D2.2.1 SPECIFICATION OF POST QUERYING PROCESSING FUNCTIONALITIES

Advanced Search Services and Enhanced Technological Solutions for the European Digital Library

Grant Agreement Number: 250527

Funding schema: **Best Practice Network**

Deliverable ASSETS.D2.2.1.CNR.WP2.2.V1.0

Report

V1.0 – 11 May 2011

Document. ref.: ASSETS.D2.2.1.CNR.WP2.2.V1.0

assets

Programme Name: .... ICT PSP
Project Number: 250527
Project Title: .... ASSETS
Partners: .... Coordinator: ENG (IT)
Contractors:
Document Number: ASSETS.D2.2.1.CNR.WP2.2.V0.2
Work-Package:............................... 2.2
Deliverable Type: .... Report
Contractual Date of Delivery: ...... 31 March 2011
Actual Date of Delivery: .... 11 May 2011
Title of Document: .... Specification of post querying processing functionalities
Author(s): .... Diego Ceccarelli (CNR), Claudio Lucchese (CNR), Raffaele Perego (CNR), Òscar Paytuv (BMAT), Serkan Demirel (Europeana), Sergiu Gordea (AIT)

Approval of this report .... APPROVED

Summary of this report:................ see Executive Summary

History: .... see Change History

Keyword List: .... Post Query Processing

Availability.................................... This report is:
X public
limited to ASSETS consortium distribution
limited to EU Programme distribution
restricted

Change History

Version	Date	Status	Author (Partner)	Description
0.1	01.03.2011	Draft	Claudio Lucchese (CNR)	Initial draft
0.2	31.03.2011	Peer-review	Diego Ceccarelli (CNR)	
0.3	26.04.2011	Pre-final	Claudio Lucchese (CNR)	Addressing reviewers comments
0.4	11.05.2011	Final	Claudio Lucchese (CNR)	Addressing reviewers comments
1.0	11.05.2011	Final	Claudio Lucchese (CNR)	Approved and Released

assets

Table of Contents

EXECUTIVE SUMMARY **1**

1. T2.2.1: POST QUERYING PROCESSING **2**

1.1 SEARCH SHORTCUTS: AN EFFECTIVE QUERY SUGGESTION ALGORITHM 2
1.1.1 Related work 2
1.1.2 Search Shortcuts Problem definition 3
1.1.3 Generating query suggestions with Search Shortcuts 4
1.1.4 Quality Assessment 6
1.2 SEARCH SHORTCUTS SPECIFICATION FOR ASSETS 11

2. T2.2.2: METADATA-BASED RANKING **13**

2.1 DESCRIPTION OF BM25F 13
2.2 LEARNING TO RANK 14
2.2.1 Ranking SVM 15
2.2.2 RankNet 16
2.2.3 LambdaRank 17
2.2.4 LineSearch or direct optimization of BM25F 18
2.3 METADATA BASED RANKING FOR ASSETS 19
2.3.1 BM25F Solr Plugin 21

3. T2.2.3: TEXT INDEXING AND RETRIEVAL **23**

3.1 QUERY LOG ANALYSIS 23
3.1.1 Query Analysis 24
3.1.2 Session Analysis 25
3.2 INDEXING AND RETRIEVAL OF QUERY LOG INFORMATION FOR ASSETS 26

4. CONCLUSIONS **30**

5. REFERENCES **31**

assets

Executive Summary

This document contains the specification of the services that are going to be developed within tasks "T2.2.1 Post Querying Processing", "T2.2.2 Metadata based ranking" and "T2.2.3 Text Indexing and Retrieval". All these tasks fall under the responsibility of CNR, but their detailed definition is the result of an agreement reached with the other partners of the ASSETS project having a technical role. For each activity, a scientific analysis and a detailed technical specification at the API level is provided.

Regarding task "T2.2.1 Post Querying Processing", in Section 1 we report on the state of the art in query recommendation systems, and we discuss an effective query suggestion algorithm we are going to integrate into the ASSETS platform. Such recommendation algorithm has proven to be effective on standard Web query logs and evaluation datasets, and we expect that a similar performance can be achieved in the Europeana case.

Section 2 deals instead with effective techniques for ranking metadata objects in the Europeana context as planned within task "T2.2.2 Metadata based ranking". We first review the state of the art in multi-field document retrieval, and then we provide a specification of an advanced metadata based ranking to be included into the ASSETS engine. In particular, we propose to adopt the BM25F ranking function, and to exploit machine learning algorithms to best tune its parameters.

Both the previous activities require models about user behaviours and preferences to be exploited. Since the knowledge needed to build these models can be mined from the logs storing past user activities, in Section 3 we discuss the deep analysis conducted on Europeana query logs in order to devise opportunities for improving query suggestion and the Europeana ranking function. The software support to query log cleaning and mining are the subject of activities within task "T2.2.3 Text Indexing and Retrieval", that will provide an ensemble of tools for storing, mining and searching Europeana usage information.

1. T2.2.1: Post Querying Processing

Providing users of Web Search Engines (WSEs) systems with suggestions is a common practice aimed at “driving” users toward the information bits they may need. Suggestions are normally provided as queries that are, to some extent, related to those recently submitted by the user. The generation process of such queries, basically, exploits the expertise of “skilled” users to help inexperienced ones. The knowledge mined for making this possible is contained in WSEs’ logs which store all the past interactions of users with the system. The more the users that satisfied the same information need in the past, the more precise and effective the related suggestions provided by the query recommendation technique. On the other hand, to generate effective suggestions for queries that are rare or have never been seen in the past is an open issue poorly addressed by state-of-the-art query suggestion techniques.

In the past months we conducted a thorough preliminary study on its query log to understand the common behaviours of Europeana users, and to evaluate opportunities for integrating an effective query suggestion service into ASSETS. This analysis, which is reported in Section 3.1, justifies the adoption of such a solution since it shows that many users do not succeed in finding promptly the results they are looking for.

Below, we report on the state of the art in query recommendation systems, and we propose to include into ASSETS an effective query suggestion algorithm that can improve the users' interaction with the ASSETS (and Europeana) search portal. Such recommendation algorithm was proven to be effective on standard Web query logs and evaluation datasets. Finally, we provide a specification of the software to be integrated into the ASSETS platform.

1.1 Search Shortcuts: an effective query suggestion algorithm

1.1.1 Related work

The problem of query suggestion is related to two related research fields that have been traditionally addressed from different points of view: query suggestion algorithms and recommender systems. Recommender systems are used in several domains, being especially successful in electronic commerce. They can be divided in two broad classes: those based on content filtering, and those on collaborative filtering. As the name suggests, content filtering approaches base their recommendations on the content of the items to be suggested. On the other side, collaborative filtering solutions are based on the preferences expressed by the users.

Due to their characteristic features, query suggestion calls for specifically tailored algorithm being able to exploit all the additional information available in this scenario, such users session, click, query results, etc. Techniques proposed during last years are very different, yet they have in common the exploitation of usage information recorded in query logs [S10]. Many approaches extract the information used from the plain set of queries recorded in the log, although there are several works that take into account the chains of subsequent queries that belong to the same search session. The two most effective approaches are the ones based on the concept of ***Cover Graph (CG)*** and ***Query Flow Graph(QFG)***, which we described in the following.

The authors of [BT07] exploit click-through data as a way to provide recommendations. The method is based on the concept of Cover Graph. A CG is a bipartite graph of queries and URLs, where a query q and an URL u are connected if a user issued q and clicked on u that was an answer for the query. Suggestions for a query q are thus obtained by accessing the corresponding node in the CG and by extracting the related queries sharing more URLs. The sharing of clicked URLs results to be very effective for devising related queries.

[BB+08] introduced the concept of Query Flow Graph (QFG), an aggregated representation of the information contained in a query log. A QFG is a directed graph in which nodes are queries, and the edge connecting node q_1 to q_2 is weighted by the probability that users issue query q_2 after issuing q_1. Authors highlight the utility of the model in two concrete applications, namely, devising logical sessions and generating query recommendation. The authors refine the previous studies in [BB+09a] and [BB+09b] where a query suggestion scheme based on a random walk with restart model on the QFG is proposed.

We took into consideration both two algorithms based on CG and QFG to validate our proposal for Europeana.

1.1.2 Search Shortcuts Problem definition

We formalize the problem of recommending good queries as a problem of generating "search shortcuts", where we call shortcuts those queries that can help the user to access earlier the content she is looking for.

The ***Search Shortcut Problem (SSP)*** is formally defined as a problem related to the recommendation of queries in search engines and the potential reductions in the users session length. This problem formulation allows a precise goal for query suggestion to be devised: recommend queries that allowed “similar” users, i.e., users which in the past followed a similar search process, to successfully find early the information they were looking for. The problem has a nice parallel in computer systems: prefetching. Similarly to prefetching, search shortcuts anticipate requests to the search engine with suggestion of queries that a user would have likely issued at the end of her session.

Let U be the set of users of a Web Search Engine (WSE) whose activities are recorded in a query log QL, and Q be the set of queries in QL. We suppose QL is pre-processed by using some session splitting method (e.g. [JK08],[LO+11]) in order to extract query sessions, i.e., sequences of queries which are related to the same user search task. Formally, we denote by S the set of all sessions in QL, and σ^u a session issued by user u. Moreover, let us denote with σ_i^u the i-th query of σ^u. For a session σ^u of length n its final query is the query σ_n^u, i.e. the last query issued by u in the session. To simplify the notation, in the following we will drop the superscript u whenever the user u is clear from the context.

We say that a session σ is ***satisfactory*** if and only if the user has clicked on at least one link shown in the result page returned by the WSE for the final query σ_n, ***unsatisfactory*** otherwise. Clearly, it may happen to have a user click leading to an unsatisfactory result page, but we can safely rely on the so called "wisdom of the crowds": good queries generate a larger number of clicks, thus having a much significant impact on the recommendation algorithm.

Finally, given a session σ of length n we denote $\sigma_{t|}$ the ***head*** of σ, i.e., the sequence of the first t, $t < n$, queries, and $\sigma_{|t}$ the ***tail*** of σ given by the sequence of the remaining $n - t$ queries.

Definition 1. We define a k-way shortcut a function h taking as argument the head of a session $\sigma_{t|}$, and returning as result a set $h(\sigma_{t|})$ of k queries belonging to Q.

