WiRe - ”What Is the RElation?” Finding the best way to present search engine results

Filipe Brito
Instituto Superior Técnico & INESC-ID
Universidade Técnica de Lisboa
Av. Rovisco Pais, 1
1049-001 Lisboa
Portugal
filipe.brito@ist.utl.pt

ABSTRACT

Since there’s increasingly more information being put on the web, it has resulted in the need to produce more refined searches that reflect the users personal interests. Allowing to effectively relevant information of their interests. There are several techniques for capturing this information and use it as an aid on web searches. Traditionally, the used techniques fall into: monitorization of user behavior in applications; use of browsing history; queries and click logs; links followed/clicks; physical location; collaborative filtering and semantic knowledge of words. This work, not only explores the use of social data, since it wasn’t explored in the state of art, but also uses other types of personal information, like: e-mails, personal documents, bookmarks and Calendar events. In order to explore social information, we intend to capture a range of information (for example: posts, tweets and comments) which together explain the users’ preferences or interests. Afterwards we process that information according to some metrics, and deduce which keywords best represent the users’ current interests, so that they will be used as an aid on internet searches, to achieve personalization. Finally, experimental tests with users would be carried in order to evaluate the significance, precision and quality of the obtained search results, comparing them with the ones obtained on the baseline - Google search.

Keywords

Results personalization; search engines; user interests; context; personal information; social networks.

INTRODUCTION

With the advent of the internet and its underlying services and increasing popularity, it’s well known that people use and enjoy the available information on the internet. However, many times we need to know whose services are available nowadays, or which set of information (be it related to the personal or professional scope) is at our disposal according to our personal interests, and for that we need to make searches on search engines. As from the results that they retrieve we try to find the ones (the links) that meet our preferences.

It is certain that search engines satisfy the user needs for search of information, altough the biggest problem is that they don’t distinguish people from each other, their likes or preferences [1]. The youngsters are particularly affected, since search engines ignore the reading difficulty of documents and the reading proeficiency of their users [5]. Allied to these facts, it is people lack of sensibility to better express their intentions in the form of queries [11]. This fact makes the task of searching, analysing and finding the relevant results, more expensive in terms of time and in the worst scenarios we don’t even find relevant results at all. It’s very important to notice the time that people expend on search for information. They don’t like to waste a lot of time on this task, mostly only analyses the first two pages of results [9]. If they don’t find what they were looking for, they’d rather give up searching, instead of proceeding to the remaining pages.

The difficulty for obtaining correct results is more clearer when the search is formed by ambiguous words. For instance, searching for “**jaguar**”, we cannot know in advance if the search refers to the animal or to the car brand. Using the user personal information, we may be able to identify which of these categories is relevant for the user, upon the search. So, we can retrieve the more relevant results for their interests.

A more complex example will be to imaginate that **Kingson** is the search query. It can be the peripherals brand or informatic components, as well as of several locations. If, using the personal information allow us to investigate that the brand has more importance for the user, then we expect that the results are related to the brand. On the other hand, we want results about locations. However, there is many locations for “Kingston”. This way, which one of them is more relevant to the user? To answer this question we rely to a subset of the user personal information - the social one. If the user has contacts that have travelled to one of those different places, for instance Kingston (Tennessee), then we can disambiguate these places and assume that the user is interested on a specific location.

There's also another kind of prolbems, not only related with the ambiguity in the words, but also with the topic that a user refers to, within a specific domain. The social information is also important to solve this problem. For instance, search for "Best **brands**" of 2013 and the user did some *posts, tweets* or comments about informatic/computers, then he pretends that the primary results will be about brands belonging to the computer or technologies scope (Intel, Logitech, amongst others).

So, in order to solve these problems, various techniques can be used to acquire automatically the user preferences, so that without their direct intervention, we can customize their search results.

Personalize information according to the user preferences and profile, it's the hardest tasks to accomplish in the area of "Information Retrieval" [3]. There's a subset of systems known as *Information Management Systems* (IMAs) [4], whose function is to automatically find material related to the user preferences (serving as intermediate between the user and information retrieval systems), through the observation of their interactions with applications, allowing this way to antecipate in advance which information meets user needs. Another set of systems are the *Recommender systems* [10], whose job is to recommend items to the users, using their preferences or profiles as a basis. There's two approacges to build systems like those: *Content-based systems* (CBS) [10] and *collaborative filtering-based systems* (CFS) [10]. The first one recommends items, comparing their contents with representations of contents that user is interested. The second, recommends through collaboration between people, wherein each one filters through the register of their reactions to the documents they've read. These systems can be used on various domains, for instance on web page and articles recommendation. Belonging to this type we have some software agents, these are defined as: *Software agents are essentially software components with specific traits such as proactiveness, reactiveness, autonomy, and sociability* [1]. As examples of these systems we have Letizia [7] and the WebWatcher [2], [8], [6].

Finally, the *social bookmarking* is similar to the previous one in the sense that exists cooperativity between users, having as basis their relations (i.e: friendship), but centered on slope of tagging or bookmarking done by users.

However, the state-of-art doesn't mencion all the possibilites, particularly on which concerns the different kinds of personal information available - RSS feeds, browser speed dial, e-mails, images and video descriptions - all of them could be used to customize user search results. Although the biggest gap of these works is the poor use of relations and social information available on todays user applications (more specifically, social networks): posts, tweets, bookmarks, groups, events, contacts/friends and pages that I "Liked".

