoted.

4.8 Alternative retrieval models

In their basic form the three types of retrieval model are still being used extensively thanks to the simplicity of the underlying formalism (Manning, Raghavan and Schutze 2008). Many researchers have however proposed variant models as part of an effort to overcome the limitations of the original proposals and to extend their range of application. For example the Extended Boolean Model is a combination of some of the features of the Vector Space Model with Boolean algebra. This enhancement of the Boolean model allows for the return of results based on the ranking of similarity and on the partial matching of terms in query and document (Skorkovská and Ircing 2009).

Latent Semantic Indexing (LSI) is based on the VSM and was introduced as a method for reducing the negative impact of synonymy and polysemy on the retrieval process (Deerwester et al. 1990). It consists in applying a mathematical technique, the Singular Value Decomposition (SVD) to terms and term frequency in order to identify patterns of relationships. This method can be fully automated, is independent of language and leads to shorter vector representation. Empirical studies have confirmed the viability of the method and its applicability to different contexts (Bradford 2008).

A statistical language model falls within the probabilistic category. In information retrieval, language modelling consists in estimating the likelihood that a common language can generate the query and the document under consideration. The corresponding probability distribution is the language model. Evidence indicates that language models can be applied to a variety of retrieval problems and that they can produce better performance than traditional methods (ChengXiang 2008).

Feature-based retrieval models represent a significant departure from the three classical models presented earlier. In these models a document is seen as a vector of values of feature functions and the aim is to generate a single relevance score by manipulating and combining these features. It is claimed that with the right selection of features feature-based models can outperform most existing retrieval models (Metzler and Croft 2007).

4.9 Summary

Three models supporting information retrieval were covered, with a particular emphasis on their mode of representation of the documents and their processing algorithms. As the oldest model the Boolean Retrieval model has the advantage of simplicity and convenience. It is however restrictive in the formulation of similarities. Similarity can be in two states only: true or false. At the other extreme, the probabilistic approach is an attempt to model the subjective features of the information retrieval process over a range of probabilities. The calculation of probabilities requires the specification of assumptions that can be highly biased and inconsistent. The document representation in the probabilistic model is very simple and ignores terms frequency or position. The Vector Space Model on the other hand combines clarity with flexibility. The underlying algebraic model is well specified and well understood, and the documents are represented with more details. The VSM offers a viable compromise in information retrieval processing.

Chapter 5

A Mediation Framework

5.1 Introduction

The aim of this chapter is to present an approach to personalisation in Web Search and its application in the design and implementation of a mediation framework. The rationale and the context of the approach are presented first. The approach is motivated by the need to investigate the impact of personalisation and different modes of profile generation on the retrieval process, in order to enhance its effectiveness in terms of precision and recall. In the approach, specific user profile formulation and document content are given equal consideration, a characteristic that favours the adoption of a content-based method.

The framework has been designed and implemented in a way that allows the evaluation of different information retrieval techniques without implementing a full search engine. This framework will be the foundation for the evaluation and comparison of explicit, implicit and hybrid user profiling. The same framework can be used by other researchers or students to evaluate their ideas, by simply replacing the subsystem responsible for filtering document.

5.2 Rationale and context

The research on user profiles has identified two extremes of profile generation, explicit and implicit (Frias-martinez et al. 2009).

Their characteristics are often compared and contrasted in terms of the level of user intervention and in terms of system support and interpretation. Some studies have highlighted the fact that users prefer transparency and control in the systems they use. These studies also indicate that too much flexibility in the customisation process, such as editing profiles, can have an adverse effect on personalisation (Ahn et al. 2007).

One of the key issues in the personalisation process is how to address 'the cold start problem'. The assumption that a significant amount of explicit feedback is required in order to build a profile has led to more emphasis on implicit feedback and on the synergy of user communities, rather than rely on explicitly formulated profiles (Zigoris and Zhang 2006).

Besides the dismissal of what is considered the 'brittle models' of the explicit profiles and their lack of relevance, many of the systems on user personalisation are increasingly relying on social networks to provide additional implicit information on user behaviour, and by implication pave the way for recommendation procedures (Cayzer and Michlmayr 2008).

Although this approach has the advantage of creating a richer context of interaction, it has the drawback of postulating the existence of a social network, an assumption that may affect its operational dependence. Another disadvantage of this approach is the undue weight it gives to the implicitly generated user information**.** A fact that many controlled studies have reported is the correlation between the usefulness of documents to users and many of their interactive activities such as time spent viewing a document and other operations such as saving and printing them (Fox et al. 2005).

It was however reported that the information that a user is searching for has a significant impact on the usefulness of the implicit feedback (Kelly and Belkin 2004).

Despite the strong contrast between implicit profile and explicit profiles that seem to drive some studies, many researchers have however pointed out that the implicit and the explicit positive feedback are complementary (Jawaheer, Szomszor and Kostkova 2010).

An approach is proposed that seeks to overcome the limitations identified earlier and to capitalise on the complementary nature of explicit and implicit profiles. It also marks a departure from the 'feedback' related to explicit profiles, in order to minimise user intrusion and inconvenience. In contrast the focus is on the profile formulation by the user. This shift of emphasis means that the user has some control over the personalisation, while the concurrent implicit profile generation maintains the currency of the user interests. In the proposed approach, prominence is given to the user, the document and their interaction. This perspective is well served by a content-based approach rather than a collaborative approach. It provides focus, control and wider application. The content based approach allows the system to harvest relevant user information without the need of a community of users.

The novelty of the work lies in the seamless and balanced combination of discrete intervention and transparent implicit profile generation. No explicit feedback is required during the interaction with the documents such as for example rating the relevance of each document. Instead the user is allowed to state relevant interests in terms keywords. A number of key factors form the basis of the implicit feedback mechanism.

5.3 Design requirements and issues

There are a number issues related to the evaluation of any information retrieval system designed for the web. The major issue is the amount of

documents available. Implementing and running a full web search engine is usually unfeasible. However for some techniques, instead of developing the search engine, an information retrieval technique can be evaluated by filtering only a subset of web documents, where this subset would be retrieved from a classic web search engine API.

For the framework to be useful, it has to meet several objectives. First it has to allow a programmer to implement custom methods for learning. The framework should provide an interface that allows tracking all actions detected in a web browser, like navigating or clicking. A second important requirement is that the framework should support transparency for retrieving search results from a base web search engine. A person modifying the framework in order to evaluate different filtering techniques should be able to evaluate different filtering techniques by modifying only the filtering method, and by handling events from the browser if needed. The framework should provide a way of entering user's explicit profile, however only a basic method of explicit profile generation needs to be supported in the framework, and for more advanced methods some changes in the user interface may be required. By default users provide explicit information only by entering keywords of interest; however a programmer should be allowed to extend the user interface if more explicit information is required.

5.4 Overall Architecture of the mediation framework

The framework should provide a platform for addressing a number of issues. The ability to accommodate different modes of mediation, the prominence of content and of patterns of behaviour as well as efficient representation should be the guiding factors in the design of the framework. The refined requirements for the mediation framework are expressed as follows:

- Three forms of user profile generation will be provided: explicit, implicit and hybrid. This will enhance the flexibility of the system and

will offer a wider scope for the investigation of the impact of personalisation on Web search.

- A content-based approach will be adopted for the mediation framework because of its focus on the interaction between the specification of the profile of a single user and the content of documents. It will allow direct evaluation of different modes of profile generation. It has also the benefit of a clear identification of the factors that affect the search process.
- The Vector Space Model will be used for the determination of the similarity between users and documents and hence for the filtering process. The VSM combines clarity with simplicity and offers an efficient method for document representation. It also allows for the consistent use of weights for the terms in the query representations for the three modes of profile generation.

These design decisions have been translated into the architecture for the framework as shown in Figure 5.1. It presents a diagram of the mediation framework which is made up of three fundamental components: user profile generation and representation, document keyword extraction and representation, and document filtering.

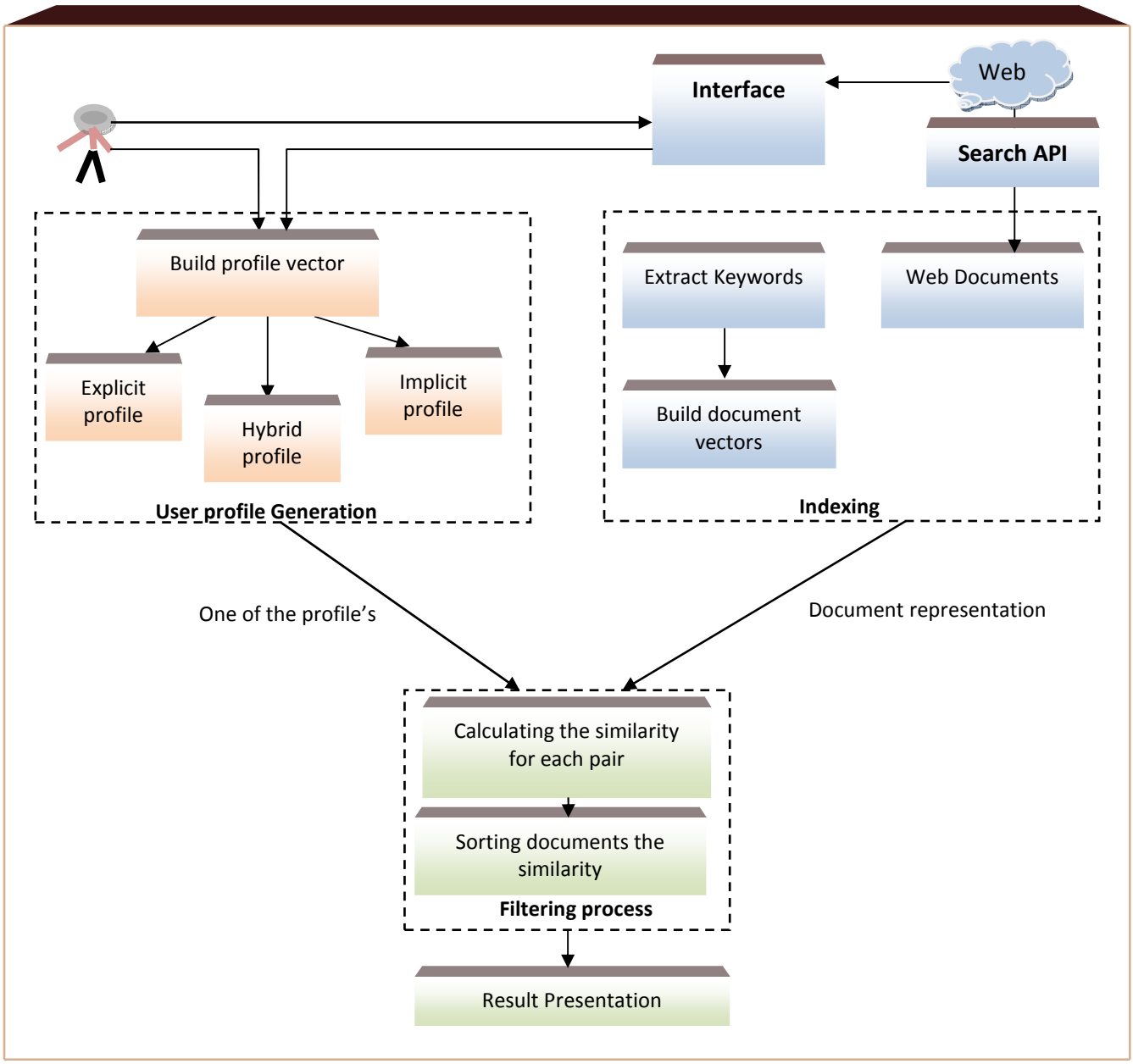


Figure 5.1: Overall Mediation framework

In the framework, a user is interacting with the web through a provided web browser. A user profile can be generated from data provided explicitly by the user or from events generated by the web browser. Once the search has to be performed, a group of documents returned by base search API is indexed and filtered with use of the created previously user profile.

5.4.1 User profile generation

Three different methods of profile generation have been investigated – explicit, implicit, and hybrid. In each case a user profile is represented in the VSM by a list of keywords with weights stored as a term vector:

$$P = (<p_1, w_1>,\ldots, <p_i, w_i>\ldots, <p_n, w_n>) \quad \text{[Equation 5.1]}$$

In equation 5.1 the keyword is presented by p_i and its weight by w_i. The vector representation of the profiles is the same is in all the three mediation systems; however the way in which weights for each term are determined may vary.

5.4.1.1 Explicit profile

An explicit user profile is an instantiation preferences and interests provided explicitly by the users in terms of keywords. The keywords are stored as a term vector where all the weights are equal. Single user profile can be composed of several keywords groups, which are stored separately in a database and retrieved by name. A user can build several profiles for finer customisation. In a most simple form, the explicit user profiling can take a form, in which a user is entering a new profile as a list of keywords before every search.

5.4.1.2 Implicit profile

An implicit profile is based on observation of user behaviour and browsing history. A system monitors the user's browsing activities by checking the history of visited pages, the time spent on each page and the document print and save actions. The time spent on each page is assumed to be a good way of estimating the user interest in that web page. The total time spent on each page is determined by the time when the user starts reading a page and the time the user moves away from the page. Saving a web page, or printing it, also indicates higher user interest towards that particular page. The system is tracking this browsing behaviour, and stores it together with a keyword

vector created for each of the documents. Based on the collected information, the system selects the keywords from documents that are considered relevant documents, and builds the implicit profile of the user from these keywords, by adding the vectors together. Before adding, each vector can be scaled to reflect different estimated importance of each of the documents, e.g. weights for every keyword in a vector representing a printed document can be multiplied by some arbitrary value to reflect that these keywords are more likely to describe relevant documents.

5.4.1.3 Hybrid profile

In the hybrid profile the explicit and implicit profiles are generated independently and combined into a single term vector. The weights of the keyword are adjusted according to the mode of mediation.

5.4.2 Document representation

In the VSM a document is represented by a term vector. Each word in a document is represented by a separate dimension of the vector. The keywords are extracted during web documents analysis from the title and meta-data to build the term vector.

$$D = (<d_1, w_1>, <d_i, w_i>..., <d_n, w_n>)$$ [Equation 5.2]

For the implicit and hybrid system the frequency vectors generated for each document may be ranked according to importance depending on the user activities such as time spent reading the document and browsing, printing and saving.

5.4.3 Document filtering

The VSM model can be applied to filter the results by determining the degree of similarity between individual user profiles and documents content. Each dimension of a vector represents a word (keyword) and a weight value in that dimension determines the importance of that word. Based on the weights of

the corresponding matching terms the similarity between a document and a query can be measured. The cosine measure is used for this purpose. The cosine similarity function is determined by the following formula:

$$Sim\ (D,P) = \frac{D \cdot P}{\|D\|\ \|P\|} = \frac{\sum_{i=1}^{m} d_i p_i}{\sqrt{\sum_{i=1}^{m} d_i^2}\ \sqrt{\sum_{i=1}^{m} p_i^2}} \quad \text{[Equation 5.3]}$$

Equation 3 defines the similarity function where, D is a document vector (D = (d_i…..d_m)) and P is a user profile vector (P = p_i…..p_m)).

If vectors D and P are normalised then ||D|| = ||P|| = 1 and the formula can be simplified to:

$$Sim\ (D,P)\ = \sum_{i=1}^{m} d_i p_i \quad \text{[Equation 5.4]}$$

The keywords that appear only in one of the two vectors are ignored (as if weight value for not existing keywords is equal to zero). The determination of the similarity is illustrated by an example. For example, if the user profile P = (<science, 0.74>, <museum, 0.55>) – term “science” has a weight 0.74 and term “museum” has a weight 0.55, and all others terms weight will be consider as 0. For the document frequency vector D = (<museum, 0.82>, <history, 0.51>, <nature, 0.31>,) the similarity is equal to 0.55*0.82 (word 'museum') + P_i *0 (other words from vector P not existing in vector D) + 0*W_i (other words from vector D, not existing in vector P) which gives the similarity value 0.451.

5.4.4 Implementation

The system has been implemented in Java. Java supports heterogeneous programming and provides an easy way for interacting with web search engines APIs. The system utilizes several components to perform a web search based on explicit profiling of user, implicit profiling of user and combination of both profiles. As the system is operating on vectors, the IGLU-Java package has been used in the implementation. This package offers facilities for creating and manipulating various types of vectors,

including vectors capable of assigning values to words. The current implementation of the prototype relies on the TermVector class from the IGLU package. The same package is providing methods for calculating the cosine similarity between vectors. The implemented code and UML diagrams are provided in Appendix A.

All the systems use the same techniques for interacting with a classic web search API (both Google and Yahoo have been used) and for extracting the keywords. This section describes the methods and techniques that are common to the three systems. The main classes and the methods which are particular to each system will be presented later.

5.4.4.1 Web search

The web search component provides the facilities for interacting with a traditional web search engine API. The interaction is made by sending standard HTTP request with parameters appended to URLs, and the response is usually returned in XML (Extensible Markup Language) or JSON (JavaScript Object Notation) format – depending on the API used. JSON is a text-based open standard designed for human-readable data interchange.

To get the basic search engine API result the following method is called:

```
/**
 * Main method for searching in Google on Yahoo API.
 * Maximum 'NrOfResultsFormBaseAPI' results will be returned.
 */
public static List<String> searchForUrl(String[] keywords, API_TYPE api)
{
    if( api == API_TYPE.API_GOOGLE )
    {
        // Google API only allows 8 results per page, therefore
        //   request has to be send N/8 times to get N results – once for each page
        int noOfPages = NrOfResultsFormBaseAPI/8;
        List<String> results = new LinkedList<String>();

        for(int i=0; i <= noOfPages; i++)
        {
            String json = search_Google(keywords, i);
            List<String> pageResults = parseResultsFromGoogleJSON(json);

            for(String url:pageResults )
                if(results.contains(url)) continue;
                else results.add(url);
        }
        return results;
```

```
        }
        else
        if( api == API_TYPE.API_YAHOO )
        {
            // In Yahoo API all results can be received in one step
            String searchResult = search_Yahoo(keywords, NrOfResultsFormBaseAPI);
            return findURLsFromYahooResponse(searchResult);
        }

        return null;
    }
```

Parameter 'api' defines which search API should be used and the parameter 'keywords' is an array of keywords. The code for making request to Yahoo! Web search API is shown below:

```
    private static String search_Google(String[] keywords, int pageNumber)
    {
        // Encode keywords array to URL compatible string
        String encodedKeywords = [...];


        String request = "http://api.search.yahoo.com/WebSearchService/V1/webSearch?"
         +"appid=YahooDemo&results=" + nrOfYahooResults
         + "&query=" + encodedKeywords;

        // use HTTP client to send the request
        HttpClient client = new HttpClient();
        GetMethod method = new GetMethod(request);

        // read HTTP response from the API
        String result = [...];

        return result;
    }
```

In the response for the Yahoo API the following XML script is used:

```
<Result>
   <Title>[…]</Title>
   <Summary>[…]</Summary>
      <Url>http://www.neopets.com/userinfo.phtml</Url>
      […]
   </Result>
[…]
```

The response is parsed in the findURLfromYahooResponse method just like any other XML file. Values inside <Url></Url> tags are returned as a list of the search API result. The procedure for Google API is very similar:

```
    private static String search_Google(String[] keywords, int pageNumber)
    {
        // Encode keywords array to URL compatible string
        String encodedKeywords = [...];

        String request = "https://ajax.googleapis.com/ajax/services/search/web"
          +
"?v=1.0&key=AIzaSyBk2lMNQ2RfKR3l5TirizdIb2KdByrVySI&userip=194.66.32.16&rsz=8&q="
          + encodedKeywords + "&start=" + pageNumber;

        // use HTTP client to send the request
        HttpClient client = new HttpClient();
        GetMethod method = new GetMethod(request);

        // read HTTP response from the API
        String result = [...];

        return result;
    }
```

As Google API is limited to return only 8 responses per request, the request has to be made multiple times with different values for 'pageNumber'. The responses from each request are joined together.

To get the list of ULRs from the Google search API the following JSON (JavaScript Object Notation) document has to be processed.

```
"responseData":
{
   "results":
   [
      {
         "GsearchResultClass":"GwebSearch",
         "unescapedUrl":"http://en.wikipedia.org/",
         "url":"http://en.wikipedia.org/",
         […]
      },
      […]
   ]
}
```

5.4.4.2 Keywords extraction

Once the list of URLs from the base search API has been obtained, the system extracts the keywords from the documents in the list of returned URLs. Three frequency vectors are built individually from the title, the meta-tag keywords and the meta-tag description. The three vectors are then scaled by different weights and combined into a single vector. The final vector is created by selecting the top 5 keywords from the combined vector and then normalised as shown in the code fragment below:

```
public static TermVector findKeywords(String url) {
   [...]

   String document = MyUtils.UtilsWeb.getURL(url);

   String title = [read title tag];
   String metakeywords = [read meta tag with keywords];
   String metadescription = [read meta tag with description];

   // build vetors
   FrequencyVectorCreator fvc = new FrequencyVectorCreator();

   TermVector vectTitle = buildVectorFromString(title);
   TermVector vectKeyw = buildVectorFromString(metakeywords);
   TermVector vectDesc = buildVectorFromString(metadescription);

   // scale vectors
   vectTitle.scaleBy(0.3);
   vectKeyw.scaleBy(0.5);
   vectDesc.scaleBy(0.2);

   // combine three vectors into one
   HashFigure<String, Double> pairs = new HashFigure<String, Double>();
   addVector(pairs, vectTitle);
   addVector(pairs, vectKeyw);
   addVector(pairs, vectDesc);

   TermVector combinedVector = new TermVector();
   Iterator<Entry<String, Double>> it = pairs.entrySet().iterator();

   while(it.hasNext())
   {
      Entry<String, Double> entry = it.next();
      combinedVector.put(entry.getKey(), entry.getValue());
   }

   // take top N keywords and normalize

   combinedVector.truncateTo(5);
   combinedVector.normalize();

   return combinedVector;
   [...]
}
```

The sequence diagram for the findKeywords method is presented in Figure 5.2. That method is creating a vector containing all term from the given string, with weight for each term equal to the number of its occurrences in the string.

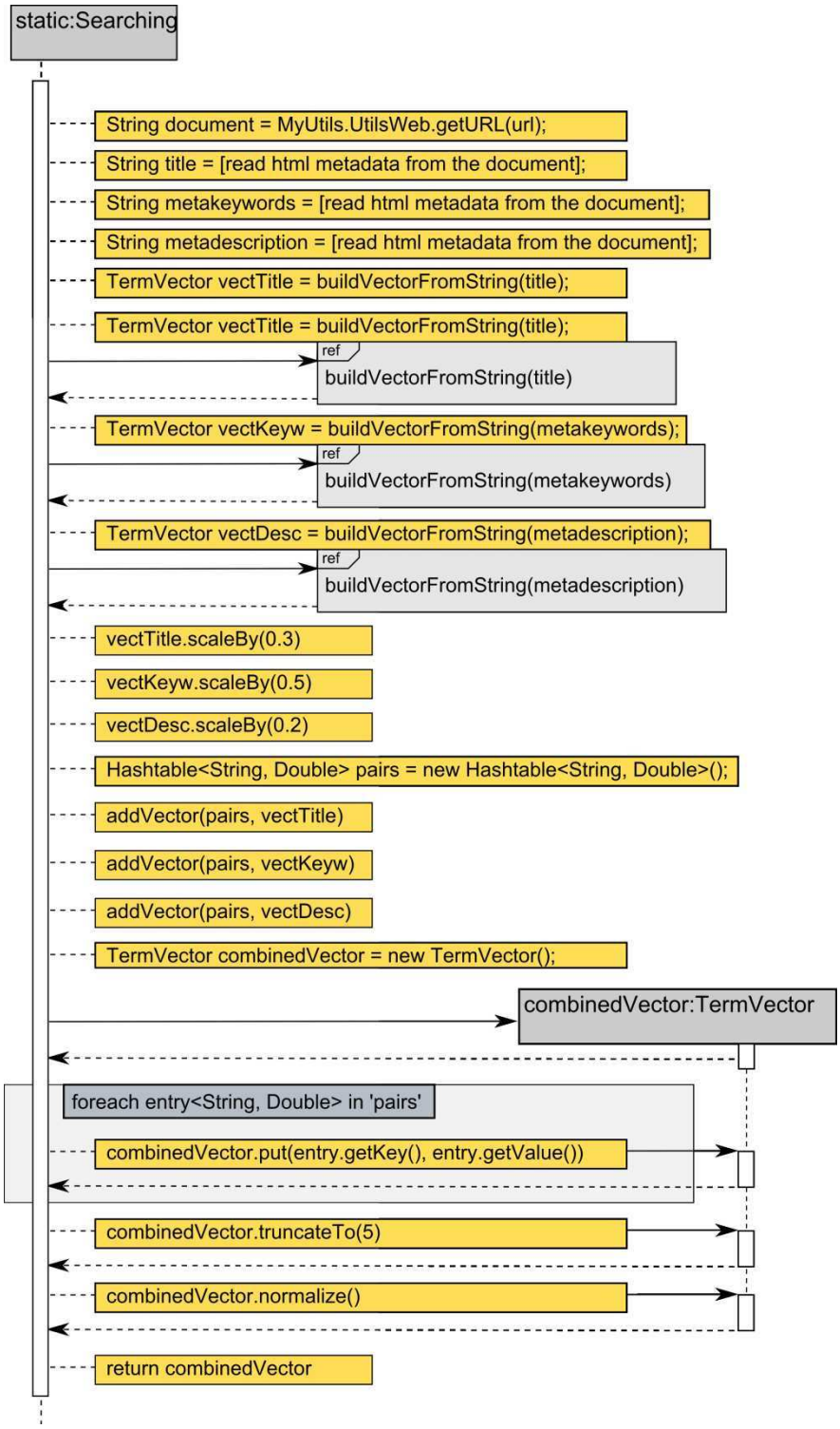


Figure 5.2: The sequence diagram for the findKeywords method

The diagram for the referenced 'buildVectorFromString' method is presented in Figure 5.3). Indexable words from a given document are counted and returned as a vector, with weight for each term equal to the number of its occurrences.

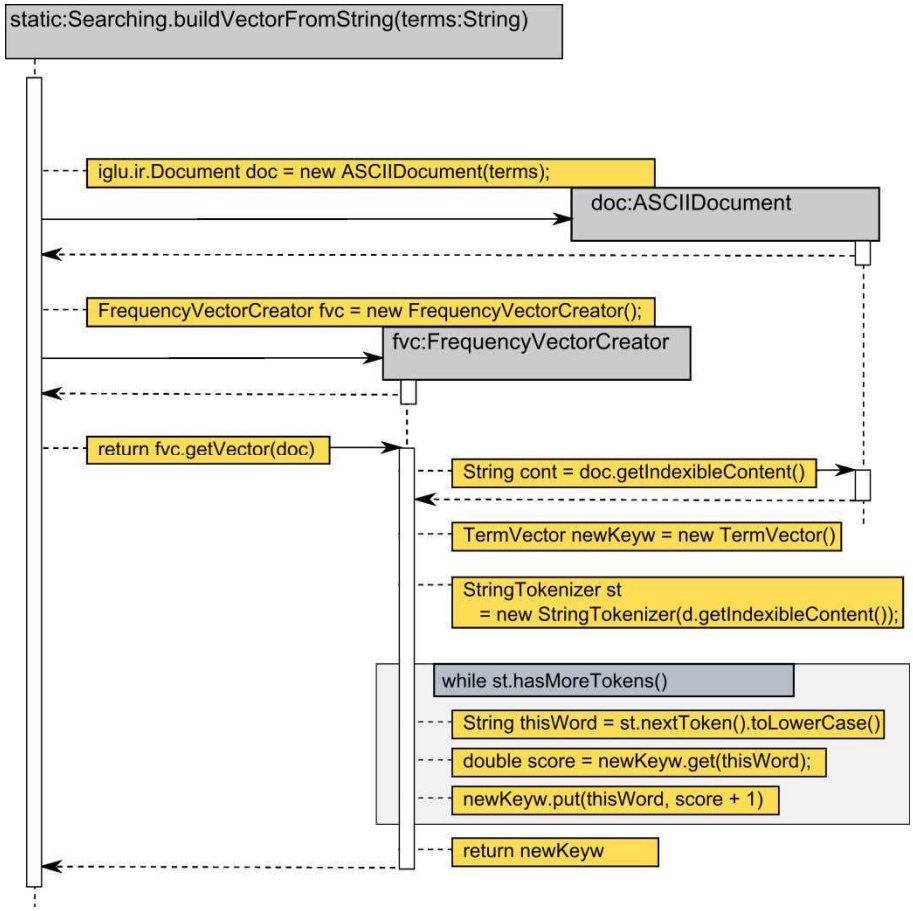


Figure 5.3: The sequence diagram for the buildVectorFromString method

The same process of extracting the keywords is repeated for every URL in the search results - one vector is created for each page. If a document cannot be accessed then a null value is returned. The exception handling code has been removed from the fragments for simplicity.

5.4.4.3 Documents filtering

The vectors are created for both selected user profile and the documents found by the base web search API. The term vector (user profile) and

frequency vector (document) are normalised before the similarity for each document is calculated. Once all the vectors are generated, the VSM model can be applied to filter the results by determining the degree of similarity between term vector and documents vectors. Each dimension of the VSM vector represents a single word (keyword) and its weight determines how important that word is in that vector. If a keyword in the document vector exists also in the user profile vector then its importance weight is dependent on the keyword weight in both vectors, otherwise its importance is zero. If a word exists in both vectors then the corresponding values in each vector are multiplied otherwise the word is ignored. The total similarity is the sum of the values calculated for each word separately.

After applying the VSM similarity calculation the system filters the documents by removing all but top M with the highest similarity. The following code is used to determine the similarity between two vectors:

```
public double getSimilarityScore (TermVector profileVector,
                                                    TermVector docFreqVector)
{
   // normalize both vectors
   profileVector.normalize ();
   docFreqVector.normalize ();

   // calculate cosine similarity
   double result = 0;
   Iterator it = profileVector.termIterator ();
   while (it.hasNext ())
   {
      // for each term in first vector multiply its value with its value in another vector
      //   if a term only exists in one vector, its value in other is zero therefore
      //   the term is ignored
      String thisTerm = (String) it.next ();
      result += profileVector.get (thisTerm) * docFreqVector.get (thisTerm);
   }

   return result;
}
```

The filtering of the web documents relies on the similarity between profile vector and document vector. The higher is the value of the similarity, the higher will be the position of the document in the results.