Such definition allows a simple ex-post evaluation methodology to be introduced by means of the following similarity function:

Definition 2. Given a satisfactory session $\sigma \in S$ of length n, and a k-way shortcut function h, the similarity between $h(\sigma_{t|})$ and a tail $\sigma_{|t}$ is defined as:

$$s\left(h\left(\sigma_{t|}\right), \sigma_{|t}\right) = \frac{\sum_{q \in h(\sigma_{t|})} \sum_{m=1}^{n-t} [\![q = \left(\sigma_{|t}\right)_m]\!] f\left(m\right)}{|h(\sigma_{t|})|}$$

where $f(m)$ is a monotonic increasing function, and function $[q=\sigma_m]=1$ if and only if q is equal to σ_m.

In order to evaluate the effectiveness of a given shortcut function h, the average value of s on all satisfactory sessions in S can be computed .

Definition 3. Given the set of all possible shortcut functions H, we define Search Shortcut Problem (SSP) the problem of finding a function $h \in H$ which maximizes the sum of the values computed according to Definition 2 on all satisfactory sessions in S.

A difference between search shortcuts and query suggestion is actually represented by the function $[q=(\sigma_{|t})_m]$, Definition 2. By relaxing the strict equality requirement, and by replacing it with a similarity relation – i.e., $[q \sim (\sigma_{|t})_m]=1$ if and only if the similarity between q and σ_m is greater than some threshold – the problem reduces, basically, to query suggestion. By defining appropriate similarity functions, Definition 2 the can be thus used to evaluate query suggestion effectiveness as well.

Finally, we should consider the influence the function $f(m)$ has in the definition of scoring functions. Actually, depending on how f is chosen, different features of a shortcut generating algorithm may be tested. For instance, by setting $f(m)$ to be the constant function $f(m)=c$, we measure simply the number of queries in common between the query shortcut set and the queries submitted by the user. A non-constant function can be used to give an higher score to queries that a user would have submitted later in the session, i.e. queries closer to the last successful one. An exponential function $f(m)=e^m$ can be exploited instead to assign an higher score to shortcuts suggested early. Smoother f functions can be used to modulate positional effects.

1.1.3 Generating query suggestions with Search Shortcuts

Inspired by the above SSP, we define a novel algorithm that aims to generate suggestions containing only those queries appearing as final in satisfactory sessions. The goal is to suggest queries having a high potentiality of being useful for people to reach their initial goal. As hinted by the problem definition, suggesting queries appearing as final in satisfactory sessions, in our view is a good strategy to accomplish this task. In order to validate this hypothesis, we analyzed the Microsoft RFP 2006 dataset, a query log from the MSN Search engine containing about 15 million queries sampled over one month of 2006 (hereinafter *QL*).

First, we measured that the number of distinct queries that appear as final query in satisfactory sessions of *QL* is relatively small if compared to the overall number of submitted queries: only about 10% of the total number of distinct queries in *QL* occur in the last position of satisfactory user sessions. As expected, the distribution of the occurrences of such final queries in satisfactory user sessions is quite skewed (as shown in Figure 1), thus confirming once more that the set of final queries actually used by people is limited.

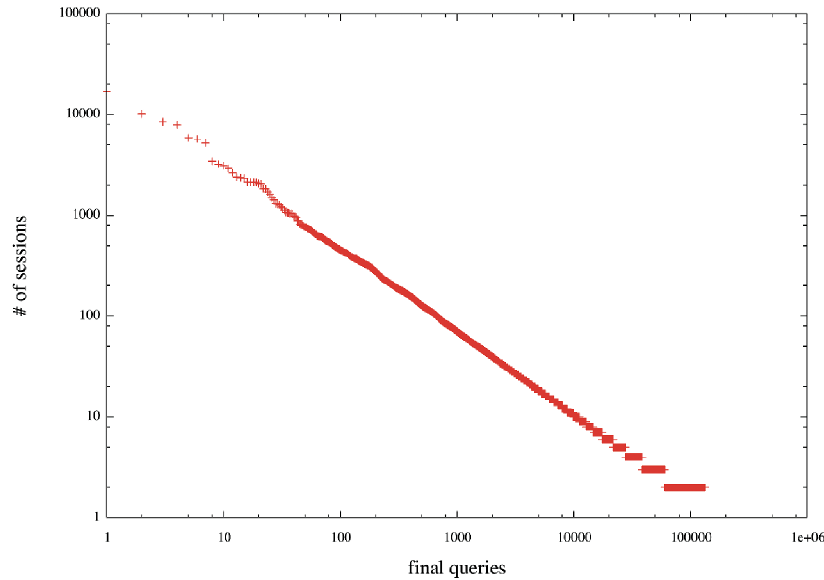

Figure 1 Popularity of final queries in satisfactory sessions

queries which are final in some satisfactory sessions may obviously appear also in positions different from the last in other satisfactory sessions. We verified that, when this happens, these queries appear much more frequently in positions very close to the final one. About 60% of the distinct queries appearing in the penultimate position of satisfactory sessions are also among the final queries; about 40% in positions second to the last; 20% as third to the last, and so on. We can thus argue that final queries are usually close to the achievement of the user information goal. We consider these queries as highly valued and high quality short pieces of text expressing actual user needs.

The SSP algorithm we propose works by computing, efficiently, similarities between partial user sessions (the one currently performed) and historical satisfactory sessions recorded in a query log. Final queries of most similar satisfactory sessions are suggested to users as search shortcuts.

Let σ' be the current session performed by the user, and let us consider the sequence τ of the concatenation of all terms with possible repetitions appearing in $\sigma'_{t|}$, i.e. the head of length t of session σ'. We now compute the value of a scoring function $\delta(\tau, \sigma^s)$, which measures the similarity between the set of terms τ and current queries (i.e. queries used in the current session) and, for each satisfactory session. Intuitively, this similarity value measures to which extent a previous session overlaps with the user's information need expressed so far (represented as a bag-of-words computed through the concatenation of terms τ). The sessions are ranked according to δ scores, and final queries of the top n ranked sessions are used in the list of query suggestions. It is obvious that we may have different recommendation methods, depending on how the function δ is chosen. In our particular case, we chose δ to be computed with the similarity function used in the BM25 algorithm [RZ09]. We opt for an IR-like metric, because we want to take increase the importace of high discriminative words found in the context of the past sessions. BM25, and other IR-related metrics, have been designed specifically to leverage this aspect in the context of query or documents similarity computation. The shortcuts generation problem has been, thus, reduced to the information retrieval task of finding highly similar sessions in response to a

given sequence of queries. In our current experiments, we compute the similarity function δ only on the current query issued by the user instead of using the whole head of the session. This will allow us to compare the results of our work with other algorithms which produce recommendations starting from a single query.

The idea described above is thus translated into the following process. For each unique ***final query*** $\boldsymbol{q_f}$ contained in satisfactory sessions we define what we have called a ***virtual document*** identified by its title and its content. The title, i.e. the identifier of the document, is exactly query string q_f . The content of the virtual document is instead composed of all the terms that have appeared in queries of all the satisfactory sessions ending with q_f . At the end of this procedure we have a set of virtual documents, one for each distinct final query occurring in some satisfactory sessions. Just to make things more clear, let us consider a toy example. Consider the two following satisfactory sessions: *(dante alighieri→ divina commedia→ paolo and francesca*, and *(divina commedia→ inferno canto V→ paolo and francesca)*. We create the virtual document identified by title *paolo and francesca* and whose content is the text *(dante alighieri divina commedia divina commedia inferno canto V)*. The virtual document actually contains also repetitions of the same term that are considered in the context of the BM25 ranking metrics. All virtual documents are indexed with the preferred Information Retrieval system, and generating shortcuts for a given user session σ' becomes simply processing the query $\sigma'_{t|}$ over the inverted file indexing such virtual documents. We know that processing queries over inverted indexes is very fast and scalable, and these important characteristics are inherited by our query suggestion technique as well.

The other important feature of our query suggestion technique is its robustness with respect to rare and singleton queries. Singleton queries account for almost 50% of the submitted queries [S10], and their presence causes the issue of the sparseness of models [AT05]. Since we match τ with the virtual documents obtained by concatenating all the queries in each session, even previously unseen queries can match a virtual document and generate high quality suggestions. Most suggestion algorithms, instead, can match only previously submitted queries. Therefore we can generate suggestions for queries in the long tail of the distribution those terms have some context in the query log used to build the model.

1.1.4 Quality Assessment

Evaluating the effectiveness of a recommender system is a difficult task. We tried to avoid the cost of expensive user-studies, and to exploit the implicit feedback present in the query log. Thus, in our experimental evaluation we use the similarity metric defined in Definition 2, and we compute the average value of similarity over a set of satisfactory sessions. This performance index objectively measures the effectiveness of a query suggestion algorithm in foreseeing the satisfactory query for the session.

In particular, we measured the values of this performance index over suggestions generated by using our Search Shortcuts (SS) solution and by using in exactly the same conditions two other state-of-the-art algorithms: Cover Graph (CG) proposed by [BT07] and Query Flow Graph (QFG) proposed by [BB+09a]. These algorithms are recent and highly reputed representatives of the best practice in the field of query recommendation.

Related to the evaluation methodology based on user-studies, we propose an approach that measures coverage and the effectiveness of suggestions against a manually assessed and publicly available dataset. To this purpose, we exploited the query topics and the human judgements provided by NIST for running the TREC 2009 Web Track's Diversity Task

(http://trec.nist.gov/data/web09.html). For the purposes of the TREC diversity track, NIST provided 50 queries to a group of human assessors. Assuming each TREC query as a topic, assessors were asked to identify a representative set of subtopics covering the whole spectrum of different user needs/intentions. Subtopics are based on information extracted from the logs of a commercial search engine, and are roughly balanced in terms of popularity. Obviously the queries chosen are very different and cover different search aspects (e.g. difficulty, ambiguity, and/or faceted search) in order to allow the overall performance of diversification methods to be evaluated and compared. Since diversity and topic coverage are key issues also for the query recommendation task [MLK10], we propose to use the same third-party dataset for evaluating query suggestion effectiveness as well.

Let's now introduce the definitions of coverage, and effectiveness.

Definition 4 (Coverage). Given a query topic *A* with *n* subtopics $\{a_1, a_2, \ldots, a_n\}$, and a query suggestion technique *T*, we say that *T* has coverage equal to *c* if $n \cdot c$ subtopics match suggestions generated by *T*.

Explanatory example: A coverage of 0.8 for the top-10 suggestions generated for a query *q* having 5 subtopics means that 4 subtopics of *q* are covered by at least one suggestion.

Definition 5 (Effectiveness). Given a query topic A with *n* subtopics $\{a_1, a_2, \ldots, a_n\}$, and a query suggestion technique *T* generating *k* suggestions, we say that *T* has effectiveness equal to *e* if $e \cdot k$ suggestions cover at least one subtopic. An effectiveness of 0.1 on the top-10 suggestions generated for a query *q* means that only one suggestion is relevant for one of the subtopics of *q*.

