D2.2.3 METADATA-BASED INDEXING AND RANKING – 2ND RELEASE

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

Grant Agreement Number: 250527
Funding schema: **Best Practice Network**

Deliverable D2.2.3 WP2.2

Prototype

V 1.3 – March 02 2012
Document. ref.: ASSETS.D2.2.3.CNR.WP2.2.V1.3

Programme Name: ..... ICT PSP
Project Number: 250527
Project Title: ..... ASSETS
Partners: ..... Coordinator: ENG (IT)
Contractors:
Document Number: D2.2.3
Work-Package:..... 2.2
Deliverable Type: ..... Prototype
Contractual Date of Delivery: ..... 31 January 2012
Actual Date of Delivery: ..... 02 March 2012
Title of Document: ..... Metadata-Based Indexing and Ranking
Author(s): ..... Diego Ceccarelli (CNR), Claudio Lucchese(CNR), Raffaele Perego (CNR)

Approval of this report ..... APPROVED – Luigi Briguglio (ENG)

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

History: ..... see Change History

Keyword List: ..... ASSETS, metadata, learning to rank, bm25f

Availability..... public

Change History

Version	Date	Status	Author (Partner)	Description
0.1	13/01/2012	Draft	ISTI-CNR	Initial draft
0.2	27/01/2012	Draft	ISTI-CNR	Finalizing draft
0.3	01/02/2012	Draft	ENG	Peer reviewing by Massimiliano Nigrelli (ENG)
1.0	20/02/2012	Pre-Final	ISTI-CNR	Integration of the reviews
1.1	29/02/2012	Pre-Final	AIT	Peer reviewing by Sergiu Gordea
1.2	01/03/2012	Final	ISTI-CNR	Finalization.
1.3	02/03/2012	Final	ENG	Approval and Release

Table of Contents

Executive Summary

This document illustrates the software components resulting from the activities conducted within the tasks "T2.2.1 Post Querying Processing", "T2.2.2 Metadata based ranking" and "T2.2.3 Text Indexing and Retrieval". For each delivered service, it includes:

- the software requirements overview;
- the technical documentation (UML diagrams, services description and API documentation, software packaging and installation instruction);
- the user manual

This document is intending to support the integration in Europeana and maintenance related activities. This document describes the backed part of the services, the integration in the Assets Portal is documented in the D2.5.4 Improved User Interface.

1. This Introduction

Task "T2.2.1 Post Querying Processing" proposes a new query recommendation algorithm specifically tailored for the ASSETS project and Europeana users.
Task "T2.2.2 Metadata based ranking" deals with techniques for effective ranking of metadata objects in the for the Europeana search engine.
The Europeana query log analysis that we conducted with the aid of the tools developed within task "T2.2.3 Text Indexing and Retrieval", and the literature on multi-field document retrieval, suggests that the ranking function currently adopted by Europeana can be improved.
This deliverable provides the technical documentation needed to install, configure and use the software that has been produced during the above mentioned three tasks.
The document is divided into three main parts that describe in detail the three text processing services:

- The **Query Suggestion Service** (T.2.2.1): provides useful suggestions for related search queries given the last query used by a user.
- The **Metadata Based Ranking Service** (T.2.2.2) which provides a new ranking function (BM25F), together with a “learning to rank” method to be used for learning the free parameters of the BM25F ranking function.
- The **Query Log indexing Service** (T.2.2.3) which performs the cleaning, analysis and indexing of the Europeana query logs. It provides statistics over the past users’ queries.

This document introduces technical aspects of the services: the software requirements, the UML diagrams, the API documentation, the software installation and configuration, and the user manuals.

2. T2.2.1 Post Query Processing

2.1 Introduction

User's queries are often ambiguous or they do not suffice to describe the user's *information need*. For this purpose specific post-query processing techniques are important to provide the user with additional or enhanced information: **query suggestion** is one of such kind.

Providing users of Web Search Engines (WSEs) systems with suggestions is a common practice aimed at "driving" users towards the "piece of information" that 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 who should help inexperienced ones. The mined-knowledge for making this possible is contained in WSEs' logs which store all the previous interactions between the users and the system. The more good queries (i.e. queries that enable users to satisfy their information need) were used in the past, the more precise and effective the related suggestions provided by the query recommendation technique will be for future searches. On the other hand, generating effective suggestions for queries that are rare or have never been submitted in the past is an open issue poorly addressed by state-of-the-art query suggestion techniques.

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 quickly access the content he/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 in advance the information they were looking for. The problem has a nice parallel in computer systems: pre-fetching. Similarly to pre-fetching, search shortcuts anticipate future requests (made by the users) to the search engine with suggestion of queries that a user would have likely issued at the end of his/her session.

2.2 Software Requirements Overview

2.2.1 Requirements

Usability: The system will nicely integrate a set of related queries within the Europeana portal. Users are already used to query suggestion mechanism.

Reliability: The service must be able to provide meaningful related links for any given query submitted by a user to the search engine.