The documents sorting and filtering algorithm is represented by the following pseudo code:

```
    // create term vector for stored user preferences
    TermVector profileVector = [get user profile created by one of the methods]

    // list to store documents in descending similarity order
    ValueSortedMap results = new ValueSortedMap();

    Foreach retrieved 'document':
    {
        // get frequency vector for the document
        TermVector docFreqVector = (TermVector)idVectorMap.get(document);

        // measure the similarity
        double similarity = getSimilarityScore(profileVector, docFreqVector);

        // only include documents with positive similarity
        if(similarity > 0) results.put(document, similarity);
    }

    // results list is automatically sorter. To return top-N results:
    result.truncateTo(N);
```

The algorithm is implemented with the following java code:

```
private List<String> sortDocumentBySimilarity(List<String> webSearchAPIResult,
                                                          TermVector preferences)
{
   // create search engine for similarity comparison
   RAMSearchEngine rse = new RAMSearchEngine(); // parseXML(yahooAPISearchResult);

   // find keywords for every document
   HashMap<String, TermVector> documents = findKeywords(webSearchAPIResult);

   //-- add results to rse
   for(String url:webSearchAPIResult)
      rse.addDocument(url, "", documentKeywords);

   // get document sorted by similarity to preference vector
   iglu.util.ValueSortedMap vsm = rse.retrieveDocuments(preferences, 20);

   // return vsm as a list
   List<String> results = new LinkedList<String>();
   java.util.Iterator itr = vsm.keyIterator();
   while (itr.hasNext())
      results.add(itr.next().toString());

   return results;
}

// from RAMSearchEngine.java
public ValueSortedMap retrieveDocuments(TermVector vector, int numSimilar)
{
   ValueSortedMap results = new ValueSortedMap();

   // for each doc
   Iterator docIt = idVectorMap.keySet().iterator();
   while(docIt.hasNext())
   {
      // get similarity to vector
      Object thisItem = docIt.next();
      TermVector thisVec = (TermVector)idVectorMap.get(thisItem);
      double similarity = getSimilarityScore(vector, thisVec);
```

```
      if(similarity > 0)
         results.put(thisItem, similarity);
   }

   if(numSimilar > 0)
      results.truncateTo(numSimilar);

   return results;
}

// from RAMSearchEngine.java
public double getSimilarityScore(TermVector vector1, TermVector vector2)
{
   double result = 0;

   Iterator it = vector1.termIterator();
   while(it.hasNext())
   {
      String thisTerm = (String)it.next();
      result = result + vector1.get(thisTerm) * vector2.get(thisTerm);
   }

   return result;
}
```

A simplified sequence diagram for this code is presented on the figure 5.4

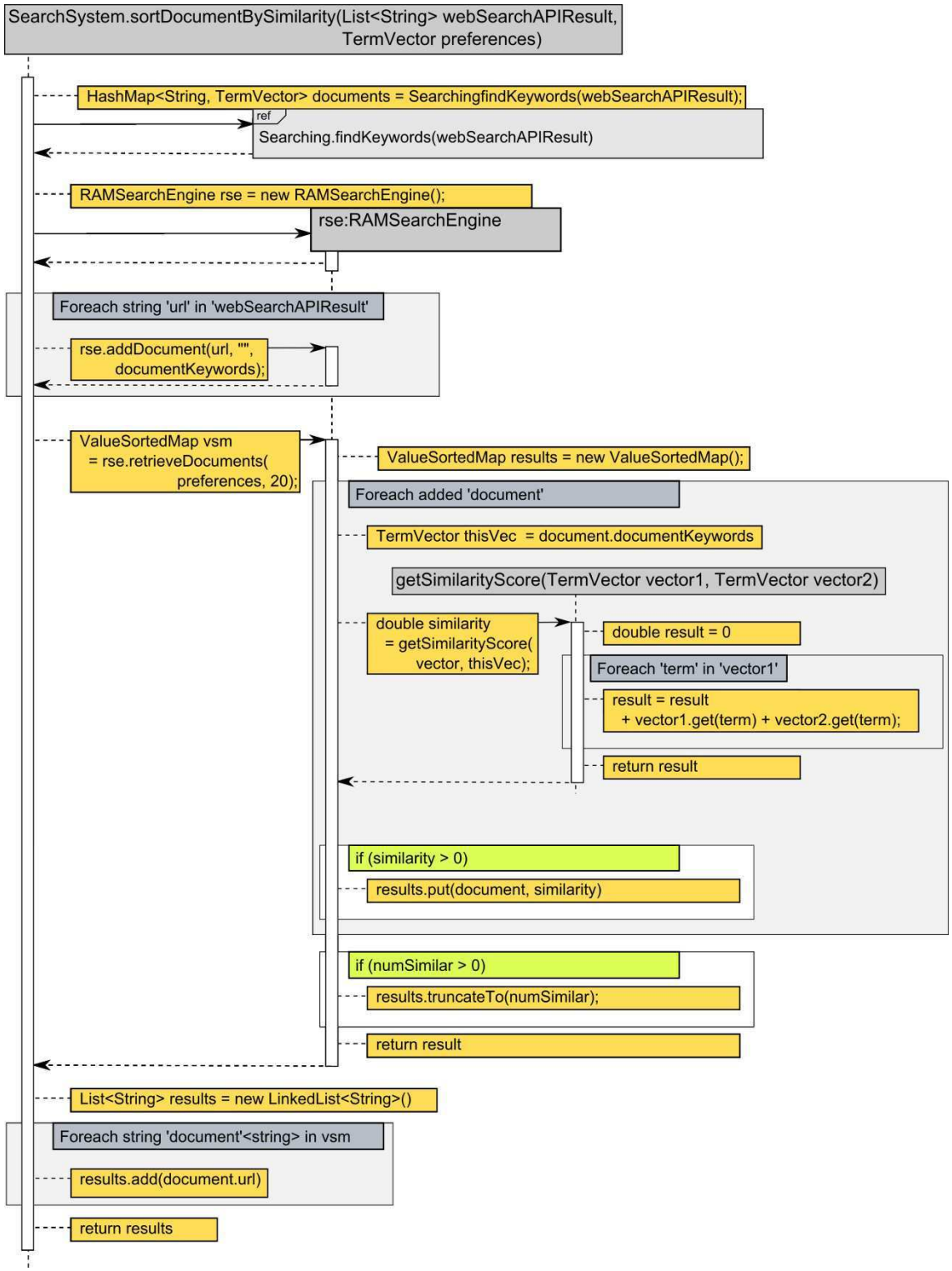


Figure 5.4: The sequence diagram for calculating the similarity between vectors

The code is calculating the similarity between vectors representing each of the documents and the profile vector individually (getSimilarityScore), and afterwards, it sorts the documents according to the calculated similarity (by Using ValueSortedMap).

5.5 Mediation systems

This section is concerned with the description of the three different mediation systems which were designed and implemented. The prototypes were implemented in JAVA in the NetBeans IDE.

5.5.1 Explicit mediation system

The explicit mediation system requires the user to provide a list of keywords in order to generate the explicit profile vector. All keywords are assumed to be equally important and have the same weight in the vector. A user can have more than one vector. The user profile can be modified later, at any time by adding, deleting or modifying existing vectors. After creating the profile the system stores it in the database.

Figure 5.5 presents the overall architecture of the explicit mediation system. The explicit user profile vector which can be used to retrieve a list of documents URLs by using a web search engine API (Yahoo and Google APIs are available in the implemented prototype). The system creates a frequency vector for each document. After building the vectors for the user profile and for each document, the system applies the VSM to calculate the cosine similarity between the profile vector and each of the document vectors. The documents are sorted in descending order of similarity and the system presents the top 20 of the documents to the users.

Figure 5.5: Explicit system

5.5.1.1 Implementation of the explicit system

The class diagram of the explicit system provides the better understanding of the system behaviour. The current prototype consists of two main classes shown below:

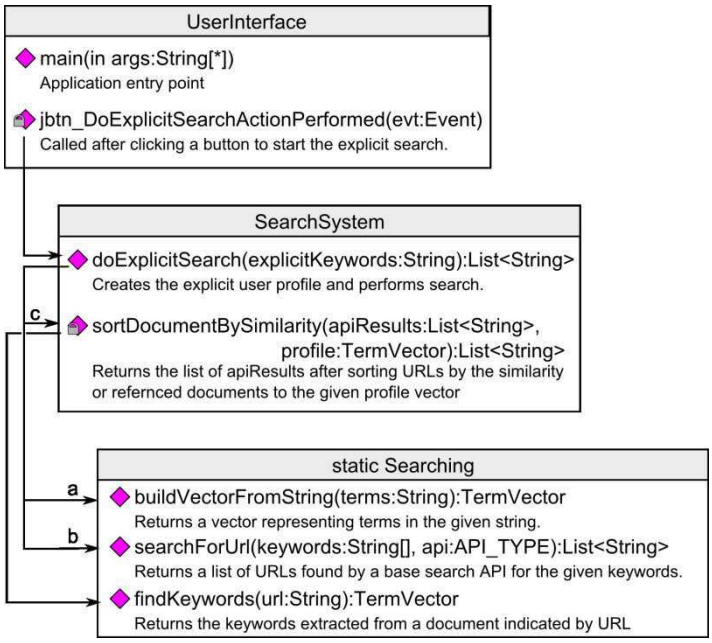


Figure 5.6: Simplified class diagram for the explicit system

Figure 5.6 presents the explicit system class diagram. The 'UserInterface' class is the main interface class for the use of the system explicitly. 'UserInterface' class provides the GUI (Graphic User Interface) and use the 'SearchSystem class for the searching functionality. The 'Searching' class provides static methods for calling Yahoo or Google web search API as well as methods for retrieving keywords for a web document. The explicit search is performed by the 'doExplicitSearch' method, which builds the explicit user profile (a) from keywords provided as a string, performs search in the base search API (b), and sorts the retrieved URLs by the similarity to the user profile (c).

Additional classes – TermVector, RAMSearchEngine, ValueSortedMap are used (in order) for storing frequency and term vectors, calculating documents similarities, and storing a list of documents in order of similarity to a user profile.

5.5.1.2 Explicit profile system database

Explicit user profiles are stored in a single database table.

Figure 5.7: Explicit profile table

The 'ID' column values are generated as auto increment numbers. The 'userName' column value is the text used to distinguish different user profiles. Finally, the 'keywords' column holds the list of keywords separated with spaces.

5.5.1.3 Creating explicit profile – pseudo code

The detailed description of the main system function – explicit search - is provided in the following pseudo code:

1. Allow user to choose explicit profile from one of the profiles the database
2. Retrieve list of URLs suggested by a classic web search API (either Google or Yahoo) for the keywords in the profile term vector
3. Normalise the profile term vector
4. For each of the suggested document URL
    a. Calculate the keywords freq. vector for the document
        i. Read the title, keywords and description from the document metadata
        ii. Create frequency vector for title, keywords and description
        iii. Scale each of the vectors by its importance
            1. Scale the vector made from the title by 0.3
            2. Scale the vector made from the description 0.5
            3. Scale the vector made from the meta-keywords 0.2
        iv. Add terms from all vectors into one combined document keywords vector - if a keyword exists in more than one vector, then its value in new vector is a sum of the value from each

vectors

v. Remove all but top N raked keywords

b. Normalise the document keywords vector

i. length = 0

ii. For each keywords length += keyword_rating2

iii. scale the vector by [1/square_root(length)]

c. Similarity = 0

d. For each keyword in either vectors

i. Get keyword rating from the term vector

ii. Get keyword rating from the freq. vector

iii. Add the result of multiplication to the similarity

iv. Store document similarity

5. Sort the documents by similarity
6. Return top N documents with highest similarity

5.5.2 Implicit mediation system

The implicit system architecture is shown in Figure 5.8. Following is a description of the implicit system components.

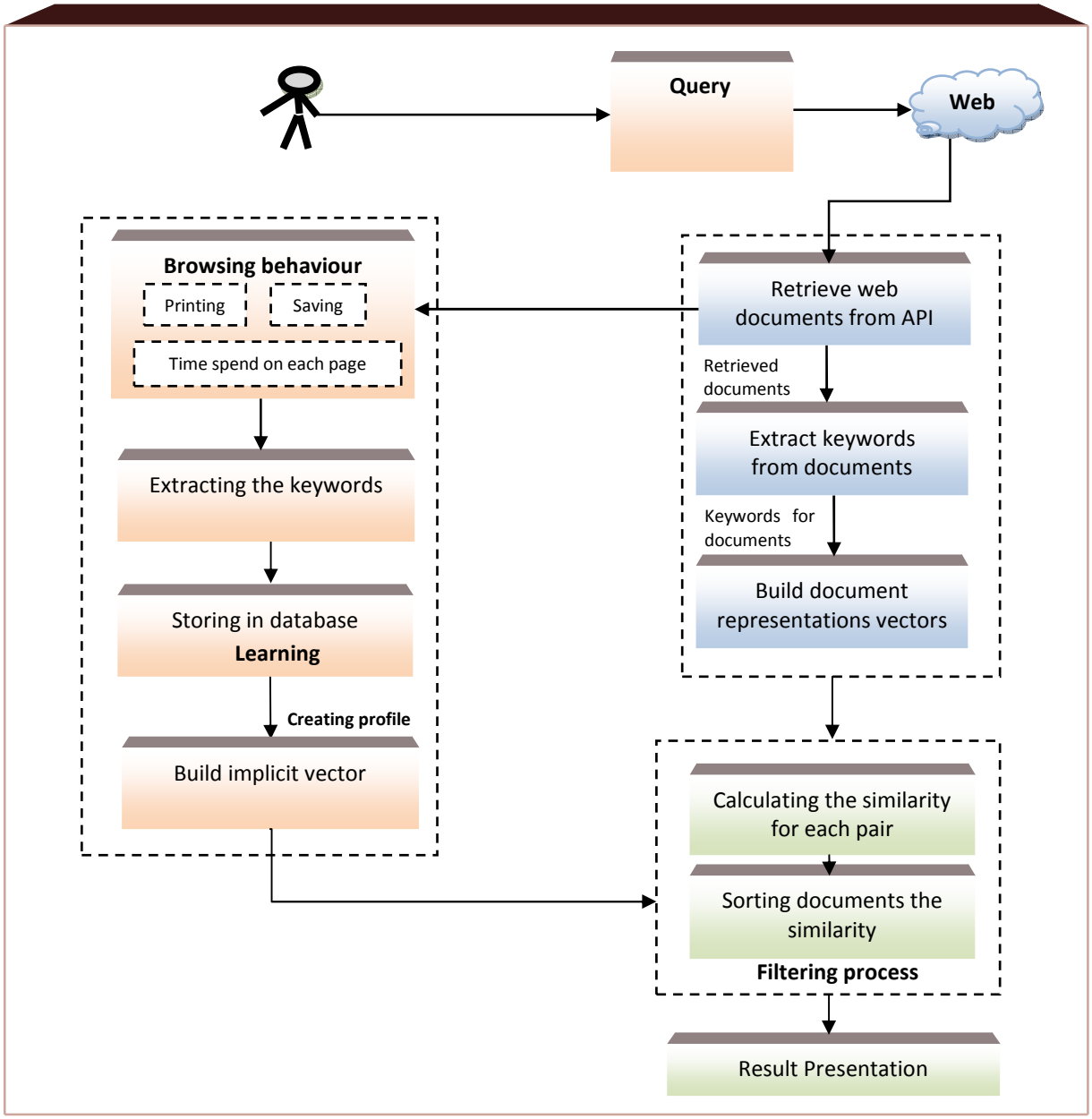


Figure 5.8: Implicit system

Figure 5.8 depicts the architecture of the implicit system. The system provides a web browser and monitors the user actions within the browser by tracking the events sent by the browser. It extracts keywords from each visited document. The keywords are extracted from the documents title and metadata (keywords and description). The system treats keywords with different importance depending on the source – for instance keywords extracted from a document title are more important that these extracted from the description. The keywords with higher importance and keywords that are repeated in different section have higher weights in the frequency. The vector is stored in the database.

In addition to the vector containing the keywords that describe the document content, the system also stores the activity type (whether the document was viewed, printed or saved). Information about the time of the event is also stored – for activities such as printing or saving only the start time is provided, while for viewing both the start and end time are saved to allow for the calculation of the time spent on each document. This time together with other activities (printing and saving) will be then used to calculate importance of that document when the implicit vector is generated.

After collecting the information regarding user browsing behaviour, the system is able to generate the implicit user profile in the form of keywords and weights for each document. Keywords from documents that were opened for only a short time are ignored, while keywords from documents that were saved or printed are considered especially important. The implicit profile vector is created for every included document after scaling it by the importance calculated for that document. After creating the summarised vector the 5 keywords with highest weights are used and returned as the implicit profile vector. The number of keywords is restricted as too many keywords would result in limiting the number of results retrieved from the base API, rendering sorting document by similarity to the implicit profile vector unfeasible.

The implicit user profile vector can now be used to call one of the base Web Search APIs. The APIs results can be filtered by sorting the returned documents by similarity between the implicit profile vector and the keywords vector extracted for each document. The implicit profile vector is normalised before filtering to simplify the calculation of the cosine similarity.

5.5.2.1 Implementation of the implicit system

Figure 5.9 describes the implicit system class diagram. The current prototype contains five main classes. 'UserInterface' class is the main user instance class. 'Searching' class provides facilities for searching in base web search API, and creates the implicit user profile. The 'WebBrowserListener' class tracks user activities and redirect events to 'MyActivityLogger' class. 'MyActivityLogger' class stores the user activities in the database.

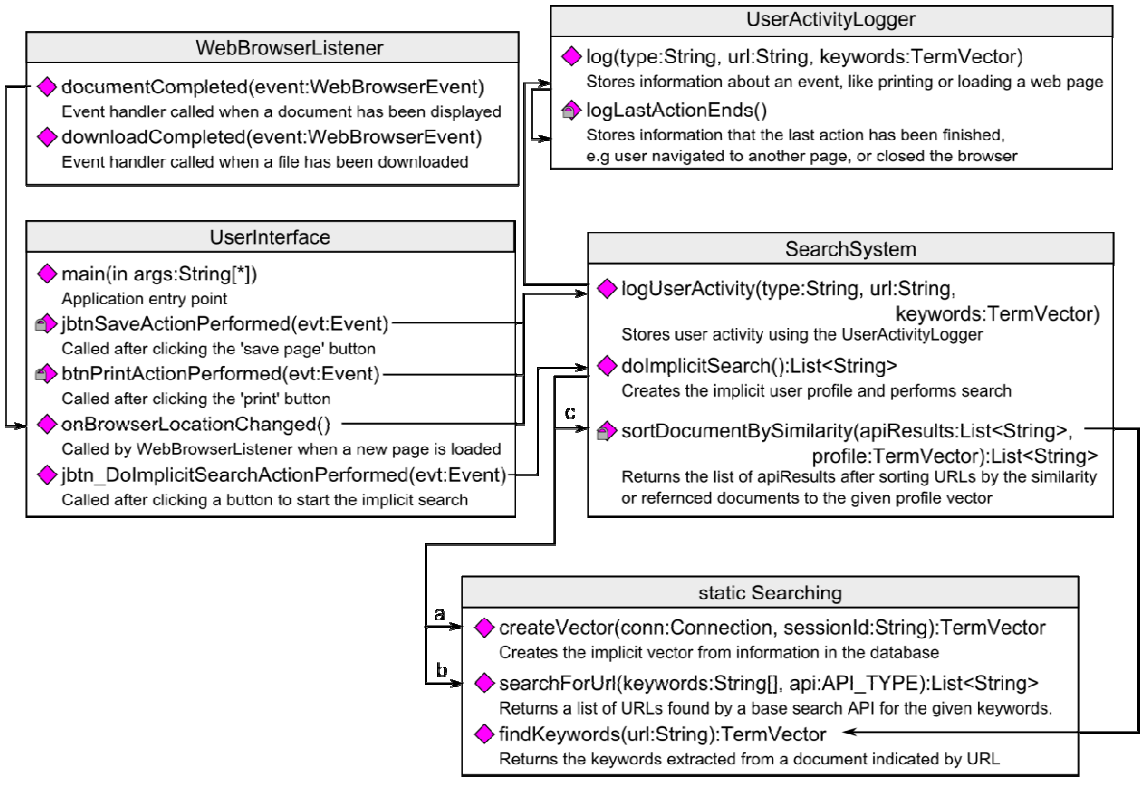


Figure 5.9: Simplified class diagram for the implicit system

The implicit system is learning by tracking browser events, particularly the 'documentCompleted' event, and by storing information when 'save' or 'print'

buttons are clicked. The information is stored by the 'UserActivityLogger' class, by calling the 'log' method. The search is performed by calling the 'do ImplicitSearch' method. The method creates the implicit vector (a), searches for URLs in the base web search API (b), and filters the results (c).

5.5.2.2 Implicit profile system database

User browsing behaviour is stored in a database, as shown in Figure 5.10. The table 'ActivityType' is a dictionary of activity types that can be inserted in the 'SessionActivity'. Each 'SessionActivity' entry has an ID which is an auto incremented number, a sessionID text which is a name of the session (as entered by the user). Each action has the startTime, but the finishTime is only set for the browsing action (printing and saving actions are considered instantaneous and therefore have only startTime set).

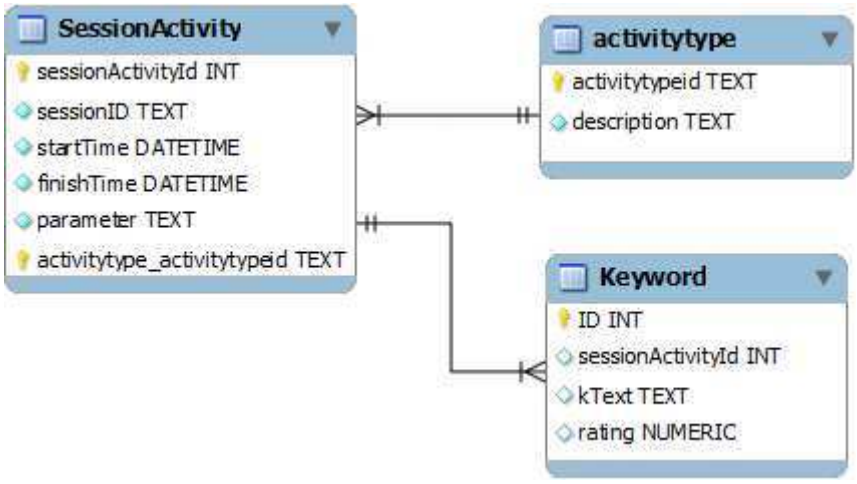


Figure 5.10: Implicit profile table

5.5.2.3 Creating implicit profile – pseudo code

The detailed description about system learning behaviour is provided in the following pseudo code:

1. For each visited 'document':
2. When user is opening a page
    a. calculate the keywords freq. vector
        i. Read the title, keywords and description from the document metadata
        ii. Create frequency vector for title, keywords and description
        iii. Scale each of the vectors by its importance
            a) Scale the vector made from the title by 0.3
            b) Scale the vector made from the description 0.5
            c) Scale the vector made from the meta-keywords 0.2
    iv. Add terms from all these vectors to create one vector
    v. Remove keywords with lowest ranking
    vi. Normalise the vector
        a) length = 0
        b) For each keywords length += keyword_rating^2
        c) scale the vector by [1/square_root(length)]
    i. Store the keywords in the database
3. When user is leaving a page
    a.i. Store time of the visit in the database
4. When user is printing or saving
    a.i. Store that event the database

The formula for creating the term vector for implicit user profile:

1. For each document stored in the database
    a. Get the average time the user spent on each page from the database
    b. Get keywords (with ratings) for all pages that have been visited for longer than average
        i. Retrieve the browsing history from the database
            a.a) each action contains action id, start time, and end time
            a.b) only select records where end time – start time > average time
            a.i. For every action id
            a.ii. retrieve keywords associated with this action, each keyword has a rating
b. Get keywords (with ratings) from pages that were printed or saved
    a Retrieve the browsing history from the database
        i. each action contains action id, action time, and action type
        ii. only select records with activity type 'saving' or 'printing'
    b For every action id
        b.i. retrieve keywords associated with this action, each keyword has a rating
        b.ii. multiply the rating by 10 to consider printed/saved pages

more important
c. Combine both keywords vectors from step 2 and 3
i. If a keyword exist in both vectors, then its new rating is a sum of rating from both vectors
d. Normalise the vector
i. length = 0
ii. For each keywords length += keyword_rating^2
iii. scale the vector by [1/square_root(length)]
2. Return the vector as it can now be used for searching

The implicit profile is generated once each time when the system has to perform a search.

5.5.2.4 Implicit program code

The system interaction with the search API and keywords extraction from the metadata and title is described in section 5.2.4.2. This section describes how to store user activities. For saving and printing the 'logCurrentActivity' method is called.

```
// from MyWebBrowserListener
private String ignoreLastURL = "";

// called when a document has been opened
public void documentCompleted(WebBrowserEvent event)
{
   String url = this.myWebBrowser.getURL().toString();

   if(url == null || url.equalsIgnoreCase(ignoreLastURL))
      return;

// check if the main part of the url is the same and ignore change
//    if the main part has not changed
   if(ignoreLastURL != null && url.contains("#"))
   {
      if(ignoreLastURL.startsWith(url.substring(0, url.indexOf("#"))))
         return;
   }
 ignoreLastURL = url;
 this.mySimpleBrowser.logCurrentActivity("Browsing", "Some URL BROWSING");
}

[…]

// from UserJnterface.java

public void logCurrentActivity(String activityType, String description)
{
   // find keywords

   TermVector keywords = Searching.findKeywords(webBrowser.getURL().toString());

   // save to the database

   ActivitiLogger.log(getSessionID(), activityType, description,
                      webBrowser.getURL().toString(), keywords);
}
```

The 'documentCompleted' method is called as an event from the browser, each time after the browser has finished loading a web page.

5.5.2.5 Code for creating the implicit user profile

Creating implicit user profile - error handling code has been removed for simplicity.

```
public static TermVector createVector(Connection conn, int sessionId)
{
   Statement stmt = conn.createStatement();

   String sql = "select max(DateDiff('s', starttime, finishtime)) as maxTime, "
              + " avg(DateDiff('s', starttime, finishtime)) as avgTime"
              + " from SessionActivity where sessionid = " + sessionId
                + " and activitytypeId='Browsing' and finishtime is not null "
                + " and starttime is not null ";

   ResultSet rs = stmt.executeQuery(sql);
   rs.next();
   double maxTime = rs.getDouble(1);
   double avgTime = rs.getDouble(2);

   //-- build the vector

   HashMap<String, Double> keywords = new HashMap<String, Double>();

   // get keywords from the database – browsed pages

   sql = "select t2.kText, t2.rating from SessionActivity as t1 "
       + " INNER join keyword as t2 on t1.sessionactivityid = t2.sessionactivityid"
       + " where t1.sessionid = " + sessionId
       + " and t1.starttime is not null and t1.finishtime is not null "
       + "   and DateDiff('s', starttime, finishtime) >= " + avgTime + " ";

   rs = stmt.executeQuery(sql);
   while(rs.next())
      addKeyword(keywords, rs.getString(1), rs.getDouble(2));

   // get keywords from printed and saved pages

   sql = "select t2.kText, t2.rating*" + modForStoredPages
      + " from SessionActivity as t1 "
      + " INNER join keyword as t2 on t1.sessionactivityid = t2.sessionactivityid "
      + " where t1.sessionid = " + sessionId
      + " and (activitytypeId ='Printing' or activitytypeId='Saving' )";

   rs = stmt.executeQuery(sql);
   while(rs.next())
      addKeyword(keywords, rs.getString(1), rs.getDouble(2));

   // make a vector from N most popular keywords

   TermVector vector = new TermVector();
   for(String s:keywords.keySet())
      vector.put(s, keywords.get(s));

   return vector.topN(maxKeywordsUsed);
}
```

The returned TermVector object contains the generated implicit profile information.

5.5.2.6 Retrieving search results for the implicit system

Retrieving search results for the implicit user profile is presented in the next fragment of code.

```
public void doSearch()
{
   TermVector vector = Searching.createVector( ActivitiLogger.getConnection(),
               getSessionID());

   […]

   //-- search for documents in base web search API

   // create simple list of keywords ordered by values
   String[] list = new String[vector.size()];
   Iterator it = vector.termIterator();
   int idx = 0;
   while(it.hasNext())
      list[idx++] = it.next().toString();

   // find urls for the keywords by using base api

   java.util.List<String> urls = Searching.searchForUrl(list, ChoosedSearchAPI);


   //-- sort documents by similarity to the implicit vector

   RAMSearchEngine rse = new RAMSearchEngine();

   for(String url:urls)
   {
      HTMLDocument objDoc = new HTMLDocument("<html></html>");
      TermVector documentKeywords = Searching.findKeywords(url);
      // add document with keywords
      rse.addDocument(url, objDoc.getFullContent(), documentKeywords);
   }

   // retrieve documents with highest similarity to the fiven vector
   ValueSortedMap map = rse.retrieveDocuments(vector, 20);

   // display result
   setText_Result(map);
}
```

For each of the URLs returned by the base search API, the system retrieves the web document in order to parse it to create a keywords vector. The same process is repeated for all the documents returned in the base search API results and a vector is created for every page so it can be compared to the user profile vector.

5.5.3 Hybrid system

The hybrid system is shown in Figure 5.11. Following is a description of the components used in the architecture of the hybrid system, which utilizes a combination of explicit and implicit user profiles.

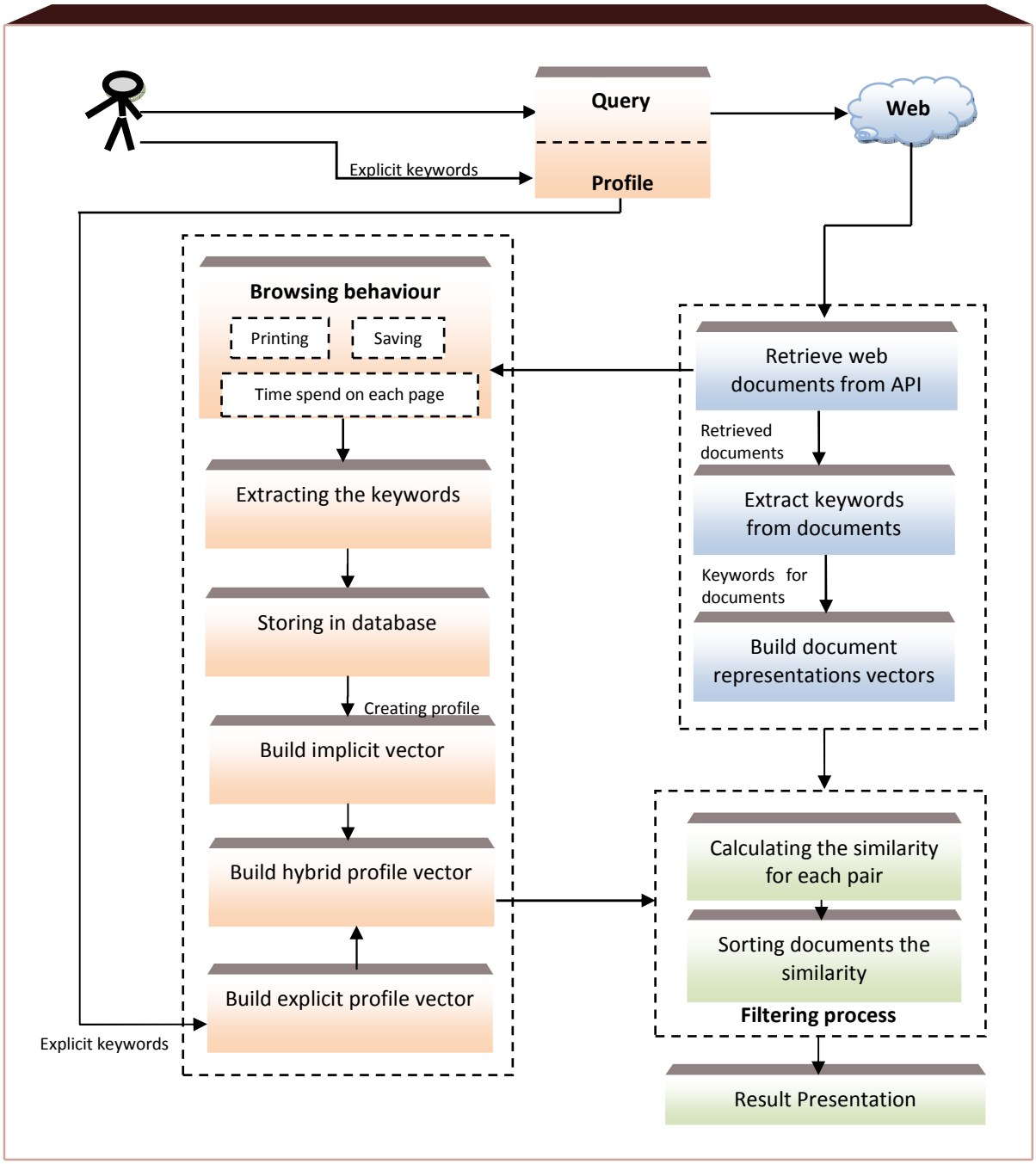


Figure 5.11: Hybrid system

In the hybrid mediation system the explicit profile and implicit profile are generated separately and combined into a single term vector. For the explicit user profile the system asks the users to add their preferences to the profile in terms of keywords. The explicit profile is stored in a term vector.

The implicit user profile monitors the user's search activities by checking the history, the time spend on the each page, the printing and saving etc. The system analyse the collected information to generate a list of keywords (with weights) for the implicit profile.

After building the vectors for both profiles individually, the system combines both vectors. The system gets the highest keyword weight from implicit user profile and gives the same weight to every keyword in the explicit user profile vector. Both vectors are then added – if a keyword appears in both vectors, the system sets its new weight to the sum of weights from the both vectors. After that the system normalise the combined vector and returns it.

5.7.1.1 Implementation of the hybrid system

The prototype for the hybrid system has been implemented in JAVA programming language using NetBeans IDE. The class diagram of the hybrid system provides the better understanding of the system architecture. The prototype has five main classes:

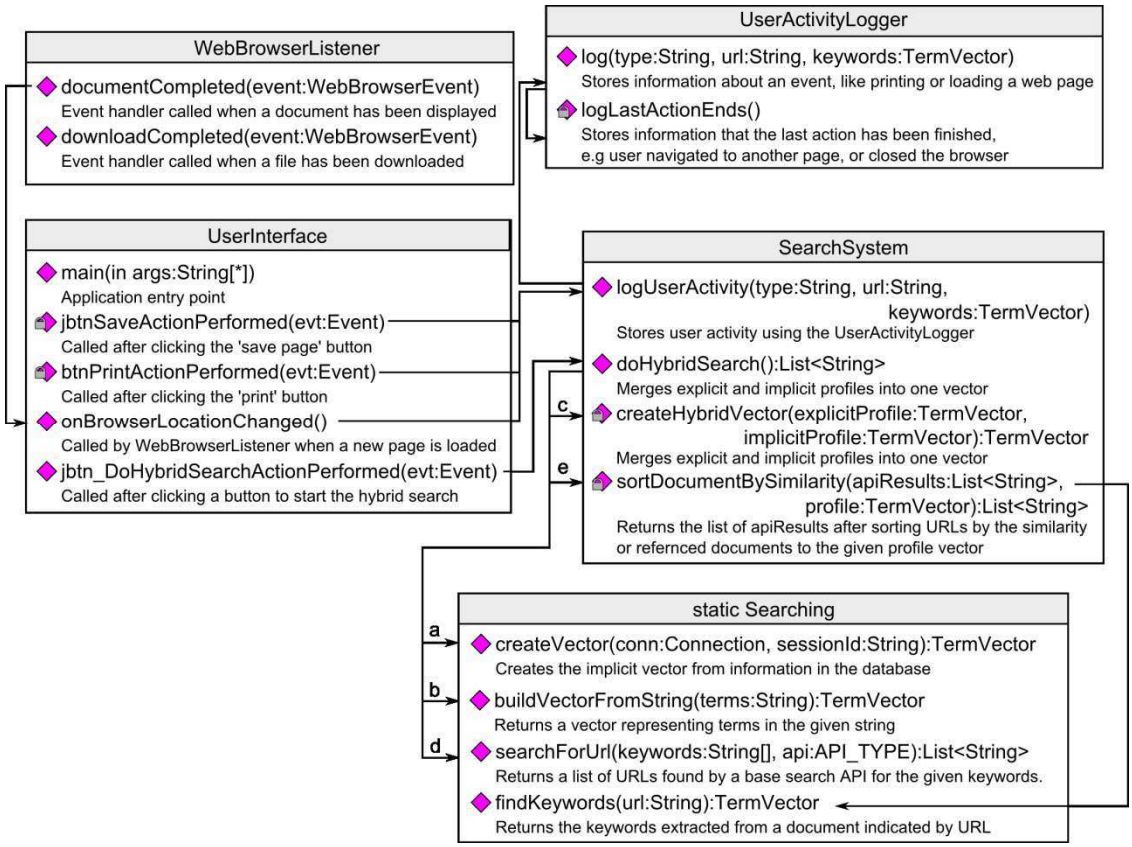


Figure 5.12: Simplified class diagram for the hybrid system

Figure 5.12 presents the hybrid system class diagram. 'UserInterface' class creates the 'WebBrowserListener' to track user activities inside the browser. Browsing, printing and saving activates are stored by calling the 'log' method of the SearchSystem object, which calls the appropriate method from the 'UserActivityLogger' object. The hybrid searching is performed in four steps. Firstly, the system creates the explicit and implicit profiles (a, b), which are then merged into a hybrid profile (c), the hybrid profile is then used to retrieve results from the base web search API (d) and to filter these results (e).