The methodology just described has some net advantages. It is based on a publicly-available test collection which is provided by a well reputed third-party organization. Moreover, it grants to all the researchers the possibility of measuring the performance of their solution under exactly the same conditions, with the same dataset and the same reproducible evaluation criterion.

The experiments were conducted using the Microsoft RFP 2006 query log which was preliminary pre-processed by converting all queries to lowercase, and by removing stop-words and punctuation/control characters. The queries in the log were then sorted by user and timestamp, and segmented into sessions on the basis of a splitting algorithm which uses a time discriminator. We grouped into the same session all the queries issued by the same users in a time span of 30 minutes. Noisy sessions (i.e. likely performed by software robots) were removed. The remaining entries correspond to approximately 9M sessions. These were split into two subsets: training set with 6M sessions and a test set with the remaining 3M sessions. The training log was used to build the recommendation models needed by CG and QFG and used for performance comparison.

Instead, to implement our SS solution we extracted satisfactory sessions present in the training log and grouped them on the basis of the final query. Then, for each distinct final query its corresponding virtual document was built with the terms (with possible repetitions) belonging to all the queries of all the associated satisfactory sessions. Finally, by means of the Terrier search engine[1], we indexed the resulting 1,191,143 virtual documents. The possibility of processing queries on such index is provided to interested readers through a simple web interface[2]. The web-based wrapper accepts user queries, interact with Terrier to get the list of final queries (id of virtual documents) provided as top-k results, and

[1] See http://terrier.org/

[2] See http://searchshortcuts.isti.cnr.it

retrieves and visualizes the associated query strings.

We used Definition 2 to measure the similarity between the suggestions generated by SS, CG, and QFG for the first queries issued by a user during a satisfactory session belonging to the test set, and the final queries actually submitted by the same user during the same session. We conducted experiments by setting the number *k* of suggestions generated to 10, and, we chose the exponential function $f(m)=e^m$ to assign an higher score to shortcuts suggested early. Moreover, the length *t* of the head of the session was set to $\lceil n/2 \rceil$, where *n* is the length of the session considered. Finally, the metric used to assess the similarity between two queries was the Jaccard index computed over the set of tri-grams of characters contained in the queries [JJJ07], while the similarity threshold used was 0.9.

Due to the long execution times required by CG, and QFG for generating suggestions, it was not possible to evaluate suggestion effectiveness by processing all the satisfactory sessions in the test set. We thus considered a sample of the test set constituted by a randomly selected group of 100 satisfactory sessions having a length strictly greater than 3. The histogram in Figure 2 shows the distribution of the number of sessions vs. the quality of the top-10 recommendations produced by the three algorithms. SS produces recommendations having a quality score greater than 60% for 18 sessions out of 100. Moreover, in 36 cases out of 100, SS generates useful suggestions when its competitors CG and QFG fails to produce even a single effective suggestion. Indeed, CG and QFG can hardly propose good recommendation of less frequent queries, as discussed later. On average, over the 100 sessions considered, SS obtains an average quality score equal to 0.32, while QFG and CG achieves 0.15 and 0.10, respectively.

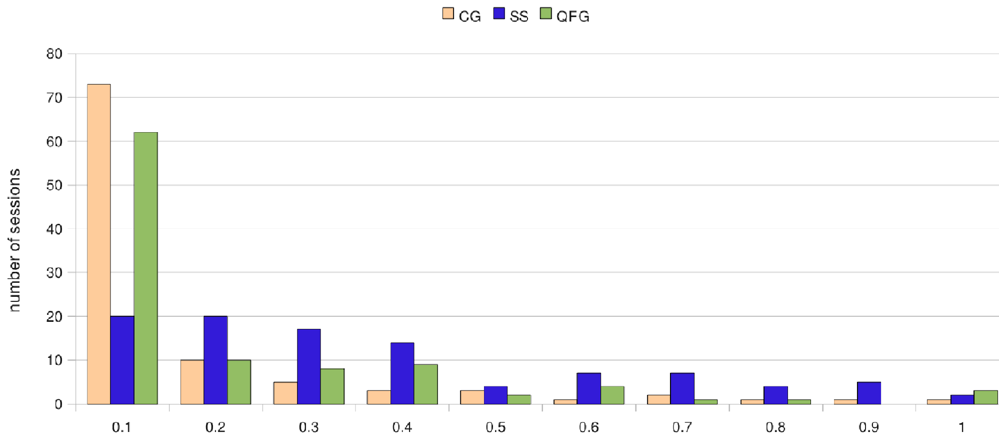

Figure 2 Query suggestion quality

The relevance of the suggestions generated by SS, CG, and QFG w.r.t. the TREC query subtopics was manually assessed. The evaluation consisted in asking assessors to assign the top-10 suggestions returned by SS, CG, and QFG to their related subtopic, for each given TREC query. Editors were also able to explicitly highlight that no subtopic can be associated with a particular recommendation. The evaluation process was blind, in the sense that all the suggestions produced by the three methods were presented to editors in a single, lexicographically ordered sequence where the algorithm which generated any specific suggestion was hidden. Given the limited number of queries and the precise definition of subtopics provided by NIST assessors, the task was not particularly cumbersome, and the evaluations generated by the assessors largely agree.

The histogram shown in Figure 3 plots the average coverage (Definition 4) of the associated subtopics measured on the basis of assessor's evaluations for the top-10 suggestions.

The comparison of the results returned by SS, CG, and QFG was performed by using all 50 TREC topics. In this Figure it cand be easily observed that SS outperforms its competitors. On 36 queries out of 50 SS was able to cover at least half of the subtopics, while CG only in two cases reached the 50% of coverage, and QFG on 8 queries out of 50. Moreover, SS covers the same number or more subtopics than its competitors in all the cases but 6. Only in 5 cases QFG outperforms SS in subtopic coverage (query topics 12, 15, 19, 25, 45), while in one case (query topic 22) CG outperforms SS. Furthermore, while SS is always able to cover one or some subtopics for all the cases, in 15 (27) cases for QFG (CG) the two methods are not able to cover any of the subtopics. The average fraction of subtopics covered by the three methods is: 0.49, 0.24, and 0.12 for SS, QFG, and CG, respectively.

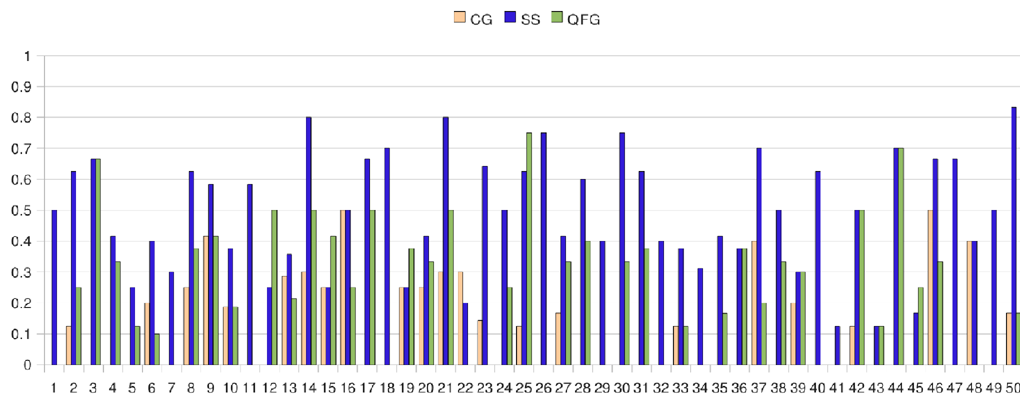

Figure 3 Recommendation Coverage

Figure 4 reports the effectiveness (Definition 5) of the top-10 suggestions generated by SS, QFG, and CG. Also considering this performance metric the Search Shortcuts solution results are the better ones. SS outperforms its competitors in 31 cases out of 50. The average effectiveness is 0.83, 0.43, and 0.42 for SS, QFG, and CG, respectively. The large difference measured is mainly due to the fact that both CG and QFG are not able to generate good suggestions for queries that are not popular in the training log.

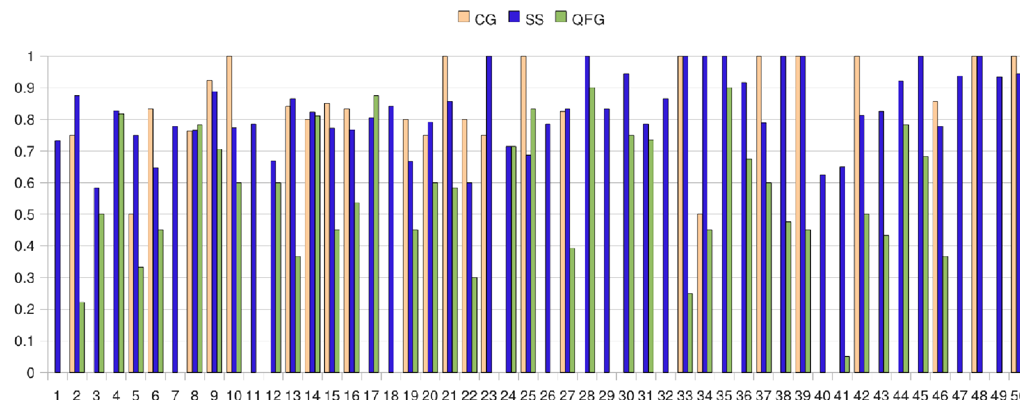

Figure 4 Recommendation Effectiveness

Regarding this aspect, the histogram in Figure 5 shows the average effectiveness of the top-10 suggestions returned by SS, CG and QFG measured for groups of TREC queries arranged by their frequency in the training log. SS remarkably outperforms its competitors. In fact, it is able to produce high-quality recommendations for all analyzed query categories, while CG

and QFG can not produce recommendations for unseen queries. Furthermore, while SS produces constant quality recommendations with respect to the frequency of the TREC queries, CG and QFG show an increasing trend in the quality of recommendations as the frequency of the TREC queries increases.