Performance: The query suggestion system should be faster than the query answering system, i.e. it should be able to provide a result within a few hundred milliseconds.

***Look &* Feel:** The displayed results list should be easily recognizable but not intrusive.

Layout and Navigation Requirements: The suggested links should appear near the search box and/or at the bottom of the search result page.

System Constraints: The system will require the SOLR search engine.

Licensing Requirements: The post query processing service is written in Java and requires the Java Runtime Environment. There are no requirements to acquire a license for commercial third party software. Third party components are several open-source Java libraries.

System Documentation: (a) Code will be commented in a professional manner so that API documentation can be automatically generated, and (b) service documentation will also be provided, detailing the installation, configuration, and use of the service.

2.2.2 Use Cases

Actors

- User: the user submits his/her own queries to the search system, that transparently forward them to the query suggestion tool

Stakeholders

- Users: need advanced tools for improving their query browsing and interaction experience with Europeana.
- Europeana: creates user friendly access to European heritage.

Preconditions

- Europeana query logs are available to the Assets portal administrator.

Basic flow of events

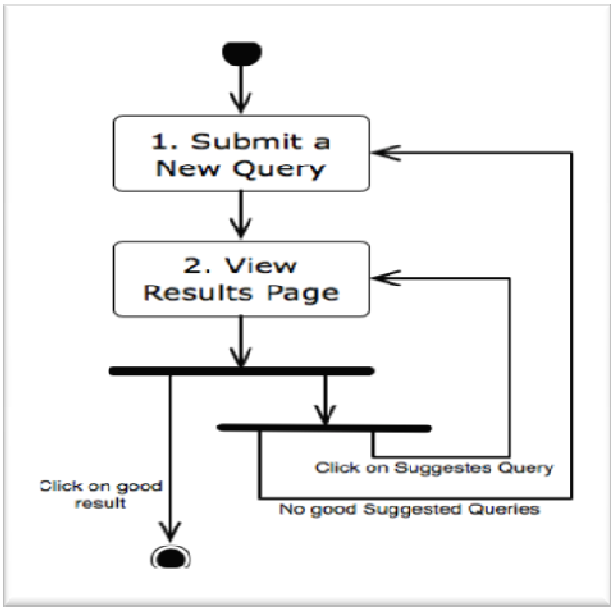


Figure 1 Post Query Processing use case

1. The use case begins when user visits the Europeana Home Page;
2. At step one he/she submits a query;
3. At step two he/she receives the results pages;
4. If there is a good result, this is clicked and the use case ends;
5. Otherwise the user may find an interesting query suggestion, which is used as a new query to the system;
6. Or the user re-writes a new query until an interesting result is returned.

Post- conditions

- The system must log the queries submitted and the clicks issued during the interaction with the user.

Special requirements

- User sessions must be available to train the underlying model;
- The query suggestions must be integrated within the ASSETS/EUROPEANA web portal.

Content requirements for Europeana portal:

- Europeana must log the activity of the users visiting the portal. The activity should be split into users' sessions.

2.3 Technical Documentation

2.3.1 UML Diagrams

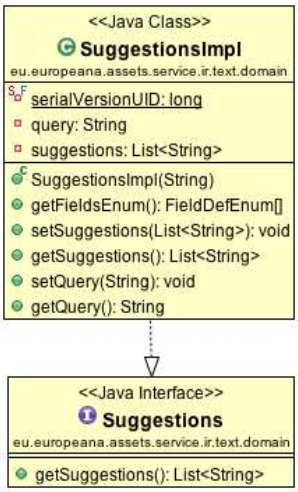


Figure 2 Suggestions Class Diagram

Figure 2 shows 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 for encapsulating a particular query (e.g., *Pablo Picasso*) and the suggestions for the query *ranked for relevance* (e.g., *Pablo Picasso life, Guernica, Cubism …)*.
The interface exposes only the method *GetSuggestion()* which returns the list of the suggestions to be displayed to the user.

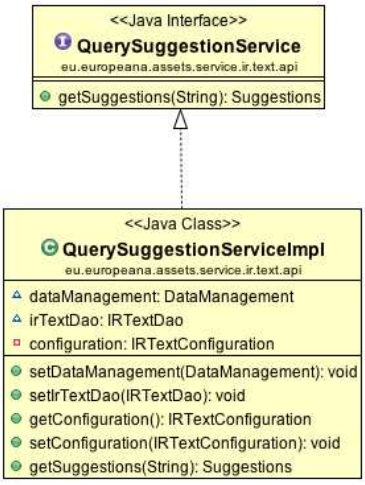


Figure 3 Query Suggestion Service Class Diagram

Figure 3 shows 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.