RELATED WORK

More specifically, we can group part of the alternatives for personalization discussed earlier, into a into six big sets. They are the following:

- **Proactive recommendation** - Gives sugestions to the user, accordingly to its interests. User has **always** free will to accept or reject the suggestions given to him. The philosophy behind is that no one knows better the user interests, than the the user itself;
- **Context** - Refers to what the user is "working at the moment", to his recent activities. For instance: my physical location, searches that i've made, and so forth;
- **Semantic analysis** - Points to the interpretation of the word meaning, tipically grouping them into a set of categories (example: "jaguar" can be animal category aswell on the car category). This is done, usually by querying a hierarchy of categories, such as ODP[1] or WordNet[2]. With these approaches we can explore the semantic relations between the concepts to find;
- **Collaboration** - This methodology, has as its basis to analyse the other users preferences, with likes (or profiles) similar to the user itself. So, the preference model is generated not only with the user personal preferences but also with the interests of the community of users. This collaborative information, can be used to customize user search results.
- **Results ranking** - It allows to order the search results in order of their relevance to the user, the first ones are the more relevant. These sort is done by different criterias, for example on the selection of a more refined search query;
- **Reading level** - This method consists on filtering and selecting the results that corresponds to the user reading proeficiency (their knowledge about a specific topic). For example "agriculture" can be the domain and basic the user reading level. Using this technique means that all results corresponding to the topic "agriculture" that have "normal" or "advanced/expert" as the technical level will be discarted (thereby considered irrelevant to the user).

WIRE SYSTEM

Our system is intitled "Wire". His job is to extract a set of different kinds of information (will it be social or local) from different sources and use it to customize users search results. To do that the system is divided into two big modules: "Back-end" and "Front-end".

The "Back-end" is totally written in pure Java and can be viewed has the "server side" (since it runs on a local server) and has the responsability to extract, process, manage and store all the different types of information. The extraction is done using several API's (i.e: Twitter, gmail and so on)

[1]Open Directory Project - **http://www.dmoz.org/**. It's the biggest hierarchy of categories or topics edited by humans, available on web. Each word is correctly grouped on specific categories, inside of this huge taxonomy.

[2]Similar to ODP, but it's only a database for the portuguese linguistic knowledge -**http://www.clul.ul.pt/clg/wordnetpt/index.html**

or accessing the users local hard drive and crawl the whole filesystem for documents. Still that information comes many times in an unsuable way, so we need to process it in some way to create usable and tretable information for us, in other words we need clean **keywords**. These two processes will be explained further on the document. The management process consists on identifying which keywords need a special treatment, for example the user Linkedin skills should be treated as "special keywords" in the sense that is long time but also short time interests, and so that information will likely be relevant for the user (this is done individually by each plugin, not it's not precisely a separate "module"). Finally, the storage process is in fact the object database itself (DB4O[3]) with all the corresponding implemented methods that query the database or to store information, or to check if the keyword already exists or more importantly to extract keywords based on some criteria (for instance, query for all keywords that occured more than 300 times; or that have recency bigger than 0.9; and so forth).

The "Front-end" is in fact the "client-side" (it runs on a web browser) and was totally written in HTML, CSS and javascript (using JQuery) and it corresponds to the user interface. It will send many requests to the server, either for logging on the social networks, or asking for access on user hard drives, or querying the database for relevant information/keywords (since this module is the one where users make searches), or simply for updating the status of the individual progress bars, that help the user to know how's it going the extraction of its information on each one of the platforms available.

Architecture

Our framework's constituted by different modules and is based on plugins. These allow the extraction of information from different sources, in order to use it on user search results personalization. An overview of the architecture of our framework is shown on figure 1.

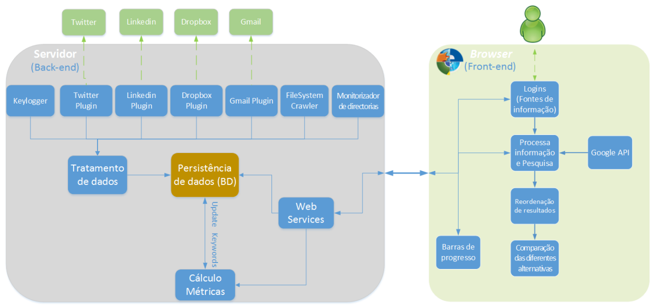


Figure 1. Overview of the system architecture

This framework is divided on two main components: Back-end and Front-end. Back-end is purely written in Java and it's basically a local web server. On other hand, Front-end is written in HTML+CSS and javascript (with JQuery[4] and fancybox[5] libraries) and it runs on a web browser.

[3]`http://www.db4o.com/`
[4]`http://jquery.com/`
[5]`http://fancyapps.com/fancybox/`

Two different components that interact with each other on a client-side style. Back-end is more like a domain logic with three main objectives: extract, process and store information. Front-end is the user interface and has three main objectives: allow user interaction, retrieve relevant information from back-end database and make custom searches (using that very same information).

Back-end
As said previously the Back-end component is formed by several distinc modules. So, on figure 1 we can see all modules of the component. Let's begin with the description of each one of its main modules.

Server
All Back-end components run on a local server. It's on this module that we've implemented mechanisms that abstract, the important component, of client-server communication. It defines the various types of treatments for each request sent by the client and its response, as well as the identification of the request method (GET, PUT, POST or OPTIONS) and also writing the respective response headers for each one of them.