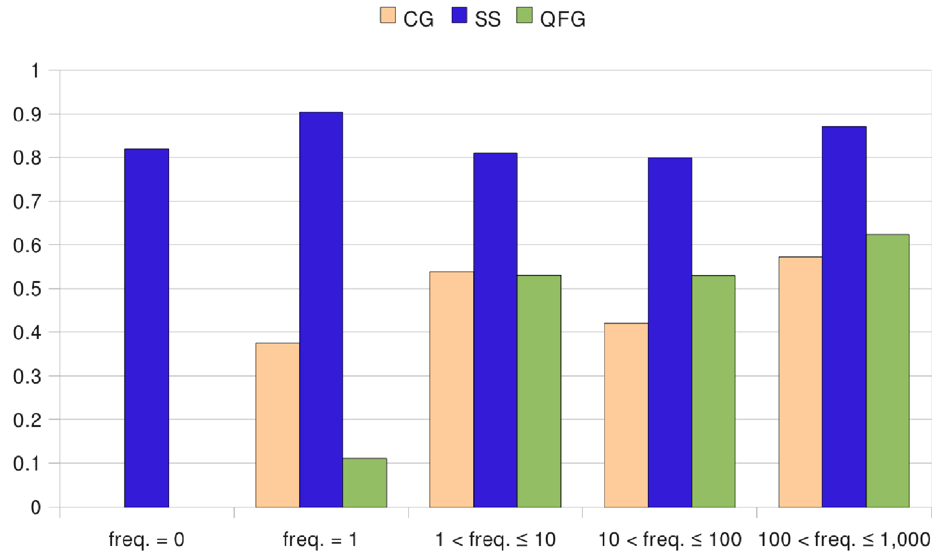

Figure 5 Average Recommendation Effectiveness on varying query frequencies

For this reason, we can assert that the SS method is very robust to data sparseness which strongly penalizes the other two algorithms, and is able to effectively produce significant suggestions also for singleton queries which were not previously submitted to the WSE. We recall that singleton queries account for almost half of the whole volume of unique queries submitted to a WSE. These are often the hardest to answer since they ask for “rare” or poorly expressed information needs. The possibility of suggesting relevant alternatives to these queries is more valuable than the one of suggesting relevant alternatives to frequent queries which express common and often easier to satisfy needs.

Table 1 Suggestion example

Subtopics for query: "diversity"	**Suggestions provided**		
	SearchShortcuts	**Cover Graph**	**Query Flow Graph**
1. How is workplace diversity achieved and managed? 2. Find free activities and materials for running a diversity training program in my office. 3. What is cultural diversity? What is prejudice? 4. Find quotes, poems, and/or artwork illustrating and	• diversity in education • diversity inclusion • cultural diversity • diversity test • accepting diversity • diversity poem • diversity skills • diverse learners presentation • picture of diverse children • advantages of	*no suggestions*	• accepting diversity • disparaging remarks • diverse world • diversity director • diversity poem • diversity test • minority & women • civil liberties • inclusion • gender and racial bias

promoting diversity.	diversity		

Finally, we report in Table 1 an example of the suggestion produced by the three algorithms for the query "diversity" which occurs only 27 times in training log, and the subtopics taken into consideration by the TREC diversity task.

1.2 Search shortcuts specification for ASSETS

Our proposal is to integrate into ASSETS the Search Shortcuts algorithm, which was proven to be very effective in the Web search scenario. The recommendation algorithm has a very simple interface, but it requires model training to be performed off-line. The model training should be repeated when a significant topic shift occurs in the query log, and therefore the algorithm is not well tuned to answer such new queries. Usually, the training is run during weekends when the search infrastructure has low query load. The software to be developed for the query log analysis and indexing is described in Section 3. Below we provide the specification of the core query recommendation service, which is also included in Deliverable D2.0.4.

Service Name	**Query Suggestion service**
Responsibility	*Query Suggestion*
Provided Interfaces	*Suggest*
Dependencies	*ASSETS Common, ASSETS Core, Query Logs, BM25F*

Interface Name	***Suggest***
Key concepts	*Queries, Shortcuts, Ranking*
Operation	*getSuggestions*

The interface *Suggest* hides the implementation details of the recommendation service. It provides a simple method named *getSuggestions*, which provides a set of recommended queries that the Graphical User Interface should present to the user. Below we show the corresponding class diagrams that provide a more detailed specification of this service.

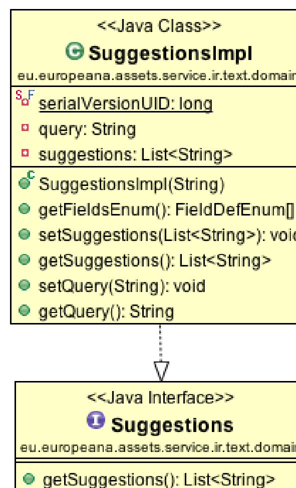

Figure 6 SuggestionsImpl class diagram

In Figure 6 we show the class diagram of the domain object Suggestions which is used to store the set of queries suggested to the user. The object is used to encapsulate for a particular query (e.g., *Pablo Picasso*), the suggestions for the query *ranked for relevance* (e.g., *Pablo Picasso life, Guernica, Cubism …)*.

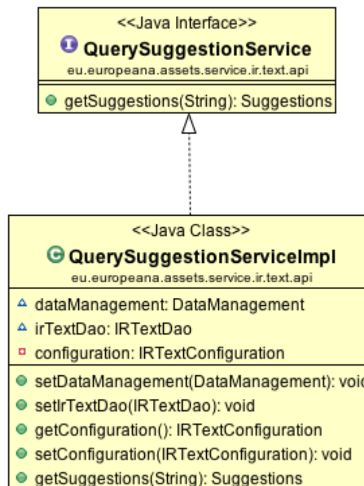

Figure 7 QuerySuggestionServiceImpl class diagram

In Figure 7 we show the class diagram of the query suggestion service implementation, which exploits an index of virtual documents to provide recommendations in response to a given query. For each received query, the Query Suggestion Service produces a Suggestions Object containing the ranked list of suggestions.

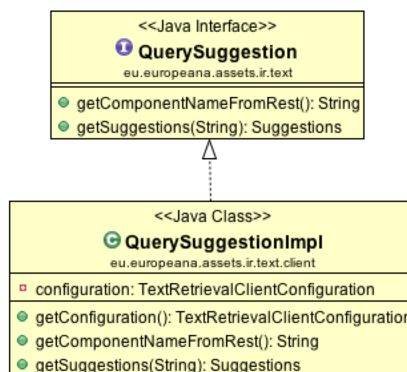

Figure 8 QuerySuggestion client class diagram

Finally, in Figure 8, we illustrate the class diagram of the query suggestion client and its implementation. The client defines how the other components of the ASSETS platform will interact with the query recommendation component. Its task is to receive from the other components the queries, then submits the queries to the query suggestion service and returns the suggestions to the applicants. If needed, this service could be exposed externally as a specific API-call available to third-parties.

2. T2.2.2: Metadata-based Ranking

Task T2.2.2 deals with effective techniques for ranking metadata objects in the Europeana context. The Europeana query log analysis that we conducted thanks to the tools developed within task T2.2.3, and the literature on multi-field document retrieval, suggests that the ranking function currently adopted by Europeana can be improved. The results of such query log analysis are illustrated in Section 3. In the following, we first review the state of the art in multi-field document retrieval, and we provide a specification of an advanced metadata based ranking to be included into the ASSETS engine. In particular, we propose to adopt the BM25F ranking function, and to exploit machine learning algorithms to best tune its parameters. The learning step exploits the output of query log processing tools, which is described in Section 3.

2.1 Description of BM25F

Ranking functions are one of the most important components of a document retrieval system. A ranking function answers to the question "what is the relevance of a document *d* for the user query *q*?". Therefore, the goodness of the ranking function adopted determines the quality of the results returned.

The probably most widely used ranking function is **BM25** (RW94), and it is still considered the most relevant baseline. Grounded in the probabilistic language modelling theory, BM25 was designed as a non-linear combination of three important document attributes: term frequency, document frequency, and document length. Even if originally, Web documents where considered as composed of few fields, such as body, title, URL, BM25 uses a flat representation of a document, where its fields are simply concatenated into a single textual description. But we know that Europeana documents have a very rich structure and they are described by means of many fields, each possibly playing a different role in the document retrieval task.

BM25F (RZT04) is an extension of BM25 that exploits a document description having multiple fields, and it is still a non-linear function, thus capable of modelling non-trivial factors that determine the relevance of a document for a given query. Given a document *d*, having fields *F*, and a query *q*, BM25F produces a score of the document computed as follows:

$$\mathrm{BM25F}(q,d) = \sum_{t \in q} \mathrm{TF}(t,d) \ \cdot \ \mathrm{IDF}(t)$$

where *TF(t,d)* measures the importance of term *t* for document *d*, and *IDF(t)* is the usual inverse document frequency measuring the importance of term *t* in the whole collection of document. Let *df(t)* be the document frequency of term *t*, i.e. the number of documents in the collection containing the term *t*, the IDF function is defined as follows:

$$\mathrm{IDF}(t) = \log \frac{N - df(t) + 0.5}{df(t) + 0.5}$$

More precisely, BM25 and BM25F adopt a term frequency saturation function which accounts for the fact that finding twice the term *t* in *d*, is not twice as surprising (i.e. relevant) as finding the same term once. We can update the BM25F formula as follows:

$$\mathrm{BM25F}(q,d) = \sum_{t \in q} \frac{TF(t,d)}{k + TF(t,d)} \cdot \ IDF(t)$$

The parameter *k* realizes the saturation: the larger *k*, the more important is the variation of term frequency. As we mentioned above, BM25F takes into account the multiple fields of the document, and this is done when computing the term frequency component *TF(t,d)*. Indeed, the term frequency is computed independently for each field, and a linear combination is computed as follows:

$$\mathrm{TF(t,d)} = \sum_{f \in F} w_f \cdot \mathrm{TF}(t,d,f)$$

where w_f is a weight (or boosting factor) for the field *f*, and *TF(t,d,f)* is the frequency-based contribution of term *t* in the field *f* of document *d*. Finally, the frequency *TF(t,d,f)* is normalized on the basis of the length of the document field *f*:

$$\mathrm{TF}(t,d,f) = \frac{\mathrm{occurs}(t,d,f)}{1 + b_f \left(\frac{l_{d,f}}{l_f} - 1 \right)}$$

where *occurs(t,d,f)* is the actual number of occurrences of term *t* in the field *f* of document d, $l_{d,f}$ is the length of the field *f* of document *d*, l_f is the average length of field *f* across the whole collection, and b_f is a model parameter tuning the impact of document length normalization.

BM25F can be considered the state of the art of ranking functions in multi-field document retrieval. However, its accuracy depends on the ability to fine-tune its parameters k, w_f, b_f. Note that for *|F|* fields there are *2|F|+1* parameters to be tuned.