The prototype incorporates the three modes of mediation and is capable of performing all three types of searches – for the implicit vector, for the explicit vector and for a combined vector. In the hybrid search the initial list of documents is retrieved from the base web search API in the same way as in the explicit system – by using the terms from the explicit vector.

The detailed description of the searching algorithm used by the hybrid system is outlined in the following pseudo code:

1. Get explicit keywords from the user
2. Get implicit user profile vector from the browsing history
3. Create combined vector
    a. Get the highest keyword rating from the implicit vector
    b. Scale the explicit vector by the highest implicit rating (calculated in point a.)
    c. Add explicit and implicit vectors
        If a keyword exist in both vectors, then its new rating is a sum of rating from both vectors
4. Trim the combined vector to 5 keywords with the highest rating as too many could prevent from getting any results from base web search API

i. Normalise the vector
    a. length = 0
    b. For each keywords length += keyword_rating^2
    c. scale the vector by [1/square_root(length)]

ii. Return the normalised vector

The new part that identifies the hybrid system is how the explicit and implicit vectors are combined before the similarity for each document is calculated. The system gets the highest keyword value from implicit user profile and scale the explicit user profile by that value.

The code used to combine explicit and implicit user profiles:

```
private TermVector createCombinedVector(
                     TermVector explicitUserPreferences,
                     TermVector implicitUserPreferences)
{
   // if implicit vector is empty, then return the explicit vector
   if(implicitUserPreferences.size() == 0)
      return explicitUserPreferences.topN(explicitUserPreferences.size());

   // find the maximum keyword rating from implicit vector (top keyword)
   //    - use the fact that the term vector is always ordered descending by rating
   String bestImplicitKeyword =
                   (String)implicitUserPreferences.termIterator().next();
   double bestImplicitValue = implicitUserPreferences.get(bestImplicitKeyword);

   // create a combined vector
   TermVector result = new TermVector();

   // add all keywords from explicit results
   //  the rating will be changed if this keywords exists in the
   //  impicit vector as well
   result.putAll(explicitUserPreferences);
   // explicitly entered keywords ratings are scaled to be as hight as higher implicit rating
   result.scaleBy(bestImplicitValue);
```

```
   // add all keywords from implicit preferences to keywords
   //   from explicit preferences
   Iterator it = implicitUserPreferences.termIterator();
   while(it.hasNext())
   {
      String key = (String)it.next();
      double rating = (explicitUserPreferences.get(key) * bestImplicitValue)
                    + implicitUserPreferences.get(key);
      // if already added from the explicit vector, the keyword will be replaced
      ///   with the new value which includes implicit and explicit rating
      result.put(key, rating);
   }
   return result;
}
```

The code for doing hybrid search is as follows:

```
public List<String> doHybridSearch(String strKeywords)
    {
        // replace '=', ',', ' ' to '+'
        strKeywords = strKeywords.replace('=', '+').replace(',', '+').replace(' ', '+');

        // create explicit vector from the string containing keywords
        TermVector explicitUserPreferences = createUserPreferenceVector(strKeywords);
        // create implicit term vector from the browsing history (as in implicit prototype)
        TermVector implicitUserPreferences =
                            Searching.createVector(userActivityLogger.getConnection(), 1);

        if(explicitUserPreferences == null)
        {
            List<String> result = new LinkedList<String>();
            result.add("[error in getting explicit keywords]");
            return result;
        }

        if(implicitUserPreferences == null)
        {
            List<String> result = new LinkedList<String>();
            result.add("[cannot get implicit keyword from the database]");
            return result;
        }

        //-----------------------------------------
        // create hybrid vector (as presented in the beginning of this section)
        TermVector combinedPreferences = createCombinedVector(explicitUserPreferences,
                                                               implicitUserPreferences);
        //-----------------------------------------

        // search in base web search API - Yahoo or Google
        String[] keywords = strKeywords.split("\\+"); // only use explicit keywords
        List<String> APISearchResult = Searching.search(keywords, chosenAPI);

        // order results by similarity to the combined (hybrid) vector
        List<String> results = sortDocumentBySimilarity(APISearchResult,
                                                        combinedPreferences);

        return results;
    }
```

The method returns the list of strings, each of them representing one URL.

5.8 System interaction

Three different system interfaces were designed, one for each of the system prototypes. The browser component is based on Internet Explorer that is installed on the user computer. In the explicit system, user can enter the keywords to create profiles which can be modified at a later stage if required. In the implicit and hybrid systems, users can browse the Web using a build-in web browser. After the search the system displays the most relevant search results as a list of URLs. User can enter URLs directly, or follow hyper-links on any page. All information about the browsing history (including the time spent on a page) is stored in the database. The system learns from users searching behaviour in the background and improves the search according to individual interest. The hybrid system prototype's user interface enables users to perform an explicit or implicit only search as well as a hybrid search.

5.8.1 Explicit system Interface

The interface of the explicit profiles is shown in Figure 5.13. The interface enables users to enter their preferences and to view the search results at different stages of the filtering process.

The user can to enter name to save/load the profile (a new user profile is created if the entered user name did not existed in the database). One user is allowed to create any number of vectors by choosing '<new>' from the 'Stored keywords' list and entering the list in to the 'keywords' text box.

The system allows users to add, update, save, and remove vectors based on their changing interests. To make change to a vector, user has to select it from the list and either click 'delete' to remove it or change the list of keywords and pres 'save' to update the vector.

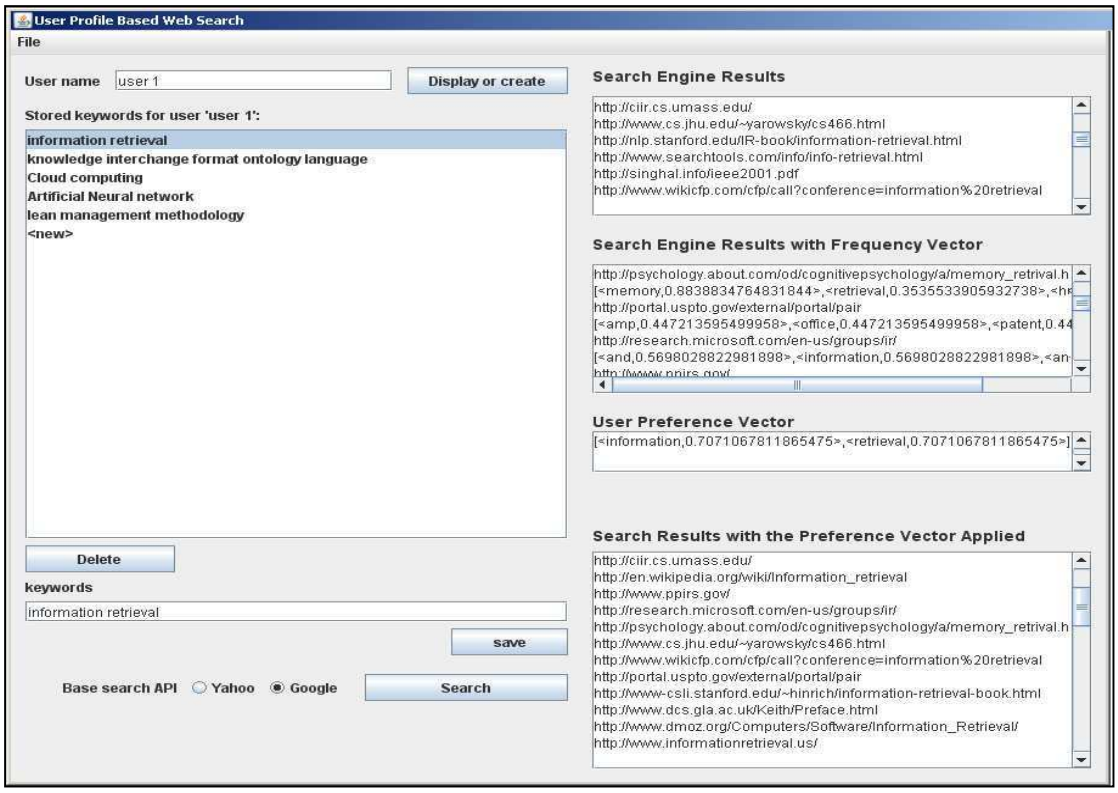


Figure 5.13: Explicit system interface

To perform a search, a user has to select the vector from the list, choose one of the base web search APIs and press the search button. The output information consists of the list of the URL returned by the base web search API, frequency vectors created for each of the documents returned, and the list of URLs after the VSM filtering is applied.

5.8.2 Implicit System Interface

The interface of implicit system is shown in figure 5.14. The main component of the implicit system prototype is the web browser which is capable of tracking the user actions and learning from it. All the learning

actions are performed in the background without additional interaction with the user.

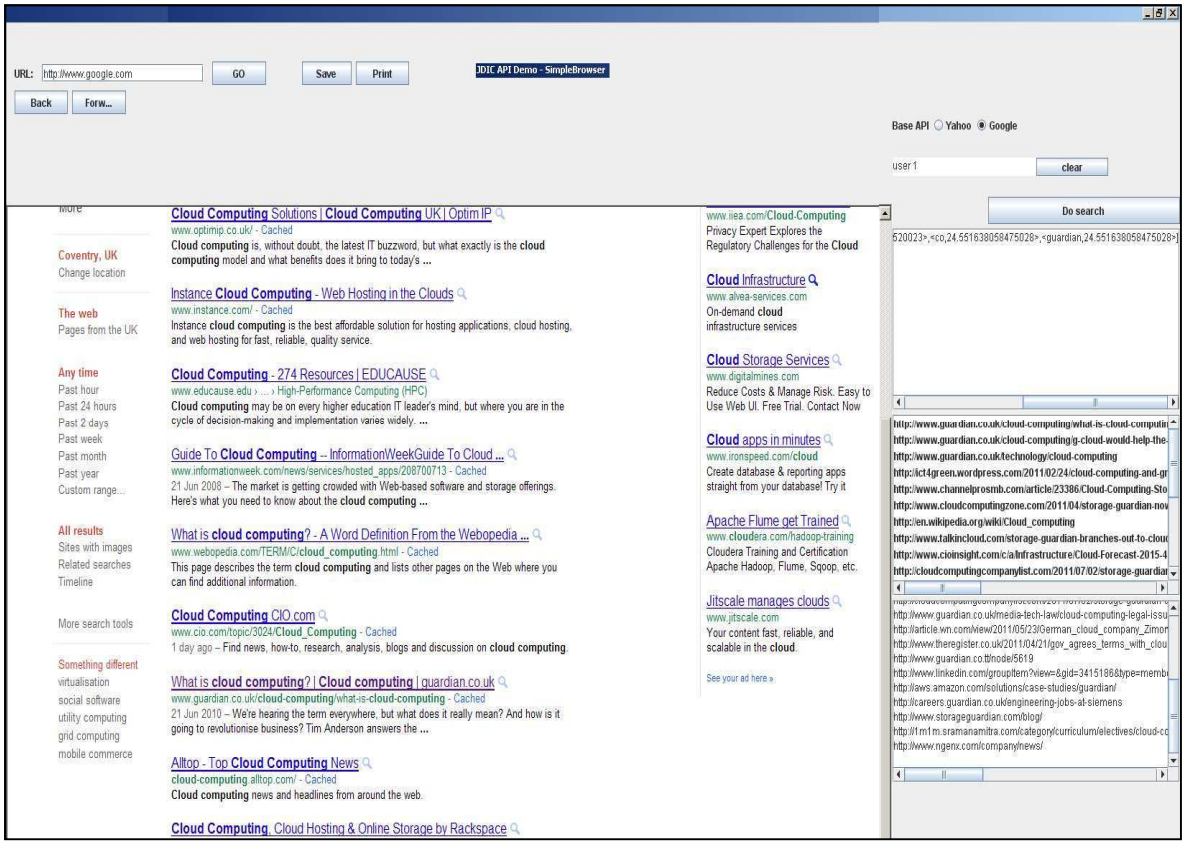


Figure 5.14: Implicit system interface

The user can provide a name for a session which allows creating different profiles depending on the current user interest. The application can be closed without losing any acquired knowledge, but the user is able to clear session data by clicking on the 'clear' button next to the textbox with the session name.

After browsing if the user wants to perform a search operation the base web search API has to be selected and the 'do search' button clicked. After the search the implicit profile vector is displayed together with frequency vector generated for each of the document returned by the base search API. Finally the list of suggested URLs is displayed in the bottom right text area.

5.8.3 Hybrid System Interface

The hybrid system prototype caters for both explicit and implicit search. It allows tracking the user actions and learning the user preferences to create an implicit user profile vector whenever the user wants to start search operation. The user can provide a name for the implicit profile session which allows storing multiple implicit profiles for the user depending on the current interest. The keywords for the explicit profile vector have to be provided in a text box. The system supports the three kinds of search: explicit, implicit and hybrid.

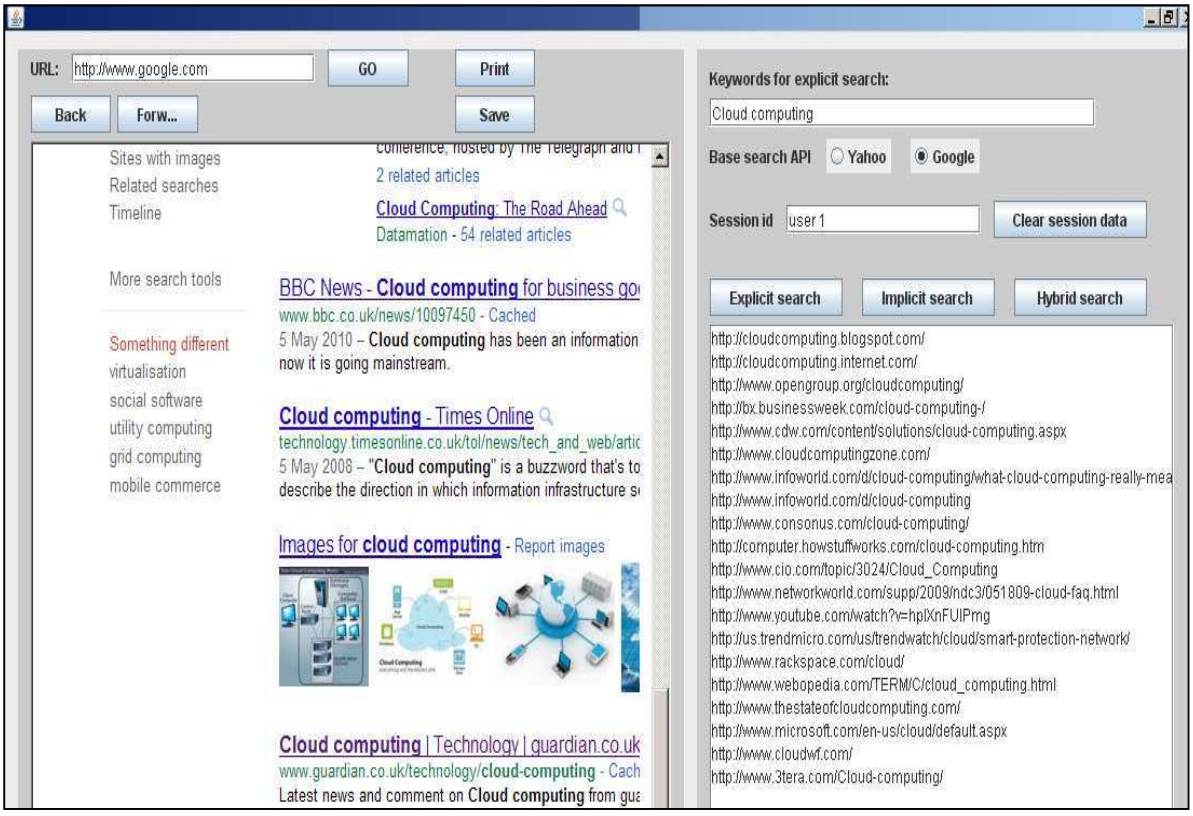


Figure 5.15: Hybrid system interface

5.9 Summary

As an integral part of the endeavour to address the limitations of the base search engines a mediation framework was proposed and implemented. The provision of three mediation systems with their own distinctive profile generation ensures that the framework offers quality of service in document filtering. These requirements are well served by the adoption of the content-based method and the VSM model. The overall flexibility of the framework is supported by appropriate user interfaces.

Chapter 6

Evaluation

6.1 Introduction

The aim of this chapter is to present a comparative evaluation of the proposed framework. It involves a quantitative evaluation to determine the effectiveness of the three mediation systems and a qualitative evaluation of the framework to demonstrate the contribution of this research. The measure of effectiveness of the systems will be in terms of precision and recall, and will be with respect to the base Google and Yahoo search engines. In addition the three modes of profile generation will be compared to each other.

6.2 Evaluation methodology

The experiment was conducted to measure the effectiveness of the explicit, implicit, and hybrid systems in terms of recall, precision and F-measure. The experiments have been divided into two stages. The first phase concerns the evaluation of the three mediation systems and the base API used, and the second phase deals with the impact of additional learning time

of the implicit and the hybrid systems on the retrieval precision and recall. The systems are tested with Google and Yahoo! Web Search APIs as base search engines.

6.2.1 Experiment setup

The experiment was performed with 30 real users with their own choice of keywords. Each user has provided one set of keywords for the first phase which gives a total number of 30 queries. To measure the system effectiveness the evaluation was conducted on the following retrieval systems during January 2011 to May 2011:

- Google Web Search API (base Google)
- Yahoo Web Search API (base Yahoo)
- Explicit system using Yahoo (explicit with Google)
- Explicit system using Google (explicit with Yahoo)
- Implicit system using Yahoo (implicit with Google)
- Implicit system using Google (implicit with Yahoo)
- hybrid system using Yahoo (hybrid with Google)
- hybrid system using Google (hybrid with Yahoo)

The experiment was conducted in two phases. In the first phase only short time was given to build implicit profiles, while in the second phase this time was extended.

6.2.1.1 Experiment phase 1

In the first phase the users were instructed to use a provided web browser for 15 minutes so that the browsing behaviour could be recorded in the database and be available to the implicit and hybrid system. The browser recorded the time spent on each page and activities such as printing and saving a document. After the browsing session users proceeded to enter the keywords for the explicit profile. After the explicit keywords were provided all the search systems have been started. The documents retrieved by each of

the implemented systems from the base Web Search APIs were combined into one list, sorted randomly and pre-opened in a web browser to avoid the situation where the ratings given by the users are affected by their opinions of the retrieval systems. It was decided to select first 20 web sites returned from each system for evaluation: 160 documents were opened – 80 through Google API and 80 through Yahoo API as the basis for mediation systems. The documents were presented to the users who were asked to give to each document a score that will indicate how relevant it is to the entered query. The full test usually took from one to one and a half hour per user.

6.2.1.2 Experiment phase 2

The second phase of the experiment was performed with the same 30 users; all of them had already taken part in the first phase. The experiment has been performed to check how the retrieval effectiveness changes when a system has additional time to learn from the user behaviour. Each user has used the same user name as in the previous part of the experiment so that new information was added to already stored browsing history captured during the first phase of the experiment. The additional learning time was set to 15 minutes per user, so that the total time allowed for system learning was 30 minutes. The users have rated search results in the same way as in the first phase of the experiment. Only the implicit and hybrid systems were retested in the second phase as only these systems are using the stored browsing information.

6.2.2 Documents rating

To ensure that all users are using the same scale of scores, the users were presented with an indication on how to assess a page depending on whether it was relevant or not. Five categories were created to assess search results, these are “relevant”, “less relevant”, “irrelevant”, “links” and “no access” (Kumar and Prakash 2009, Shafi and Rather 2005). This scale represents the main characteristic of relevance as continuous rather than binary. Based

on the Kumar and Prakash (2009) research, the following rating instruction was created and provided to the user:

Category	Description	Score
Relevant	Related Conference paper, journal paper or web document fully related to the query	2
Less relevant	Document not fully concerned on to the query topic, but having the required information as part of its contents	1
URLs/Links	Page that provides a list of URLs where at least two URLs are redirecting to a page with the relevant information	0.5
Irrelevant	Documents totally irrelevant to the user intentions	0
No access	Web pages that for any reason cannot be accessed (e.g. 'page not found error').	Error (0)

Figure 6.1: Rating instruction provided to each user

This information was used by users to assess the relevance of documents retrieved by the base search engines and the retrieval systems. Users were giving scores to documents on the basis of relevance to the query. These score allows us to determine the effectiveness of the retrieval systems and the search engines in terms of both precision and recall.

The relevance scores given by users to a document could be influenced by the awareness of the system or search engine that returned the document. As mentioned in the first phase of the experiment, to ensure that the result are only based on relevance rather than users expectations or opinions about the retrieval method, a small evaluation application was introduced that displayed a series of documents in random order and enabled users to assign scores to documents without knowing which system has returned them. If a document was retrieved by more than one search engine then it was displayed only once. This process makes the scoring fairer and also easier for users. After one page was rated the testing system was automatically switching to the next document.

Figure 6.2 shows the evaluation system with documents opened. Each document is displayed in a separate tab.

Figure 6.2: Evaluation system during documents rating

Once a user clicks on one of the rating buttons placed under the document, the rating for the displayed document is saved, and the next tab is brought forward. Once all documents are rated they are closed and the results are displayed in a spreadsheet as shown in Figure 6.3.

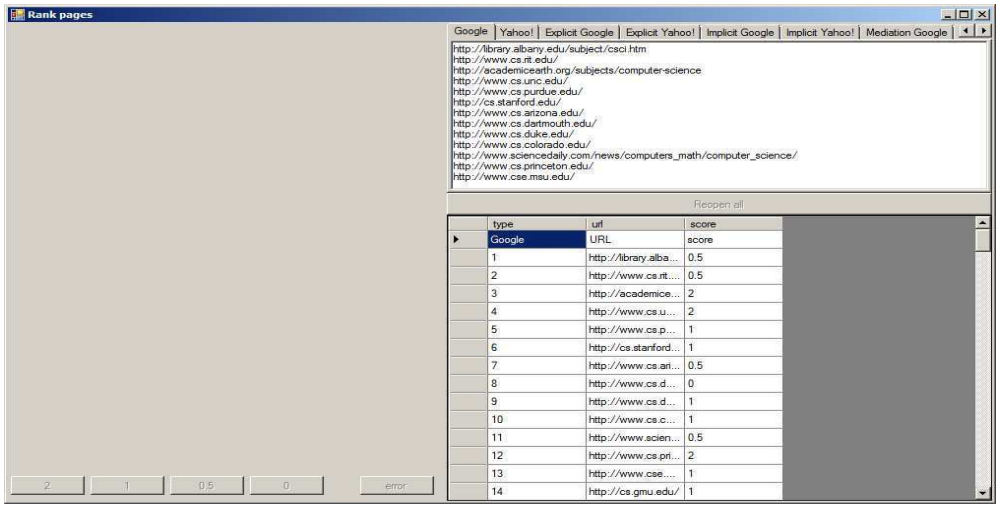


Figure 6.3 Evaluation system showing documents scores

Each user had to rate up to 20 web pages from each system, and altogether they had to rate up to 160 web pages.

6.2.3 Measures of effectiveness

This section describes the measures used to assess the performance of each of the systems. All the systems are based on the same base Web Search APIs and are using the same keywords extraction method to eliminate any spurious factors. The performance of the retrieval process of the three systems will be determined in terms of precision, recall and F-measure.

6.2.3.1 Precision

The precision of a retrieval system for a given query is the number of relevant documents retrieved over the total number of documents retrieved for that query. As a document can be classified as relevant, partially relevant or irrelevant instead of using number of documents, a value (score) will be assigned to each document as a reflection of the degree of its relevance. The precision is calculated as the total score assigned for retrieved documents divided by the maximum score that would be given if all documents were relevant (Kumar and Prakash 2009).

$$\text{Precision} = \frac{total\ scores\ from\ relevant\ document\ retrieved}{maximum\ score\ of\ retrieved\ documents} \quad \text{[Equation 6.1]}$$

If more documents are deemed irrelevant then the precision is low, but if more documents match the expectations of the users then the precision is high (for that particular query).

6.2.3.2 Recall

Recall is the total score of all relevant documents retrieved by a search engine over the total score for all relevant documents held in the database.

$$Recall = \frac{total\ score\ from\ relevant\ retrieved\ documents}{total\ score\ of\ all\ relevant\ documents} \quad \text{[Equation 6.2]}$$

Users should be able to view all relevant documents that may meet their information requirements. If the relevance scores from retrieved documents is close to the total score of all documents in the database, then the recall will be high, otherwise it will be low.

Recall is often nontrivial to measure because usually it is difficult to determine the number of relevant documents in the whole database. The issue is how to identify an acceptable pool of relevant documents. One approach is to combine all the relevant documents returned by more than one search engine (Kumar and Prakash 2009, Shafi and Rather 2005). The score for each search system is calculated using the following equation:

$$Recall = \frac{score\ of\ documents\ retrieved\ by\ \ evaluated\ search\ system}{maximun\ score\ for\ documents\ retrieved\ by\ both\ search\ system} \quad \text{[Equation 6.3]}$$

As this measure provides only an approximation to the true value of recall, it is often referred to as relative recall. For example if two systems have to be compared, Google and Yahoo! Web search APIs, then the Google API relative recall can be measured by dividing the total score for document retrieved by Google API by the maximum score for documents retrieved by either Yahoo! or Google API. If the same document is returned by both search engines its score is counted only once.

6.2.3.3 F-measure

The F-measure can be used to combine precision and recall to obtain a single efficiency measure. F-measure score is defined as the harmonic mean of precision and recall.

$$F\text{-}measure\ = 2\frac{Precision \cdot Recall}{Precision + Recall} \quad \text{[Equation 6.4]}$$

This is also known as F1 measure as precision and recall are weighted equally.

6.2.4 Statistical significance of the results

As the number of users taking part in the experiment is limited to 30, a statistical analysis is required to asses if the results are significant. The goal is to determinate whether the hybrid system performs better than the base APIs. As the number of samples is not very high, the 'Student`s' T-test is applicable. This test allows for testing a hypothesis on the basis of a difference between two sample means. The underlying statistical theoretical background is presented below.

A group of user have generated one series of samples for each of the systems. Each user have provided two samples for each of the series, one sample was provided by rating documents returned with use of the Google API and other provided by rating the results from a system based on Yahoo API.

The following parameters have to be defined before calculating the significance level:

- n_1 – size of the first sample
- n_2 – size of the second sample
- $\overline{X_1}$– average value from the first sample
- $\overline{X_2}$ – average value from the second sample
- σ_1 – standard deviation in the first sample
- σ_2 – standard deviation in the second sample

In the next step, two hypotheses have to be defined. The first hypothesis states that the increase (or decrease) of the value (e.g. precision or recall) is not significant, and the alternative hypothesis stating that this difference is significant. If the first hypothesis (also called null hypothesis) is rejected after the test, then the second hypothesis is accepted.

- H_0 – The average value in the first sample is lower or equal to the average value in the second sample

- H_1 – The average value in the first sample is higher than the average value in the second sample

The test statistic can be expressed as:

$$t = \frac{\overline{X_1} - \overline{X_2}}{\sqrt{\frac{\sigma_1}{n_1} + \frac{\sigma_2}{n_2}}} \quad \text{[Equation 6.5]}$$

Finally the critical value (t_c) of the test has to be specified. This value can be read from tables, for a given value of required significance level (α) and degree of freedom (d_f). The degree of freedom for two samples, each containing 60 elements, is 118 (total number of samples, minus the number of series). The significance level can be arbitrarily chosen. The value 0.01 would mean that if the null hypothesis is accepted, then there is 99% confidence that the difference stated by that hypothesis have occurred by chance.

Once all these steps are completed, the decision has to be made either the null hypothesis should be accepted or rejected. The null hypothesis cannot be rejected if the calculated value of t is lower than the critical value t_c. Otherwise the null hypothesis is rejected, and the alternative hypothesis can be accepted instead.

$$if\,(t < t_c)\ then \text{ do not reject } H_0\ otherwise\ reject\ H_0 \text{ and accept hypothesis } H_1$$

The significance of the results will be calculated for the comparison of the hybrid system with the base search APIs, the explicit system and with the implicit system.

6.3 Experiment phase 1 results

The experiments results are collected from Google, Yahoo, explicit system using Yahoo, explicit system using Google, implicit system using Yahoo, Implicit system using Google, hybrid system using Yahoo and hybrid system using Google. The retrieval effectiveness has been measured with 30 real

users with their own choice of queries. A spreadsheet was prepared to calculate the precision for each search, based on the collected data. Detailed description regarding collected data, users and calculations are provided in Appendix B.

There is no absolute measurement of recall as it is not feasible to assess and rate all the documents in the Google or Yahoo databases. Instead a method of calculating the relative recall, which has already been adopted by Kumar and Prakash (2009) and Shafi and Rather (2005), will be used in the measurement of recall. The total of the relevant documents is created by the combination of the relevant documents returned by the base Google and the base Yahoo search engines. Duplicates, i.e. the documents returned by both search engines are counted once only. This method was used in the evaluation of the recall of the three mediation systems. The spreadsheet used in the recall calculation is provided in the Appendix B.

6.3.1 Precision and relative recall with Google and Yahoo! APIs

After the submission of the keywords to search engine API the first 20 returned documents are retrieved for the determination of the precision and relevance score.

6.3.1.1 Precision of base Google and Yahoo! APIs

Figure 6.4 shows the scores for the documents returned by the Google Web Search API for user queries.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	214	122	58	198	8	600
%	35.7%	20.3%	9.7%	33.0%	1.3%	
Total score	428	122	29	0	0	579
Overall precision						**0.48**

Figure 6.4: Precision of Google API

To obtain the overall Google precision the total score for all documents was divided by the maximum score that could be assigned if all documents were fully relevant. As 20 documents were rated by each of 30 users, and the maximum score given for each document is 2, the maximum possible score is 1200. Figure 6.5 displays the search results with Yahoo search engine.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	239	90	73	186	12	600
%	39.8%	15.0%	12.2%	31.0%	2.0%	
Total score	478	90	36.5	0	0	604.5
Overall precision						**0.50**

Figure 6.5: Precision of Yahoo! API

The results indicate that the base Google has returned slightly higher number of irrelevant documents, while Yahoo returns a higher combination of relevant and less relevant documents.

6.3.1.2 Relative recall of base Google and Yahoo! APIs

The recall was calculated with the same data used to calculate the precision. As stated earlier, to the recall for Google or Yahoo! APIs, the total relevance

score for the documents retrieved by Google or Yahoo, divided by the total document score retrieved from both Yahoo and Google.

Description	Google	Yahoo	Duplicated
Documents	600	600	139
Documents Score	579	604.5	174.5
Recall	**0.57**	**0.60**	

Figure 6.6: Relative recall of base Google and Yahoo! APIs

Figure 6.6 presents the recall for Google and Yahoo. The detailed recall calculations are provided in Appendix B. In terms of the recall the base Yahoo API performs slightly better than Google API.

6.3.1.3 Overall precision and relative recall of base Google and Yahoo

Figure 6.7 presents a graph for the precision and relative recall of the base search APIs used in the mediation systems. The graphs indicate that base Yahoo! API returns more relevant documents than base Google API.

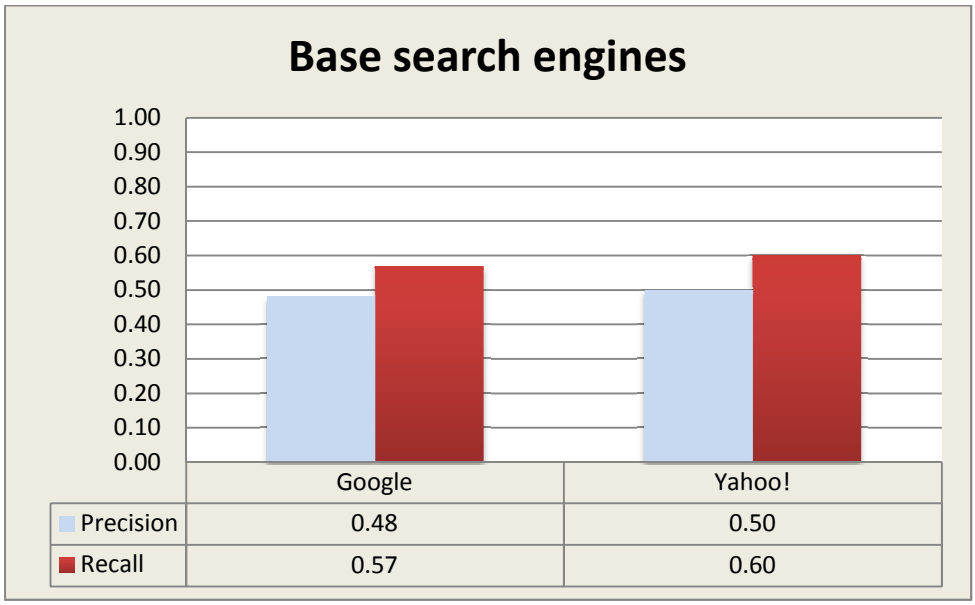


Figure 6.7: Precision and relative recall results for Google and Yahoo! APIs

It can be seen, that for both APIs the relative recall is higher than the precision, and that about half of the returned document were relevant to the queries.