Plugins
They are responsible for using API's to extract information from those different platforms. Each plugin is responsible for the information that it extracts from the source on which he is connected. After the information is selected and processed (depending on the plugin it might need additional modules like the "documentos" one, to accomplish part of its processing) and its stored on the database.

It's very easy to extend more functionality to several sources of information, since it allows adding more plugins that will use the corresponding API or libraries to access the information.

Documents
Module that easily allows adding new types of documents, by simply respecting the well defined interface and implement it accordingly with what we intend to process and obtain on what regards information level of other types of documents. Currently the contemplated (and which makes more sense) ones are: PDF, doc, docx and txt. The first three are accessed through apache libraries.

Directory monitor
Module responsible for using a thread to pool a directory (previously specified) on the file system. We only detect events of file **modification** or file **creation**.

When this event is detected, the corresponding file (document) is identified (the type to which it belongs to) and the module " documents" will process and "clean" it, storing it in the database. Finally, when you do not want to continue monitoring it's possible to terminate the process (stopping the thread responsible for pooling).

FileSystem Crawler
Detects how many drives are in the user computer and from such the hierarchy of directories each one has. Recursively traverses each directory and its subdirectories and identifies

the files (documents) present in each. It's, once again, the responsibility of the "document" modulus process each one of these files.

During this process are we filter files and directories that we've considered irrelevant. The relevant files are: PDF, doc, docx, and txt. Directories are all accepted **except** the directories and their subdirectories containing in its path these words (case insensitive):

Windows; Programs; Programs; Dropbox.

This filtering enables not only a more efficient extraction of information, but also the elimination of information which is "useless" and noisy. Operating system or installed programs documents (English or Portuguese version) doesn't interest us. In turn, it doesn't concern us to process dropbox through this way (synchronization of local files), because we already have a plugin (with respective API) for this purpose and is therefore inconsistent to have a duplicate source.

Finally, how the module is implemented becomes trivial to add more paths that are to be excluded in information processing.

Keylogger

Since it performs a "low-level" task - the detection of keyboard global events - its implementation fell on JNI (*Java Native Interface*) which allows the loading of C code, which in turn can access "low level" functions and capture keystrokes, the loose and its type. We consider only "releases" events for the purpose of constructing words, since having both gave incorrect (duplicate) results and have the "pressure" event will make more complicated to treat the problem of " key" repetition. For example: "SSSSSSSS" with event " pressure" we'd obtain: "SSSSSSSS", but with the event " release" we'd obtain: " S". And in most cases we only aim for " S". Only by a matter of milliseconds we might leave our fingers on the button longer than necessary. By the way we've implemented it, it's possible to disable and enable this *listener* anytime.

StopWords

Module responsible for accepting text files, containing the various stopwords for each language. It loads them in memory, and thus used by all modules responsible for extracting information to check if keywords currently under consideration are or aren't irrelevant.

Because of the way we've implemented it, it's trivial to add new stopwords, either to the languages already available, or for a new language (i.e.: French, German, etc.) by simply adding a new text file (.txt) which will contain the list of stopwords.

Metrics calculation

Module that contains the various options for calculating metrics (already presented on further sections). Given how we implemented this module, it's not very difficult to add new calculation methods, as long as they fit in the well defined interface, implementing these methods.

Metrics test

Extracts all keywords from the database (module "Persistence Data (BD)"), sorts the dates of each keyword from smallest to largest (from the oldest one to the recent one) and calculates their metrics. Finally, generates a test file (.txt) that contains all the information for each keyword: ¡name, number of occurrences, timestamps, latest timestamp, recency and regularity¿. Also there would be generated several HTML's (up to 10 MB each) that contain all this information "graphically", which helps us to visualize and validate if metrics values makes sense.

Data Persistency (DB)

Database module. Contains all settings to be made to the database from the fields to have as an index to the sizes of the blocks and defragmentation. Mainly, it contains implemented a wide range of methods of searching for information on various criteria (i.e: keyword name; timestamp greater than, for example - 14-08-2013; etc.), aswell efficiency alternatives in terms of query execution, either: " Query By Example (QBE)", "SODA" or " Native Query (NQ)" [6]. The tests we have done, SODA is globally the most efficient and Native Query showed a significant performance degradation. SODA is the API for querying, the other two alternatives are translated into SODA on runtime, so it is no wonder that we've obtained SODA as the more efficient querying mechanism.

Front-end

On figure 2 we present the *interface* structure (of web pages).

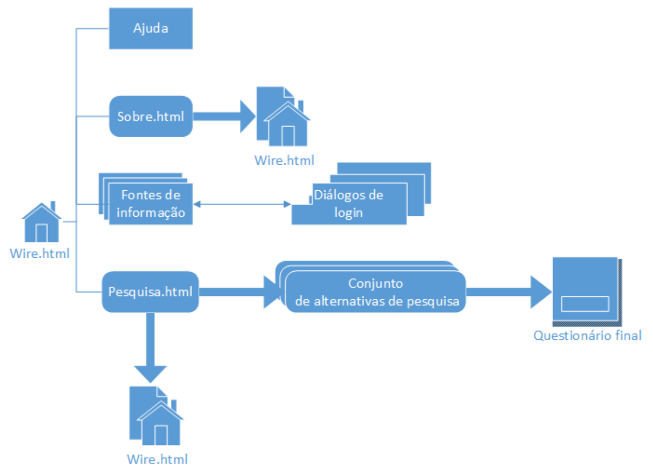


Figure 2. Estrutura do Front-end

- **Wire.html** - It's the Home page, place where we can find several buttons that lead us to the authentication process on social plataforms. Home page also allows access to "help" menu, "Sobre.html" page and search page (pesquisa.html);
- **Sobre.html** - Descrives the scope of this thesis and technologies used;
- **Pesquisa.html** - Leads to the various alternative search pages (google, query expansion and results reordering);