Before describing how to fine-tune these parameters automatically, we add some comments on the Lucene ranking function. Lucene is a popular open source search engine, being at the core of SOLR, which is the search infrastructure adopted by Europeana and ASSETS. Lucene is able to rank multi-field documents by exploiting the following scoring function:

$$LUCENE(q,d) = \sum_{t \in q} \sum_{f \in F} w_f \cdot \sqrt{\mathrm{FREQ}(t,d,f)} \cdot IDF(t)$$

Notice that the frequency of each term is saturated with the square root function, and that the score is a linear combination of a per-field contribution. The effect is that a document matching a single query term over several fields could have a larger score than a document matching several query terms in one field only, and this may significantly decrease the retrieval precision compared to BM25F, as showed in (PA+10).

2.2 Learning To Rank

In the previous section, we have described BM25F that can be considered as a non-trivial combination of many relevance measures to a given query, one for each field of the document. We have also mentioned that BM25F has a number of parameters that need to be fine tuned in order to achieve an optimal ranking function. This parameter search task consists in optimizing some cost function that measures the goodness of the rankings.

This learning problem is strictly related to another problem arising in modern information retrieval systems. Traditionally, only a small number of features have been used to devise ranking functions. BM25F is among these traditional methods, since it includes simply term frequency, inversed document frequency, and document length. More recently, a number of additional features have proved their utility. They include structural features such as title, anchor text, URL of a Web page, and also query-independent features such as PageRank, URL length, and many others. The large number of features makes it impossible to empirically fine-tune a ranking function accounting for all of them. This trend calls for new supervised methods for building effective ranking functions, which we call *Learning to Rank* algorithms.

Existing methods fall into two categories, "point-wise training" and "pair-wise training". In the former, single documents are labelled in relation to a given query, e.g. relevant or irrelevant. In the latter, the labelling consists pair-wise preferences, e.g. document *a* is more relevant than document *b*. Both allow to exploit click-through data, which are a gold-mine of implicit user relevance feedback.

Click-through data can be thought as triples (*q, r, c*) where *q* is a query, *r* is the ranked list of documents appearing in the result page presented to the user, and c is a subset of r containing the list of documents the user clicked on (sorted by timestamp). Pair-wise preferences can be simply computed by exploiting such data. Given a ranked list of results r = r_1, r_2, … , r_{10} , and the user clicks on some of them, let for example c = r_2, r_6 and r_7, we can state that:

1. Clicked documents r_2, r_6, and r_7 are relevant for query *q*;
2. The document with rank r_2 is more relevant than document with rank r_1, and document with rank r_6 more than document with rank r_5;
3. The document with rank r_6 is more relevant than documents with ranks r_1, … , r_5

Hence a *relevance* relation R on pairs of documents can be devised: for a query *q* and a collection of ranked documents *D* = {r_1, r_2, … , r_m} if a document r_i is more important than a document r_j the couple (r_i, r_j) is in R. In the previous example, we would have R = ((r_2, r_1), (r_6, r_5), (r_6, r_4), (r_6, r_3), (r_6, r_2), (r_6, r_1)).

In the following we review some important learning to rank approaches that can exploit such pair-wise preferences to optimize the ranking function parameters.

2.2.1 Ranking SVM

In (HGO00), a method called Ranking SVM is proposed. The idea is to transform pair-wise preferences into a classification problem solved with a support vector machine.

First assume that the relatedness of a query *q* to a document *d* can be expressed as a function of a vector of *N* features *x*, thanks to a mapping function $\Phi(q,d)=x$. The vector *x* may be very complex, containing hundreds of features, e.g. BM25 computed on every single document field. The training set consists in a collection of pair relations in the form $x_i \succ x_j$ meaning that x_i is more important than x_j. A ranking function *f* should be such that: $x_i \succ x_j \Leftrightarrow f(x_i)>f(x_j)$. Second, we assume that *f* is a linear function of *x* defined as follows:

$$f_w(x) = \langle w, x \rangle$$

where *w* is a set of weights, and $\langle \cdot , \cdot \rangle$ stands for the inner product. We can thus obtain:

$$x_i \succ x_j \Leftrightarrow \langle w, x_i - x_j \rangle > 0$$

In principle, e.g. due to noise in the training set, there might not exist a function *f*, or equivalently a weight vector *w*, such that the above inequality is satisfied for every instance of the training set. The goal is to find *w* such that maximizes the number of training instances for which the corresponding inequality is fulfilled. This is done by introducing non negative slack variables $\xi_{i,j}$ and reformulating the learning to rank problem into an optimization problem:

$$\min_{w} M(w) = \frac{1}{2}\|w\|^2 + C\sum \xi_{i,j}$$

subject to: $\langle w, x_i - x_j \rangle \geq 1 - \xi_{i,j}, \forall x_i \succ x_j$ and $\xi_{i,j} \geq 0$.

This formulation is equivalent to an SVM classification problem formulation, where the goal is to build a classifier recognizing correctly ordered versus incorrectly ordered document pairs. The SVM classifiers can thus be turned in a ranking function. Let ω be the solution of the SVM classification problem, the score of a document *d* relative to a query *q* is computed as follows:

$$f_\omega(q, x) = \langle \omega, \Phi(q, x) \rangle$$

In HGO00 it is also shown that this SVM formulation allows to adopt any other non-linear kernels, and not necessarily the linear inner product. Although, a non linear kernel may produce a less efficient ranking function.

2.2.2 RankNet

In BS+05, a similar approach based on neural networks is presented. As with Ranking SVM, the training is done with preference pairs. Given $x_i \succ x_j$, we denote with s_i and s_j their score $f(x_i)$ and $f(x_j)$. Then, the probability that x_i should be ranked higher than x_j in the result list, is computed with a sigmoid function:

$$P_{ij} \equiv \frac{1}{1 + e^{-\sigma(s_i - s_j)}}$$

The sigmoid function is widely used in neural networks and it is known to produce good probability estimates. The predicted probability is compared against the target probability Π_{ij} which is learnt from the training. A cross entropy cost function *C* is introduced:

$$C = -\Pi_{ij} \log P_{ij} - (1 - \Pi_{ij}) \log(1 - P_{ij})$$

Since we assume that our training pairs are in the form $x_i \succ x_j$ then Π_{ij} is always equal to 1, i.e. document x_i has (observed) 100% probability of being more important than document x_j. Thus the model simplifies to:

$$C = \log\left(1 + e^{-\sigma(s_i - s_j)}\right)$$

In the above, we did not specify the ranking function *f*, but we can assume it is a function of some model parameters *w*, similarly to what we did with Ranking SVM, without fixing any specific ranking function. Our goal is to find the model parameter that minimizes the cost function *C*. The cost can be optimized via gradient descent by taking the derivative of *C* with respect to each parameter of the model w_k:

$$\frac{\partial C}{\partial w_k} = \sum_{ij} \frac{\partial C}{\partial s_i}\frac{\partial s_i}{\partial w_k} + \frac{\partial C}{\partial s_j}\frac{\partial s_j}{\partial w_k}$$

Therefore, to update the model parameters it is needed to be able differentiate the cost function with respect to the scores, and more importantly, to differentiate the ranking function *f* with respect to each model parameter. Not only it is possible to plug the above derivative in a neural network, but also to exploit a gradient descend considering BM25F as a scoring function [TZ+06].

2.2.3 LambdaRank

RankNet optimizes a specific cost function to improve the scoring function of the retrieval system. The assumption is that the optimization measure matches the target measure, but unfortunately typical IR costs are non differentiable everywhere and require sorting by model score, which is itself a non-differentiable operation. In other words, we would prefer to specify how the rank order of documents needs to be changed, rather than tuning a ranking function leading to the desired ranking order. ***LambdaRank*** [BRL06] exploits the fact that a neural network only needs to know the gradients of the optimized function w.r.t. the model parameters, i.e. the IR measure w.r.t. the document scores, and does not need to know the cost function itself. The gradients can be defined by specifying rules describing how swapping the position of two documents, after sorting them by their score relative to a given query, affects the IR quality measure. In conclusion, we do not need to find a derivable function approximating a target IR evaluation measure, but it is sufficient to define the how a change of position in a ranked list effects the evaluation measure.

Let's first introduce the one of the most commonly used IR quality measure. The *Normalized Discounted Cumulative Gain* (NDCG) of a result list for a query *q* is:

$$\mathrm{NDCG} = N_i \sum_{j=1}^{L} \frac{(2^{r_j} - 1)}{\log(1 + j)}$$

where r_j is a relevance level of the *j*-th result, *L* is a truncation level determining the number of results evaluated, and N_i is a normalization constant such that a perfect ordering would achieve NCDG=1.

The LambdaRank gradient λ_j, is defined to be a smooth approximation to the gradient of the evaluation measure w.r.t. the score of the document ranked at position *j*, that is an approximation of how the ranking quality would change by moving the document from position *j*.

We report the LamdaRank gradient discussed in [BRL06], but it has been shown that LamdaRank gradients can be devised for many other IR evaluation function [DSB09]. The proposed gradient is a combination of the derivative of the RankNet cost scaled by the NDCG gain resulting from swapping two documents. In detail, assume that document *i* and *j* have scores s_i and s_j, relevance level r_i and r_j, and let o_{ij} be the cost difference s_i-s_j. Note that differently from RankNet we do not assume that documents pair are always such that $x_i \succ x_j$. Then, the RankNet cost is equivalent to:

$$C_{ij} = -S_{ij} o_{ij} + \log\left(1 + e^{S_{ij} o_{ij}}\right)$$

where S_{ij} equals 1 if $l_i > l_j$, and -1 vice versa. Its derivative according to the score difference is:

$$\partial C_{ij}/\partial o_{ij} = \partial C_{ij}/\partial s_{ij} = -S_{ij}/\left(1 + e^{S_{ij}o_{ij}}\right)$$

The LamdaRank gradient is then defined as:

$$\lambda_{ij} \equiv S_{ij}\left|\Delta NDCG\,\frac{\partial C_{ij}}{\partial o_{ij}}\right| = S_{ij}\left|N(2^{l_i} - 2^{l_j})\left(\frac{1}{\log(1+r_i)} - \frac{1}{\log(1+r_j)}\right)\left(\frac{1}{1+e^{S_{ij}o_{ij}}}\right)\right|$$

where *N* is the reciprocal of the maximum DCG for the given query, and r_i and r_j are the rank position of the document *i* and *j*. Note that the sign of λ_{ij} depends only on S_{ij}, i.e. on their relative importance.

The LamdaRank gradient λ_i for a single document *i* is computed by marginalizing over all the available pair-wise gradients λ_{ij} as follows:

$$\lambda_i = \sum_j \lambda_{ij}$$

To sum up, we are training a neural network to score document with respect to a given query. The neural network optimizes a cost function which is a smooth approximation of the NDCG, and which can be extended to other IR evaluation functions. In particular, it is sufficient to specify how the cost changes when the order of the ranked document changes, by introduced the so called LambdaRank gradients. These gradients are finally plugged into a neural network learning process.