6.3.2 Precision and relative recall for the explicit system

The explicit mediation system is based on the keywords provided by users directly to specify their profile. Each user submits the keywords to the system to retrieve the search results.

6.3.2.1 Precision of the explicit system using Google and Yahoo

The approach used to calculate the precision of the explicit system using Yahoo is the same as used for the base APIs. The system has produced 20 ranked web documents from explicit system based on Yahoo, and 20 documents from explicit system based on Google API in as returns to each query.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	220	124	70	184	2	600
%	36.7%	20.7%	11.7%	30.7%	0.3%	
Total score	440	124	35	0	0	599
Overall precision						**0.50**

Figure 6.8: Precision of Explicit system using Google

The results of the explicit system using Google API are presented on Figure 6.8. The results show that there is slight improvement in precision in the explicit system using Google over base Google API.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	268	113	67	151	1	600
%	44.7%	18.8%	11.2%	25.2%	0.2%	
Total score	536	113	33.5	0	0	682.5
Overall precision						**0.57**

Figure 6.9: Precision of Explicit system using Yahoo

Figure 6.9 presents the search results with explicit system using Yahoo. The table indicates that that the explicit Yahoo performs better than the base Yahoo! API and better than explicit system using Google API.

6.3.2.2 Relative recall of explicit system

To measure the relative recall, the documents from the explicit Google and the explicit Yahoo have been combined into one set.

Description	Google	Yahoo	Duplicated
Documents	600	600	99
Documents Score	599	682.5	137
Recall	**0.52**	**0.60**	

Figure 6.10: Relative recall of explicit system

Figure 6.10 provides the results of Google and Yahoo relative recall calculations.

6.3.2.3 Overall precision and relative recall of the explicit system

Figure 6.11 presents the precision and relative recall of the explicit mediation system. The system retrieves more accurate results compared to base search engines, however users had to provide their preferences explicitly. The precision and relative recall is higher in the explicit system

using Yahoo! API than the explicit system using Google API. The explicit Yahoo performs better than base Yahoo API, but that the recall for the explicit system based on Google API has decreased when compared to base Google API.

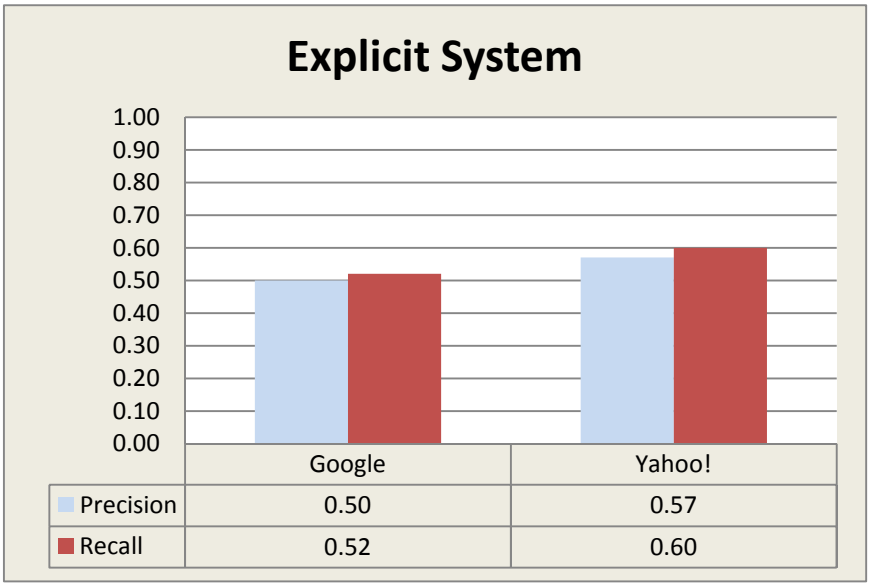


Figure 6.11: Precision and relative recall results for the explicit system

The explicit system performs better with the Yahoo search engine API than with Google API, however comparison between the base APIs is not the aim of this experiment. Instead the average precision and recall of each kind of mediation system will be provided in section 6.4.1.

6.3.3 Precision and relative recall with the implicit system

The implicit system is based only on the observed user browsing behaviour. In particular it is based on the recording of the time spent on each document, printing or saving documents, and the content of these documents.

6.3.3.1 Precision of the implicit system using Google and Yahoo

Figure 6.12 presents the precision of implicit system search results. The precision is the same as of base Google API but much lower than the precision of the explicit system using Google API.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	228	93	65	210	4	600
%	38.0%	15.5%	10.8%	35.0%	0.7%	
Total score	456	93	32.5	0	0	581.5
Overall precision						**0.48**

Figure 6.12: Precision of implicit system using Google

The next figure (Figure 6.13) presents the precision for the implicit system with Yahoo API.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	188	101	83	225	3	600
%	31.3%	16.8%	13.8%	37.5%	0.5%	
Total score	376	101	41.5	0	0	518.5
Overall precision						**0.43**

Figure 6.13: Precision of Implicit system using Yahoo API

The precision is lower than precision of the base Yahoo! API and much lower than precision of the explicit system Yahoo! API. The implicit system based on Yahoo! API returns has larger proportion of irrelevant documents to relevant documents. This indicates that at this stage the keywords extracted form the documents do not reflect accurately the interest of the user.

6.3.3.2 Relative recall of the implicit system

For the implicit system, the same approach has been used to calculate the relative recall as was used for base APIs and for the explicit system.

Description	Google	Yahoo	Duplicated
Documents	600	600	99
Documents Score	581.5	518.5	137
Recall	**0.56**	**0.50**	

Figure 6.14: Recall of explicit system

Figure 6.14 shows the overall recall of implicit system. The recall is slightly higher with for the version based on Google API.

6.3.3.3 Overall precision and relative recall of implicit system

Figure 6.15 shows the precision and relative recall for the implicit system.

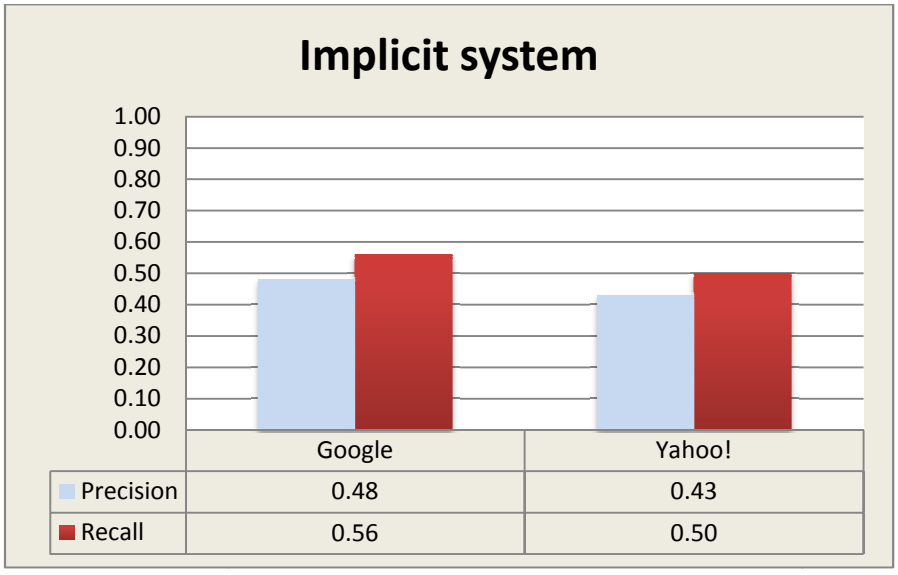


Figure 6.15: Precision and relative recall results for implicit system

The results give an initial indication that the Google-based implicit system performs better than the Yahoo-based system. Its performance is however

very close to the base Google search API, an indication that the mediation is not very effective.

6.3.4 Precision and relative recall with the hybrid system

The hybrid system combines both kinds of user profiles – explicit and implicit. In this system the user provides the keywords explicitly, but the information about the users browsing history is also used to build a combined vector. The system retrieved the search results by using the combined profile vector.

6.3.4.1 Precision of hybrid system using Google and Yahoo

Figure 6.16 shows the hybrid system precision. The precision is close to that of the explicit system based on Google API. The percentage of irrelevant documents is still high.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	233	118	66	180	3	600
%	38.8%	19.7%	11.0%	30.0%	0.5%	
Total score	466	118	33	0	0	617
Overall precision						**0.51**

Figure 6.16: Precision of hybrid system using Google

The precision of the hybrid system using Yahoo! API is presented on Figure 6.17.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	287	100	71	140	2	600
%	47.8%	16.7%	11.8%	23.3%	0.3%	
Total score	574	100	35.5	0	0	709.5
Overall precision						**0.59**

Figure 6.17: Precision of hybrid system using Yahoo! API

The results indicate that the hybrid system with Yahoo! API returns the largest percentage of relevant documents so far. Its precision is even higher than the precision measured for the explicit system Yahoo! API. The combination of the two types of profile has enhanced the effectiveness of the system.

6.3.4.2 Relative recall of the hybrid system

Description	Google	Yahoo	Duplicated
Documents	600	600	111
Documents Score	617	709.5	157
Recall	**0.53**	**0.61**	

Figure 6.18: Relative recall of hybrid system

Figure 6.18 describes the overall recall of hybrid system. The recall for system based on Yahoo! is the highest so far. This reflects an overall improvement in the retrieval of relevant documents.

6.3.4.3 Overall precision and relative recall of hybrid system

Figure 6.19 illustrates the results of the precision and recall measured for the hybrid system. The system combines both implicit and explicit approaches to improve search results.

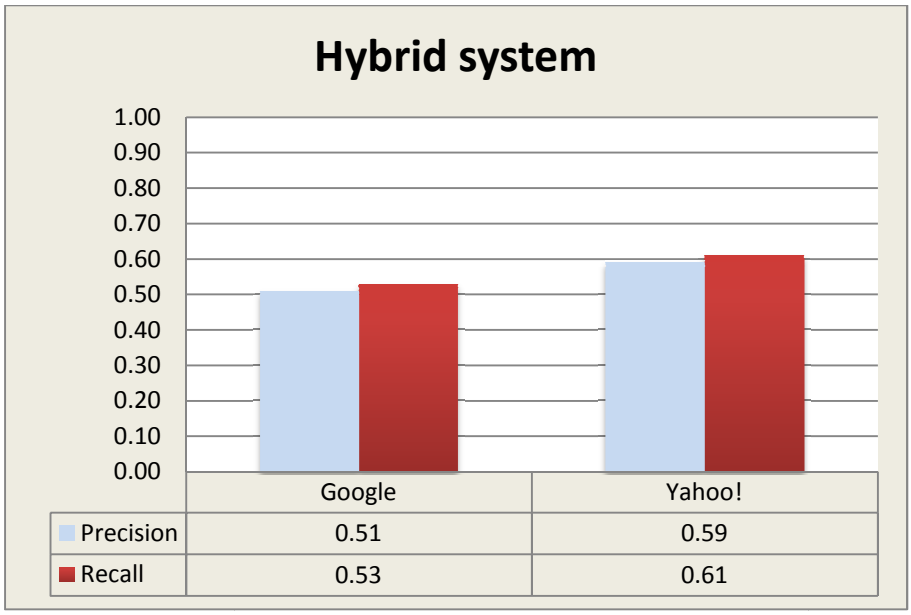


Figure 6.19: Precision and relative recall results for hybrid system

The hybrid system based on Yahoo! API has retrieved results with several percent better precision and recall than hybrid system using Google API. Generally, the hybrid system performs well compared to the other kinds of mediation systems. Retrieved documents are mostly relevant to the query of the users. The precision is high for the hybrid system in comparison with the explicit system, the implicit system, and the base API used.

6.4 Analysis of phase 1 results

This section presents an analysis of the results of experiment phase 1, by collating and comparing the results of the three mediation systems.

6.4.1 Precision results for all systems

Figure 6.20 shows the precision results for all the implemented systems and for the base search engines. The explicit system has a slightly higher precision than the corresponding base search engines. In the first experiments the users had limited time and spent about 15 minutes on the initial browsing.

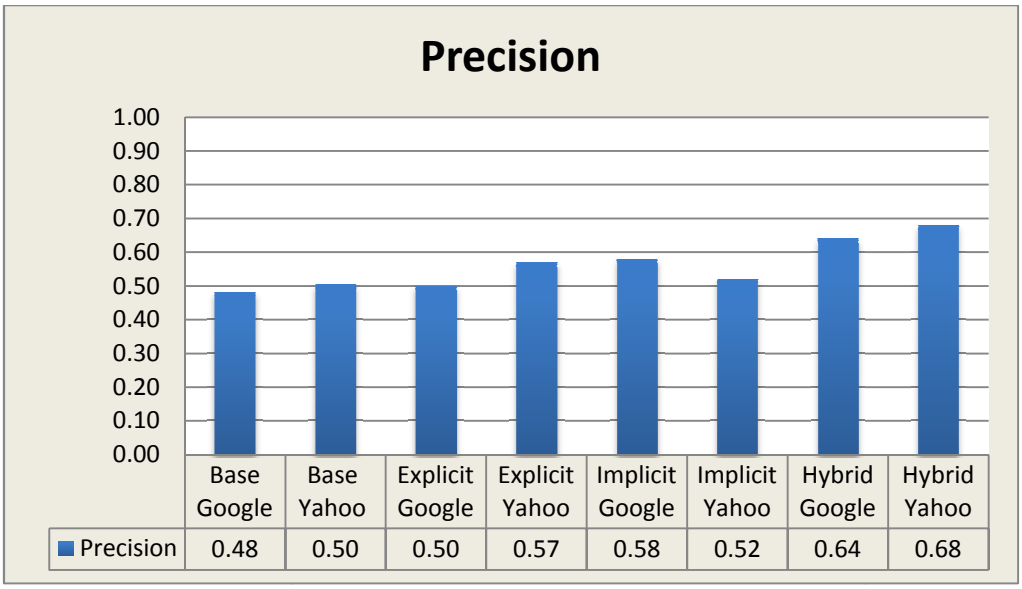


Figure 6.20: Precision results for all systems

The hybrid system improves the search results even after the short learning time. The system appears to be effective in terms of precision, but the difference with the explicit system or base system is minimal. It appears that overall the hybrid system performs better with Yahoo API than with Google API.

The precision results of the each system based on Yahoo! and Google APIs were combined in order to obtain an average precision for each kind of mediation systems, as shown in Figure 6.21. The combined similarity can be either calculated from all the documents retrieved by each system or as the average of the precision calculated individually for the system based on Google or Yahoo APIs. As the number of documents retrieved by each version of system is in every case the same, for all of the systems the results will be the same irrespective of the calculation method.

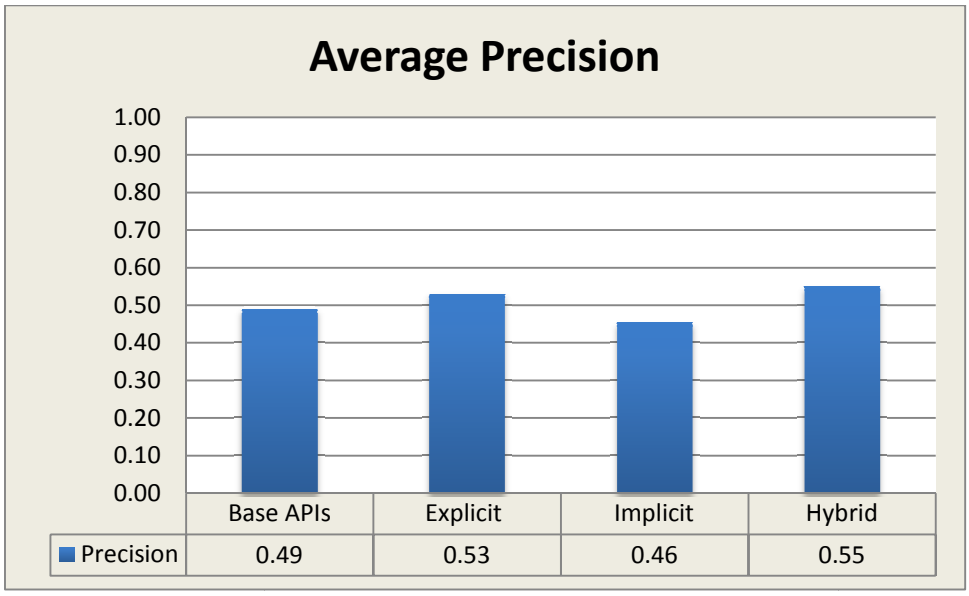


Figure 6.21: Average precision

This confirms that the hybrid system performs better than the other systems. In the first phase of the experiment, the implicit system and hybrid system were creating the implicit vector based on the behaviour collected over 15 minutes and the measured precision may be different after the second phase of the experiments where more learning time was available to the systems.

The experiment results indicate that the explicit system performs generally better that the base Google and Yahoo search engines in terms of precision. Also the hybrid profile provides better precision than the implicit system. The ability of the customers to customise explicitly the retrieval process is clearly an advantage. The combination of the explicit profile and the implicit profile has slightly enhanced the overall profile.

6.4.2 Recall results for all systems

Figure 6.22 shows the relative recall results for all systems, with the two base engines. Although the calculated recall is the relative recall, it has been consistently applied to all the systems and thus it is a useful indicator for effectiveness comparison.

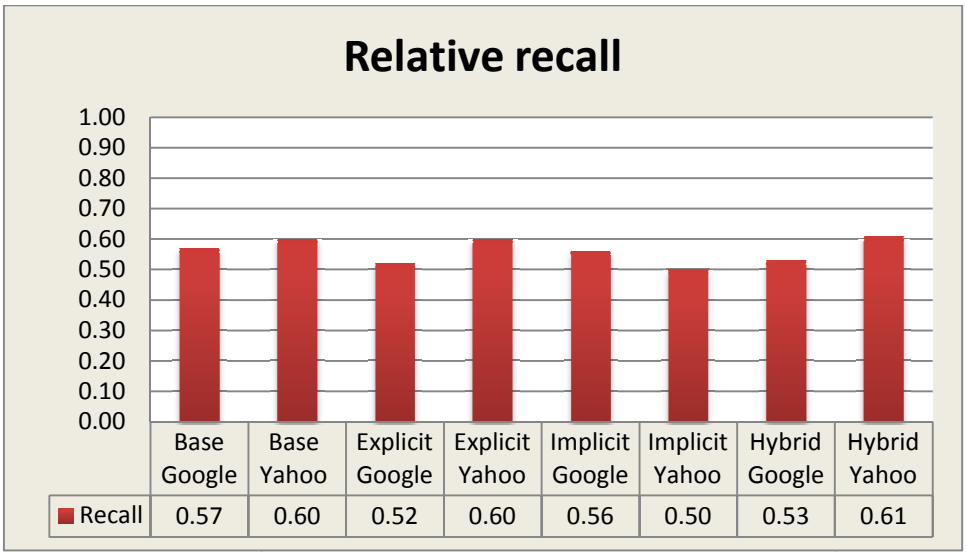


Figure 6.22: Relative recall results for all the systems

The relative recall is high if both versions of each system retrieved the same relevant documents (while irrelevant documents do not affect the relative recall). It can be seen in Figure 6.22 that the highest values for recall is achieved by hybrid, explicit and base system using Yahoo! API, and that the lowest values were measured for the implicit and explicit systems using Google API. The average relative recall for all the systems was also calculated and the results displayed in Figure 6.23.

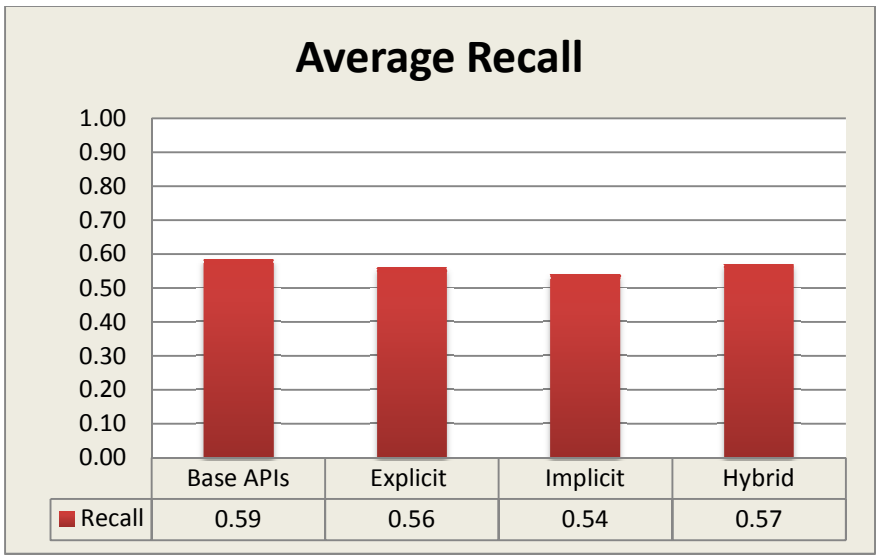


Figure 6.23: Average relative recall

In contrast with precision, the base APIs perform best in terms of the recall. The recall of the hybrid system is lower than the recall for the base APIs by 0.02, however it is higher than the recall of the implicit systems by 0.04.

6.4.3 F-measure results for different systems

The F-measure combines precision and recall into a single measure of effectiveness and it was calculated for all the systems.

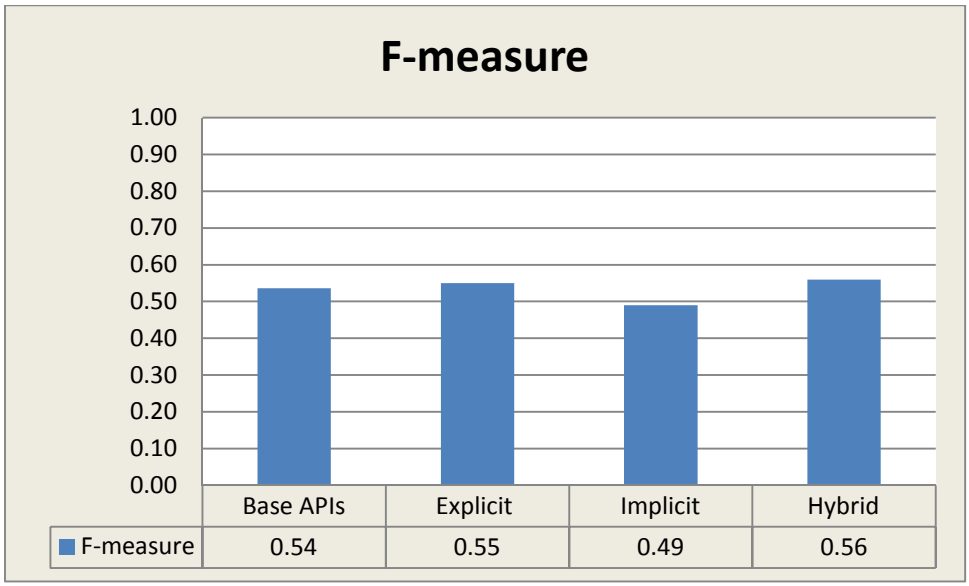


Figure 6.24: F-measure

The Figure 6.24 shows that there is little variation between the F-measure values for any of the systems in phase 1, except for the implicit system which has lower F-measure after the short learning session.

If the user decides to change its interest the hybrid system performance may be affected. To avoid this problem, a user is able to change the session name to create a new separate profile, and can switch between profiles at any time.

6.5 Experiment phase 2 results

The second phase of the experiment includes only the implicit and the hybrid systems as the results will only change for the systems that can learn from the users browsing behaviour. The users who have taken part in the first phase of the experiment were invited for the second phase.

6.5.1 Precision and relative recall with the implicit system

The procedures used to measure the precision and recall of the systems is the same as in the first phase of the experiment. Every user has been using the provided web browser by another15 minutes, which gives total time of learning 30 minutes.

6.5.1.1 Precision of Implicit system using Google and Yahoo APIs

Figure 6.25 shows the precision of implicit system using Google API. The precision is significantly higher than the precession calculated after the first phase of the experiment.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	260	148	61	128	3	600
%	44.7	26.3	11.8	17.7	0.3	
Total score	520	148	30.5	0	0	698.5
Overall precision						**0.58**

Figure 6.25: Precision of the implicit system using Google

Figure 6.26 shows the precision of implicit system based on the Yahoo! API. The precision is much lower than for implicit Google above, but it is

significant improvement compared to the same system results in the first phase.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	218	139	100	141	2	600
%	37.0	26.0	17.8	19.3	0	
Total score	436	139	50	0	0	625
Overall precision						**0.52**

Figure 6.26: Precision of implicit system using Yahoo

6.5.1.2 Relative recall of the implicit system

Figure 6.27 illustrates the relative recall of implicit system. The recall for implicit system is similar to the results obtained in phase 1.

Description	Google	Yahoo	Duplicated
Documents	600	600	70
Documents Score	698.5	625	97.5
Recall	**0.57**	**0.51**	

Figure 6.27: Recall of the implicit system

6.5.1.3 Overall precision and relative recall of implicit system

Figure 6.28 shows the overall retrieval effectiveness results with the implicit system. The results indicate that there is a significant improvement in precision after additional learning but the recall values are similar to those obtained in phase 1. The implicit system performs better with Google API than with Yahoo! API.

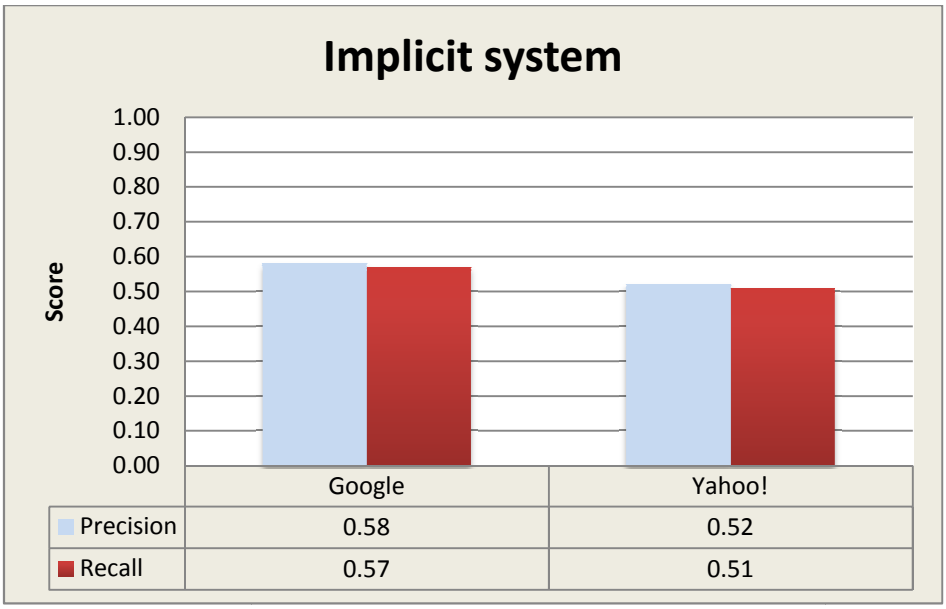


Figure 6.28: Precision and relative recall results for the implicit system

Both versions of the system have returned document from which over half were relevant to queries.

6.5.2 Precision and relative recall with hybrid system

The hybrid system combines the explicit and implicit modes in order to achieve better results in terms of precision and relative recall.

6.5.2.1 Precision of the hybrid system using Google and Yahoo

The results presented in Figure 6.29 indicate that the precision of the hybrid system based on Google API is very high compared to the first phase results, and that the percentage of relevant documents is high.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	305	123	69	103	0	600
% relevant	50.8	20.5	11.5	17.2	0	
Total score	610	123	34.5	0	0	767.5
Overall precision						**0.64**

Figure 6.29: Precision of hybrid system using Google

Figure 6.30 shows the overall precision of the hybrid system based on Yahoo! API. The value is also very high and close to that of the hybrid system using Google API.

The results indicate that the hybrid system is maintaining a more accurate profile and that more relevant documents are retrieved.

Description	Relevant	Less relevant	URLs	Irrelevant	Page cannot be accessed	Total
Number of document	331	116	62	89	2	600
%	55.2	19.4	10.4	14.9	< 0.1	
Total score	662	116	31	0	0	809
Overall precision						**0.67**

Figure 6.30: Precision of hybrid system using Yahoo! API

6.5.2.2 Relative recall of the hybrid system

Figure 6.31 shows the relative recall for the hybrid system. The recall values for both versions of hybrid are close.

Measurement	Google	Yahoo	Duplicated
Documents	600	600	142
Documents Score	767.5	809	228
Recall	**0.57**	**0.60**	

Figure 6.31: Recall of the hybrid system

6.5.2.3 Overall precision and relative recall of hybrid system

The results for hybrid system are similar for both versions – based on Google and Yahoo API, as shown in Figure 6.32.

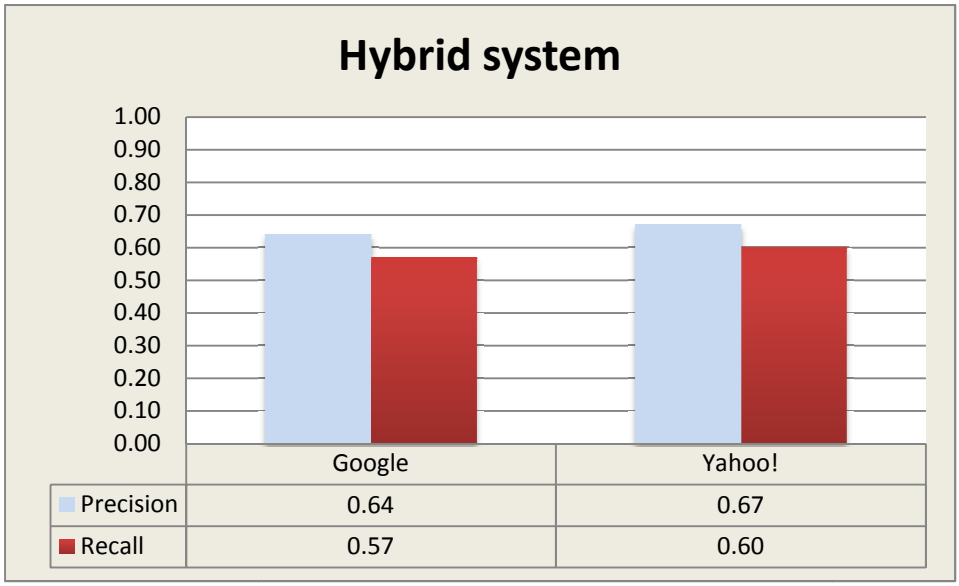


Figure 6.32: Precision and relative recall results for the hybrid system

In both versions of the hybrid system – using Google and Yahoo API, the precision is higher that recall. The retrieved documents are mostly relevant to the queries, but there are still some relevant documents that have not been retrieved (systems are retrieving different relevant documents). The Google API based version has improved in both precision and recall, compared to the results of phase 1, while the Yahoo version has improved the precision but not the recall.

6.6 Comparison of the results from both phases of the experiment

This section presents the effect of the learning process on the precision and the recall of the implicit and the hybrid systems.

6.6.1 Precision

The overall precision of all the systems, measured in phase 1 and 2 is presented in Figure 6.33.

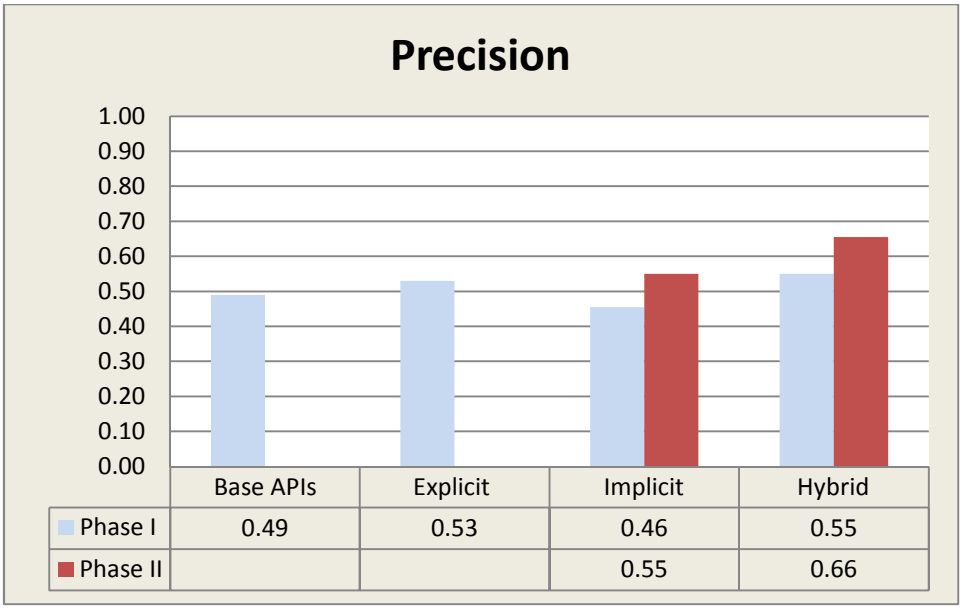


Figure 6.33: Overall precision

It can be seen from the graphs that the precision for both the implicit and the hybrid systems have improved after the additional learning opportunity. The improvement of the implicit system is 19.6% (from 0.46 in the first phase to 0.55 in the second phase) while in the hybrid system the precision have improved by 20% (from 0.55 to 0.66).