[6]A brief comparison of the three search engines disponibilizdos by DB4O - `http://community.versant.com/documentation/reference/db4o-7.13-flare/java/Content/object_lifecycle/querying/soda_query.htm`

- **Questionário final** - Page with the survey that user needs to fill, immediatly after it finishes experimenting the system and making its searches;

- **Fontes de informação** - Set of buttons, that allows to open the different login dialogs ("dialogos de login"), which allows the user to authenticate on the distinct social applications (authentication is **always** made on server side - Back-end - through the AJAX requests), or request the extraction of local information;

Extract and process information

On figure 3 we can see an overview of the information flow on Back-end.

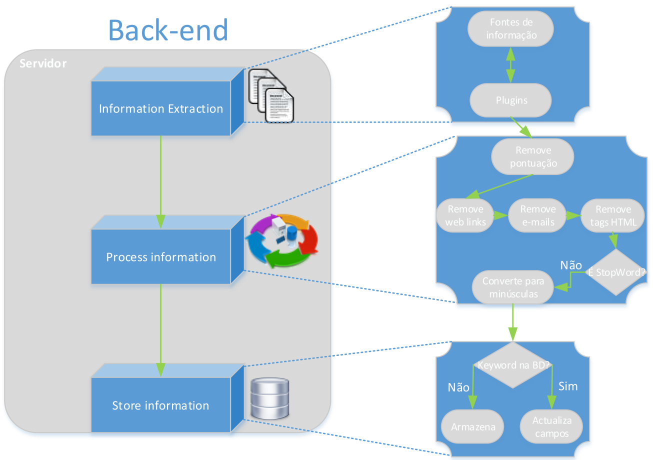


Figure 3. Back-end structure

Wire system is responsible for capturing many kinds of user personal information, whatever it is personal or social or contextual (anything that the user is working or interested most recently) 4. Using several API's: Twitter, Dropbox, Linkedin and Gmail we are able to extract users information from that social sources. We also capture (with a Keylogger) "live text" that the user is writing and we use it as an important source of contextual information. Besides this we analyse and process every PDF, txt, doc or docx document from users local computer, with the capability of also monitor some working directory chosen by the user itself, that detects which files is the user working at the moment (detects events of file creation and modification, which are the only two that concerns us).

After we process all the information, which can be summarized on the following steps:

1. Remove of all punctuation;

2. Delete all HTML tags (particularly in case of e-mails);

3. Remove all e-mail addresses;

4. Remove all hyperlinks (www; http; https);

5. Separate text by space (" ") so we can get only **keywords**;

6. Convert keywords to lowercase;

7. Check, for each keyword, if it's a portuguese or english stop word (we have two documents that lists all stop words for each language);

8. Store on the object database the keywords that respect the previous points.

After this, we need to provide some "meaning" to the keywords. We do that by calculating some metrics - number of occurences, most recent timestamp, recency and regularity. The number of times is a simple calculus, whenever we find a word that's already stored on the database, we increment its occurence counter. The most recent timestamp, represents the most recent occurence for each keyword. Recency represents the importance of the word according to its evolution in time (like a timeline), the more the keyword occurs close to the "todays date" the more recent it is, else it's less recent and we estimate this with using different weight functions that gives more or less weight to close or distant dates of occurence. Finally, regularity represents the pattern of occurence of the keyword, which means it doesn't really matter the distance of the occurence dates to the todays date, but the "stability" of the interval between the two. The more stable it is, or by other words the less variation distance has the more regular a keyword is. Maximum value of 1.0 is reached when the distance/interval between adjacent dates is always the same (i.e: 1 Jan, 2 Jan, 3 jan - all are distant from each other by 1 day. So regularity = 1.0).

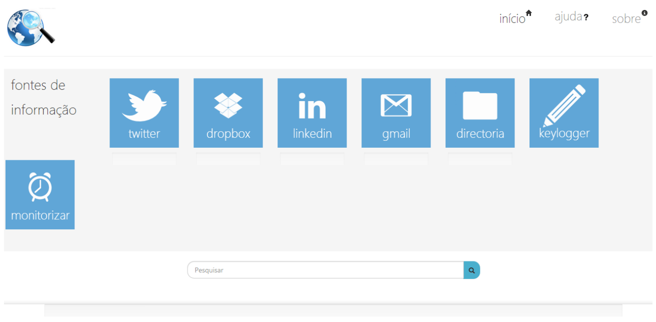


Figure 4. Wire system HTML+CSS interface

INFORMATION AS A PATH FOR PERSONALIZATION

A module of great importance on the Wire "Back-end" architecture is the one, that we call "Metrics processor", whose function is to give some kind of meaning or relevance to each one of the big set of keywords stored on our database. And how do we calculate that relevance? The answer is with a set of metrics (discussed previously), but the important and the more complex ones are **regularity** and **recency**.