The authors of [SB09] show that it is possible to achieve interesting result by training such a neural network on the signals given by the BM25 from each single field. One disadvantage of the approach, is that the generated ranking function is a neural network from which it is not easy to understand the importance of each attribute. Also, implementing a neural network as a ranking function may not be an efficient solution.

2.2.4 LineSearch or direct optimization of BM25F

The above learning to rank approaches exploit BM25 by using the BM25 rank on each document field as a feature of the document/query pair. Indeed, they are able to find possibly non-linear function that may resemble BM25F, but they cannot be considered as a tuning process of the BM25F parameters. An interesting approach for Europeana would be to find the best parameter set for BM25F, and this can be done with a greedy exploration of the parameter space. An interesting approach based on the *line search* algorithm is presented in [TZ+06].

The advantage of the line search algorithm is that it can be adapted so as to exclude any dependency on the gradients of the cost function to be optimized, and rather to consider actual IR evaluation functions such as NDCG.

The algorithm works as follows. Given an initial point in the parameter space, a search along each co-ordinate axis is performed by varying one parameter only and keeping fixed the others. For each sample point, the NDCG is computed, and the location corresponding to the best NDCG is recorded. Such location identifies a promising search direction. Therefore, a line search is performed along the direction from the starting point to the best NDCG location. If the parameter space has dimension *k*, we need to perform *k+1* line searches to complete an iteration, or epoch, and possibly move to an improved solution. The new solution is then used as the starting point of the next iteration, and the sampling scale is reduced. This iterative process continues until no improvement is found, or a maximum number of epochs is reached.

The authors of [TZ+06] show that it is thus possible to tune the parameters of BM25F in

order to obtain better ranking results.

2.3 Metadata based ranking for ASSETS

We propose to replace the Lucene ranking function with BM25F. The latter has proven to be for effective, and has efficiency guarantees compared to neural network based approaches. The new ranking service should exploit a learning process where its parameters are fine-tuned on the basis of click through information contained in the query logs. The learning process can be costly, depending on the size of the query log and on the size of the collection. But this training is performed off-line, and its cost is justified by the improvement in the efficacy of the search system. We plan to devise a ranking function that best suits Europeana actual users, starting from an evaluation of some of the above described techniques.

Service Name	**BM25F Scoring function**
Responsibility	*(i) Search & Retrieval, (ii) Learn from query logs BM25F's parameters*
Provided Interfaces	*BM25F*
Dependencies	*ASSETS Common, ASSETS Core, Query Logs*

Interface Name	***BM25F***
Key concepts	*Queries, Learning to rank, Ranking, Query Logs*
Operation	*search , learning to rank*

The service this provides to main operation: *search* and *learning to rank*.

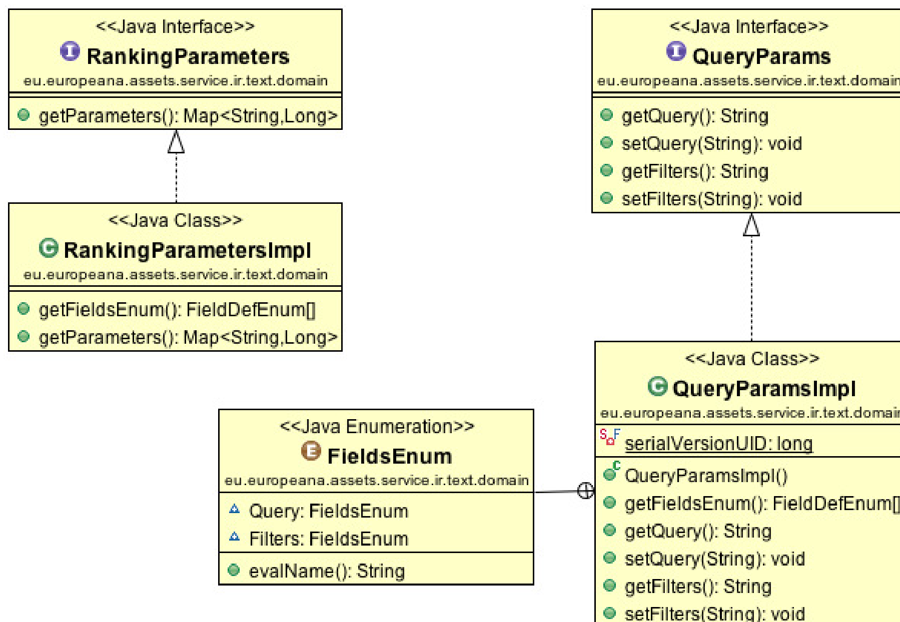

Figure 9 Ranking parameters class diagram

Figure 9 shows the domain objects for the service:

- **QueryParams** models the user query: it contains the text of the query and the filters

possibly added by the user to refine the query (for example TYPE:IMAGE filters only documents containing images).

- **RankingParameters** models the set of free parameters for the ranking function. The method getParameters() returns a dictionary where, for each parameter, there is the value optimizing the quality of the ranking function, learned from the query logs.

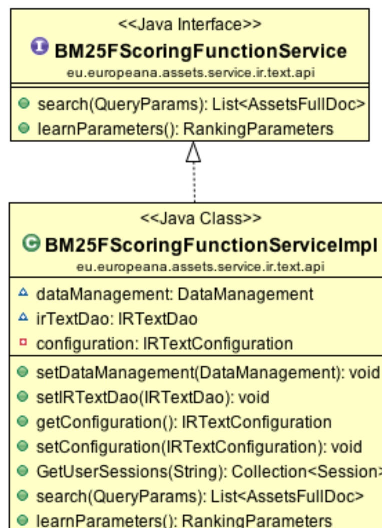

Figure 10 BM25F Scoring function class diagram

Figure 10 shows the class diagram of the BM25F scoring function implementation. The service allows to process a query using the BM25F scoring function (method search) and returns a list of *AssetsFullDoc*. Furthermore, the service exposes a method to retrieve a good tuning for the parameters in the scoring function (that the developer has to set in the SOLR configuration file).

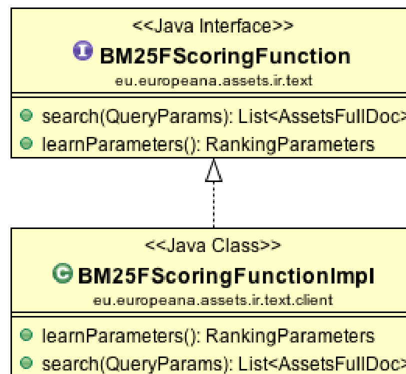

Figure 11 BM25F Scoring function client class diagram

In Figure 11, we show the class diagram of the BM25F client and its implementation. The client defines how the other components of the ASSETS platform will interact with the BM25F component. Its task is to receive from the other components the queries (encapsulated in a QueryParams object, that may contain also filters and other parameters), then submits the queries to the SOLR engine and returns the results to the applicants (function search()). Furthermore, the client also exposes a method to require the BM25F's parameters learning process (method learnParameters()). If a user invokes this method, (s)he obtains a list of parameters with their respective tuned values (encapsulated in a RankingParameters object).

2.3.1 BM25F Solr Plugin

For performance reasons, we decided to implement the BM25F ranking function inside Solr.

The ranking function needs to access several values that can be found only in the document index, that are:

- The field term frequency, i.e., how many times a term occurs in a field of a document (e.g., "description");
- The inverse document frequency, i.e., how many documents contain a specified term;
- The average length of a field, i.e., the average length (in terms) of a field computed on the whole collection.

We integrated the ranking function as a Solr *plugin*, without touching the code core. This will allow to update Solr to new versions without applying any patch. The admin can import the plugin from the Solr's configuration file (solrconfig.xml) by simply adding this few lines to the file:

```xml
<queryParser name="bm25f"
class="bm25f.parser.BM25FQParserPlugin">
<float name="k1">1.0</float>
<str name="mainField">text</str>
<lst name="averageLengthFields">
<float name="text">500</float>
<float name="title">20</float>
<float name="description">300</float>
<float name="YEAR">4</float>
<float name="date">10</float>
</lst>
<lst name="fieldsBoost">
<float name="text">1.0</float>
<float name="title">5.0</float>
<float name="description">3.0</float>
<float name="YEAR">1.0</float>
<float name="date">1.0</float>
</lst>
<lst name="fieldsB">
<float name="text">0.75</float>
<float name="title">0.75</float>
<float name="description">0.75</float>
<float name="YEAR">0.75</float>
<float name="date">0.75</float>
</lst>
</queryParser>
```

The configuration file allows the admin to change the parameters of the ranking function by using his domain knowledge or by calling the learnParameters() method. The customizable parameters are:

- K1, the saturation factor (default 1.0)
- fieldBoost, containing the boosts to apply on the various fields;
- fieldB, containing the boosts to apply to the length of a field;
- averageLengths, the average lengths of the fields, because solr does not have this data. The method learnParameters will also return an estimation of these lengths.

Once the plugin has been plugged in, the BM25F ranking function can be called by simply adding to the get request the parameter defType=bm25f, e.g. :

```
http://mysolrmachine:8983/solr/select/?defType=bm25f&q=leonardo%20da%20vinci
```

3. T2.2.3: Text Indexing and Retrieval

3.1 Query Log Analysis

A query log keeps track of historical information regarding past interactions between users and the retrieval system. It usually contains tuples $\langle q_i, u_i, t_i, V_i, C_i \rangle$ where for each submitted query q_i, the following information is available: i) the anonymized identifier of the user u_i, ii) the submission timestamp t_i, iii) the set V_i of documents returned by the search engine, and iv) the set C_i of documents clicked by u_i. Therefore, a query log records both the activities conducted by users, e.g. the submitted queries, and an implicit feedback on the quality of the retrieval system, e.g. the clicks.

Here, we consider a query log coming from Europeana portal, relative to the time interval ranging from August 27, 2010 to February, 24, 2011. This is a six months worth of users' interactions, resulting in 1,382,069 distinct queries issued by users from 180 countries (3,024,162 is the total number of queries). We pre-processed the entire query log in order to remove noise (e.g., stream of queries submitted by software robots instead of humans).

It is worth noticing that 1,059,470 queries (i.e., 35% out of the total) also contain a *filter* (e.g., YEAR:1840). These filters are used to implement *faceted search*. Users can filter results by *type*, *year* or *provider* simply by clicking on a button, so it is reasonable that they try to refine retrieved results by applying a filter, whenever they are not satisfied. Furthermore, we find that users prefer filtering results by type, i.e., images, texts, videos or sounds. Indeed, we measured that 20% of the submitted queries contains a filter by type. This is a proof of the skilfulness of Europeana users and their willingness to exploit non trivial search tools to find the desired contents. This also means that advanced search aids, such as query recommendation, would be surely exploited.