The precision of the hybrid system is not worse than the precision of any other systems even after a short learning.

If a user changes the interest, the effectiveness of the hybrid and the implicit system may decrease. Depending on the change, the implicit and hybrid systems could still be useful. The user is allowed to create a new profile without deleting the old one – the system allows creating any numbers of profiles so the user can go back to previously created one.

6.6.2 Statistical significance of comparison of the systems precision

As described in the methodology, the T-test was used to analyse the statistical importance of the experiment results. The statistic has been calculated for the following comparison:

- Hybrid system precision with base APIs precision
- Hybrid system precision with explicit system precision
- Hybrid system precision with implicit system precision

The t-test could be misleading if the distribution of the variable (measured precision for one user) is very different from the normal distribution.

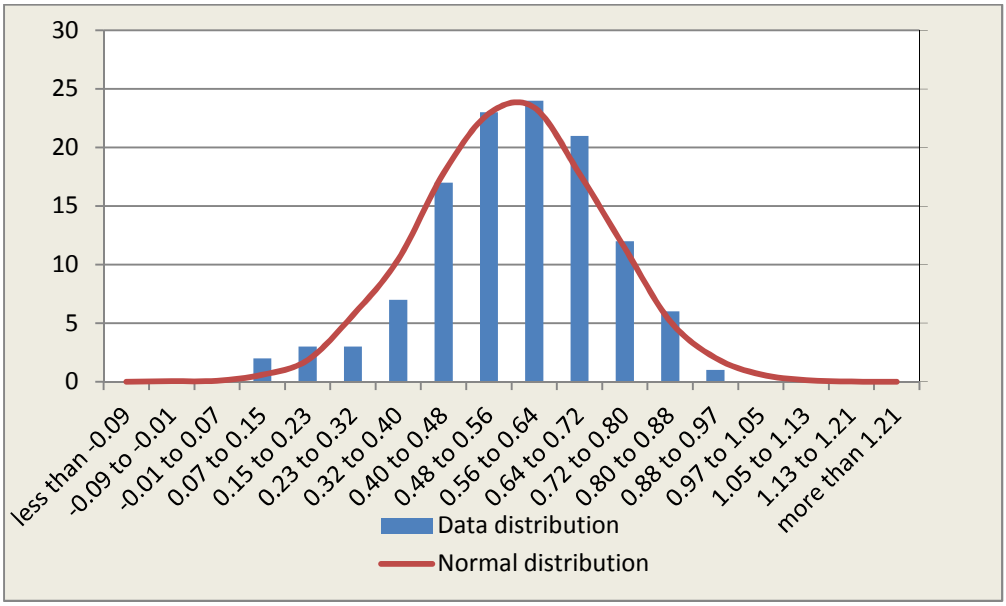


Figure 6.34: Distribution results

The figure 6.34 is presenting the distribution of the precision measured for different users for all systems to show that it is in fact close to the normal distribution, and therefore a t-test can be been applied correctly.

6.6.2.1 Comparison of the precision for hybrid system and the base APIs

The calculated precision for the base API is 0.49, while the precision for the hybrid system is 0.66. The T-test can be used to estimate the degree of trust in the calculated values. For that purpose the null hypothesis is defined as "The increase of precision in the hybrid system over the base API is not significant" and the alternative hypothesis is stating that this increase is significant.

		Hybrid	Base API
Average value	$\bar{x}$	0.657	0.493
Standard deviation	σ	0.029	0.039
Sample size	n	60	60

Figure 6.35: Parameters for the precision comparison for hybrid system and the base APIs

The t statistics is calculated below:

$$t = \frac{\overline{x_1}+\overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1}+\frac{\sigma_2}{n_2}}} = 4.86 \qquad \text{[Equation 6.6]}$$

The null hypothesis can be rejected if the calculated value for t is higher than the critical t-value (t_c), which can be read from tables for given samples size and significance level. The critical t-value for importance level 0.01 is equal to 3.1607.

For significance level $\alpha = 0.01$ the t_c value is 3.1607.

4.86 is higher than 3.1607, therefore t is larger than t_c and the null hypothesis has to be rejected. Therefore the alternative hypothesis, stating that the increase of precision in the hybrid system over the base API is significant, can be accepted with 0.01 significance level (99% confidence).

6.6.2.2 Comparison of the precision for hybrid system and the explicit system

The calculated precision for the explicit system is 0.53, while the precision for the hybrid system is 0.66. The null hypothesis in the T-test is defined as "The increase of precision in the hybrid system over the explicit system is not significant" and the alternative hypothesis is stating that this increase is significant.

		Hybrid	Base API
Average value	$\bar{x}$	0.657	0.533
Standard deviation	σ	0.029	0.032
Sample size	n	60	60

Figure 6.36: Parameters for the precision comparison for hybrid system and the explicit system

The t statistics is calculated below:

$$t = \frac{\overline{x_1}+\overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1}+\frac{\sigma_2}{n_2}}} = 3.872 \qquad \text{[Equation 6.7]}$$

The null hypothesis can be rejected if the calculated value for t is higher than the critical t-value (t_c) The critical t-value for importance level 0.01 can be read from tables as 3.1607.

For significance level $\alpha = 0.01$ the t_c value is 3.1607.

4.86 is higher than 3.1607, therefore t is larger than t_c and the null hypothesis has to be rejected. Therefore the alternative hypothesis, stating

that the increase of precision in the hybrid system over the explicit system is significant, and can be accepted with 0.01 significance level.

6.6.2.3 Comparison of the precision for hybrid system and the implicit system

The calculated precision for the implicit system is 0.55, while the precision for the hybrid system is 0.66. The null hypothesis is defined as "The increase of precision in the hybrid system over the implicit system is not significant" and the alternative hypothesis is stating that this increase is significant.

		Hybrid	Base API
Average value	$\bar{x}$	0.657	0.549
Standard deviation	σ	0.029	0.026
Sample size	n	60	60

Figure 6.37: Parameters for the precision comparison for hybrid system and the implicit system

The t statistics is calculated below:

$$t = \frac{\overline{x_1}+\overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1}+\frac{\sigma_2}{n_2}}} = 3.562 \qquad \text{[Equation 6.8]}$$

The null hypothesis can be rejected if the calculated value for t is higher than the critical t-value (t_c), which can be read from tables for given samples size and significance level. The critical t-value for importance level 0.01 is equal to 3.1607.

For significance level $\alpha = 0.01$ the t_c value is 3.1607.

3.562 is higher than 3.1607, therefore t is larger than t_c and the null hypothesis has to be rejected, and instead the alternative hypothesis, stating

that the increase of precision in the hybrid system over the base API is significant, can be accepted with 0.01 significance level (99% confidence).

6.6.3 Recall

Recall is a measure of the completeness of the retrieval process. The higher the recall value, the lower will be the number of relevant documents not retrieved. The overall recall of all the systems, measured in phase 1 and 2 is presented on Figure 6.38. Although the determination of recall may be approximate because of the small pool of relevant documents, it has been consistently applied in the experiments with all the systems to allow comparison.

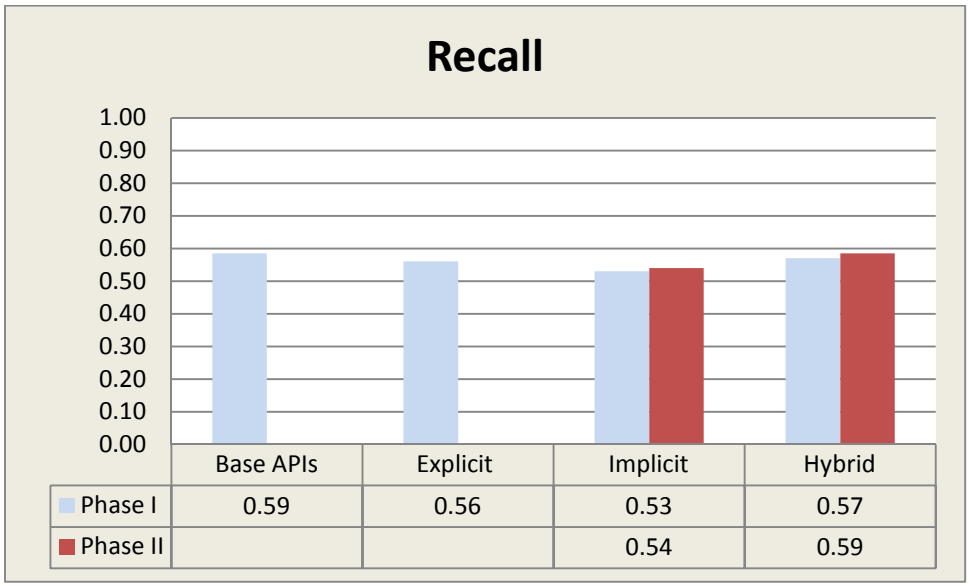


Figure 6.38: Overall recall

It can be seen from the graph that all systems have similar recall.

The hybrid system recall has improved, but the improvement is only 3.5% (0.59 versus 0.57 measured in the first phase). The effectiveness of all the systems is similar in terms of recall, and there is no visible recall advantage

in using any of them. The highest value achieved by the hybrid system is equal to that achieved by the base search engines.

6.6.4 Statistical significance of comparison of the systems recall

As described in the methodology, the T-test was used to analyse the statistical importance of the experiment results. The statistic has been calculated for the following comparison:

- Hybrid system recall with base APIs recall
- Hybrid system recall with explicit system recall
- Hybrid system recall with implicit system recall

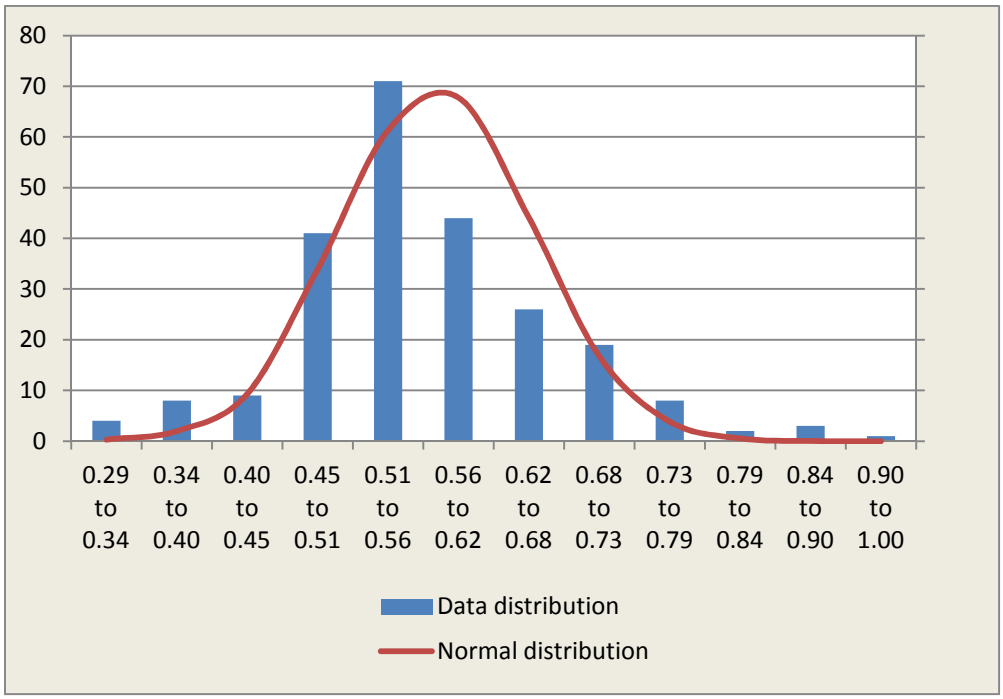


Figure 6.39: Distribution results

The figure 6.39 is presenting the distribution of the recall measured for different users for all systems to show that it is in fact close to the normal distribution, and therefore t-test can be been applied correctly.

6.6.4.1 Comparison of the recall for hybrid system and the base APIs

The calculated recall for the base API is 0.592, while the recall for the hybrid system is 0.594. The T-test can be used to estimate the degree of trust in the calculated values. For that purpose the null hypothesis is defined as "The increase of recall in the hybrid system over the base API is not significant" and the alternative hypothesis is stating that this increase is significant.

		Hybrid	Base API
Average value	$\bar{x}$	0.592	0.594
Standard deviation	σ	0.012	0.013
Sample size	n	60	60

Figure 6.40: Parameters for the recall comparison for hybrid system and the base APIs

The t statistics is calculated below:

$$t = \frac{\overline{x_1}+\overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1}+\frac{\sigma_2}{n_2}}} = 0.085 \qquad \text{[Equation 6.9]}$$

The null hypothesis can be rejected if the calculated value for t is higher than the critical t-value (t_c), which can be read from tables for given samples size and significance level.

For significance level $\alpha = 0.01$ the t_c value is 2.3583.

The value 0.085 is lower than 2.3583, therefore t is lower than t_c and the null hypothesis cannot be rejected. Therefore the null hypothesis, stating that the increase of recall in the hybrid system over the base API is not significant, cannot be rejected (with 99% confidence).

6.6.4.2 Comparison of the recall for hybrid system and the explicit system

The calculated recall for the explicit system is 0.56. The recall for the hybrid system is 0.592. The null hypothesis in the T-test is defined as "The increase of recall in the hybrid system over the average recall for the explicit system is not significant" and the alternative hypothesis is stating that this increase is significant.

		Hybrid	Base API
Average value	$\bar{x}$	0.592	0.579
Standard deviation	σ	0.0122	0.0241
Sample size	n	60	60

Figure 6.41: Parameters for the recall comparison for hybrid system and the explicit system

The t statistics is calculated below:

$$t = \frac{\overline{x_1}+\overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1}+\frac{\sigma_2}{n_2}}} = 1.307 \qquad \text{[Equation 6.10]}$$

For significance level α = 0.01 the t_c value is 2.3583, therefore the null hypothesis cannot be rejected, as the t value is lower – the hypothesis that the recall of the hybrid system has not improved over the recall for the explicit system cannot be rejected (with 99% confidence).

6.6.4.3 Comparison of the recall for hybrid system and the implicit system

The calculated recall for the implicit system is 0.54, while the recall for the hybrid system is 0.59. The null hypothesis is defined as "The increase of recall in the hybrid system over the implicit system is not significant" and the alternative hypothesis is stating that this increase is significant.

		Hybrid	Base API
Average value	$\bar{x}$	0.592	0.542
Standard deviation	σ	0.012	0.011
Sample size	n	60	60

Figure 6.42: Parameters for the recall comparison for hybrid system and the implicit system

The t statistics is calculated below:

$$t = \frac{\overline{x_1}+\overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1}+\frac{\sigma_2}{n_2}}} = 2.550 \qquad \text{[Equation 6.11]}$$

The null hypothesis can be rejected if the calculated value for t is higher than the critical t-value (t_c), which can be read from tables for given samples size and significance level. The critical t-value for importance level 0.01 is equal to 2.3583.

2.250 is lower than 2.3583, therefore t is lower than t_c and the null hypothesis cannot be rejected with 99% confidence.

The statistical analysis indicates that there is no significant improvement in recall with the hybrid system.

6.6.5 F-measure

The F-measure combines precision and recall into a single measure of effectiveness and it was calculated for all the systems.

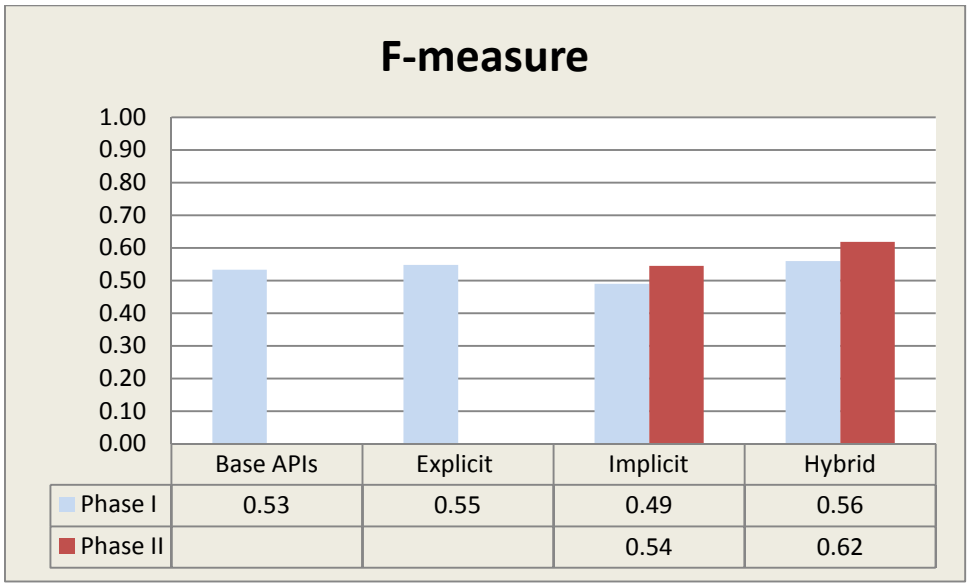


Figure 6.43: F-measure

The Figure 6.43 shows that there is little variation between the F-measure values for any of the systems in phase 1, except for the implicit system which needs additional learning to achieve similar results to the other systems.

It can be clearly seen that the performance of the hybrid system has improved after the additional learning (by 10.7% - the new value is 0.62 while before it was 0.56). Even without the extra learning its performance was no worse than the performance of any of the other systems. The users can benefit from the hybrid system even after short learning (15 minutes).

If the user decides to change its interest the hybrid system performance may be affected. A user is able to change session name to create a new separate profile, and can switch between profiles at any time.

6.6.6 Statistical significance of comparison of the systems F-measure

As described in the methodology, the T-test was used to analyse the statistical importance of the experiment results. The statistic has been calculated for the following comparison:

- Hybrid system F-measure with base APIs F-measure
- Hybrid system F-measure with explicit system F-measure
- Hybrid system F-measure with implicit system F-measure

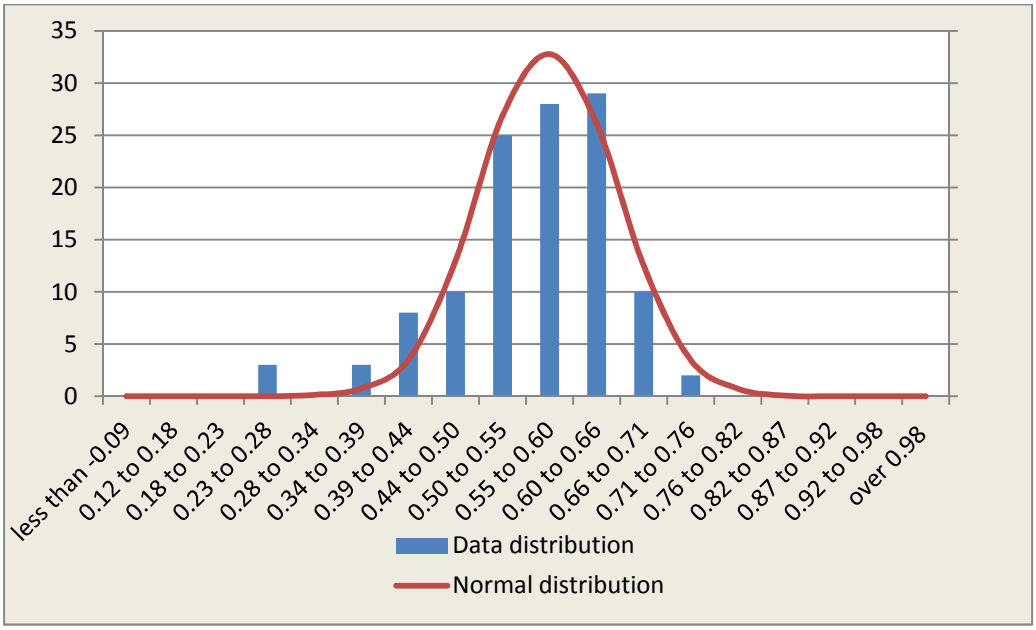


Figure 6.44: Distribution results

The figure 6.44 is presenting the distribution of the precision measured for different users for all systems to show that it is in fact close to the normal distribution, and therefore t-test can be been applied correctly.

6.6.6.1 Comparison of the F-measure for hybrid system and the base APIs

The F-measure parameters calculated for the base APIs and the hybrid systems are presented in table 6.45.

		Hybrid	Base API
Average value	$\bar{x}$	0.560	0.536
Standard deviation	σ	0.0144	0.0260
Sample size	n	60	60

Figure 6.45: Parameters for the F-measure comparison for hybrid system and the base APIs

The null hypothesis is defined as "The increase of F-measure in the hybrid system over the base API is not significant" and the alternative hypothesis is stating that this increase is significant.

The t statistics is calculated below:

$$t = \frac{\overline{x_1}+\overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1}+\frac{\sigma_2}{n_2}}} = 0.927 \qquad \text{[Equation 6.12]}$$

The null hypothesis can be rejected if the calculated value for t is higher than the critical t-value (t_c), which for importance level 0.01 is equal to 2.3583, therefore the null hypothesis cannot be rejected with 99% confidence.

6.6.6.2 Comparison of the F-measure for hybrid system and the explicit system

The F-measure parameters calculated for the explicit system and the hybrid systems are presented in table 6.46.

		Hybrid	Base API
Average value	$\bar{x}$	0.560	0.519
Standard deviation	σ	0.0144	0.0240
Sample size	n	60	60

Figure 6.46: Parameters for the F-measure comparison for hybrid system and the explicit system

The null hypothesis is defined as "The increase of F-measure in the hybrid system over the base API is not significant" and the alternative hypothesis is stating that this increase is significant.

The t statistics is calculated below:

$$t = \frac{\overline{x_1}+\overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1}+\frac{\sigma_2}{n_2}}} = 1.593 \qquad \text{[Equation 6.13]}$$

The null hypothesis can be rejected if the calculated value for t is higher than the critical t-value (t_c) The critical t-value for importance level 0.01 can be read from tables as 2.3583.

The null hypothesis can be rejected if the calculated value for t is higher than the critical t-value (t_c), therefore in this case the null hypothesis cannot be rejected with 99% confidence – the difference is not significant enough.

6.6.6.3 Comparison of the F-measure for hybrid system and the implicit system

The F-measure parameters calculated for the implicit system and the hybrid systems are presented in table 6.47.

		Hybrid	Base API
Average value	$\bar{x}$	0.560	0.534
Standard deviation	σ	0.0144	0.0133
Sample size	n	60	60

Figure 6.47: Parameters for the F-measure comparison for hybrid system and the implicit system

The null hypothesis is defined as "The increase of F-measure in the hybrid system over the base API is not significant" and the alternative hypothesis is stating that this increase is significant.

The t statistics is calculated below:

$$t = \frac{\overline{x_1} + \overline{x_2}}{\sqrt{\frac{\sigma_1}{n_1} + \frac{\sigma_2}{n_2}}} = 1.198 \qquad \text{[Equation 6.14]}$$

The critical t-value for importance level 0.01 can be read from tables as 2.3583, therefore the null hypothesis cannot be rejected with 99% confidence.

6.6.7 Summary of quantitative evaluation

The summary of the evaluation (after the second phase of the experiment) is presented in the table below.

Parameter	System	Value	Relation to the Hybrid system
Precision	Base API	0.49	Significantly lower (99% confidence)
	Explicit	0.53	Significantly lower (99% confidence)
	Implicit	0.55	Significantly lower (99% confidence)
	Hybrid	**0.66**	N/A
Recall	Base API	0.59	Not significant (or less than 99% confidence)
	Explicit	0.56	Not significant (or less than 99% confidence)
	Implicit	0.54	Not significant (or less than 99% confidence)
	Hybrid	**0.59**	N/A
F-measure	Base API	0.54	Not significant (or less than 99% confidence)
	Explicit	0.55	Not significant (or less than 99% confidence)
	Implicit	0.55	Not significant (or less than 99% confidence)
	Hybrid	**0.62**	N/A

Figure 6.48 Summary of the evaluation results

It can be seen from figure 6.48 that the after the learning phase, the hybrid system outperforms all other kinds of mediation systems in terms of precision. The ability of the users to formulate their profile explicitly is enhanced by the implicit analysis of previously visited documents - both modes of profile generation reinforce each other; while the implicit profile narrows the scope of the search, the explicit profile ensures that there is clear focus on user interest during the profile generation. The evidence indicates that the hybrid mediation system highly enhances the documents retrieval precision.

The recall and the F-measure (which is partially delivered from the recall) have not improved significantly. The difference in the values of recall and F-measure calculated for each of the systems is visible, however high deviation of the data samples does not allow telling whether this difference is

significant. The lack of improvement in recall may be due mainly to the difficulty of measuring recall. The use of relative recall, based on two search engines only where almost all the documents are similar is essentially equivalent to the use of one search engine.

Given the fact that precision and recall are equally weighted, despite some increase in precision the lack of improvement in recall has had a dampening effect on the calculation of the F-measure.

6.7 Qualitative evaluation

In this section a number of personalised systems are presented in order to identify the contribution of the proposed hybrid mediation framework. Recommender systems are designed to recommend content based on learning algorithms. In general in a content–based filtering system items are selected according to an explicit or implicit profile and the content of document visited or ranked.

Syskill & Webert is a content-based filtering system based only on explicit profile generated when the user provides a feedback for visited items. It is designed to improve the item recommendations by selecting items that are matched either on the basis of the generated explicit user profile or the query of the user (Garden and Dudek 2006). The user has to rate a number of web pages for the system to be able to analyse the page content and deduce the interest of the user interest (Pazzani, Muramatsu and Billsus 1996). The main shortcoming of this system is that the user has to rank explicitly visited pages. The system relies on this explicit feedback to generate a profile. In contrast the hybrid system can benefit from all visited documents by analysing in the background parameters such as time spent on the page without any additional intervention from the user. The Syskill & Webert system is ineffective if the domain of search changes because a new profile has to be generated (Pazzani, Muramatsu and Billsus 1996).

Lieberman (1995) have developed a system called Letizia which creates implicit users profiles from the analysis of the individual browsing behaviour. It assumes that the user is interested in a document if the documents is saved or bookmarked and weak interest if the document is left without following links inside the document. The system gives weight to documents that are linked to the documents that the user is currently viewing and suggests similar documents that match the implicit profile. The system does not make use of any explicit data for the recommendations. The proposed hybrid system on the other hand incorporates both an implicit user profile as well as an explicit profile.

The WebWatcher system monitors the choice of links by the user for the future recommendation of links. The system does not require the submission of keywords or explicit ratings. It considers the documents that were retrieved through a link as examples of documents of interest to the user, and the documents that were available through links but not visited as examples of documents not relevant. These documents are also included into the building of the implicit profile as negative examples (Mladenic 1996). If a document is considered as a negative example the system will not suggest similar documents in the future. There may be however many reasons for not visiting a links, e.g. user have already found relevant information or there is more than one link of interest to the user and only one of them was followed. The negative factor may influence a system to ignore interesting documents. The proposed system does not consider links as one of the criteria in the document analysis and is relatively open on the content of documents – documents are never considered as negative (irrelevant) examples.

Stegmann (2005) presented an approach to explicit user profiling that complies with personal interests by means of an adaptive natural language dialogue. The system captures the information provided by users during a dialogue session and stores it in an explicit user profile. This kind of interaction requires however very high attention from the user.

Although this brief review has only covered a subset of filtering systems it has highlighted the advantages and shortcomings of the proposed framework. Its ability to incorporate different modes of profile generation and to accommodate some learning is one of its most attractive features. It creates a context where user and system can collaborate in retrieving relevant documents. In contrast most systems tend to focus on one aspect of profiling. The proposed hybrid mediation system combines the explicit and implicit profiles to create a more effective mediation system.

6.8 Summary

The experimental results have indicated that the combination of different modes of mediation is a viable option in filtering documents in Web search. The experiment has shown that the explicit system performs well compared to the base API used. It allows the users to formulate their search interests with immediate effect. Although the implicit system alone does not perform as well, its combination with the explicit features into a hybrid system appears to be the best of the investigated modes of mediation, especially after the system had more time for learning. This system performs consistently better in terms of precision than other systems, without decreasing the recall and F-measure performance.

Compared to some content-based systems the mediation framework is able to combine different modes of mediation to provide more relevant irrespective of the base search engine.

The performance of the hybrid system could improve further over the performance of the base APIs, if the methods of gathering implicit and explicit information are improved. Jung, Herlocker and Webster (2007) has developed a system that besides of gathering the basic browsing information was also analysing the clicking, and claimed that the clicking is the most accurate indicator for predicting the user's behaviour (Jung, Herlocker and Webster 2007). Rastegari and Shamsuddin (2010) have also agreed that the clicking can be the most accurate indicator. Therefore adding the support of

recording clicking can be a way of further improving the implicit profile generation (Rastegari and Shamsuddin 2010). The explicit profile generation can also be improved with features like the explicit rating of documents, as it was done by Claypool, Waseda and Brown (2001) who has also added the possibility of implicitly predicting the explicit rating for documents that were not rated by the user.

Chapter 7

Conclusions and Further Work

7.1 Introduction

The main objective of this research was to investigate the impact of different modes of mediation on the Web Search process. It involved three main tasks. First, the investigation of methods and mechanisms in user profile generation and in filtering search results. Second task was focused on the design and implementation of a mediation framework as a layer between a user and classic Web Search engines. Finally the third task was to provide the comparative evaluation of the impact of the different types of mediation systems on web search results in terms of precision and recall.

The aim of this chapter is to provide concise conclusions of the research presented in this thesis and to determine to what extent the research objectives were met. An assessment of the current status of the mediation framework will also help to identify the contribution of this work and it will offer pointers for further work.

7.2 Research contribution and conclusions

The main objectives of this research as were identified in section 1.5. They were stated as follows:

- To identify and investigate issues related to the web and search engines.
- To investigate the role of different personalisation techniques and retrieval models in the enhancement of the quality of retrieval process.
- To propose a novel approach for enhancing the filtering of search results by combining selectively different methods.
- To design and implement a mediation framework that enables the deployment of three different user profiling methods.
- To perform a quantitative evaluation of the mediation framework in terms of precision recall and F-measure as well as a qualitative evaluation.

This research is an integral part of the effort aimed at overcoming the limitations of the classic search engines. In addressing this issue a critical evaluation of various profiling techniques and of retrieval models was carried. The investigation has led to the proposal of a mediation approach which was applied in the development of a mediation framework. It involved the integration of three modes of user profiling within a content-based information retrieval method, and it was facilitated by the adoption of the Vector Space Model.

The developed framework acted as a vehicle for the investigation of the impact of the modes of user profiling mediation on Web Search results. Explicit, implicit and hybrid profile generation were incorporated into three mediation systems to represent prevailing forms of user profiling. The combination of explicit and implicit methods into the hybrid method has ensured that document filtering was performed according to context and without incurring the shortcomings of cold start. The performance of the

systems was evaluated with the help of a large number of users in terms of precision and recall.

The explicit mediation system enables users to formulate and change easily their search interests. It has also the advantage that it does not suffer from the cold start. Its evaluation shows that the precision of this system is only slightly better than the precision calculated for base Google APIs, while the recall is slightly lower. It can be stated that there is no visible improvement in using the explicit user profile alone, with Google API. Its performance is however much better with the Yahoo! API for both precision and recall.

The implicit mediation system was designed with the assumption that users browsing activities can indicate whether a currently opened document is of interest to the user; observed activities include the time spent reading a page, printing a page or saving it. The system learns from the browsing behaviour, and can filter search results to find documents that are similar to documents that were of interest to the user in previous browsing sessions. The use of the implicit mediation system yields less accurate results than the base Web search APIs, in terms of both precision and recall. However after the learning time was doubled the precision improved and was higher than precision of the base APIs and of the explicit system.

The hybrid mediation system combines the explicitly stated interests with the observation of user behaviour. The experimental results indicate that the hybrid system yields better and more accurate results than the other two mediation systems or the base APIs. In addition, after the learning time was doubled the precision of the system increased in relation to its previous precision and in relation to the base APIs. This system appears to be the best of the compared approaches of enhancing the retrieval effectiveness. While the precision improved, the recall and F-measure have not been significantly affected. These results indicate clearly that a hybrid system can enhance the quality of the search results. The hybrid system has managed to retrieve documents from which a large proportion was relevant to user intentions.

The framework was carefully evaluated with real users using the three systems with the Google and the Yahoo! APIs. The results expressed in terms of precision and recall and were validated by a statistical analysis. The significance tests confirm that the mediation framework enhances the quality of the retrieval process, and that it performs better than the basic APIs.

The investigation into mediation systems and related techniques, and the development of the mediation framework, as well as its evaluation can help form an objective assessment on the main contribution of this research. The contribution lies essentially in the provision of three different mediation systems and the evaluation of their impact on the web search process. More specifically, the combination of different modes of mediation within a content-based method represents one of the distinctive features of this work. This research contributes to the validation of the view that personalisation can offer an effective way of dealing with information overload. This view is supported by significance tests.