About regularity, we tried some different methods to calculate it. But we're going to describe the only one that achieved better results, more consistent values. It can be summarized on the following steps:

1. Identification of occurence dates (without repeated dates);

2. Convert dates from milliseconds to seconds;

3. Convertem-se as datas de milissegundos para segundos;

4. Calculte the difference between adjacent dates (i.e: 2^a - 1^a; 3^a - 2^a; 4^a - 3^a; and so forth);

5. These differences/distances between successive dates are stored in a list;

6. We identify all outliers from this list (as explained previously);

   (a) Or we remove them;

   (b) Or we need to treat them through a mathematical form;

7. Select the maximum value of the list (the biggest of the differences);

8. Apply the three simple rule to normalize the dates (values in rage of: 0 to 1);

9. Calculate the arithmetic mean thereby obtaining regularity.

Relatively to recency, we've also tried a few methods (fewer than on regularity) and like previsouly we will pick and explain the one that achieved best results:

1. Identification of occurence dates (without repeated dates);

2. Subtract each occurence date with the start date and store the results on a new list (i.e. 1500 - 1000 = 500 s; and so forth);

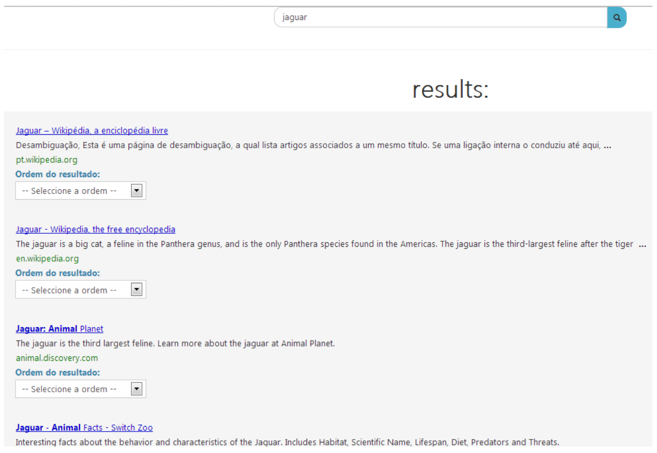
3. Divide each element of this last list by the "todays date" (i.e.: $\frac{500}{3000} = 0.25$; and so on) thereby obtaining a new list, lets call it "Normalized Timestamps" (range between: [0,1]);

4. Calculate the arithmetic mean (weight = 1 for all timestamps), which gives the same ponderation to distinct dates. Or, on the other hand, calculate the geometric mean which assigns diferent weighing to different dates (this last one makes more sense to use and th weigth calculus will be explained on point 5);

5. For each normalized element of the "Normalized timestamps" list, calculate the respective weight according to a specific weight function;

6. Calculate the geometric mean thereby obtaining recency.

After we're done calculating these metrics for each keyword we can now proceed to the extraction of the most relevant ones (by different criteria, for instance regularity or recency, or a mean of both, or a new metric that combines the remaining metrics through a specific calculus). The keywords that we've selected (because they were the most relevants) will now be used as an aid on users search, as we can see on figure 5, which is an example of query expansion and returned first the results that corresponds to the "animal" and not the "car brand".



jaguar

results:

Jaguar – Wikipédia, a enciclopédia livre
Desambiguação, Esta é uma página de desambiguação, a qual lista artigos associados a um mesmo título. Se uma ligação interna o conduziu até aqui, ...
pt.wikipedia.org
Ordem do resultado:
-- Seleccione a ordem --

Jaguar - Wikipedia, the free encyclopedia
The jaguar is a big cat, a feline in the Panthera genus, and is the only Panthera species found in the Americas. The jaguar is the third-largest feline after the tiger ...
en.wikipedia.org
Ordem do resultado:
-- Seleccione a ordem --

Jaguar: Animal Planet
The jaguar is the third largest feline. Learn more about the jaguar at Animal Planet.
animal.discovery.com
Ordem do resultado:
-- Seleccione a ordem --

Jaguar - Animal Facts - Switch Zoo
Interesting facts about the behavior and characteristics of the Jaguar. Includes Habitat, Scientific Name, Lifespan, Diet, Predators and Threats.

Figure 5. Example: search results for the query "jaguar"

EVALUATION

We carried an user evaluation with 18 participants, where our main objective is to find out which one of our group of methods for search (query expansion with variants, results ordering with variants and google baseline) returns more relevant results for the user, in a proper order (firstly the more relevant ones and the last ones the more irrelevant ones).

The first objective is to make the user do a regular search (input text on the text box and hits enter) and in the background we made four searches, each represents one of those variants. For each search (google search and query expansion) eight results are returned, but we want to display on the web page at maximum the 16 distinct results (because the alternative of results ordering doesn't return **distinct** results than the others alternatives. It only reorders what was returned previously). So, having those distinct results the user finds a combo-box beneath each result and must order it according to its own preferences, and then submits the form (as shown on figure 5.

After that, we present each one of the alternatives/variants to the user and he must decide which one of them is the one that best suits its interests, in other words which one of the alternatives presented the best results order according to the user preferences. Finally, we ask all participants to answer to a final survey[7] about their experience on searching tasks, their habit of using search engines and how did they evaluate our system interface.

Results discussion

Graphic 6 shows the results for each alternative on all three tasks.