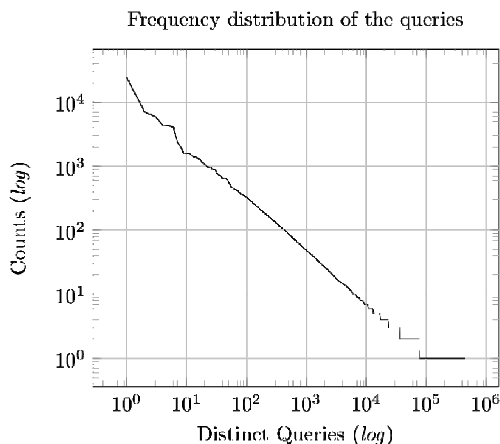

Figure 12 Query Frequency Distribution

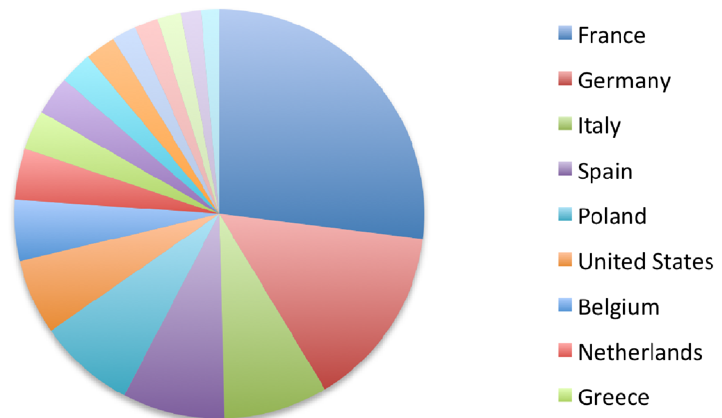

Figure 13 Country query distribution

Similarly to Web query log analysis [SM+99], we discuss two aspects of the analysis task: i) an analysis on the *query set* (e.g., average query length, query distribution, etc.) and ii) a higher level analysis of *search sessions*, i.e., sequences of queries issued by users for satisfying specific information needs.

3.1.1 Query Analysis

First we analyzed the load distribution on the Europeana portal. An interesting analysis can be done on the queries themselves. Figure 12 shows the frequency distribution of queries. As expected, the popularity of the queries follows a power-law distribution ($p(x) \propto k \cdot x^{-\alpha}$), where x is the popularity rank. The best fitting α parameter is α= 0.86, which gives a hint about the skew in the frequency distribution. The larger α the larger is the portion of the log covered by the top frequent queries. Both [M00] and [BG+07] report a much larger α value of 2.4 and 1.84 respectively from a Excite and a Yahoo! query log.

Such small value of α means that the most popular queries submitted to Europeana do not account for a significantly large portion of the query log. Indeed, since Europeana is strongly focussed on the specific context of cultural heritage, its users are likely to be more skilled and therefore they tend to use a more diverse vocabulary.

In addition, we found that the average length of queries is 1.86 terms, which is again a smaller value than the typical value observed in Web search engine logs. We can argue that the Europeana user has a more rich vocabulary, with discriminative queries made of specific terms.

Figure 13 shows the distribution of the queries grouped by country. France, Germany, and Italy are the three major countries accounting for about the 50% of the total traffic of queries submitted to the Europeana portal.

Figure 14 reports the number of queries submitted per day. We observe a periodic behaviour over a week basis, with a number of peaks probably related to some Europeana dissemination or advertisement activities. For example, we observe several peaks between the 18th and the 22nd November, probably due to the fact that, in those days, Europeana announced to have reached a threshold of 14 million of indexed documents.

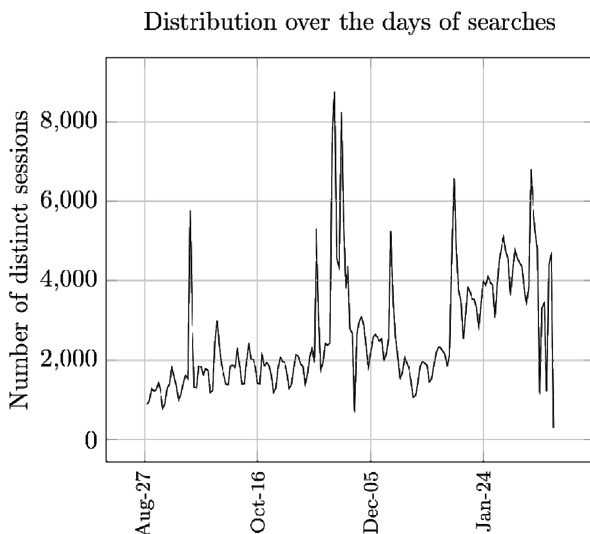

Figure 14 Daily Query Frequency Distribution

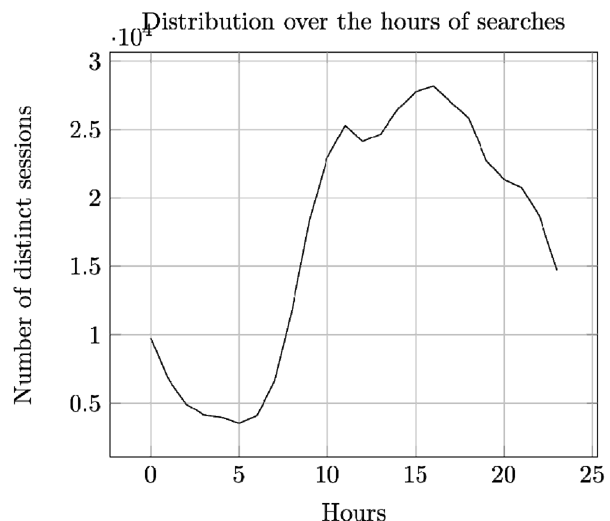

Figure 15 Hourly Query Frequency Distribution

Figure 15 shows the load on the Europeana portal on a per hour basis. We observe a particular trend. The peak of load on the Europeana portal is in the afternoon, between 15 and 17. It is different from commercial Web search engines where the peak is reached in the evening, between the 19 and the 23 [BJ+04]. A possible explanation of this phenomenon

could be that the Europeana portal is mainly used by people working in the field and thus, mainly accessed during working hours. From the other side, a commercial Web search engine is used by a wider range of users looking for the most disparate information needs and using it through all the day.

3.1.2 Session Analysis

To fully understand user behaviour, it is important to analyze also the sequence of queries she submits. Indeed, every query can be considered as an improvement of the previously done by the user to better specify her information need.

Several techniques have been developed to split the queries submitted by a single user into a set of sessions [BB+08,JK08,LO+11]. We adopted a very simple approach which has proved to be fairly effective [SM+99]. We exploit a 5 minutes inactivity time threshold in order to split the stream of queries coming from each user. We assume that if two consecutive queries coming from the same user are submitted within five minutes they belong to the same logical session, whereas if the time distance between the queries is larger, the two queries belong to two different interactions with the retrieval system.

By exploiting the above time threshold, we are able to devise 404,237 sessions in the Europeana query log. On average a session lasts about 276 sec, i.e., less than 5 minutes, meaning that, under our assumption, Europeana's users complete a search activity for satisfying an information need within 5 minutes. The average session length, i.e., the average number of queries within a session, is 7.48 queries. This number of queries is a interesting evidence that the user is engaged by the Europeana portal, and she is willing to submit many queries to find the desired result.

Moreover, we distinguish between ***successful*** and ***unsuccessful*** sessions. According to [BC+09], a session is supposed to be successful if its *last* query has got a click associated. To this end, we find 182,280 occurrences of successful sessions in the Europeana query log, that is about 45% of the total. We notice that in [BG+07] it was observed a much larger fraction of successful sessions, about 65%.

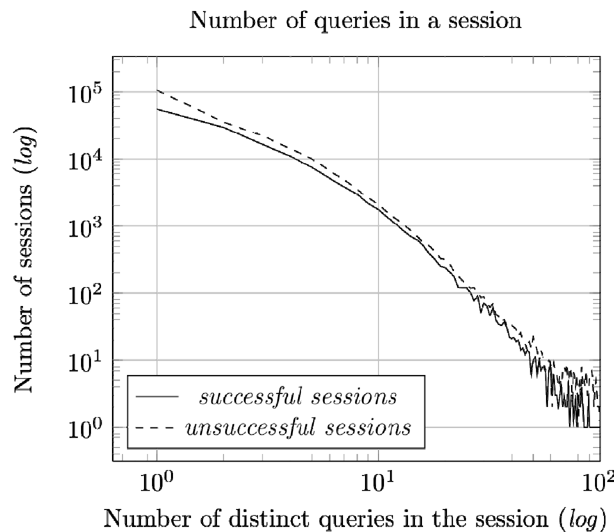

Figure 16 Successful vs. Unsuccessful sessions' length distribution

Figure 16 shows the distributions of session lengths, both for successful and unsuccessful sessions. On the x-axis the number of queries within a session is plotted, while on the y-axis

the frequencies, i.e., how many sessions to contain a specific number of queries are reported. We expect successful sessions contain on average less queries than unsuccessful ones, due to the ability of the retrieval system to return early high quality results in successful session. The fact that the session length distributions are very similar, suggests that high quality results are not in the top pages, and that the Europeana ranking can be improved in order to present interesting results to the user earlier, thus reducing the successful session length with a general improvement of the user experience.

	Europeana	***Web Search Engines***
avg. query terms	1.86	2.35 [M00] 2.55 [SM+99]
query distribution (i.e., power-law's α)	0.86	2.40 [M00] 1.84 [BG+07]
avg. queries per session	7.48	2.02 [SM+99]
% of successful sessions	45	65 [BM+10]

Table 2 Comparison of Europeana and Web users

Finally, in Table 2 we summarize some statistics extracted both from the analysis of the Europeana query log as well as from general purpose Web Search Engines historical search data.

3.2 Indexing and retrieval of query log information for ASSETS

The goal of task T2.2.3 is to devise a set of query log processing tools needed by other services, in particular for extracting user behavioural patterns needed for improving the ASSETS ranking function and for providing the model used by the query recommendation service.

Service Name	**Query Log Indexing**
Responsibility	Cleaning and indexing of query log information for learning.
Provided Interfaces	*BuildIndex, GetUserSession, GetQueryPopularity*
Dependencies	ASSETS Common, ASSETS Core, Query Logs

Interface Name	**QueryLogIndexing**
Key concepts	Session Detection, Data cleaning
Operation	Analysis and indexing of query log index

This includes non-trivial activities such as query log cleaning, analysis and indexing accessible by any ASSETS component via the *QueryLogIndexing* service interface.