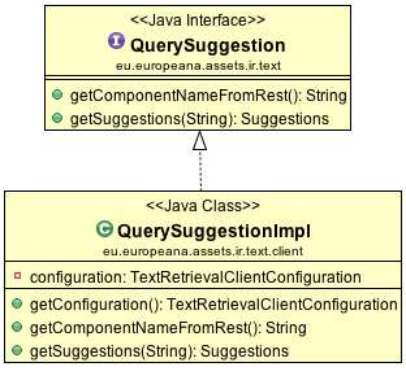


Figure 4 Query Suggestion Client Class Diagram

Finally, Figure 4 illustrates the class diagram of the query suggestion client and its implementation. The client defines how the other components of the ASSETS platform would interact with the query recommendation component. Its first task is to receive the queries from the other components; then it submits the queries to the query suggestion service and finally returns the suggestions to the applicants. If needed, this service could be exposed externally as a specific API-call available to third parties.

2.3.2 Service APIs: REST services

The suggestions for a query are also provided as a REST service.

Post Query Processing

Method	Response type	Path	Function
GET	XML/JSON	/assets/ir-text /QuerySuggestionService /rest/suggestions	Returns the suggestion given a query (parameter *query*)

Figure 5 IR-Text Post Query Processing REST API

The service takes a string containing the query performed by the user (with parameter name *query*) as input and returns a list of possible suggestions for the query, encoded in XML or in JSON.

The path of the service is **/assets/ir-text/QuerySuggestionService/rest/suggestions**

Table 1 shows the main service information needed to call it:

Method	Response type	Name	Parameters	Function
GET	XML/JSON	**suggestions**	**@query,** the query performed by the user	Returns related queries for the query performed by the user

Table 1 Post Query Processing REST API

2.3.3 Service APIs: Client API

This API provides a method to invoke the suggestion service.

API	**QuerySuggestion**
Responsibility	*Provides related queries given a query performed by the user*
Provided methods	**Suggestions getSuggestions (String query)** *Takes a user query (e.g., "Pablo Picasso"), and it returns the suggestions* • *(e.g., "Pablo Picasso blu's period","cubism"...)* ***@param** query : the user query* ***@returns** the domain object Suggestions, containing the list of all possible related queries (the list is empty if there are not related queries).*

2.3.4 Software Packaging

The query suggestion module is 100% java code and it is developed using Maven.

The backend and the client depend on

- solr-solrj 1.4.0 - to interface with the solr server where the suggestions are stored
- common-data-model 1.0
- gson 1.7.1 – Google library for parsing json

The configuration files are:

- **assets-ir-text.properties** (/ir-text/src/main/resources/assets-ir-text.properties)
- **assets-ir-text-client.properties** (/ir-text-client/src/main/resources/assets-ir-text-client.properties)

2.3.5 Installation and configuration

Since the service uses Solr to store its model, the developer will have to add a new core in the solr configurations for soring and indexing the suggestions[1].
Then the url of the solr suggestion server has to be set in the property file (**assets-ir-text.properties**) e.g.:

[1] For adding a new core in solr configurations, please refer to http://wiki.apache.org/solr/CoreAdmin

```
# the solr suggestion server
solr.server.suggestion = http://localhost:8989/solr/suggestion
```

In order to enable the suggestions the admin will have run the log analysis and to compute the query suggestions:

1. Compute the query suggestions with the command:

```
java –cp ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar
 eu.europeana.assets.service.ir.text.cli.GetSuggestionsFromJsonLogsCLI
–input querylogFolder
```

For each query log file in the query log folder, this command will generate a json file in the sessions folder. The folder name is set in the property file (**assets-ir-text.properties**) with the name **session.folder,** e.g.,

```
#The folder containing the parsed sessions in the query logs
session.folder = /tmp/europeana_sessions
```

2. Finally each json file containing the parsed session will be indexed on the Solr Server using the command:

```
java –cp ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar
 eu.europeana.assets.service.ir.text.cli.IndexSuggestionsCLI
–input sessionFile
```

where:

- **sessionFile** is a file in the $(**session.folder)** folder.

3. T 2.2.2 Metadata Based Ranking

3.1 Introduction

Task T2.2.2 deals with techniques for effective ranking of results in the context of Europeana search. The Europeana query log analysis that we conducted with the aid of the tools developed within task "T2.2.3 Text Indexing and Retrieval", and the literature on multi-field document retrieval, suggests that the ranking function currently adopted by Europeana can be improved.

In the following paragraphs, we provide a specification for an advanced metadata based ranking. In particular, we will describe our implementation of the BM25F ranking function, and the machine-learning module for best tuning its parameters. The learning step exploits the output of query log processing tools, which is described in Section 4.

3.2 Software Requirements Overview

3.2.1 Requirements

Reliability: A novel ranking function will not affect the reliability of the search system.

Performance: The ranking function should not harm the performance of the underlying search system.

User Interface: The service will change the list of results returned to the user, but not their presentation.

Look & Feel: No user interface to be provided.

Layout and Navigation Requirements: No user interface to be provided.

Interfaces to External Systems or Devices: The system will provide a set of results for any given query submitted by the user.

Software Interfaces: No external software will be used. The service will be embedded as an additional ranking function (i.e., a Solr plugin) within the query processing component. Access to full documents, and to historical data about the user interaction with the system is needed to fine-tune the ranking function. This analysis is done off-line with no interaction with external components.