Through this graphic we can see that for task one, 50% of the participants selected Google has the approach that best ordered their results. Another half of the participants selected one of our personalization approach (33.3% Query Expansion and 16.7% results reordering). So, if we analyse this individually Google was the most selected one, which means that we didn't had so much success on this task. For task two, no one

[7]`https://docs.google.com/forms/d/1OlfiLVhbzXGXYGJ44sfE3hnOWNxB2fEbsAaEMTgJ9ZA/edit`

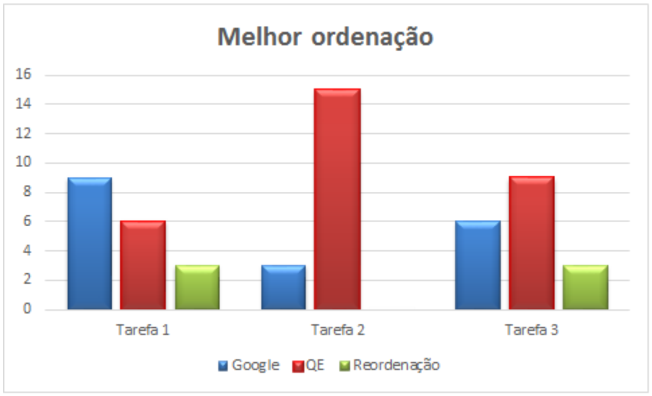


Figure 6. Best ordering alternative for each task

selected results reordering as best alternative, however 83.3% selected Query Expansion and only 16.7% selected Google. We can prove that our personalization mechanism based on user recent activities, had great success. Finally, for task three we had 66.7% (50% of Query Expansion and 16.7% results reordering) of the participants selecting one of our approaches as best ordering alternative and only 33.3% selected Google. With this task, we can conclude that our tools of short-term and contextual information extraction proved to be useful, by showing good results and an improvement to Google's results order.

Resuming, with task one we couldn't achieve better results than Google, and therefore the long-term and social information is less important in terms of obtaining good results, unlike short-term and contextual information that shows nice results (like was the case of task one and two).

Table 1 and figure 7 shows the final results obtained for the MAP (*Mean Average Precision*) calculus.

	Tarefa 1			Tarefa 2			Tarefa 3		
	Google	QE	Reordenao	Google	QE	Reordenao	Google	QE	Reordenao
MAP	**0,58**	0,33	0,55	0,27	**0.59**	0.27	0.42	**0,43**	0.39
Spearman	**0,29**	0,12	0,18	0,05	**0,10**	0.05	**0,30**	0.19	0.11

Table 1. MAP - Results of each approach for each task

Trough this metric it's possible to find some conclusions about the effectiveness of our system and consequently of it's personalization approaches for each one of the tasks performed on user tests.

On task 1, Google was the approach that on average find more precise results for the user approximately 55%. Google was the one that achieved the best performance in terms of results ordering (figure 6), but also in terms of results precision. In spite of QE haven't been so far from Google and also the reordering showed positive results sometimes, we were expecting that on this task Google take the league by showing best results. This is due to the fact of dealing with many sources of information (local and social) with different kinds of information (social and long term mainly) and interest topics for the user. Of this mixture, results many "noise/garbage" and less system efficiency in determining which of those topics/interests found and filtred, will the user be interested and choose upon performing user test.

Task 2, was the most "closed" in terms of search liberty and served to prove undoubtedly that our assumption that the contextual and short-term information could be very useful to personalize user searches. Our QE approach was without question, the one that showed more precise results for the users. Google and results reordering precision were similar, since we hadn't enough keywords to be able to sort results (thus presenting, and on the same order, same results as Google). These two approaches exhibit some precision, because some results returned by QE cease not to be relevant to the users (although the topic was the same), implying even if the users were more interested in, for instance "coffee", that they'll assign less importance to some of these results and prefer some others about "programming". Other users simply prefered to sort, almost totally, by programming results, which represented better their interests.

On task 3 search liberty was huge, and one more time, we focused on using contextual and short-term information to capture user interests. We've obtained that QE and Google were nearly tied on what concerns precision (however like we've said previously, QE showed better results ordering). By these results we can understand that, the fact of the set of information is very broad (local files) and all user interaction (searches, writings, and so on) leading to many topics of interest (even restricting the search to one topic), resulting on more "noise" and, consequently the relevant keyword extraction is less effective. However, we've obtained very good results for this task and we verified that our personalization mechanisms can overcome Google.

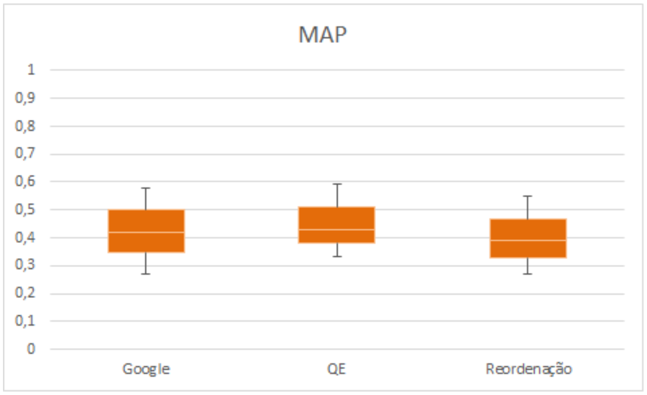


Figure 7. MAP - BoxPlot

To evaluate results relevance we've used MAP (*Mean Average Precision*) metric, which function is to allow us to conclude if the relevant results (or the important ones) for the user have appeared on each alternative or not, independently of their order.