In Figure 17 we show the class diagram modelling query log data and session information. *QueryLogRecord* describes the object modelling a record in the query log. It representsRepresents a user interaction with the portal (submitting a query, clicking on a results, etc.). Session models a user query session. Queries by the same user are split in different sessions on the basis of the time interval between consecutive queries.

Figure 17 QueryLogRecImpl class diagram

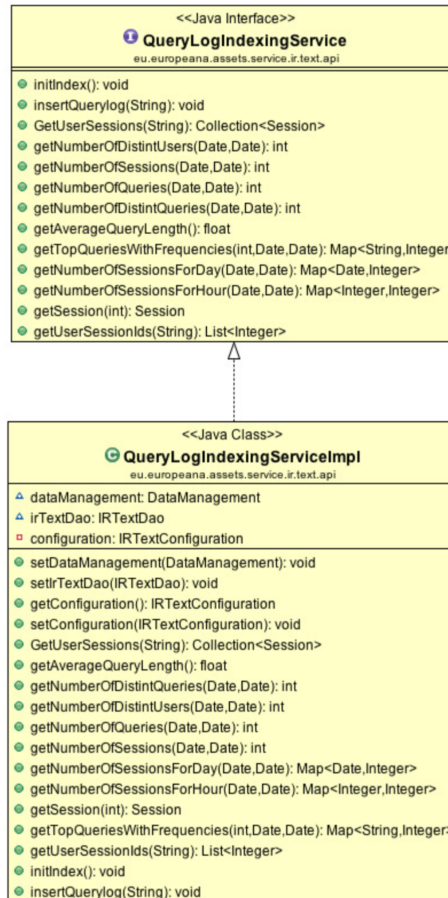

Figure 18 QueryLogIndexing class diagram

Finally, in Figure 18 we report the class diagram of the query log indexing service. The service accomplishes the task of retrieving the query log, splitting the log into user sessions, computing relevant statistics and other info to be used by the query suggestion service and by the metadata based ranking. More in detail:

- *initIndex* and *insertQueryLog* allow to create a *newIndex*, and to add new query log.
- *getUserSessions, getNumberOfDistintUsers, getNumberOfSessions, getNumberOfQueries, getNumberOfDistintQueries, getAverageQueryLength, getTopQueriesWithFrequencies, getNumberOfSessionsForDay, getNumberOfSessionsForHour, getSession , getUserSessionIds* allow to retrieve statistics on the indexed query logs (also filtering on the date)

Notice that the service executes implicitly the removal of noise from the query log, e.g. bots interactions.

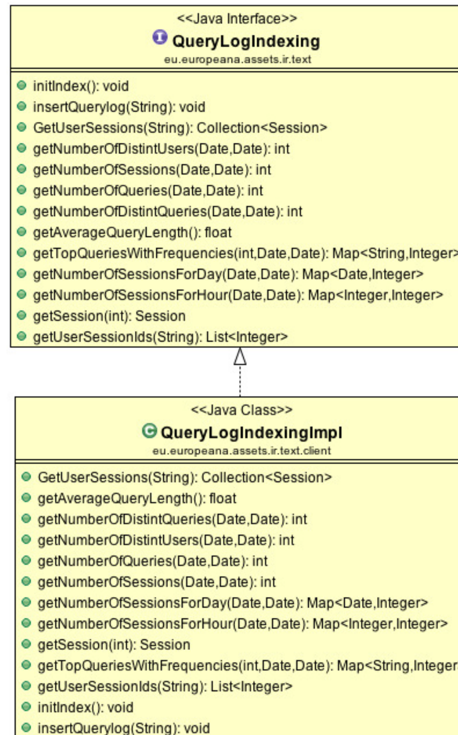

Figure 19 QueryLogIndexing client class diagram

In Figure 19, we illustrate the class diagram of the Query Log Indexing client and its implementation. The client defines how the other components of the ASSETS platform will interact with the Query Log Indexing component. This component allows the users to obtain useful statistics on the user searches. More in detail, a user can retrieve:

- the session ids by a particular user (getUserSessions);
- the distinct number of users (getNumberOfDistintUsers);
- the distinct number of sessions (getNumberOfSessions);
- the number of queries (getNumberOfQueries);
- the number of distinct queries (getNumberOfDistintQueries);
- the average length of a query in number of terms (getAverageQueryLength);
- the number of sessions per day (getNumberOfSessionsForDay);
- the number of sessions per hour (getNumberOfSessionsForHour) ;
- the most frequent queries (getTopQueriesWIthFrequencies);
- the sessions associated with a specific IP address (GetUserSessionsIds);
- a particular session object (getSession).

The user can also refine his search by specifying in the method a starting and an ending date. In this way, the statistics will concern only the queries submitted in the specified time period. Finally, the user can build a new query log index (initIndex), or index a new file containing query log records from Europeana (insertQueryLog).

4. Conclusions

This deliverable details the ASSETS services that are developed by the CNR team within tasks "T2.2.1 Post Querying Processing", "T2.2.2 Metadata based ranking" and "T2.2.3 Text Indexing and Retrieval". For each activity, the deliverable contains a related work section surveying the state of the art on the particular topic; a detailed description of the solution devised for implementing the service under the ASSETS project umbrella; and the specification of the APIs designed for service invocation. The development of the prototypes for these services is undergoing. At the time of writing this document we are about to have a first implementation of all the services described above. The first prototypes do not support the full service functionality, but a connected subset of them is already developed in order to showcase the overall post query processing process. Moreover, the current implementations are already integrated in the ASSETS platform.

5. References

AT05	Adomavicius, G., Tuzhilin, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. IEEE TKDE 17 (6), 734–749. 2005.
BB+08	Boldi,P.,Bonchi,F.,Castillo,C.,Donato,D.,Gionis,A.,Vigna,S. The query-flow graph: model and applications. In: Proc. CIKM’08. ACM 2008.
BB+09a	Boldi, P., Bonchi, F., Castill2o0, C., Donato, D., Vigna, S.. Query suggestions using query-flow graphs. In: Proc. WSCD’09. ACM. 2009.
BB+09b	Boldi, P., Bonchi, F., Castil1lo0, C., Vigna, S. From ’dango’ to ’japanese cakes’: Query reformulation models and patterns. In: Proc. WI’09. IEEE. 2009.
BC+09	Baraglia, R., Cacheda, F., Carneiro, V., Fernandez, D., Formoso, V., Perego, R., Silvestri, F. Search shortcuts: a new approach to the recommendation of queries. In: Proc. RecSys’09. ACM, New York, NY, USA. 2009.
BG+07	Baeza-Yates, R., Gionis, A., Junqueira, F., Murdock, V., Plachouras, V., Silvestri, F. The impact of caching on search engines. In: Proc. SIGIR’07. pp. 183–190. ACM, New York, NY, USA. 2007.
BJ+04	Beitzel, S.M., Jensen, E.C., Chowdhury, A., Grossman, D., Frieder, O.. Hourly analysis of a very large topically categorized web query log. In: Proc. SIGIR’04. ACM Press. 2004.
BM+10	Broccolo, D., Marcon, L., Nardini, F.M., Perego, R., Silvestri, F.: An efficient algorithm to generate search shortcuts. Tech. Rep. 2010-TR-017, CNR ISTI Pisa. 2010.
BRL06	C. Burges, R. Ragno, and Q.V. Le. Learning to rank with non-smooth cost functions. In Advances in Neural Information Processing Systems (NIPS), 2006.
BS+05	Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In Proceedings of the 22nd international conference on Machine learning (ICML '05), 2005.
BT07	Baeza-Yates, R., Tiberi, A. Extracting semantic relations from query logs. In: Proc. KDD’07. ACM. 2007.
DSB09	Pinar Donmez, Krysta M. Svore, and Christopher J.C. Burges. On the local optimality of LambdaRank. In Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval (SIGIR '09), 2009.
HGO00	R. Herbrich, T. Graepel, and K. Obermayer. Large Margin Rank Boundaries for Ordinal Regression. Advances in Large Margin Classifiers, pages 115-132, 2000.
JJJ07	Jarvelin, A., Jarvelin, A., Jarvelin, K. s-grams: Defining generalized n-grams for information retrieval. IPM 43 (4), 1005– 1019. 2007.
JK08	Jones, R., Klinkner, K.L.. Beyond the session timeout: automatic hierarchical segmentation of search topics in query logs. In: CIKM ’08. pp. 699–708. ACM 2008.
LO+11	Lucchese, C., Orlando, S., Perego, R., Silvestri, F., Tolomei, G.: Identifying task-based sessions in search engine query logs. In: Proc. WSDM’11. pp. 277–286.

	ACM, New York, NY, USA 2011.
M00	Markatos, E.P. On caching search engine query results. In: Computer Communications, 2000.
MLK10	Ma, H., Lyu, M. R., King, I. Diversifying query suggestion results. In: Proc. AAAI'10. AAAI. 2010.
PA+10	José Pérez-Agüera, Javier Arroyo, Jane Greenberg, Joaquin Iglesias, Victor Fresno. Using BM25F for semantic search. SEMSEARCH '10: Proceedings of the 3rd International Semantic Search Workshop (2010)
RW94	S. Robertson and S. Walker. Some simple effective approximations to the 2-Poisson model for probabilistic weighted retrieval. In ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR), pages 345–354, 1994.
RZ09	Robertson, S., Zaragoza, H. The probabilistic relevance framework: Bm25 and beyond. Found. Trends Inf. Retr. 3 (4), 333–389. 2009.
RZT04	S. Robertson, H. Zaragoza, and M. Taylor. Simple BM25 extension to multiple weighted fields. In ACM Conference on Information Knowledge Management (CIKM), pages 42–49, 2004.
S10	Silvestri, F. Mining query logs: Turning search usage data into knowledge. Foundations and Trends in Information Retrieval 1 (1-2), 1–174. 2010.
SB09	Krysta M. Svore and Christopher J.C. Burges. 2009. A machine learning approach for improved BM25 retrieval. In Proceeding of the 18th ACM conference on Information and knowledge management (CIKM '09).
SM+99	Silverstein, C., Marais, H., Henzinger, M., Moricz, M. Analysis of a very large web search engine query log. SIGIR Forum 33, 6–12. September 1999.
TZ+06	Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. Optimisation methods for ranking functions with multiple parameters. In Proceedings of the 15th ACM international conference on Information and knowledge management (CIKM '06), 2006.