From this review of the work that was carried out and the identification of the contribution of this research it can be stated that all the objectives of the research were met.

7.3 Limitations of the research

Although the aims and objectives of this research were met, a number of limitations have been identified in the resulting system and the process. These are detailed below:

- The implicit profiling makes use of three variables only to help generate a profile. This restriction can have an adverse effect on the accuracy of the profile.
- The calculation of the similarity is performed by exact match only. Useful documents can be ignored by this linguistic constraint.
- The number of keywords is limited to 5. An imbalance may result from the competition for space by the explicit and implicit profiles.

- Use of relative recall as an approximation performed with two search engines only. The pool of documents is very small and many documents are not accessed.
- Although the number of users is statically significant, 30 users only were involved in the testing. The sample may not give a true reflection of the performance of the system.
- Efficiency issues are important but were not addressed, especially in the hybrid system. The overheads of the framework and its systems were not investigated.
- The learning process has been investigated properly. It is difficult to assess precisely at what time the learning takes place and is most effective.

7.4 Further Work

Although the proposed framework appears to be a viable mediator between users and the Web, there is still scope for enhancing its effectiveness. Some of the issues that are considered for further work include:

- The criteria used for implicit generation are limited to time spent, printing and saving. Further work will seek to generate implicitly more accurate profiles, e.g. by widening the criteria of observation to include bookmarking and link selection.
- The proposed framework is based on using the exact match for the keywords. Instead of using an exact match for keywords, their synonyms could be considered as well. Using ontology offers a way of expanding the scope of the proposed system. It would help identify the terms that are related to those stored in a user profile.
- The three systems operate as mediators between two search engines APIs and the users. The framework can incorporate more search engines, including domain specific search engines to provide better access to sources. This will also provide a larger pool of documents and improve the calculation of recall.
- The proposed framework relies on a content-based approach. The scope of the framework can be expanded by including collaborative

features such as the clustering of users according to common interests.

7.5 Summary

This research has provided the opportunity to gain a deeper insight into mediation systems. An assessment of the contribution of this work indicates that the research programme has met successfully all its objectives. The research has confirmed that mediation frameworks can improve the quality of the web search results and that the choice of the mode of mediation, whether explicit, implicit or hybrid is an important factor in enhancing precision and recall.

References

Abual-Rub, M. S., Abdullah, R., and Rashid, N. A. (2007) 'A Modified Vector Space Model for Protein Retrieval'. *Journal of Computer Science and Network Security* 7 (9), 85-89

Ahn, J. W., Brusilovsky, P., Grady, j., He, D., and Syn, S. Y. (2007) *''Open User Profiles for Adaptive News Systems: Help or Harm?*" 'Proceedings of the Sixteenth International World Wide Web Conference'. held at Alberta, Canada

Amati, G., Crestani, F., Ubaldini, F., and Nardis, S. D. (1997) *Probabilistic Learning for Information Filtering*. 'RIAO, 5th International Conference'. held at McGill University, Montreal. Canada

Aoidh, E. M., Bertolotto, M., and Wilson, D. C. (2007) *Implicit Profiling for Contextual Reasoning about Users.* '7th International Conference on Case Based Reasoning (ICCBR)'. held at Belfast, Northern, Ireland

Balabanovic, M., and Shoham, Y. (1997) 'Fab: Content-Based, Collaborative Recommendation'. *Journal on Communications of the ACM* 40 (3), 66-72

Bernard, J. J., and Spink, A. (2006) 'How are we Searching the World Wide Web? A Comparison of Nine Search Engine Transaction Logs'. *Journal on Information Processing and Management* 42 (1), 248-163

Berners-Lee, T., Cailliau, R., Nielsen, H. F., and Secret, A. (1994) 'The World-Wide Web'. *Journal on Communication of the ACM* 37 (8), 76-82

Berry, M. W., Drmac, Z., and Elizabeth, J. R. (1999) 'Matrices, Vector Spaces, and Information Retrieval'. *Journal on SIAM Review* 41 (2), 335-362

Beza-Yates, R., and Ribeiro-Neto, B. (1999) *Modern Information Retrieval*. ACM Press: USA

References

Blachman, N., and Peek, J. (2007) How Google Works [online] available from <http://comptechnoportal.files.wordpress.com/2009/11/how-google-works-google-guide.pdf> [Dec. 2007]

Bradford, R. (2008) *An Empirical Study of Required Dimensionality for Large-scale Latent Semantic Indexing Applications*. 'Proceedings of the 17th ACM Conference on Information and Knowledge Management'. held at Napa Valley, California, USA. 153–162

Brin, S., and Page, L. (1998) 'The Anatomy of a Large-Scale Hypertextual Web Search Engine'. *Journal on Computer Networks* 30 (1-7), 107-117

Brusilovsky, P. and Tasso, C. (2004) 'Preface to Special Issue on User Modeling for Web Information Retrieval'. *Journal on User Modeling for Web Information Retrieval* 14 (2-3), 147-157

Budzik, J., and Hammond, K. (1999) *Watson: Anticipating and Contextualizing Information Needs*. 'Proceedings of the Sixty-Second Annual Meeting of the American Society for Information Science'. held at Medford, NJ, 727-740

Burright, M. (2006) Database Reviews and Reports Google Scholar Science & Technology. [online] available from http://www.library.ucsb.edu/istl/06-winter/databases2.html

Busby, M. (2003) Learn Google. Plano, Texas: Wordware Publishing, Inc.

Cayzer, S., and Michlmayr, E. (2008) Adaptive User Profiles. HP Laboratories. [online] available from <http://www.hpl.hp.com/techreports/2008/HPL-2008-201.pdf> [2008]

Clarke, S., and Willett, P. (1997) Estimating the recall performance of search engines. Association of Special Libraries (ASLIB) Proceedings 49 (7), 184-189.

Claypool, M., Gokhale, A., Miranda, T., Murnikov, P., Netes, D., and Sartin, M. (1999) *Combining Content-Based and Collaborative Filters in an Online*

Newspaper. '*Proceedings of ACM SIGIR Workshop on Recommender Systems'*. held at Berkeley, California

Claypool, M., Le, P., Waseda, P., and Brown, D. (2001) *Implicit Interest Indicators.* 'Proceeding of the 6th international conference on intelligent user interface'. held at Santa Fe, New Mexico, United States. 33-40

Dean, J., and Ghemawat, S. (2008) 'MapReduce: Simplified Data Processing on Large Clusters'. *Journal on Communication of the ACM* 6 (1), 107-113

Deerwester, S., Dumais, S. T., Furnas, G. W., Landauer, T. K., and Harshman, R. (1990) 'Indexing by Latent Semantic Indexing'. *Journal of the American Society for Information Science.* 41(6), 321-407

Delgado, J., Ishii, N., and Ura, T. (1998) *Content-Based Collaborative Information Filtering: Actively Learning to Classify and Recommend Documents*. *'Proceedings of the Second International Workshop on Cooperative Information Agents II, Learning, Mobility and Electronic Commerce for Information Discovery on the Internet'*. held at London, UK

Ferragina, P., and Gulli, A. (2005) *A Personalized Search Engine Based on Web-Snippet Hierarchical Clustering.* 'Conference on International World Wide Web'. held at Chiba, Japan

Frias-Martinez, E., Cebrian, M., Moises, J.P., and Oliver, N. (2009) *Explicit vs. Implicit Tagging for User Modeling*. 'Proceedings of the Workshop on Personalization in Mobile and Pervasive Computing'. held at Rento, Italy

Fox, S., Karnawat, K., Mydland, M., Dumais, S., and White, T. (2005) 'Evaluating implicit measures to improve web search'. *Journal of ACM Transactions on Information Systems* 23(2), 147–168

Fuhr, N. (1992) 'Probabilistic Models in Information Retrieval'. *Journal on Computer* 35 (3), 243-255

Garden, M., and Dudek, G. (2006) 'Mixed Collaborative and Content-Based Filtering with User-Contributed Semantic Features'. (ed.*) Proceedings of the 21st National Conference on Artificial Intelligence.* held at Boston, Massachusetts: AAAI Press, 1307-1312

Gasparetti, F., and Micarelli, A. (2007) *Exploiting Web Browsing Histories to Identify User Needs.* 'Proceedings of the 12th International Conference on Intelligent User Interfaces'. held at Honolulu, Hawaii: ACM, 325-328

Gauch, S., Chaffee, J., and Pretschner, A. (2003) 'Ontology-Based Personalized Search and Browsing'. *Journal on Web Intelligence and Agent Systems* 1 (3-4), 219-234

Gemechu, F., Yu, Z., and Ting, Y. (2010) A Framework for Personalized Information Retrieval Model. *Computer and Network Technology (ICCNT),* 500 – 505

Ghosh, R., and Dekhil, M. (2009) *Discovering User Profiles*. Proceedings of the 18th International Conference on World Wide Web. held at Madrid, Spain.

Gils, B.V., Proper, H. A., Bommel, P. V., and Schabell, E. D. (2003) 'Profile-based retrieval on the World Wide Web'. Bra, P. D (ed.) *Proceedings of the Ninth Interdisciplinary Conference on Information Science.* held at Eindhoven University of Technology, 91-98

Google (2011) Google Help [online] available from <http://www.google.com/support/websearch/bin/static.py?hl=en&page=guide.cs&guide=1186810&answer=106230&rd=1> [June 2011]

Google (2011) Webmaster Tools Help [online] available from <http://www.google.com/support/webmasters/bin/answer.py?answer=34439> [May 2011]

Grcar, M., Mladenic, D., and Grobelnik, M. (2005) 'User Profiling for Interest-Focused Browsing History'. *Proceeding of the Workshop on End User Aspects of the Semantic Web*, 'Conjunction with the 2nd European Semantic Web Conference'. held at Heraklion, Greece

Grimmelmann, J. (2007) 'The Structure of Search Engine Law'. IOWA LAW REVIEW 93(1), 1-64 [online] available from http://www.nyu.edu/projects/nissenbaum/papers/Grimmelmann_StructureOfSearchEngineLaw.pdf [2007]

Grossman, D. A., and Frieder, O. (2004) Information Retrieval-Algorithms and Heuristics. 2nd Edition edn. Netherlands: Springer

Hendler, J., and Berners-Lee, T. (2010) 'From the Semantic Web to Social Machines: A Research Challenge for AI on the World Wide Web'. *Journal on Artificial Intelligence* 174 (2), 156-161

Hiemstra, D. (2009) 'information Retrieval Models'. Goker, A., and Davies, J. (ed.) Information Retrieval: Searching in the 21st Century. Publisher: John Wiley and Sons

Holmes, E. G. (2006) 'Google and Beyond: Finding Information using Search Engines, and Evaluating Your Results'. *Journal on Technical Services Law Librarian* 31 (2), 8-9

Hopfgartner, F., Hannah, D., Gildea, N., and Jose, J.M. (2008) *Capturing Multiple Interests in News Video Retrieval by Incorporating the Ostensive Model.* 'Proceeding of the Second International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases'. held at Auckland, New Zealand, 48–55

Huang, Z., Chen, H., and Zeng, D. D. (2004) 'Applying Associative Retrieval Techniques to Alleviate the Sparsity Problem in Collaborative Filtering'. *Journal of ACM Transactions of Information Systems*. 22, 116-142

Hunt, B. (2005) Search Engine Watch-what Exactly, is Search Engine Spam? [online] available from <http://searchenginewatch.com/article/2067496/What-Exactly-is-Search-Engine-Spam> [Feb. 2005]

Hussein, M., and Elsayed, T. (2008) Studying Facial Expressions as an Implicit Feedback in Information Retrieval Systems.

Ichikawa, Y., Nakamura, M., Hata, K., and Nakagawa, T. (2008) 'Provision of Services According to Individual User Preferences Over a Cross-Section of Sites Implemented with Personalized-Service Platform'. *NTT Information Sharing Platform Laboratories*. Musashino-shi, Japan

Jawaheer, G., Szomszor, M., Kostkova, P. (2010) *Comparison of Implicit and Explicit Feedback from an Online Music Recommendation Service.* 'Proceedings of the 1st International Workshop on Information Heterogeneity and Fusion in Recommender Systems'. held at New York, USA

Jones, K. S., Walker, S., and Robertson, S. E. (2000) 'A Probabilistic Model of Information Retrieval: Development and Comparative Experiments'. *Journal on Information Processing and Management* 36 (6), 779-808

Jung, S., Herlocker, J.L., and Webster, J. (2007) 'Click data as implicit relevance feedback in web search'. *Journal of Information Processing and Management* 43 (3), 791–807

Kagie, M., Loos, M. V. D., and Wezel, M. V. (2009) 'Including Item Characteristics in the Probabilistic Latent Semantic Analysis Model for Collaborative Filtering'. *Journal of AI Communications* 22 (4), 249-265

Kamishima, T., and Akaho, S. (2006) *Nantonac Collaborative Filtering Methods- Recommendation Based on Order Responses.* 'Proceedings of the National Institute of Advanced Industrial Science and Technology (AIST)'. International workshop on data-mining and Statistical Science (DMSS2006). held at Sapporo, Japan.

Kelly, D., and Belkin, N. J. (2001) *Reading Time, Scrolling and Interaction: Exploring Implicit Sources of User Preferences for Relevance Feedback during Interactive Information Retrieval.* 'Conference on SIGIR'. held at New Orleans, USA

Kelly, D., and Belkin N. J. (2004) *Display time as implicit feedback: understanding task effects.* 'Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval'. held at New York, USA, 377–384

Khribi, M. K., Jemni, M., and Nasraoui, O. (2009) *Automatic Recommendations for E-Learning Personalization Based on Web Usage Mining Techniques and Information Retrieval.* 'Proceedings of the Eighth IEEE International Conference on Advanced Learning Technologies'. held at Santander, Cantabria. 2, 30-42

Klusch, M. (2001) 'Information Agent Technology for the Internet: A Survey'. *Journal on Data & Knowledge Engineering* 36 (3), 337-372

Kumar, B. T. S., and Prakash, J. N. (2009) 'Precision and Relative Recall of Search Engines: A Comparative Study of Google and Yahoo'. *Journal of Library & Information Management* 38, 124-137

Lawrence, S. (2000) 'Context in Web Search'. *IEEE Data Engineering Bulletin* 23 (3), 25-32

Lemire, D., and Maclachlan, A. (2005) 'Slope One Predictors for Online Rating-Based Collaborative Filtering'. *Proceedings of the Fifth SIAM International Conference on Data Mining.* ed. by Anon, 471-480

Li, Q., and He, D. (2010) *Searching for Entities: When Retrieval Meets Extraction.* 'The Nineteenth Text Retrieval Conference' (TREC). held at Gaithersburg, MD: NIST

Li, Q., and Kim, B. M. (2003) *An Approach for Combining Content-Based and Collaborative Filters.* 'Proceedings of the Sixth International Workshop on

Information Retrieval with Asian Languages'. held at Sapporo, Japan ACM,17-24

Liu, F., Yu, F., and Meng, W. (2006) *Effective keyword search in relational databases.* 'Proceedings of the ACM SIGMOD international conference on Management of data'. held at Chicago, IL, USA

Liddy, E. D. (2005) Document Retrieval, Automatic [online] available from http://www.cnlp.org/publications/Document.Retrieval.Liz.pdf.edn: Published in the Encyclopedia of Language & Linguistics, Elsevier Limited

Lieberman, H. (1995) 'Letizia: An Agent that Assists Web Browsing'. ed. Mellish, C. San Mateo, CA: Morgan Kaufmann publishers Inc. 924-929

Manavoglu, E., Pavlov, D., and Giles, C. L. (2003) 'Probabilistic User Behavior Models'. *Proceedings of Third IEEE International Conference on Data Mining (ICDM 2003)*. ed. by Anon, 203-210

Manning, C. D., Raghavan, P., and Schütze, H. (2008) Introduction to Information Retrieval. United States: Cambridge University Press

Maron, M. E., and Kuhns, J. L. (1960). 'On relevance, probabilistic indexing and information retrieval'. *Journal of the ACM*. 7, 216-244.

Meteren, R. V., and Someren, M. V. (2000) 'Using Content-Based Filtering for Recommendation'. *Proceedings of MLnet/ECML2000 Workshop. held in Barcelona*. Spain

Metzler, D., and Croft, W.B. (2007) 'Linear feature-based models for information retrieval'. *Journal of Information Retrieval*, 10(3), 257-274

Mladenic, D. (1996) Personal Webwatcher: Design and Implementation. [online] available from [http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.2143]

Mowshowitz, A., and Kawaguchi, A. (2002) 'Bias on the Web'. *Journal of communication of the ACM* 45 (9), 56-60

Mowshowitz, A., and Kawaguchi, A. (2005) 'Measuring Search Engine Bias'. *Journal on Information Processing and Management* 41 (5), 1193-1205

Naik, N.P., and Rao, A.M. (2011) *Information Search and Retrieval System in Libraries.* 'Proceedings of *the* 8th International Caliber. held at Goa University, Goa.

Notess, G. R. (2008) Review of Yahoo! Search [online] available from <http://www.searchengineshowdown.com/features/yahoo/review.html> [Jan. 2009]

Parkes, D.C., and Seuken, S. (2011) [online lecture] CS 186 Lecture 17-Recommender Systems. Available from http://www.seas.harvard.edu/courses/cs186/doc/17-rec-sys.pdf

Paulson, P., and Tzanavari, A. (2003) 'Combining Collaborative and Content Based Filtering using Conceptual Graphs'. *Lectures Notes in Computer Science*, 168-185

Pazzani, M. J., Muramatsu, J., and Billsus, D. (1996) 'Syskill & Webert: Identifying Interesting Web Sites'. *in Proceedings of the Thirteenth National Conference on Artificial Intelligence.* ed. by Anon, Portland, US: AAAI Press, 54-61

Pazzani, M. J., and Billsus, D. (2007) 'Content-based recommendation systems'. in *Lecture Notes on the Adaptive Web: Methods and Strategies of Web Personalization.* ed. by Springer-Verlag, 325-341

Polyvyanyy, A., and Kuropka, D. (2007) 'A Quantitative Evaluation of the Enhanced Topic-Based Vector Space Model': *A Technical Report of the Hasso-Plattner-Institute*, 19

Rashid, A., Mamunur., Albert, I., Cosley, D., Lam, S. K., McNee, S. M., Konstan, J. A., and Riedl, J. (2002) *Getting to Know You: Learning New User Preferences in Recommender Systems*. 'Proceedings of the 7th International Conference on Intelligent User Interfaces'. held at San Francisco, California, USA: ACM Press

Rastegari, H., and Shamsuddin, S.M. (2010) 'Web Search Personalization Based on Browsing History by Artificial Immune System'. *Journal of Advances in Soft Computing and Its Applications* 3 (2), 282-301

Robertson, S. E., van Rijsbergen, C. J., and Porter, M. F. (1981) *Probabilistic Models of Indexing and Searching.* 'Proceedings of the 3rd Annual ACM Conference on Research and Development in Information Retrieval'. held at Kent, UK: Butterworth

Robertson, S. (2004) 'Understanding Inverse Document Frequency: On Theoretical Arguments for IDF'. *Journal of Documentation* 60 (5), 503-523

Rucker, J., and Polanco, M. J. (1997) 'Siteseer: Personalized Navigation for the Web'. *Journal on Communications of the ACM* 40 (3), 73-76

Salton, G., Fox, E. A., and Wu, H. (1983) 'Extended Boolean Information Retrieval'. *Journal of Communication of the ACM* 26 (11), 1022-1036

Salton, G., Singhal, A., Mitra, M., and Buckley, C. (1997) 'Automatic Text Structuring and Summarization'. *Journal of Information Processing and Management* 33 (2), 193-207

Sankaradass, V., and Arputharaj, K. (2011) 'An Intelligent Recommendation System for Web User Personalization with Fuzzy Temporal Association Rules'. *Journal of European Scientific Research* 51 (1), 88-96

Sarwar, B. M., Konstan, J. A., and Riedl, J. (2005) 'Distributed Recommender Systems for Internet Commerce'. *Encyclopedia of Information Science and Technology,* 907-911

Shah, C. (2009) Retrieval Models-1. USA

Shafi, S. M., and Rather, R. A. (2005) Precision and Recall of Five Search Engines for Retrieval of Scholarly Information in the Field of Biotechnology. Webology, 2 (2), Article 12. [online] available from <http://www.webology.org/2005/v2n2/a12.html> [Aug. 2005]

Shen, X., Tan, B., and Zhai, C. (2006) 'Exploiting Personal Search History to Improve Search Accuracy'. (ed.) *Proceedings of 2006 ACM Conference on Research and Development on Information Retrieval*, 'Personal Information Management Workshop'. SIGIR

Sieg, A., Mobasher, B., and Burke, R. (2004) 'Inferring User's Information Context: Integrating User Profiles and Concept Hierarchies'. *Proceedings of the 2004 Meeting of the International Federation of Classification Societies*, IFCS 2004. ed. by AnonChicago, 563-574

Singhal, A., and Salton, G. (1995) 'Automatic Text Browsing using Vector Space Model'. *Proceedings of the Dual-use Technologies and Applications Conference. 318- 324*

Singhal, A. (2001) 'Modern Information Retrieval: A Brief Overview'. *Bulletin of the IEEE Computer Society Technical Committee on Data*, 24

Skorkovská, L., and Pavel I. (2009) 'Experiments with Automatic Query Formulation in the Extended Boolean Model'. Lecture Notes in Computer Science. Publisher: Springer. 5729, 371-378

Slawski, B. (2008) Yahoo Phrase Based Indexing in a Nutshell [online] available from <http://www.seobythesea.com/2008/02/yahoo-phrase-based-indexing-in-a-nutshell/> [July 2011]

Smyth, B., and Wilson, D. (2003) 'Explicit vs. implicit profiling – a case-study in electronic programme guides'. *Proceedings of the 18th International Joint Conference on Artificial Intelligence.* held at Acapulco, Mexico

Stegmann, R. (2005) 'Improving Explicit Profile Acquisition by Means of Adaptive Natural Language Dialog'. in Lecture Notes in Computer Science. ed. by Anon, 518-520

Sugiyama, K., Hatano, K., and Yoshikawa, M. (2004) *Adaptive Web Search Based on User Profile Constructed without any Effort from Users.* 'Proceedings

of the 13th International Conference on World Wide Web'. held at New York, USA: ACM Press

Swapna, P., and Ravindran, R. B. (2008) 'Personalized Web-Page Rendering System'. Das, G., Sarda, N. L., and Reddy, K.P. (ed.) *Proceedings of COMAD* held in India: Computer Society of India, 30-39

Tanudjaja, F., and Mui, L. (2002) *Persona: A Contextualized and Personalized Web Search*. 'Proceedings of the 35th Annual Hawaii International Conference on System Science'. held at Island, Hawaii

Van Rijsbergen, C.J., (1979) *Information Retrieval*. London; Boston. Butterworth, 2nd Edition

Voorhees, M., and Harnam, D. K. (2005) *TREC Experiment and Evaluation in Information Retrieval*. Cambridge, Massachusetts: MIT Press

White, R. W., Jose, J. M., and Ruthven, I. (2003) *An Approach for Implicitly Detecting Information Needs.* 'Proceedings of the Twelfth International Conference on Information and Knowledge Management'. held at New York, USA: ACM Press

Yahoo (2011) My Yahoo! [online] available from <http://my.yahoo.com> [2011]

Yahoo (2010) Yahoo! Advertising Blog. [online] available from <http://www.yadvertisingblog.com/blog/2010/08/31/advertisers-begin-your-account-transitions/> [Dec.2010]

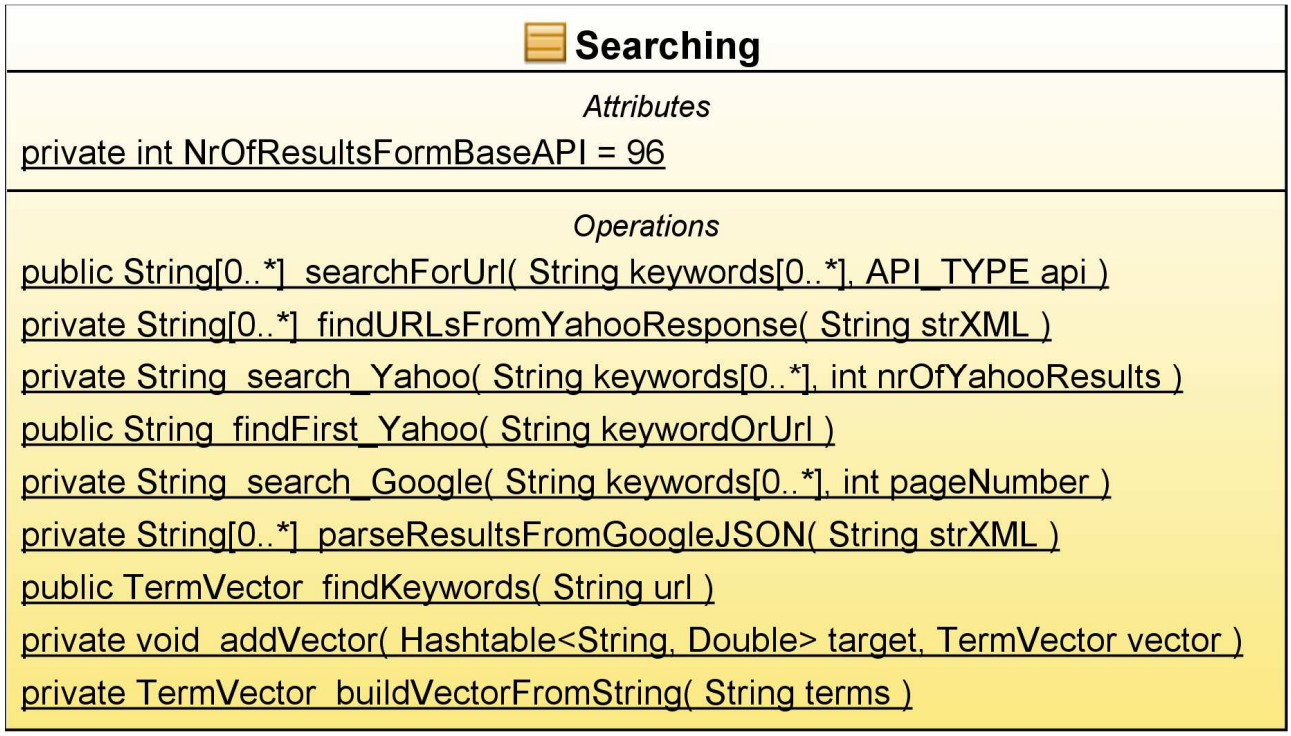
Yip, William., and Quiroga, L. (2008) Google Page Rank Algorithm, LIS 678 Personalized Information [online] available from <http://willwork.org/lis678/Special%20Topics/Report.pdf> [Oct.2008]

Zigoris, P., and Zhang, Y. (2006) *Bayesian Adaptive User Profiling with Explicit \& Implicit Feedback*. 'Proceedings of the 15th ACM International Conference on Information and Knowledge Management'. held at Arlington, Virginia, USA: ACM Press

Some code not strongly related to the mediation framework (e.g. error handling or user interface related) has been removed.

Project 'ExplicitUserProfiles'

File 'Searching.java'




```
// List of base search APIs avaliable
public enum API_TYPE
{
    API_GOOGLE,
    API_YAHOO
}
```

```
   // Number of results from Yahoo or Google.
   private final static int NrOfResultsFormBaseAPI = 100;
```

searchForUrl

```
// Main method for searching in Google on Yahoo API.
public static List<String> searchForUrl(String[] keywords, API_TYPE api)
{
   if (api == API_TYPE.API_GOOGLE)
   {
      int noOfPages = NrOfResultsFormBaseAPI / 8;
      List<String> results = new LinkedList<String>();

      for (int i = 0; i <= noOfPages; i++)
      {
         String json = search_Google(keywords, i);
         List<String> pageResults = parseResultsFromGoogleJSON(json);
         for (String url : pageResults)
            if (!results.contains(url)
              && results.length < NrOfResultsFormBaseAPI) results.add(url);
      }
      return results;
   }
   else if (api == API_TYPE.API_YAHOO)
   {
      String searchResult = search_Yahoo(keywords,NrOfResultsFormBaseAPI);
      return findURLsFromYahooResponse(searchResult);
   }
}
```

findURLsFromYahooResponse

```
// Extract the URLs from the Search Results from Yahoo
private static List<String> findURLsFromYahooResponse(String strXML)
{
   LinkedList<String> results = new LinkedList<String>();

   DocumentBuilder builder
               = DocumentBuilderFactory.newInstance().newDocumentBuilder();
   org.w3c.dom.Document doc
               = builder.parse(InputSource(new java.io.StringReader(strXML)));

   doc.getDocumentElement().normalize();
   org.w3c.dom.NodeList nodeLst = doc.getElementsByTagName("Result");

   for (int s = 0; s < nodeLst.getLength(); s++) // iterate results
   {
      org.w3c.dom.Node fstNode = nodeLst.item(s);
      if (fstNode.getNodeType() == org.w3c.dom.Node.ELEMENT_NODE)
      {
         org.w3c.dom.NodeList clickUrlsNodes
                  = ((Element)fstNode).getElementsByTagName("ClickUrl");

         if (clickUrlsNodes.getLength() > 0)
         {
            // there should be only one ClickUrl per result
            Element clickUrlElement = (Element)clickUrlsNodes.item(0);

            // get Text from that element
            org.w3c.dom.NodeList text = clickUrlElement.getChildNodes();
```

```
                // get text value
                String clickUrl
                          = ((org.w3c.dom.Node) text.item(0)).getNodeValue();

                // add text to the list
                results.add(clickUrl);
            }
        }
    }
    return results;
}
```

search_Yahoo

```
// get response from Yahoo! API
private static String search_Yahoo(String[] keywords, int nrOfYahooResults)
{
    String encodedKeywords = "";
    for (String s : keywords)
       encodedKeywords += (encodedKeywords.length() > 0 ? "+" : "")
                        + URLEncoder.encode(s, "UTF-8");

    String request = "http://api.search.yahoo.com/WebSearchService/V1/"
                   + "webSearch?appid=YahooDemo&results="
                   + nrOfYahooResults
                   + "&query=" + encodedKeywords;

    // Send GET request
    GetMethod method = new client.GetMethod(request);
    if(new HttpClient().executeMethod(method) != HttpStatus.SC_OK)return null;

    // Get the response body
    InputStream rstream = method.getResponseBodyAsStream();

    // Process the response from Yahoo! Web Services
    BufferedReader br = new BufferedReader(new InputStreamReader(rstream));
    String result = […]; // read stream line by line

    return result;
}
```

String search_Google

```
private static String search_Google(String[] keywords, int pageNumber)
{
    String encodedKeywords = […];

    String request = "https://ajax.googleapis.com/ajax/services/search/web"
                   + "?v=1.0&key=AIzaSyBk2lMNQ2RfKR3l5TirizdIb2KdByrVySI"
                   + "&userip=194.66.32.16&rsz=8&q="
                   + encodedKeywords
                   + "&start=" + pageNumber;

    org.apache.commons.httpclient.HttpClient client
          = new org.apache.commons.httpclient.HttpClient();
    GetMethod method = new GetMethod(request);
```

```
      // Send GET request
      GetMethod method = new client.GetMethod(request);
      if(new HttpClient().executeMethod(method) != HttpStatus.SC_OK)return null;

      // Get the response body
      InputStream rstream = method.getResponseBodyAsStream();
      String result = […]; // read stream line by line
      return result;
   }
```

parseResultsFromGoogleJSON

```
   // parse JSON returned by Google API to get list of URLs
   private static List<String> parseResultsFromGoogleJSON(String strXML)
   {
      //  { "responseData":
      //  {"results": [ {"unescapedUrl":"[…]", […]}, {[…]} ] }, […] }
      JSONObject obj = (JSONObject) JSONValue.parse(strXML);
      obj = (JSONObject) obj.get("responseData");
      JSONArray results = (JSONArray) obj.get("results");
      List<String> urls = new LinkedList<String>();
      for (Object o : results)
         urls.add((String)(((JSONObject)o).get("unescapedUrl")));
      return urls;
   }
```

findKeywords

```
   public static TermVector findKeywords(String url)
   {
      //=== open the document and read keywords from metadata
      String document = MyUtils.UtilsWeb.getURL(url);
      document = document.toLowerCase();

      int end = document.indexOf("</head");
      document = document.substring(0, end).replace("'", "\"");

      String title = ""; String metakeywords = ""; String metadescription = "";

      // extract the metadata 'keywords', 'description' and title
      // documents can be not vell-formated xml […]
      int start = document.indexOf("<title");
      if (start != -1)
      {
            start = document.indexOf(">", start);
            end = document.indexOf("</title", start);
            title = document.substring(start + 1, end);
      }

      int offset = document.indexOf("<meta");
      while (offset >= 0)
      {
         end = document.indexOf(">", offset);
         if (document.substring(offset, end).contains("keywords"))
         {
            start = document.indexOf("content", offset);
            start = document.indexOf("\"", start + 1);
            end = document.indexOf("\"", start + 1);
```

```
            metakeywords = document.substring(start + 1, end);
        }
        if (document.substring(offset, end).contains("description"))
        {  […]
            metadescription = document.substring(start + 1, end);
        }
        offset = document.indexOf("<meta", offset + 1);
    }

    // build vetors
    FrequencyVectorCreator fvc = new FrequencyVectorCreator();

    TermVector vectTitle = buildVectorFromString(title);
    TermVector vectKeyw = buildVectorFromString(metakeywords);
    TermVector vectDesc = buildVectorFromString(metadescription);

    // scale vectors
    vectTitle.scaleBy(0.3);
    vectKeyw.scaleBy(0.5);
    vectDesc.scaleBy(0.2);

    // combine three vectors into one
    Hashtable<String, Double> pairs = new Hashtable<String, Double>();
    addVector(pairs, vectTitle);
    addVector(pairs, vectKeyw);
    addVector(pairs, vectDesc);

    TermVector combinedVector = new TermVector();
    Iterator<Entry<String, Double>> it = pairs.entrySet().iterator();

    while (it.hasNext())
    {
        Entry<String, Double> entry = it.next();
        combinedVector.put(entry.getKey(), entry.getValue());
    }

    TermVector result = combinedVector.topN(5);
    result.normalize();

    return result; // take top N keywords
}
```