By this Box-Plot we've observed that Google has a median in the order of 40% precision and a mean precision in range of 33%-49%, also this is the appraoch that oscilates more (presence of more outliers), in part because of the achieved minimum results on the second task. QE shows a data simetry, median is centered on the box and has a value of 46% precision and in mean precision is located on the range of 40%-53%, this is also the approach less oscilates. Finally, results reordering as we're expecting has the median on 37% precision and its mean on range of 32%-46%.

Summarizing, Query Expansion presented the most consistent results and reaches the maximum value of the graphic (59%). However, Google covers a bigger extension of data, but tipically it only reaches the maximum of 58%. Results reordering approach was the one that reached the worst results, fewer levels of precision.

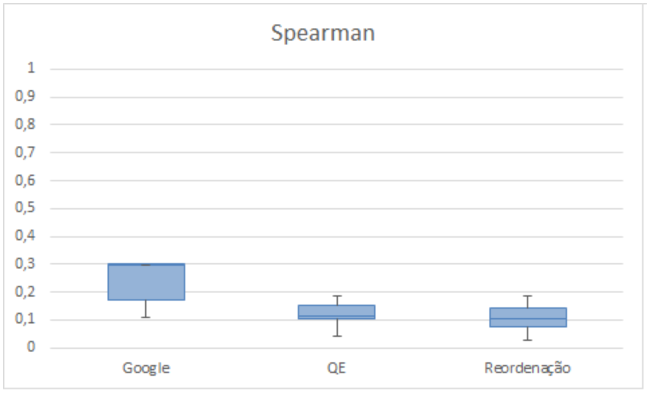


Figure 8. Spearman Rank Correlation - BoxPlot

To evaluate the importance of the results ordering for the user interests, and which order is more closer to the one that user prefered, in other words to correlate user reordering with the one offered by our approaches we need to use the statistical method *Spearman Rank Correlation*. So, the objective of this method is to measure the strength of association between these two ranked variables.

By the Spearman BoxPlot (graphic 8) observation we can see that Google and query expansion were close to each other in terms of results correlation, Google oscilated between 0.17 to 0.3, having a mean of 0.29 of results correlation. Query expansion achieved on most cases between 0.11 to 0.15, having a mean of 0.11. In other turn, reording approach was the one that showed on most cases 0.08 to 0.14 of correlation, having a mean of 0.11.

In conclusion, Google and query expansion find themselves close to each other in terms of results precision, but Google is frequently situated in higher values of correlation (0.17-0.3). So, on average Google managed to achieve more correlated results than the other two approaches. However, none of the approaches achieved strong correlations (in the order of 0.6). Whilst results reordering approach was the less useful, in terms of results personalization, both in terms of precision and results ordering. Reordenation approach was the least useless in terms of results personalization, both at the level of accuracy as sorting. So, the results points towards the use of query expansion. We conclude that our QE approach can be used as a complement to Google and our results points to the use of QE as the prefered personalization mechanism.

User's opinions

To quantify the user's satisfaction we chose some main features to be evaluated using a Likert Scale, with values varying from 1 to 5 (1- Strongly disagree; 2- Disagree; 3- Neither agree nor dis- agree; 4- agree; 5- Strongly Agree). Table 2 presents the user evaluation of our interface and system effectiveness.

	Easy Interface	interface usability	Easy navigation	System effectiveness
Average	4.72	4.28	4.67	3.72
Standard-deviation	0.06	0.06	0.05	0.06
Variance	0.33	0.33	0.24	0.33
Mode	5	4	5	4
Mean	5	4	5	4

Table 2. Classification of user satisfaction, using a 5-point Likert scale

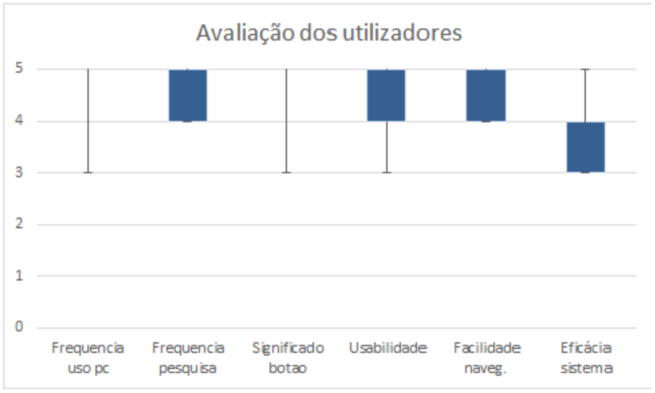


Figure 9. System usage and reccomendation

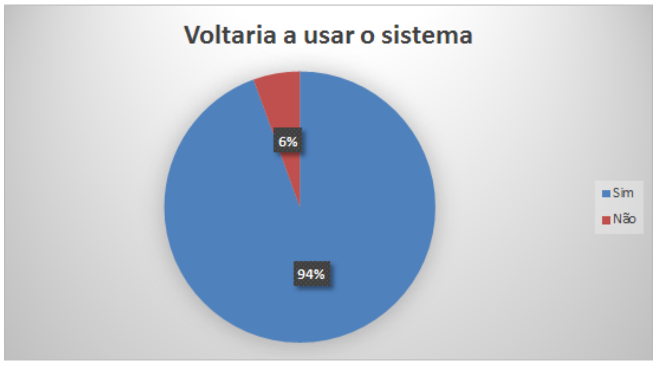


Figure 10. User evaluation results (on Likert scale)

We can observe that our interface had very good grades in terms of atractiveness and easy navigation, an average of 4.67 on 5. Although it was not our primary concern but our efforts on making it attractive and simple, showed us that was the right way to build it and to be appelative and easy understandable by the common user.