System Constraints: The system will require the SOLR search engine.

Licensing Requirements: The metadata based ranking service is written in Java and requires

the Java Runtime Environment. There are no requirements to acquire any license for commercial third party software. Third party components are open-source Java libraries.

System Documentation: (a) Code will be commented in a professional manner so that API documentation can be automatically generated, and **(b)** service documentation will also be provided, detailing the installation, configuration, and use of the service.

3.2.2 Use Cases

These use cases describe the activities that are performed during the searching. The main goal of the service is to satisfy a user information need.

Actors

- **Europeana foundation**: www.europeana.eu.
- **Assets development team**: Improve metadata based ranking.
- **Europeana Users**: Users of the Europeana portal.

Stakeholders

- **End-user**: who submits a query and navigates through a long result list to find what she was looking for.
- **Europeana**: wants to offer powerful metadata search.

Basic Flow of Events

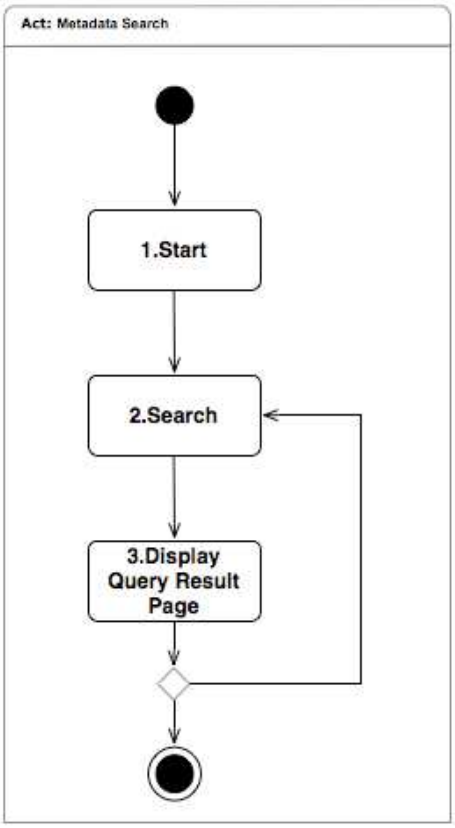


Figure 6 Metadata Based Ranking Use Case

1. **Start** : The use case begins when the users access the site.
2. **Search** : A user submits a query.
3. **Display Query Results Page** : Results are displayed to the users in a web page.

Requirements for Content Provision

- Query logs are available in order to improve results quality.
- Average length of each metadata fields is available.

Content requirements for Europeana portal

- Query logs are available in order to improve results quality.

3.3 Technical Documentation

3.3.1 UML Diagrams

The service is used to perform two main operations: *search* and *learning to rank*.

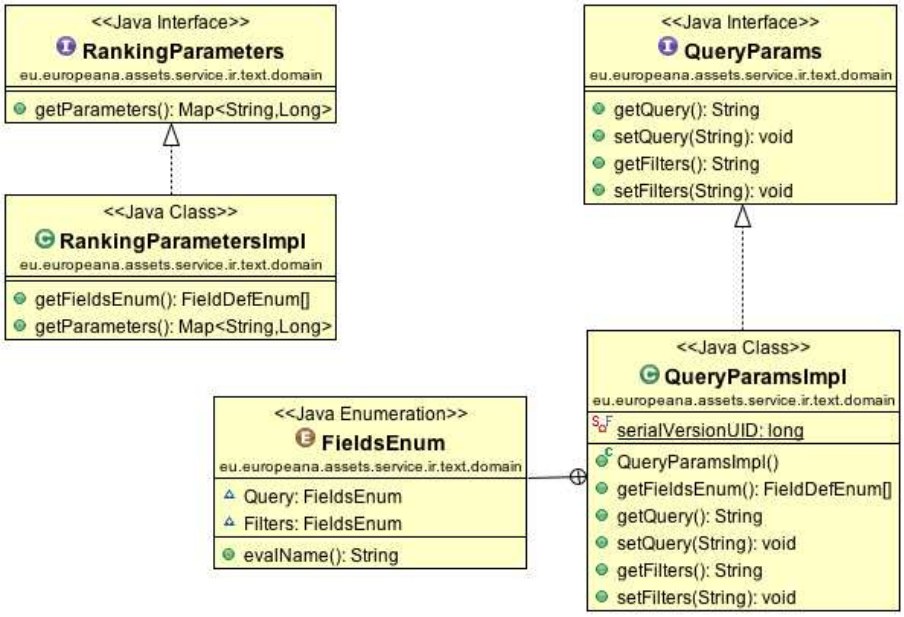


Figure 7 Ranking Parameters Domain Class Diagram

Figure 7 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.