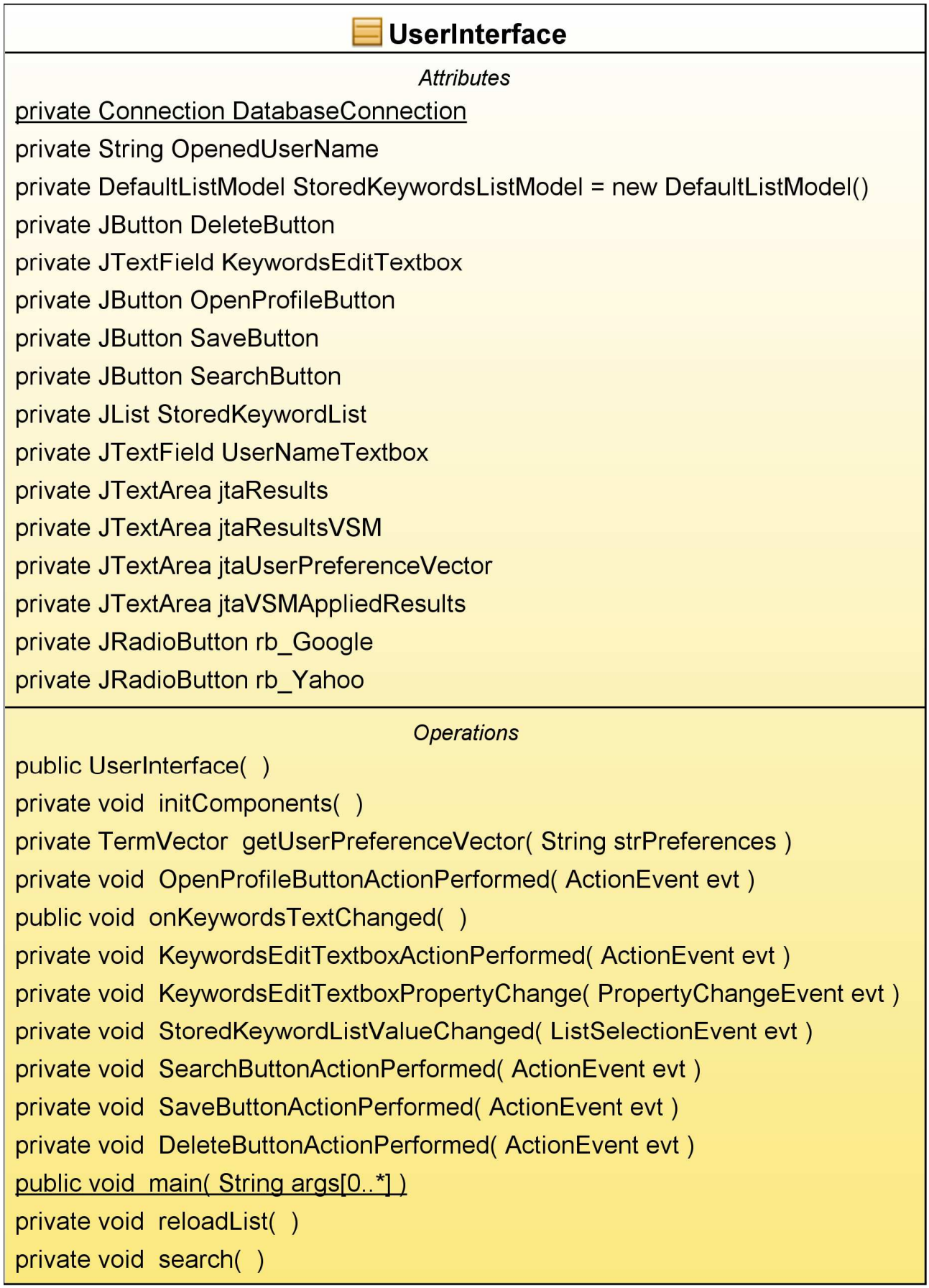
addVector

```
   private static void addVector(Hashtable<String, Double> target,
                                 TermVector vector)
   {
      Iterator it = vector.termIterator();
      while (it.hasNext())
      {
         String key = (String) it.next();
         double value = vector.get(key);

         if (target.containsKey(key))
         {
            value = (Double) target.get(key) + value;
            target.remove(key);
         }
         target.put(key, value);
      }
   }
```

buildVectorFromString

```
   private static TermVector buildVectorFromString(String terms)
   {
      FrequencyVectorCreator fvc = new FrequencyVectorCreator();
      return fvc.getVector(new ASCIIDocument(terms));
   }
```

File 'UserInterface.java'

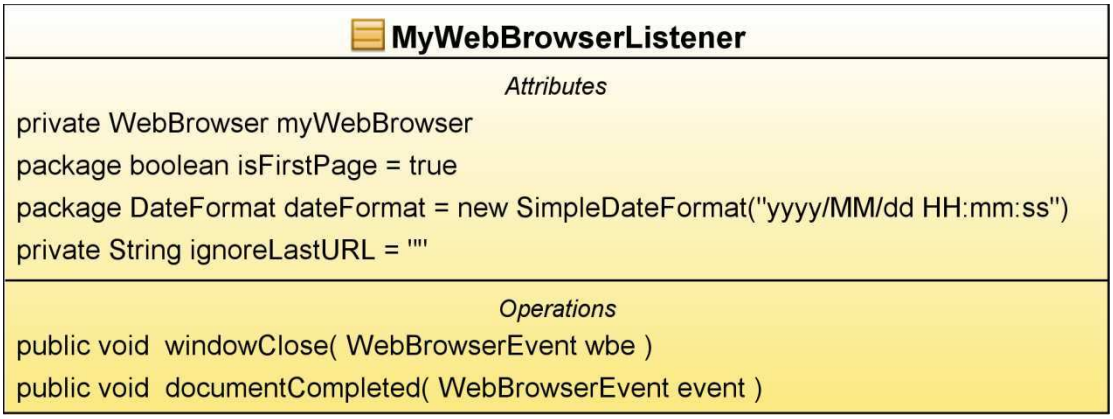


search

```
    private void search()
    {
        this.jtaResults.setText("Searching...");
        this.jtaResultsVSM.setText("");
        this.jtaVSMAppliedResults.setText("");

        //-- prepare keywords
        String strKeyword = "";
        strKeyword = this.KeywordsEditTextbox.getText()
           .replace('=', '+').replace(',', '+').replace(' ', '+');

        //-- find urls in yahoo or google
        List<String> urls = Searching.searchForUrl(
           strKeyword.split("\\+"),
           (rb_Yahoo.isSelected()
             ? Searching.API_TYPE.API_YAHOO
             : Searching.API_TYPE.API_GOOGLE));

        // display list of documents from API
        this.jtaResults.setText([…]);

        // get keywords for documents
        // and sort by similarity to the explicit vector
        RAMSearchEngine rse = new RAMSearchEngine();

        for(String url:urls)
        {
           HTMLDocument objDoc = new HTMLDocument("<html></html>");
           // findKeywords returns a normalised vector
           TermVector documentKeywords = Searching.findKeywords(url);
           rse.addDocument(url, objDoc.getFullContent(),
                                                documentKeywords);
        }

        // create user preference vector
        TermVector userPref = getUserPreferenceVector(strKeyword);
        userPref.normalize();
        jtaUserPreferenceVector.setText(userPref.toString());

        // retrieve
        ValueSortedMap vsm = rse.retrieveDocuments(userPref, 20);
        Iterator itr = vsm.keyIterator();

        fullText = "";
        while(itr.hasNext()) fullText += itr.next().toString()+"\n";
        jtaVSMAppliedResults.setText(fullText);
    }
```

Project 'ImplicitUserProfiles'

File 'MyWebBrowserListener.java'



documentCompleted

```
 private String ignoreLastURL = "";

 // called when document has been loaded into the browser
 public void documentCompleted(WebBrowserEvent event)
 {
    String url = this.myWebBrowser.getURL().toString();

    if(url == null || url.equalsIgnoreCase(ignoreLastURL)) return;

    // check if the main part of the url is the same
    if(ignoreLastURL != null && url.contains("#"))
    {
       if(ignoreLastURL
              .startsWith(url.substring(0, url.indexOf("#"))))
          return;

       ignoreLastURL = url;

       this.mySimpleBrowser.logCurrentActivity("Browsing",
                                               "Some URL BROWSING");
    }
}
```

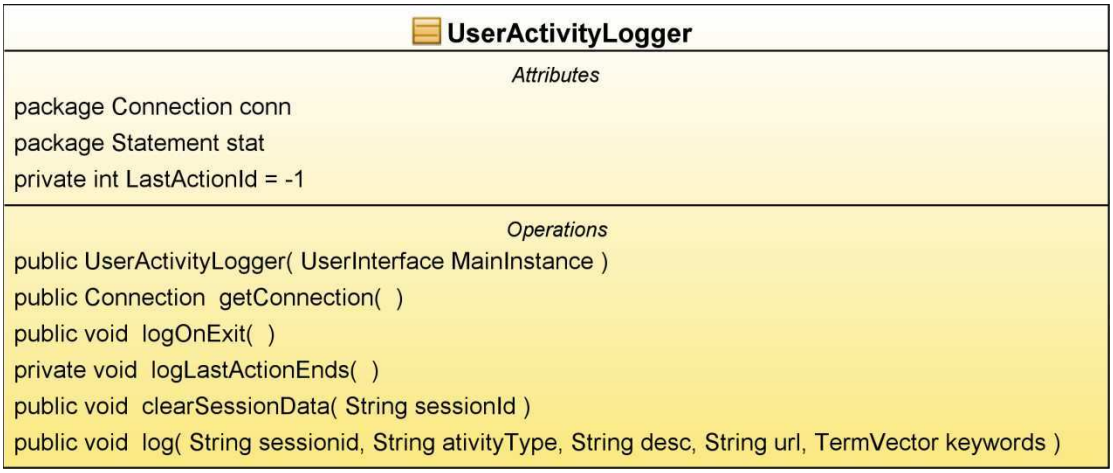
File 'Searching.java'


```
/**
 * How keyword rating is affected in a page was printed or saved
 */
private static final double modForStoredPages = 10.0;
private static final int maxKeywordsUsed = 10;

/**
 * Number of results from Yahoo or Google.
 * In Google number of results retrieved will be
 *   Math.ceil(NrOfResultsFormBaseAPI/8)*8
 */
private final static int NrOfResultsFormBaseAPI = 100;

public static List<String> searchForUrl(
                                String[] keywords, API_TYPE api)
   […] // same as in the implementation of the explicit system
}

public static TermVector findKeywords(String url)
{
   […] // same as in the implementation of the explicit system
}
```

createVector

```
public static TermVector createVector(Connection conn, String sessionId)
{
  Statement stmt = conn.createStatement();
  // find the biggest time span
  String sql = "select max(DateDiff('s', starttime, finishtime))"
      + "  as maxTime, "
      + "  avg(DateDiff('s', starttime, finishtime)) as avgTime "
      + " from SessionActivity "
      + " where sessionid = '" + sessionId.replace("'", "''") + "' "
      + "  and activitytypeId='Browsing' and finishtime is not null"
      + "  and starttime is not null ";
      // (if we only want to get newest keywords) + "
      //and DateDiff('h', starttime, now()) < " + notOlderThanHours;

  System.out.println(sql);
  ResultSet rs = stmt.executeQuery(sql);
  rs.next();
  double maxTime = rs.getDouble(1);
  double avgTime = rs.getDouble(2);


    // now get keywords
  sql = "select t2.kText, t2.rating from SessionActivity as t1 "
        + " INNER join keyword as t2 on t1.sessionactivityid "
        + "= t2.sessionactivityid "
        + " where t1.sessionid = '"
        + sessionId.replace("'", "''")
        + "' and t1.starttime is not null "
        + "and t1.finishtime is not null "
        + "   and DateDiff('s', starttime, finishtime) >= "
        + avgTime + " ";

  rs = stmt.executeQuery(sql);

  HashMap<String, Double> keywords = new HashMap<String, Double>();
  while(rs.next())
     (keywords, rs.getString(1), rs.getDouble(2));

  // printed and saved pages
  sql = "select t2.kText, t2.rating*"+modForStoredPages+" "
      + "from SessionActivity as t1 "
      + " INNER join keyword as t2 on t1.sessionactivityid = t2"
      + ".sessionactivityid "
      + " where t1.sessionid = '"
      + sessionId.replace("'", "''") + "' and (activitytypeId "
      + "='Printing' or activitytypeId='Saving' )";

  rs = stmt.executeQuery(sql);
  while(rs.next())
     addKeyword(keywords, rs.getString(1), rs.getDouble(2));

  // make a vector from n most popular keywords
  TermVector vector = new TermVector();
  for(String s:keywords.keySet())
     vector.put(s, keywords.get(s));

  return vector.topN(maxKeywordsUsed);
}
```

File 'UserActivityLogger'.java




```
UserInterface mainInstance;
Connection conn;
Statement stat;

public UserActivityLogger(UserInterface MainInstance)
{
    mainInstance = MainInstance;

    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    // set this to a MS Access DB you have on your machine
    String filename = "C:/Test/implicit_user_profile.mdb";
    String database = "jdbc:odbc:Driver={Microsoft Access Driver "
                        + "(*.mdb)};DBQ=";
    database += filename.trim()
             + ";DriverID=22;READONLY=true}"; // add on to the end
    // now we can get the connection from the DriverManager
    conn = DriverManager.getConnection(database, "", "");

}
```

logLastActionEnds

```
// Only used to save time when action was ended
private int LastActionId = -1;

private void logLastActionEnds()
{
   // save
   if( LastActionId != -1)
   {
      SimpleDateFormat sdf
                     = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

      String sql = "update SessionActivity set finishtime = '"
                 + sdf.format(new java.util.Date()) + "' "
                 + "where sessionid = '"
                 + mainInstance.getSessionID().replace("'", "''")
                 +"' and sessionactivityid = " + LastActionId+"";
      try {
         stat = conn.createStatement();
         stat.execute(sql);
      } catch (Exception ex) {
         System.out.println(ex.toString());
         ex.printStackTrace(System.err);
      }
      finally
      { try {stat.close(); } catch(SQLException e){} }
   }

   LastActionId = -1;
}
```

log

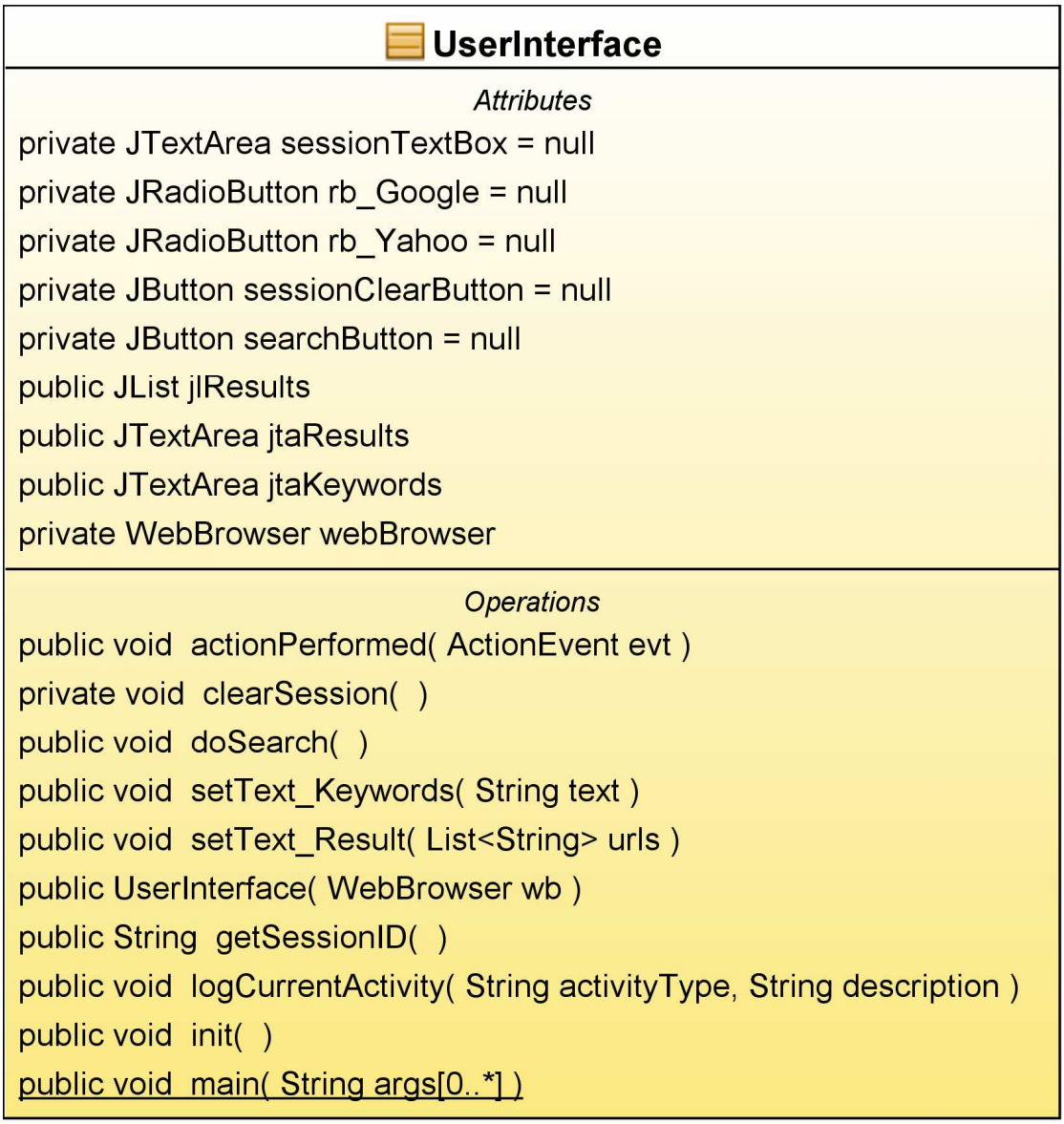
```
public void log(String sessionid, String ativityType, String desc,
               String url, TermVector keywords) {

 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd hh:mm:ss");

 String sql = "insert into SessionActivity(sessionid,activitytypeId,"
            + "starttime,finishtime,parameter) "
            + "values('" + sessionid.replace("'", "''")
            + "','" + ativityType + "','"
            + sdf.format(new java.util.Date())
            + "',NULL,'" + url + "')";

 stat = conn.createStatement();

 // Insert session activiti
 stat.execute(sql); //, Statement.RETURN_GENERATED_KEYS);
 // rs = stat.getGeneratedKeys(); // not suported in this DB
 ResultSet rs = stat.executeQuery("select max(sessionactivityid)"
              + " from SessionActivity");

 rs.next();
 int sessionactivityid = rs.getInt(1);
 rs.close();
```

```
 // remember the action ID
 if(ativityType.equals("Browsing"))
 {
    logLastActionEnds(); // save end time for previous document
    this.LastActionId = sessionactivityid;
 }

 // Insert keywords
 java.util.Iterator it = keywords.termIterator();
 while (it.hasNext())
 {
    String word = (String) it.next();
    double rating = keywords.get(word);

    sql = "insert into keyword(sessionactivityid,"
         + "kText,rating) values(" + sessionactivityid + ",'"
         + word.replace("'", "\'") + "', " + rating + ")";
    stat.execute(sql);
 }
}
```

clearSessionData

```
public void clearSessionData(String sessionId)
{
  stat.execute("delete from keyword where sessionactivityid in "
             + "(select ID from SessionActivity where sessionid='"
             + sessionId.replace("'", "''") + "')");
  stat.execute("delete from SessionActivity where sessionid='"
             + sessionId.replace("'", "''") + "'");
}
```

File 'UserInterface.java'



`actionPerformed`

```
    public void actionPerformed(java.awt.event.ActionEvent evt)
    {
        if(evt.getSource() == sessionClearButton)
        {
            clearSession();
        }
        if(evt.getSource() == searchButton)
        {
            doSearch();
        }
    }
```

doSearch

```
public void doSearch()
{
 TermVector vector = Searching.createVector(
                    ActivitiLogger.getConnection(), getSessionID());

 if(vector == null)
 {
    setText_Keywords("No data (datatabase error)");
    return;
 }

 // display vector
 setText_Keywords(vector.toString());

 if(vector.size() == 0)
 {
    setText_Keywords("No keywords for that id.");
    return;
 }

 // search for keywords in yahoo API
 String[] list = new String[vector.size()];
 Iterator it = vector.termIterator();
 int idx = 0;
 while(it.hasNext()) list[idx++] = it.next().toString();

 // find urls for keywords
 Searching.API_TYPE api = rb_Google.isSelected()
                    ? Searching.API_TYPE.API_GOOGLE :
                      Searching.API_TYPE.API_YAHOO;

 java.util.List<String> urls = Searching.searchForUrl(list, api);

 // sort by similarity to the implicit vector
 RAMSearchEngine rse = new RAMSearchEngine();

 for(String url:urls)
 {
    HTMLDocument objDoc = new HTMLDocument("<html></html>");
    TermVector documentKeywords = Searching.findKeywords(url);
    documentKeywords.normalize();
    rse.addDocument(url, objDoc.getFullContent(), documentKeywords);
 }

 vector.normalize();
 ValueSortedMap map = rse.retrieveDocuments(vector, 20);

 urls.clear();
 Iterator keyIterator = map.keyIterator();
 while(keyIterator.hasNext())
 {
    Object key = keyIterator.next();
    urls.add((String)key);
 }

 // display result
 setText_Result(urls);
}
```

logCurrentActivity

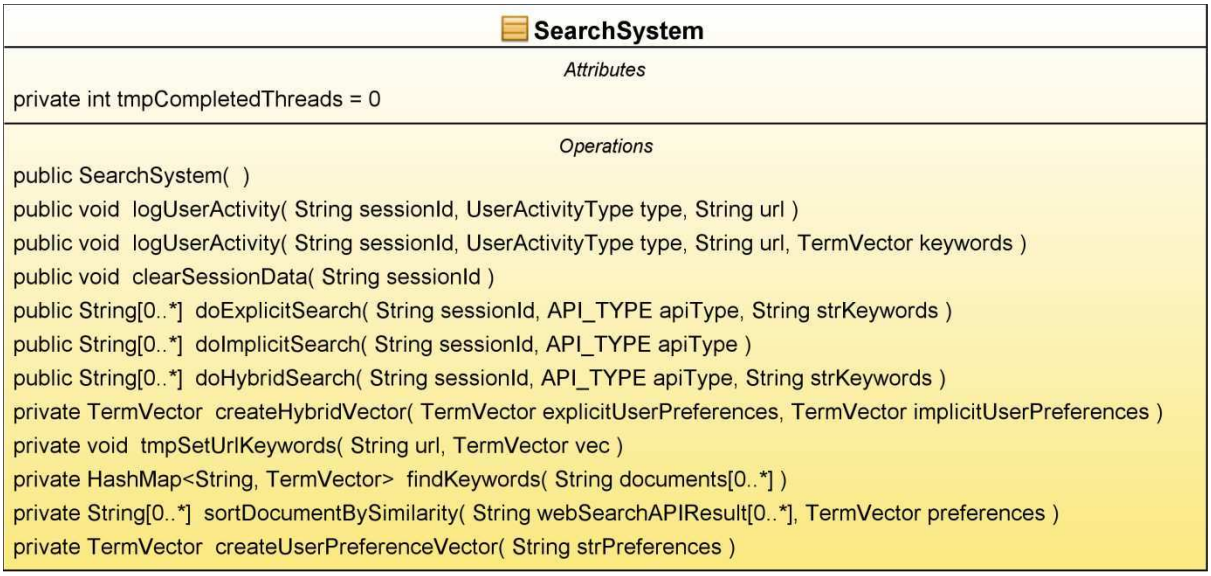
```
public void logCurrentActivity(String activityType,
                               String description)
{
   // find keywords
   TermVector keywords = Searching.findKeywords(webBrowser
                                                .getURL().toString());

   // save to the database
   ActivitiLogger.log(getSessionID(), activityType, description,
                           webBrowser.getURL().toString(), keywords);
}
```

clearSession

```
private void clearSession()
{
    ActivitiLogger.clearSessionData(sessionTextBox.getText());
}
```

Project 'HybridUserProfiles'

File 'SearchSystem.java'




```
UserActivityLogger userActivityLogger;

public enum UserActivityType
{
   Browsing,
   Printing,
   Saving,
   SearchinExplicitly,
   Exit      // when the window is closed
}

public SearchSystem()
{
   userActivityLogger = new UserActivityLogger();
}
```

logUserActivity

```
public void logUserActivity(String sessionId,
                            UserActivityType type, String url)
{
       logUserActivity(sessionId, type, url, null);
}
```

clearSessionData

```
    public void clearSessionData(String sessionId)
    {
        userActivityLogger.clearSessionData(sessionId);
    }
```

logUserActivity

```
public void logUserActivity(String sessionId,
                            UserActivityType type, String url,
                            TermVector keywords)
{
 // find keywords if not given
 if(keywords == null) keywords = Searching.findKeywords(url);

 switch(type)
 {
   case Browsing:
      userActivityLogger.log(sessionId, "Browsing", "",
                                                    url, keywords);
      break;

   case Saving:
      userActivityLogger.log(sessionId, "Saving", "",
                                                    url, keywords);
      break;

   case Printing:
      userActivityLogger.log(sessionId, "Printing", "",
                                                    url, keywords);
      break;

   case SearchinExplicitly:
      userActivityLogger.log(sessionId, "ExplicitSearch", "",
                                                    url, keywords);
      break;

   case Exit:
      userActivityLogger.logOnExit();
      break;
   }
 }
```

doExplicitSearch

```
public List<String> doExplicitSearch(String sessionId,
                        Searching.API_TYPE apiType, String strKeywords)
{
  // replace '=', ',', ' ' to '+'
  strKeywords = strKeywords.replace('=', '+')
                           .replace(',', '+').replace(' ', '+');

  TermVector tvUserPref = createUserPreferenceVector(strKeywords);

  if(tvUserPref == null)
  {
     List<String> result = new LinkedList<String>();
     result.add("[error in keywords formating]");
     return result;
  }

  System.out.println(" User Preference Vector : "
                                          + tvUserPref.toString());

  // search in yahoo
  String[] keywords = strKeywords.split("\\+");
  List<String> webSearchApiResult = Searching
                                  .searchForUrl(keywords, apiType);

  // order results by similarity
  List<String> results = sortDocumentBySimilarity
                                  (webSearchApiResult, tvUserPref);
  return results;
}
```

createUserPreferenceVector

```
private TermVector createUserPreferenceVector(String strPreferences)
{
  try
  {
     return new TermVector(strPreferences);
  }
  catch(Exception e) {System.out.println(e.toString());}

  return null;
}
```

doImplicitSearch

```
public List<String> doImplicitSearch(String sessionId,
                                      Searching.API_TYPE apiType)
{
 TermVector vector = Searching.createVector(
                      userActivityLogger.getConnection(), sessionId);

 if(vector == null)
 {
    List<String> result = new LinkedList<String>();
    result.add("[cannot get implicit keyword from the database]");
    return result;
 }

 // search for keywords in yahoo API
 String[] list = new String[vector.size()];
 Iterator it = vector.termIterator();
 int idx = 0;
 while(it.hasNext())
   list[idx++] = it.next().toString();

 // find urls for keywords
 java.util.List<String> urls = Searching.searchForUrl(list, apiType);

 // sort and limit the number of results to top maxNumberOfResults
 List<String> results = sortDocumentBySimilarity(urls, vector);

 // display result
 return results;
}
```

doHybridSearch

```
public List<String> doHybridSearch(String sessionId,
                      Searching.API_TYPE apiType, String strKeywords)
{
 // replace '=', ',', ' ' to '+'
 strKeywords = strKeywords.replace('=', '+')
                          .replace(',', '+').replace(' ', '+');

 TermVector explicitUserPreferences
                     = createUserPreferenceVector(strKeywords);

 TermVector implicitUserPreferences
                     = Searching.createVector(
                           userActivityLogger.getConnection(),
                           sessionId);

 if(explicitUserPreferences == null)
 {
    List<String> result = new LinkedList<String>();
    result.add("[error in getting explicit keywords]");
    return result;
 }

 if(implicitUserPreferences == null)
 {
    List<String> result = new LinkedList<String>();
    result.add("[cannot get implicit keyword from the database]");
    return result;
 }


 // create hybrid vector
 TermVector combinedPreferences = createHybridVector
               (explicitUserPreferences, implicitUserPreferences);

 // search in base API - only use explicit keywords
 String[] keywords = strKeywords.split("\\+");
 List<String> webSearchApiResult
                      = Searching.searchForUrl(keywords, apiType);

 // order results by similarity to the combined vector
 List<String> results = sortDocumentBySimilarity(webSearchApiResult,
                                                 combinedPreferences);

 return results;
}
```

createHybridVector

```
private TermVector createHybridVector(
                                TermVector explicitUserPreferences,
                                TermVector implicitUserPreferences)
{
 // if implicit vector is empty, then return the explicit vector
 if(implicitUserPreferences.size() == 0)
    return explicitUserPreferences // use topN to create a copy
                        .topN(explicitUserPreferences.size());

 // find the maximum weight from the implicit vector (always first)
 String bestImplicitKeyword = (String)implicitUserPreferences
                                                .termIterator().next();
 double bestImplicitValue = implicitUserPreferences
                                            .get(bestImplicitKeyword);

 // create a combined vector
 TermVector result = new TermVector();

 // add all keywords from explicit results
 //  - the rating will be changed if this keywords exists in the
 //    impicit vector
 result.putAll(explicitUserPreferences);
 result.scaleBy(bestImplicitValue);

 // add all keywords from implicit preferences to keywords
 //from explicit preferences
 Iterator it = implicitUserPreferences.termIterator();
 while(it.hasNext())
 {
    String key = (String)it.next();
    double rating = (explicitUserPreferences.get(key)
                           * bestImplicitValue)
                           + implicitUserPreferences.get(key);
    result.put(key, rating);
 }
 return result;
}
```

sortDocumentBySimilarity

```
private List<String> sortDocumentBySimilarity(List<String>
                          webSearchAPIResult, TermVector preferences)
{

  // find keywords for each of the documents
  HashMap<String, TermVector> documents
                                    = findKeywords(webSearchAPIResult);

  // create internal search engine for similarity comparison
  RAMSearchEngine rse = new RAMSearchEngine();
  for(String url:webSearchAPIResult)
  {
     TermVector documentKeywords = documents.get(url);
     // Searching.findKeywords(url);

     if( documentKeywords != null )
     {
        rse.addDocument(url, "", documentKeywords);
     }
  }

  List<String> results = new LinkedList<String>();

  // get document sorted by similarity to preference vector
  ValueSortedMap vsm = rse.retrieveDocuments(preferences, 20);

  java.util.Iterator itr = vsm.keyIterator();
  while (itr.hasNext()) // add next url
     results.add(itr.next().toString());

  return results;
}
```

retrieveDocuments - from RAMSearchEngine.jave (IGLU library)

```
public ValueSortedMap retrieveDocuments(TermVector vector, int numSimilar)
{
   ValueSortedMap results = new ValueSortedMap();

   // for each doc
   Iterator docIt = idVectorMap.keySet().iterator();
   while(docIt.hasNext())
   {
      // get similarity to vector
      Object thisItem = docIt.next();
      TermVector thisVec = (TermVector)idVectorMap.get(thisItem);
      double similarity = getSimilarityScore(vector, thisVec);

      if(similarity > 0)
         results.put(thisItem, similarity);
   }

   if(numSimilar > 0)
      results.truncateTo(numSimilar);

   return results;
}

// from RAMSearchEngine.java
public double getSimilarityScore(TermVector vector1, TermVector vector2)
{
   double result = 0;

   Iterator it = vector1.termIterator();
   while(it.hasNext())
   {
      String thisTerm = (String)it.next();
      result = result + vector1.get(thisTerm) * vector2.get(thisTerm);
   }

   return result;
}
```