About the system effectiveness we've achieved some nice grades too, which proves that our users had obtained results according to their interests.

At the end of the test we've asked users to give us their opinions and likes and dislikes about our system and what improvements we should make on our framework. The most required was to optimize the extraction and filtering processes, which took too long to finish on the first task, particularly for a user that had more than 6000 e-mails on his account. Some other opinions asked to correct some bugs on the interface, such as be able to deactivate the success, error and warning messages whenever they want, the [x] button didn't function properly. Other users suggested that after logging in on a social platform the "login window" should be closed automatically, and also on the directory watcher when the operation succeeds the confirmation dialog goes to the keylogger window, which sometimes was a bit confusing for the participants (but they quickly found it was a small error).

Resuming, we can conclude, by the graphic 9 that the majority of our users (94%) will continue to use or recommend our system to others, showing a great satisfaction with our application and the simplicity and usability of our interface.

CONCLUSION

Personalize user search results is a difficult task and has many fields of study and distinct approaches. The innovation presented here consists on four main areas: contextual information, use of local and social information (mainly through the use of social networks), query enhancement and results ordering. For contextual information we investigate what's the user writing (using a keylogger) at the current time, including what he searched recently and we also monitor what documents is he working at the moment (only the txt, pdf, doc and docx ones). Social information's obtained using social networds API's: twitter, linkedin, dropbox and gmail. Personal information is obtained by processing users hard drives and get all of its documents. Query enhancement is taking the user's intended query and enhancing it with relevant contextual information to create a more focused query (**query expansion**). Finally, results ordering is done after a user make a search, retrieve those very same results and with its relevant personal information we order those results using different criterias (use of various keywords to order, sum of all its occurences on each result, mean of occurences or maximum keyword occurence). Our hypothesis was that the marriage of these four techniques will result in a search experience that is more valuable than current tools.

The main conclusion that can be drawn from this research is that by combining the principles of search with context awareness, more relevant search results may be produced for a user. The experimental results and user study proved that using personal, social and contextual information to enhance user search queries, resulted on adding value to the user regarding on the achivement of results accordingly to its interests (including some results that they've liked and didn't expect to appear).

Despite the results and the user satisfaction were good, we could identify some problems or limitations that needed to be adjusted or corrected. Some minor problems on the interface usability and appearance, but also on the less relevant keyword retrieval, will it be due to the metrics calculation or even by the existence of a great amount of gargabe on e-mails, or documents.

We've achived better results using context and short-term information, than using social and long-term information (like we've said previously), but also because this information can referer to a bigger set of topics.

Moreover, considering the current panorama on the usage of personal information to help users achieving useful and relevant information, we belive that the work presented in this dissertation add some value and a new approach to this area.

ACKNOWLEDGMENTS

We'd like to thank all participants (test users) that gently accepted to test our system and allowed it to access their private and personal information, store them locally on a database (in an anonymous way) and operate over it to decide which part of it is relevant for the user interests.

REFERENCES

1. Anderson, N. Putting Search in Context: Using Dynamically-Weighted Information Fusion to Improve Search Results. *2011 Eighth International Conference on Information Technology: New Generations* (Apr. 2011), 66–71.
2. Armstrong, R., Freitag, D., and Joachims, T. Webwatcher: A learning apprentice for the world wide web. In *AAAI Spring symposium on Information gathering from Heterogeneous, distributed environments* (1995), 6–12.
3. Bradley, K., and Rafter, R. Case-based user profiling for content personalisation. *Adaptive Hypermedia and Adaptive Web-Based Systems* (2000), 62–72.
4. Budzik, J. Watson: Anticipating and contextualizing information needs. In *Proceedings of the Annual Meeting-American Society for Information Science*, vol. 36 (1999), 727–740.
5. Collins-Thompson, K., Bennett, P. N., White, R. W., de la Chica, S., and Sontag, D. Personalizing web search results by reading level. In *Proceedings of the 20th ACM international conference on Information and knowledge management - CIKM '11*, ACM Press (New York, New York, USA, 2011), 403–412.
6. Joachims, T., Freitag, D., and Mitchell, T. Webwatcher: A tour guide for the world wide web. In *International Joint Conference on Artificial Intelligence*, vol. 15 (1997), 770–777.
7. Lieberman, H. Letizia: An agent that assists web browsing. In *International Joint Conference on Artificial Intelligence*, vol. 14 (1995), 924–929.
8. Mladenic, D. Personal WebWatcher: design and implementation. *Slovenia, Department of Intelligent Systems, J. Stefan Institut* (1996).
9. Rafter, R., and Bradley, K. Personalised retrieval for online recruitment services. In *Proceedings of the 22nd Annual Colloquium on Information Retrieval*, Citeseer (2000), 1–13.
10. Sugiyama, K., Hatano, K., and Yoshikawa, M. Adaptive web search based on user profile constructed without

any effort from users. In *Proceedings of the 13th conference on World Wide Web - WWW '04*, ACM Press (New York, NY, USA, 2004), 675–684.

11. Teevan, J., Dumais, S. T., and Horvitz, E. Personalizing search via automated analysis of interests and activities. In *Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '05*, ACM Press (New York, New York, USA, 2005), 449–456.