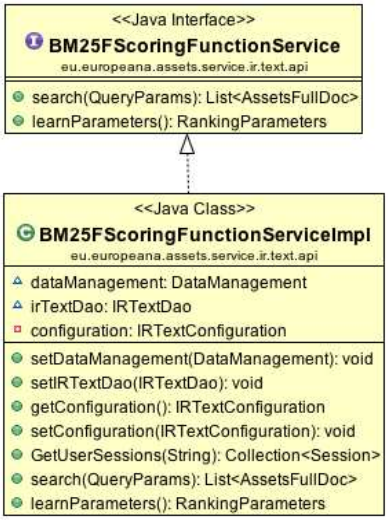


Figure 8 BM25F Scoring function class diagram

Figure 8 shows the class diagram of the BM25F scoring function implementation. The service allows processing a query using the BM25F scoring function (method search) and returns a list of *AssetsFullDoc objects*. 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).

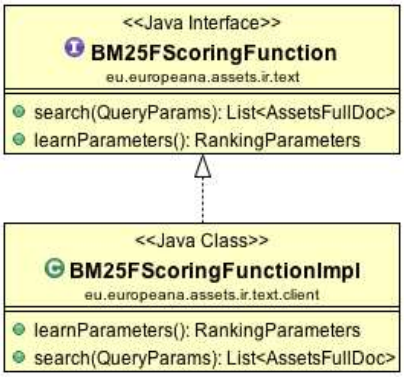


Figure 9 BM25F Scoring function client class diagram

Figure 9 shows the class diagram of the BM25F client and its implementation. The client defines how the other components of the ASSETS platform would interact with the BM25F component. Its first task is to receive from the other components the queries (encapsulated in a QueryParams object that may contain also filters and other parameters); then it submits the queries to the SOLR engine and finally it 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, he/she will

obtain a list of parameters with their respective tuned values (encapsulated in a RankingParameters object).

3.3.2 BM25F Solr Plugin

For performance reasons, we decided to implement the BM25F ranking function as a Solr plugin.

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 (i.e. number of terms) of a field computed on the whole collection.

We integrated the ranking function as a Solr *plugin*, without modifying the core code. This will allow updating Solr to new versions without applying any patch to the Solr core. The admin can enable the plugin from the Solr’s configuration file by simply adding the following few lines to the file *solrconfig.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>
```

Figure 10 How to configure the Solr BM25F plugin

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. We also provide a method to estimate the length of the parameters.

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

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

3.3.3 Service APIs: REST services

Metadata Based Ranking

Method	Response type	Path	Function
GET	XML/JSON	/assets/ir-text /MetadataBasedRankingService /rest/search	Returns the top-12 documents matching the query, ranked using bm25f ranking function (parameter *query*)
GET	XML/JSON	/assets/ir-text /MetadataBasedRankingService /rest/learn	Learns the best combination of parameters for the bm25f ranking function

Figure 11 Metadata Based Ranking REST API

The service offers two REST methods:

- **search:** receives a string containing the query performed by the user (with parameter name *query*) as input and returns the list of top documents matching the query, ranked using the bm25f ranking plugin. Results are encoded in XML or in JSON.
  The path of the method is
  /assets/ir-text/MetadataBasedRanking/rest/search

- **learning:** learns the best combination of parameters for the bm25f ranking function, returns the list of parameters with their values, encoded in XML or in JSON.
  The path of the method is
  /assets/ir-text/MetadataBasedRanking/rest/learning

Method	Response type	Name	Parameters	Function
GET	JSON/XML	**search**	**@query,** the query performed by the user	Returns the list of top-12 documents matching the query, ranked using the bm25f ranking plugin.
GET	JSON/XML	**learn**	-	Learns the best combination of parameters for the bm25f ranking function. Returns the list of parameters with their values.

Table 2 Metadata Based Ranking REST API

3.3.4 Service APIs: Client API

This API provides methods to interface with the metadata based ranking service.

API	**MetadataBasedRanking**
Responsibility	*provides techniques for improving ranking of results returned by the Europeana search engine.*
Provided methods	**List<AssetsFullDoc> search (QueryParams query)** *Takes a the user query (e.g., "Pablo Picasso"), returns the top documents matching the query, ranked using the bm25f ranking plugin.* ***@param*** *query : the user query* ***@returns*** the list of top documents matching the query, ranked using the bm25f ranking plugin **RankingParameters learnParamenters()** *learns the best combination of parameters for the bm25f ranking function, returns the list of parameters with their values, encoded in XML or in JSON.* **@returns** *returns the list of parameters with their values, encoded in XML or in JSON*

3.3.5 Software Packaging

The metadata based ranking module is 100% java code and it has been developed by using Maven.

The backend and the client depend on:

- solr-solrj 1.4.0 - to interface with the solr server where the suggestions are stored
- solr-core 1.4.0 / lucene 2.9.4 in order to extend solr with the bm25f ranking plugin.
- common-data-model 1.0
- gson 1.7.1 – Google library for parsing json

The configuration files are:

- **assets-ir-text.properties** (/ir-text/src/main/resources/assets-ir-text.properties)
- **assets-ir-text-client.properties** (/ir-text-client/src/main/resources/assets-ir-text-client.properties)

3.3.6 Installation and configuration

Installing bm25f ranking plugin

In order to install the bm25f plugin, the admin has to create the jar of the service using the command:

```
mvn assembly:assembly  -DskipTests
```

This command will produce a file called *"ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar"* in the target folder of the ir-text project.

Then copy the jar in the lib folder of solr installation:

```
cp target/ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar  $SOLRHOME/metadata/lib
```

Import the bm25f query parser from solrconfig.xml solr config file, adding in the xml:

```
<queryParser                  name="bm25f"                  class=                  "
eu.europeana.assets.service.ir.text.bm25f.parser.BM25FQParserPlugin ">
```

The plugin is then available and can be used as shown in subsection 3.3.2.

The service also provides a command line to compute efficiently the average length of each fields in the index schema (needed by the bm25f ranking function).

In order to compute such values the user may use the command:

```
java              –cp              ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar
eu.europeana.assets.service.ir.text.bm25f.util.ComputeAverageFieldLength indexPath
```

where **indexPath** is the path of the Solr index.

Learning to rank

The assessments needed for the learning to rank method, may be generated using the following command

```
java –cp ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar
 eu.europeana.assets.service.ir.text.cli.GenerateLearningToRankAssessmentsCLI
 –input sessionDir –from startDate –n assessmentsNumber
```

where:

- **sessionDir** is the directory containing the parsed sessions (in json format);
- **startDate** is the starting date of the learning interval (format is dd/MM/yyyy);
- **assessmentsNumber** is the number of assessments to generate.

The assessment contains the most popular queries available in the logs after the given date.

Assessments are generated in the folder set in the property file **assets-ir-text.properties** with the property name **assessments_folder:**

```
#the folder where the evaluation dataset resides
assessments_folder = ./services/ir-text/data/evaluation_dataset
```

The learning of BM25F parameters can be also performed by using the following command:

```
java –cp ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar
 eu.europeana.assets.service.ir.text.cli.LearningToRankCLI –input assessmentsFolder
```

where:

- **assessments_folder** is the folder containing the assessments

When the LearningToRankCLI execution is finished (i.e. may take several hours depending on the size of the query log), it will print the values for the parameters and the admin will have to update this values in the solrconfig.xml file (see Section 3.3.2).

4. T 2.2.3 Text Indexing And Retrieval

4.1 Introduction

The goal of task T2.2.3 is to devise a set of query log processing tools needed by other ASSETS services, paying attention to those that will be used to extract user behavioral patterns.

These patterns will be useful for improving the ASSETS ranking function and, at the same time, will provide the clean information to be used by the query recommendation service.

The task includes non-trivial activities such as:

- **The Query log cleaning:** consisting in parsing a raw record from the server and encoding it in a well-structured form.
- **The Analysis** consisting in removing "*spam*" records from bots, and grouping records performed by the same user in sessions.
- **Persistance**: storing the sessions and the records on a database (MongoDB).

The module also provides an API and a REST service which can be used to retrieve a singular record/session and to obtain useful statistics about the logs over a range of days or months.

4.2 Software Requirements Overview

4.2.1 Requirements

Usability: No interaction with the end user.

Reliability: It should be able to answer retrieval query about statistics over the query logs at any time.

Performance: It should be able to answer queries within a few milliseconds.

Layout and Navigation Requirements: None

Interfaces to External Systems or Devices: No external software will be used. The text files containing the query logs are used as input for the service.

Software Interfaces: This software component will provide query log indexing and retrieval to other software components of the system. In particular, it will support every tool developed within the post query processing activity. Whenever possible, RESTful HTTP interfaces will be used.

System Constraints: The system will require the MongoDB nosql database.

Licensing Requirements: Software will be licensed in compliance with EUPL.

Legal, Copyright, and Other Notices: None.

Applicable Standards: None.

System Documentation: Code will be commented in a professional manner so that API documentation can be automatically generated, and service documentation will also be provided, detailing the installation, configuration, and use of the service.

4.2.2 Use Cases

Actors

- **Europeana foundation**: www.europeana.eu.
- **Assets development team**: Index the query logs.
- **Europeana Portal administrators**: Browse the records and the statistics.

Stakeholders

- **Europeana**: wants to know better the users' needs.

This use case describes the processing steps performed by the Text Indexing and Retrieval service and its interaction with the Europeana Core service.

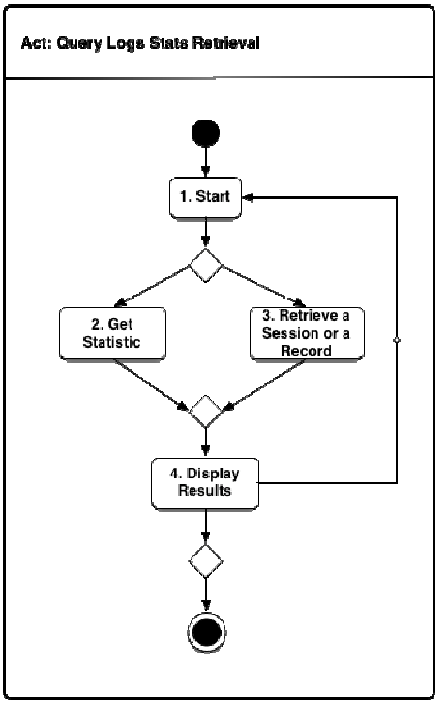


Figure 12 Query Log Statistics Retrieval Use Case

1. **Start** : The use case begins when the user accesses statistics page; then the user can:
   2. **Get a specific statistic** on the query logs,

3. **Retrieve a Session or a Record** : e.g., view a particular session performed by a user.
.4. **Display** the results.
Then the user may decide to browse other statistics or to end the browsing session.

Requirements for Content Provision

- Query logs available.

Content requirements for Europeana portal

- Query logs are available in order to improve results quality.

4.3 Technical Documentation

4.3.1 UML Diagrams

Figure 13 shows the class diagram modeling query log data and session information. *QueryLogRecord* describes the java object representation for a record in the query log. It represents a user interaction with the portal (submitting a query, clicking on a result item, etc.). The *Session* object represents a user query session. Queries by the same user are split in different sessions based on the time interval between consecutive queries.

Figure 13 QueryLogRecImpl Domain Object Class Diagram

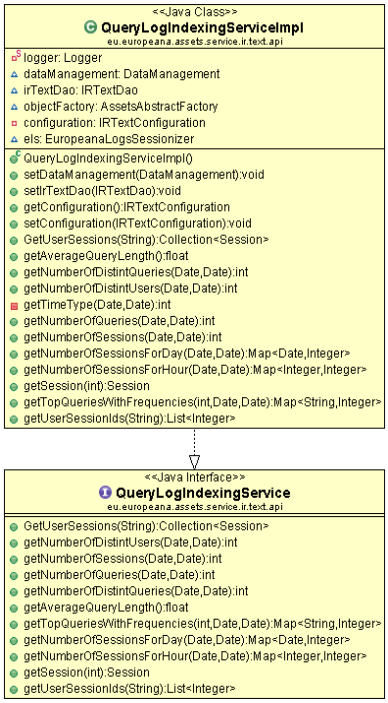


Figure 14 QueryLogIndexing Service Class Diagram

Finally, Figure 14 displays 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 information to be used by the query suggestion service and by the metadata based ranking. More in detail:

Retrieving Statistics

- *getUserSessions,* the sessions issued by a particular user;
- *getNumberOfDistinctUsers,* the distinct number of users;
- *getNumberOfSessions,* the distinct number of sessions;
- *getNumberOfQueries,* the number of queries;
- *getNumberOfDistintQueries,* the number of distinct queries;
- *getAverageQueryLength,* the average length of a query measured by the number of terms;
- *getTopQueriesWithFrequencies,* the most frequent queries;
- *getNumberOfSessionsForDay*, the frequency distribution of the sessions over the days in a month;