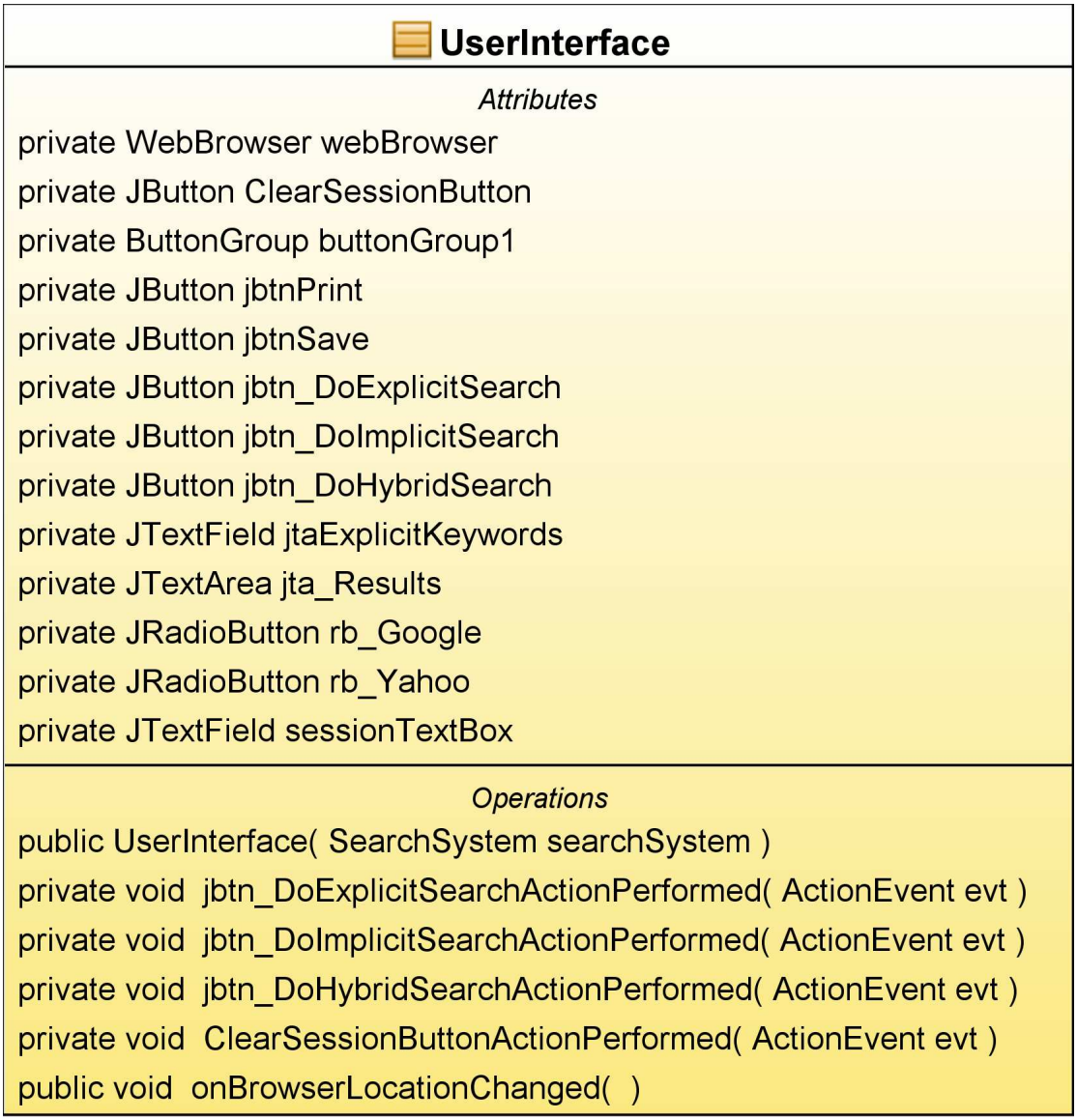
findKeywords

```
private HashMap<String, TermVector> tmpDocumentKeywords;
private int tmpCompletedThreads = 0;

// multithreaded approach to finding keywords in many documents
private HashMap<String, TermVector> findKeywords(List<String> documents)
{
 // prepare table for the results
 tmpDocumentKeywords = new HashMap<String, TermVector>();
 for(String url:documents) tmpDocumentKeywords.put(url, null);

 tmpCompletedThreads = 0;
 for(int i=0; i<documents.size(); i++)
 {
    // start one thread per document
    KeywordsThread t = new KeywordsThread();
    t.url = documents.get(i);
    t.start();
 }

 // wait till all threads are completed
 while(tmpCompletedThreads < documents.size())
    try { Thread.sleep(100); } catch(Exception e){}

 return tmpDocumentKeywords;
}
```

KeywordsThread

```
public class KeywordsThread extends Thread
{
  public String url;

  @Override
  public void run()
  {
    TermVector tv = null;
    try
    {
       tv = Searching.findKeywords(url);
    }
    catch(Exception e) { }
    finally
    {
       synchronized (tmpDocumentKeywords)
       {
          tmpDocumentKeywords.remove(url);
          tmpDocumentKeywords.put(url, vec);
          tmpCompletedThreads++;
       }
    }
  }
}
```

1.1.1.1 File 'UserInterface.java'




```
private WebBrowser webBrowser;

private SearchSystem searchSystem;

public static void main(String[] args)
{

    SearchSystem system = new SearchSystem();

    UserInterface ui = new UserInterface(system);
    ui.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    ui.pack();
    ui.setVisible(true);
}
```

DoImplicitSearchActionPerformed

```
private void jbtn_DoImplicitSearchActionPerformed(ActionEvent evt)
{
    // search
    List<String> result = this.searchSystem.doImplicitSearch(
                                    sessionTextBox.getText(),
                                    rb_Yahoo.isSelected()
                                       ? Searching.API_TYPE.API_YAHOO
                                       : Searching.API_TYPE.API_GOOGLE);
   […]
}
```

jbtn_DoHybridSearchActionPerformed

```
private void jbtn_DoHybridSearchActionPerformed(ActionEvent evt)
{
        // remove old result
        this.jta_Results.setText("");

        // search
        String keywords = this.jtaExplicitKeywords.getText();
        List<String> result = this.searchSystem.doHybridSearch
              (sessionTextBox.getText(),rb_Yahoo.isSelected()?
                 Searching.API_TYPE.API_YAHOO:Searching.API_TYPE
                 .API_GOOGLE,  keywords);

        // display the result
        […]

        //-- store search history in the database
        this.searchSystem.logUserActivity(sessionTextBox.getText(),
              SearchSystem.UserActivityType.SearchinExplicitly, "",
              Searching.buildVectorFromString(keywords));
    }
}
```

File ‘Searching.java’

The Searching class is part of the mediation framework and for the hybrid system it is the same as for the implicit or explicit systems.

1.1.1.2 File ‘UserActivityLogger.java’

The UserActivityLogger class is part of the mediation framework and for the hybrid system it is the same as for the implicit or explicit systems.

Appendix B

List of Keywords provided by Users during systems Experiments

Users	Keywords
User 1	knowledge interchange format ontology language
User 2	Coventry university history (Regarding University Courses)
User 3	Cloud computing
User 4	Digital Library
User 5	setup saltwater aquarium
User 6	Artificial Neural network
User 7	information technology
User 8	parallel computing and distributed computing
User 9	body area network for health monitoring
User 10	Connectionist Natural Language Processing
User 11	lean management methodology
User 12	Grid Computing
User 12	hsbc saving account
User 14	strategic marketing
User 15	database design
User 16	Business accounting
User 17	learn French
User 18	orange (as a mobile)
User 19	supply chain configuration
User 20	arm microprocessor
User 21	communication engineering
User 22	knowledge management
User 23	Operating system
User 24	Collaborative computing
User 25	Matlab programming
User 26	Resource based view (RSV)
User 27	Computer architecture
User 28	Java programming advantages
User 29	Solving business problems
User 30	genetic engineering

Experiment 1 – Precision results

Google Precision

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	1	3	3	13	0	20	0.16
2	4	0	1	15	0	20	0.21
3	7	9	2	2	0	20	0.60
4	14	3	1	2	0	20	0.79
5	10	2	2	6	0	20	0.58
6	7	6	2	4	1	20	0.53
7	4	5	1	10	0	20	0.34
8	11	9	0	0	0	20	0.78
9	14	4	1	0	1	20	0.81
10	8	4	3	5	0	20	0.54
11	8	8	3	1	0	20	0.64
12	10	5	0	5	0	20	0.63
13	8	3	1	8	0	20	0.49
14	6	4	2	8	0	20	0.43
15	11	3	4	2	0	20	0.68
16	4	2	1	11	2	20	0.26
17	12	4	0	4	0	20	0.70
18	1	1	0	18	0	20	0.08
19	8	6	0	6	0	20	0.55
20	7	3	1	9	0	20	0.44
21	0	4	3	13	0	20	0.14
22	6	2	4	8	0	20	0.40
23	5	5	2	8	0	20	0.40
24	6	2	3	6	3	20	0.39
25	9	1	2	8	0	20	0.50
26	12	6	1	0	1	20	0.76
27	3	3	8	6	0	20	0.33
28	5	3	1	11	0	20	0.34
29	6	6	2	6	0	20	0.48
30	7	6	4	3	0	20	0.55
Total	**214**	**122**	**58**	**198**	**8**	**600**	**0.48**

Yahoo Precision

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	1	4	5	10	0	20	0.21
2	2	0	2	16	0	20	0.13
3	5	5	5	5	0	20	0.44
4	6	4	5	5	0	20	0.46
5	10	4	1	5	0	20	0.61
6	12	5	3	0	0	20	0.76
7	4	5	3	8	0	20	0.36
8	14	6	0	0	0	20	0.85
9	8	2	5	4	1	20	0.51
10	9	2	3	5	1	20	0.54
11	11	1	4	4	0	20	0.63
12	10	6	0	4	0	20	0.65
13	16	2	1	0	1	20	0.86
14	7	3	1	8	1	20	0.44
15	11	4	1	3	1	20	0.66
16	6	2	2	9	1	20	0.38
17	8	4	0	8	0	20	0.50
18	2	2	0	16	0	20	0.15
19	6	2	0	11	1	20	0.35
20	11	3	4	2	0	20	0.68
21	3	2	2	13	0	20	0.23
22	10	2	2	6	0	20	0.58
23	9	3	5	3	0	20	0.59
24	6	2	1	8	3	20	0.36
25	8	2	3	6	1	20	0.49
26	13	2	0	4	1	20	0.70
27	6	0	7	7	0	20	0.39
28	12	3	2	3	0	20	0.70
29	7	3	1	9	0	20	0.44
30	6	5	5	4	0	20	0.49
Total	**239**	**90**	**73**	**186**	**12**	**600**	**0.50**

Explicit System Precision with Google

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	**1**	3	5	10	1	20	0.19
2	**7**	0	2	11	0	20	0.38
3	**5**	7	4	4	0	20	0.48
4	**13**	4	1	2	0	20	0.76
5	**8**	1	3	8	0	20	0.46
6	**11**	5	3	1	0	20	0.71
7	**4**	5	1	10	0	20	0.34
8	**9**	11	0	0	0	20	0.73
9	14	3	3	0	0	20	0.81
10	8	9	2	1	0	20	0.65
11	9	8	1	2	0	20	0.66
12	8	4	3	4	1	20	0.54
13	6	6	2	6	0	20	0.48
14	7	5	3	5	0	20	0.51
15	10	4	2	4	0	20	0.63
16	5	1	0	14	0	20	0.28
17	12	1	0	7	0	20	0.63
18	1	0	0	19	0	20	0.05
19	11	4	0	5	0	20	0.65
20	11	4	0	5	0	20	0.65
21	3	5	5	7	0	20	0.34
22	4	3	2	11	0	20	0.30
23	2	6	4	8	0	20	0.30
24	6	5	3	6	0	20	0.46
25	7	4	2	7	0	20	0.48
26	14	2	0	4	0	20	0.75
27	2	1	7	10	0	20	0.21
28	7	2	3	8	0	20	0.44
29	9	7	2	2	0	20	0.65
30	6	4	7	3	0	20	0.49
Total	**220**	**124**	**70**	**184**	**2**	**600**	**0.50**

Explicit System Precision with Yahoo

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	3	**4**	3	10	0	20	0.29
2	2	**1**	1	16	0	20	0.14
3	9	**2**	4	5	0	20	0.55
4	7	**7**	2	3	1	20	0.55
5	9	**7**	0	4	0	20	0.63
6	12	**4**	2	2	0	20	0.73
7	5	**11**	0	4	0	20	0.53
8	9	**7**	4	0	0	20	0.68
9	8	4	4	4	0	20	0.55
10	9	4	2	5	0	20	0.58
11	9	8	2	1	0	20	0.68
12	10	5	2	3	0	20	0.65
13	18	0	0	2	0	20	0.90
14	7	4	6	3	0	20	0.53
15	12	2	3	3	0	20	0.69
16	9	6	3	2	0	20	0.64
17	11	4	0	5	0	20	0.65
18	1	0	0	19	0	20	0.05
19	9	4	0	7	0	20	0.55
20	12	0	4	4	0	20	0.65
21	6	5	1	8	0	20	0.44
22	14	1	4	1	0	20	0.78
23	10	3	3	4	0	20	0.61
24	8	1	3	8	0	20	0.46
25	11	6	2	1	0	20	0.73
26	13	2	0	5	0	20	0.70
27	6	4	6	4	0	20	0.48
28	14	0	1	5	0	20	0.71
29	6	3	1	10	0	20	0.39
30	9	4	4	3	0	20	0.60
Total	**268**	**113**	**67**	**151**	**1**	**600**	**0.57**

Implicit System Precision with Google

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	2	2	2	13	1	20	0.18
2	3	0	3	14	0	20	0.19
3	9	1	3	7	0	20	0.51
4	4	6	3	7	0	20	0.39
5	4	4	2	10	0	20	0.33
6	7	6	4	3	0	20	0.55
7	12	4	0	4	0	20	0.70
8	10	7	2	0	1	20	0.70
9	9	2	3	6	0	20	0.54
10	13	3	2	2	0	20	0.75
11	10	1	0	9	0	20	0.53
12	15	4	0	1	0	20	0.85
13	6	3	1	9	1	20	0.39
14	8	0	0	12	0	20	0.40
15	9	3	3	5	0	20	0.56
16	2	2	1	14	1	20	0.16
17	13	5	0	2	0	20	0.78
18	14	1	4	1	0	20	0.78
19	15	2	1	2	0	20	0.81
20	9	3	1	7	0	20	0.54
21	9	3	3	5	0	20	0.56
22	9	2	2	7	0	20	0.53
23	3	3	0	14	0	20	0.23
24	6	0	3	11	0	20	0.34
25	6	6	4	4	0	20	0.50
26	6	2	1	11	0	20	0.36
27	0	2	10	8	0	20	0.18
28	9	4	1	6	0	20	0.56
29	1	7	0	12	0	20	0.23
30	5	5	6	4	0	20	0.45
Total	**228**	**93**	**65**	**210**	**4**	**600**	**0.48**

Implicit System Precision with Yahoo

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	2	2	4	12	0	20	0.20
2	2	1	1	16	0	20	0.14
3	9	6	3	2	0	20	0.64
4	12	1	2	5	0	20	0.65
5	6	7	2	5	0	20	0.50
6	10	2	4	4	0	20	0.60
7	11	4	4	1	0	20	0.70
8	1	12	5	2	0	20	0.41
9	9	5	1	5	0	20	0.59
10	11	3	2	4	0	20	0.65
11	7	6	0	7	0	20	0.50
12	13	5	2	0	0	20	0.80
13	6	0	1	12	1	20	0.31
14	6	1	3	10	0	20	0.36
15	2	0	2	15	1	20	0.13
16	1	1	4	14	0	20	0.13
17	9	5	3	3	0	20	0.61
18	12	2	2	4	0	20	0.68
19	9	4	2	5	0	20	0.58
20	7	3	2	8	0	20	0.45
21	0	1	4	15	0	20	0.08
22	9	4	4	3	0	20	0.60
23	10	1	4	5	0	20	0.58
24	4	1	6	9	0	20	0.30
25	3	5	7	5	0	20	0.36
26	1	5	0	14	0	20	0.18
27	1	0	7	12	0	20	0.14
28	9	1	0	9	1	20	0.48
29	2	8	0	10	0	20	0.30
30	4	5	2	9	0	20	0.35
Total	**188**	**101**	**83**	**225**	**3**	**600**	**0.43**

Hybrid System Precision with Google

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	1	3	5	10	1	20	0.19
2	5	1	2	12	0	20	0.30
3	7	6	4	3	0	20	0.55
4	14	3	1	2	0	20	0.79
5	9	6	0	5	0	20	0.60
6	11	5	3	1	0	20	0.71
7	4	5	1	10	0	20	0.34
8	8	11	0	0	1	20	0.68
9	13	2	4	0	1	20	0.75
10	8	5	4	3	0	20	0.58
11	9	7	1	3	0	20	0.64
12	8	5	2	5	0	20	0.55
13	7	4	2	7	0	20	0.48
14	7	5	2	6	0	20	0.50
15	10	4	2	4	0	20	0.63
16	4	1	0	15	0	20	0.23
17	15	1	0	4	0	20	0.78
18	3	0	0	17	0	20	0.15
19	13	4	0	3	0	20	0.75
20	11	4	0	5	0	20	0.65
21	2	5	4	9	0	20	0.28
22	6	2	1	11	0	20	0.36
23	2	6	4	8	0	20	0.30
24	7	4	3	6	0	20	0.49
25	7	4	2	7	0	20	0.48
26	15	2	0	3	0	20	0.80
27	2	1	8	9	0	20	0.23
28	6	3	3	8	0	20	0.41
29	12	5	2	1	0	20	0.75
30	7	4	6	3	0	20	0.53
Total	**233**	**118**	**66**	**180**	**3**	**600**	**0.51**

Hybrid System Precision with Yahoo

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	3	5	5	7	0	20	0.34
2	2	1	1	16	0	20	0.14
3	10	3	3	4	0	20	0.61
4	7	6	2	4	1	20	0.53
5	8	1	3	8	0	20	0.46
6	14	4	1	1	0	20	0.81
7	8	10	0	2	0	20	0.65
8	10	5	4	0	1	20	0.68
9	8	4	6	2	0	20	0.58
10	10	4	2	4	0	20	0.63
11	11	6	1	2	0	20	0.71
12	11	5	2	2	0	20	0.70
13	16	0	1	3	0	20	0.81
14	8	3	6	3	0	20	0.55
15	14	1	2	3	0	20	0.75
16	9	6	3	2	0	20	0.64
17	15	0	2	3	0	20	0.78
18	2	0	0	18	0	20	0.10
19	9	4	0	7	0	20	0.55
20	12	0	4	4	0	20	0.65
21	6	6	1	7	0	20	0.46
22	15	1	3	1	0	20	0.81
23	10	3	3	4	0	20	0.61
24	8	1	3	8	0	20	0.46
25	11	6	2	1	0	20	0.73
26	15	2	0	3	0	20	0.80
27	6	4	6	4	0	20	0.48
28	13	1	1	5	0	20	0.69
29	5	4	0	11	0	20	0.35
30	11	4	4	1	0	20	0.70
Total	**287**	**100**	**71**	**140**	**2**	**600**	**0.59**

Experiment 1 –Recall results

Google and Yahoo Recall

User	Google Score	Yahoo Score	Duplicated Document Score	Google recall	Yahoo Recall
1	6.5	8.5	4	0.59	0.77
2	8.5	5	2	0.74	0.43
3	24	17.5	7.5	0.71	0.51
4	31.5	18.5	1	0.64	0.38
5	23	24.5	9.5	0.61	0.64
6	21	30.5	5	0.45	0.66
7	13.5	14.5	3	0.54	0.58
8	31	34	2	0.49	0.54
9	32.5	20.5	4	0.66	0.42
10	21.5	21.5	4	0.55	0.55
11	25.5	25	8	0.60	0.59
12	25	26	16	0.71	0.74
13	19.5	34.5	4	0.39	0.69
14	17	17.5	4.5	0.57	0.58
15	27	26.5	15	0.70	0.69
16	10.5	15	2	0.45	0.64
17	28	20	12	0.78	0.56
18	3	6	3	0.50	1.00
19	22	14	6	0.73	0.47
20	17.5	27	7.5	0.47	0.73
21	5.5	9	0.5	0.39	0.64
22	16	23	6.5	0.49	0.71
23	16	23.5	7.5	0.50	0.73
24	15.5	14.5	1.5	0.54	0.51
25	20	19.5	7	0.62	0.60
26	30.5	28	15	0.70	0.64
27	13	15.5	2	0.49	0.58
28	13.5	28	3	0.35	0.73
29	19	17.5	4.5	0.59	0.55
30	22	19.5	7	0.64	0.57
Total	**579**	**604.5**	**174.5**	**0.57**	**0.60**

Explicit system Recall

User	Google Score	Yahoo Score	Duplicated Document Score	Google recall	Yahoo Recall
1	7.5	11.5	6	0.58	0.88
2	15	5.5	4	0.91	0.33
3	19	22	3	0.50	0.58
4	30.5	22	6	0.66	0.47
5	18.5	25	6	0.49	0.67
6	28.5	29	6.5	0.56	0.57
7	13.5	21	1	0.40	0.63
8	29	27	5	0.57	0.53
9	32.5	22	8.5	0.71	0.48
10	26	23	0	0.53	0.47
11	26.5	27	0	0.50	0.50
12	21.5	26	10	0.57	0.69
13	19	36	4	0.37	0.71
14	20.5	21	1	0.51	0.52
15	25	27.5	10	0.59	0.65
16	11	25.5	4	0.34	0.78
17	25	26	10	0.61	0.63
18	2	2	2	1.00	1.00
19	26	22	1	0.55	0.47
20	26	26	6	0.57	0.57
21	13.5	17.5	0	0.44	0.56
22	12	31	5	0.32	0.82
23	12	24.5	3.5	0.36	0.74
24	18.5	18.5	4	0.56	0.56
25	19	29	5	0.44	0.67
26	30	28	14	0.68	0.64
27	8.5	19	3	0.35	0.78
28	17.5	28.5	4	0.42	0.68
29	26	15.5	4	0.69	0.41
30	19.5	24	0.5	0.45	0.56
Total	**599**	**682.5**	**137**	**0.52**	**0.60**

Implicit system Recall

User	Google Score	Yahoo Score	Duplicated Document Score	Google recall	Yahoo Recall
1	7	8	3	0.58	0.67
2	7.5	5.5	2.5	0.71	0.52
3	20.5	25.5	2	0.47	0.58
4	15.5	26	2	0.39	0.66
5	13	20	2	0.42	0.65
6	22	24	0	0.48	0.52
7	28	28	0	0.50	0.50
8	28	16.5	0	0.63	0.37
9	21.5	23.5	2	0.50	0.55
10	30	26	0	0.54	0.46
11	21	20	0	0.51	0.49
12	34	32	6	0.57	0.53
13	15.5	12.5	1.5	0.58	0.47
14	16	14.5	0	0.52	0.48
15	22.5	5	0	0.82	0.18
16	6.5	5	2	0.68	0.53
17	31	24.5	7	0.64	0.51
18	31	27	5	0.58	0.51
19	32.5	23	2	0.61	0.43
20	21.5	18	4	0.61	0.51
21	22.5	3	0	0.88	0.12
22	21	24	2	0.49	0.56
23	9	23	2	0.30	0.77
24	13.5	12	0	0.53	0.47
25	20	14.5	2	0.62	0.45
26	14.5	7	0	0.67	0.33
27	7	5.5	0.5	0.58	0.46
28	22.5	19	1	0.56	0.47
29	9	12	1	0.45	0.60
30	18	14	6	0.69	0.54
Total	**581.5**	**518.5**	**55.5**	**0.56**	**0.50**

Hybrid system Recall

User	Google Score	Yahoo Score	Duplicated Document Score	Google recall	Yahoo Recall
1	7.5	13.5	6	0.50	0.90
2	12	5.5	1	0.73	0.33
3	22	24.5	2	0.49	0.55
4	31.5	21	6	0.68	0.45
5	24	18.5	8	0.70	0.54
6	28.5	32.5	8.5	0.54	0.62
7	13.5	26	3	0.37	0.71
8	27	27	3	0.53	0.53
9	30	23	6.5	0.65	0.49
10	23	25	3	0.51	0.56
11	25.5	28.5	0	0.47	0.53
12	22	28	12	0.58	0.74
13	19	32.5	4	0.40	0.68
14	20	22	1	0.49	0.54
15	25	30	12	0.58	0.70
16	9	25.5	4	0.30	0.84
17	31	31	14	0.65	0.65
18	6	4	4	1.00	0.67
19	30	22	1	0.59	0.43
20	26	26	6	0.57	0.57
21	11	18.5	0	0.37	0.63
22	14.5	32.5	9	0.38	0.86
23	12	24.5	3.5	0.36	0.74
24	19.5	18.5	4	0.57	0.54
25	19	29	5	0.44	0.67
26	32	32	14	0.64	0.64
27	9	19	3	0.36	0.76
28	16.5	27.5	5	0.42	0.71
29	30	14	4	0.75	0.35
30	21	28	4.5	0.47	0.63
Total	**617**	**709.5**	**157**	**0.53**	**0.61**

Experiment 2 – Precision results

Implicit System Precision with Google

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	9	7	2	2	0	20	0.63
2	14	2	1	3	0	20	0.76
3	6	3	5	6	0	20	0.44
4	12	3	1	4	0	20	0.69
5	12	6	2	0	0	20	0.78
6	11	7	1	1	0	20	0.74
7	5	7	4	4	0	20	0.48
8	4	4	6	6	0	20	0.38
9	3	9	1	7	0	20	0.39
10	12	2	5	1	0	20	0.71
11	5	7	3	5	0	20	0.46
12	8	7	4	1	0	20	0.63
13	7	8	0	5	0	20	0.55
14	10	5	2	2	1	20	0.65
15	14	3	1	2	0	20	0.79
16	10	5	3	2	0	20	0.66
17	10	6	2	2	0	20	0.68
18	12	3	0	5	0	20	0.68
19	11	3	2	4	0	20	0.65
20	4	8	2	6	0	20	0.43
21	10	1	1	7	1	20	0.54
22	4	5	1	9	1	20	0.34
23	14	5	0	1	0	20	0.83
24	15	2	2	1	0	20	0.83
25	9	4	3	4	0	20	0.59
26	9	3	2	6	0	20	0.55
27	3	5	0	12	0	20	0.28
28	7	5	4	4	0	20	0.53
29	9	6	1	4	0	20	0.61
30	1	7	0	12	0	20	0.23
Total	**260**	**148**	**61**	**128**	**3**	**600**	**0.58**

Implicit System Precision with Yahoo

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	7	7	5	1	0	20	0.45
2	10	3	2	5	0	20	0.6
3	10	2	7	1	0	20	0.638
4	11	4	1	4	0	20	0.663
5	15	3	2	0	0	20	0.85
6	5	5	2	8	0	20	0.4
7	5	1	8	6	0	20	0.375
8	4	6	6	4	0	20	0.425
9	5	8	5	2	0	20	0.513
10	7	7	2	4	0	20	0.55
11	7	5	6	2	0	20	0.55
12	14	6	0	0	0	20	0.85
13	7	9	2	2	0	20	0.6
14	7	8	4	1	0	20	0.6
15	11	4	1	4	0	20	0.663
16	7	3	2	8	0	20	0.45
17	1	6	9	4	0	20	0.313
18	1	6	2	11	0	20	0.225
19	7	6	5	2	0	20	0.563
20	7	5	0	8	0	20	0.475
21	9	1	1	8	1	20	0.488
22	3	1	4	12	0	20	0.225
23	11	3	3	3	0	20	0.663
24	13	2	2	3	0	20	0.725
25	1	3	4	12	0	20	0.175
26	9	5	4	2	0	20	0.625
27	10	1	4	5	0	20	0.575
28	3	6	7	4	0	20	0.388
29	9	3	0	7	1	20	0.525
30	2	10	0	8	0	20	0.35
Total	**218**	**139**	**100**	**141**	**2**	**600**	**0.52**

Hybrid System Precision with Google

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	8	3	4	5	0	20	0.49
2	6	4	3	7	0	20	0.44
3	11	3	4	2	0	20	0.68
4	14	5	1	0	0	20	0.84
5	14	2	3	1	0	20	0.79
6	9	11	0	0	0	20	0.73
7	12	1	4	3	0	20	0.68
8	14	1	5	0	0	20	0.79
9	18	2	0	0	0	20	0.95
10	13	2	3	2	0	20	0.74
11	10	8	1	1	0	20	0.71
12	7	5	3	5	0	20	0.51
13	7	5	3	5	0	20	0.51
14	13	4	3	0	0	20	0.79
15	14	5	1	0	0	20	0.84
16	16	3	1	0	0	20	0.89
17	12	6	0	2	0	20	0.75
18	13	4	1	2	0	20	0.76
19	11	2	7	0	0	20	0.69
20	12	2	4	2	0	20	0.70
21	7	5	2	6	0	20	0.50
22	7	2	0	11	0	20	0.40
23	15	3	0	2	0	20	0.83
24	5	3	0	11	0	19	0.34
25	2	7	4	7	0	20	0.33
26	7	3	1	9	0	20	0.44
27	2	7	4	7	0	20	0.33
28	7	5	2	6	0	20	0.50
29	7	4	3	6	0	20	0.49
30	12	6	2	0	0	20	0.78
Total	**305**	**123**	**69**	**102**	**1**	**600**	**0.64**

Hybrid System Precision with Yahoo

Users	Relevant	Less relevant	URLs	Irrelevant	Page can't be accessed	No. of sites evaluated	Precision
1	11	4	2	3	0	20	0.68
2	6	4	1	9	0	20	0.41
3	9	4	5	1	1	20	0.61
4	8	6	2	3	1	20	0.58
5	14	5	0	1	0	20	0.83
6	15	3	0	2	0	20	0.83
7	11	4	1	4	0	20	0.66
8	10	7	3	0	0	20	0.71
9	16	3	0	1	0	20	0.88
10	9	6	2	3	0	20	0.63
11	8	7	2	3	0	20	0.60
12	9	4	5	2	0	20	0.61
13	6	6	4	4	0	20	0.50
14	8	4	7	1	0	20	0.59
15	11	4	0	5	0	20	0.65
16	11	5	2	2	0	20	0.70
17	12	3	3	2	0	20	0.71
18	18	0	1	1	0	20	0.91
19	18	1	1	0	0	20	0.94
20	15	1	4	0	0	20	0.83
21	16	1	1	2	0	20	0.84
22	10	5	3	2	0	20	0.66
23	15	2	2	1	0	20	0.83
24	2	5	0	13	0	20	0.23
25	7	7	1	5	0	20	0.54
26	15	2	3	0	0	20	0.84
27	11	2	3	4	0	20	0.64
28	12	4	2	1	0	19	0.76
29	13	2	1	4	0	20	0.71
30	5	5	1	9	0	20	0.39
Total	**331**	**116**	**62**	**88**	**3**	**600**	**0.67**

Experiment 2 – Recall results

Implicit System Recall

User	Google Score	Yahoo Score	Duplicated Document Score	Google recall	Yahoo Recall
1	26	23.5	8	0.63	0.57
2	30.5	24	6.5	0.64	0.50
3	17.5	25.5	12	0.56	0.82
4	27.5	26.5	2	0.53	0.51
5	31	34	5.5	0.52	0.57
6	29.5	16	3	0.69	0.38
7	19	15	3	0.61	0.48
8	15	17	1.5	0.49	0.56
9	15.5	20.5	2	0.46	0.60
10	28.5	22	0	0.56	0.44
11	18.5	22	12	0.65	0.77
12	25	34	5	0.46	0.63
13	22	24	2	0.50	0.55
14	26	24	4	0.57	0.52
15	31.5	26.5	2	0.56	0.47
16	26.5	18	0	0.60	0.40
17	27	12.5	2	0.72	0.33
18	27	9	3	0.82	0.27
19	26	22.5	0	0.54	0.46
20	17	19	1	0.49	0.54
21	21.5	19.5	0	0.52	0.48
22	13.5	9	2	0.66	0.44
23	33	26.5	8	0.64	0.51
24	33	29	5	0.58	0.51
25	23.5	7	0	0.77	0.23
26	22	25	2	0.49	0.56
27	11	23	2	0.34	0.72
28	21	15.5	3	0.63	0.46
29	24.5	21	1	0.55	0.47
30	9	14	0	0.39	0.61
Total	**698.5**	**625**	**97.5**	**0.57**	**0.51**

Hybrid System Recall

User	Google Score	Yahoo Score	Duplicated Document Score	Google recall	Yahoo Recall
1	21	27	8.5	0.53	0.68
2	17.5	16.5	6	0.63	0.59
3	27	24.5	13	0.70	0.64
4	33.5	23	8	0.69	0.47
5	31.5	33	10	0.58	0.61
6	29	33	12	0.58	0.66
7	27	26.5	4	0.55	0.54
8	31.5	28.5	8.5	0.61	0.55
9	38	35	11	0.61	0.56
10	29.5	25	8	0.63	0.54
11	28.5	24	9	0.66	0.55
12	20.5	24.5	12.5	0.63	0.75
13	20.5	20	12	0.72	0.70
14	31.5	23.5	16.5	0.82	0.61
15	33.5	26	1	0.57	0.44
16	35.5	28	10	0.66	0.52
17	30	28.5	14	0.67	0.64
18	30.5	36.5	10.5	0.54	0.65
19	27.5	37.5	0	0.42	0.58
20	28	33	5	0.50	0.59
21	20	33.5	0	0.37	0.63
22	16	26.5	4	0.42	0.69
23	33	33	15	0.65	0.65
24	13	9	6	0.81	0.56
25	13	21.5	0	0.38	0.62
26	17.5	33.5	10	0.43	0.82
27	13	25.5	3.5	0.37	0.73
28	20	29	5	0.45	0.66
29	19.5	28.5	5	0.45	0.66
30	31	15.5	0	0.67	0.33
Total	**767.5**	**809**	**228**	**0.57**	**0.60**