- *getNumberOfSessionsForHour,* the frequency distribution of the sessions over the hours in a day.

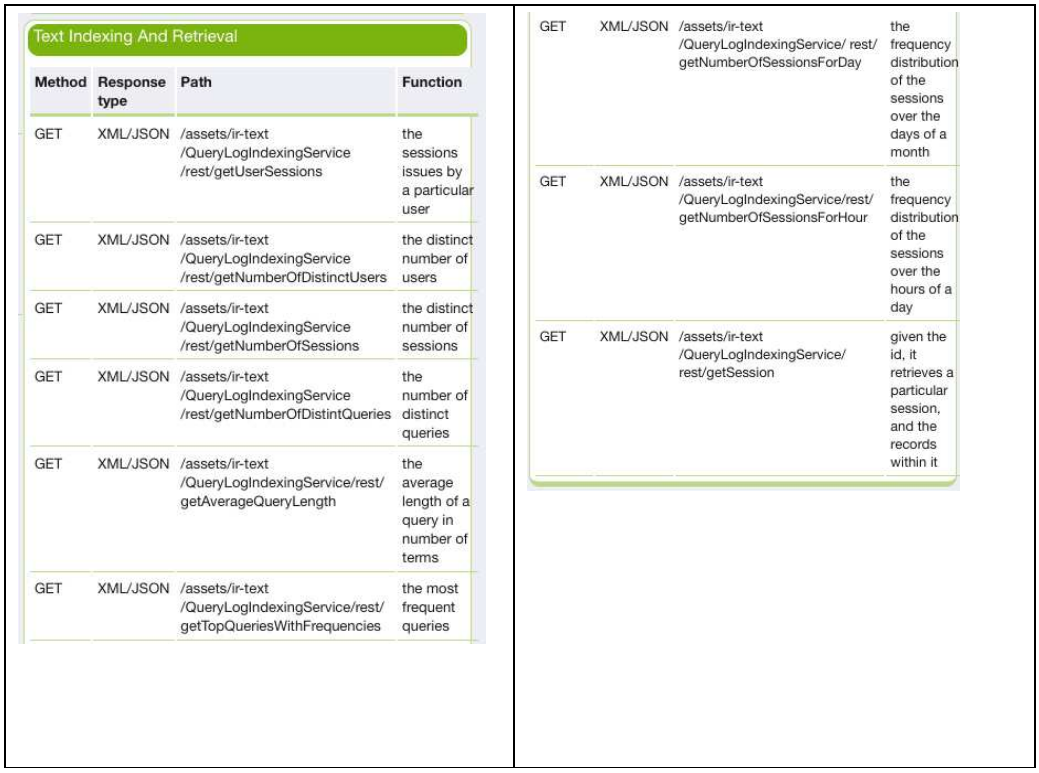
Each method receives a *start date* and an *end date as inputs* and computes the statistics over the queries in the selected time range. Users may specify a time range in months (e.g. from February 2011 to April 2011) or a time range in days (e.g., last week's queries).

Browsing Sessions and Records

- *getSession* , given the session id, it retrieves a particular session, and the records within it;

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

4.3.2 Service APIs: REST services



Text Indexing And Retrieval

Method	Response type	Path	Function
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService /rest/getUserSessions	the sessions issues by a particular user
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService /rest/getNumberOfDistinctUsers	the distinct number of users
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService /rest/getNumberOfSessions	the distinct number of sessions
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService /rest/getNumberOfDistintQueries	the number of distinct queries
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService/rest/ getAverageQueryLength	the average length of a query in number of terms
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService/rest/ getTopQueriesWithFrequencies	the most frequent queries
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService/ rest/ getNumberOfSessionsForDay	the frequency distribution of the sessions over the days of a month
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService/rest/ getNumberOfSessionsForHour	the frequency distribution of the sessions over the hours of a day
GET	XML/JSON	/assets/ir-text /QueryLogIndexingService/ rest/getSession	given the id, it retrieves a particular session, and the records within it

Figure 15 Text Indexing And Retrieval REST API

The service offers several REST methods. In Table 3 the main information needed to call the methods are shown.

Method	Response type	Name	Parameters	Function
GET	JSON/XML	*getUserSessions*	**@userId,** the user id for which to retrieve the sessions	Returns the sessions issues by a particular user
GET	JSON/XML	*getNumberOfDistinctUsers*	**@start, @end,** the period over which to calculate the statistic	Returns the distinct number of users
GET	JSON/XML	*getNumberOfS*	**@start, @end,** the period over	Returns the

		ession	which to calculate the statistic	number of sessions
GET	JSON/XML	*getNumberOfQueries*	**@start, @end,** the period over which to calculate the statistic	Returns the number of queries
GET	JSON/XML	*getNumberOfDistinctQueries*	**@start, @end,** the period over which to calculate the statistic	Returns the number of distinct queries
GET	JSON/XML	*getAverageQueryLength*	**@start, @end,** the period over which to calculate the statistic	Returns the average length of a query in number of terms
GET	JSON/XML	*getTopQueriesWithFrequencies*	**@start, @end,** the period over which to calculate the statistic	Returns the most frequent queries
GET	JSON/XML	*getNumberOfSessionsForDay*	**@start, @end,** the period over which to calculate the statistic	Returns the frequency distribution of the sessions over the days of a month
GET	JSON/XML	*getNumberOfSessionsForHour*	**@start, @end,** the period over which to calculate the statistic	Returns the frequency distribution of the sessions over the hours of a day
GET	JSON/XML	*getSession*	**@sessionId** the id of the session to show	given the id, it retrieves a particular session, and the records within it;

Table 3 Post Query Processing REST API

4.3.3 Service APIs: Client API

The client API provides methods to interface with the Query Log Indexing service,

API	**Query Log Indexing Service**
Responsibility	*Returns statistics over the query log*

Provided methods	***List<Session> getUserSessions (String userId)*** *Returns the sessions issues by a particular user;* ***@param** useriD: the user id for which to retrieve the sessions;* ***@returns** the sessions issues by a particular user* ***int getNumberOfDistinctUsers(Date start, Date end)*** *returns the distinct number of users* ***@param** start, the start date of the period over which to calculate the statistic* ***@param** end, the end date of the period over which to calculate the statistic* ***@returns** the distinct number of users* ***int getNumberOfSessions (Date start, Date end)*** *returns the distinct number of sessions* ***@param** start, the start date of the period over which to calculate the statistic* ***@param** end, the end date of the period over which to calculate the statistic* ***@returns** the distinct number of sessions* ***int getNumberOfQueries (Date start, Date end)*** *Returns the number of queries* ***@param** start, the start date of the period over which to calculate the statistic* ***@param** end, the end date of the period over which to calculate the statistic* ***@returns** the number of queries* ***int getNumberOfDistintQueries (Date start, Date end)*** *Returns the distinct number of queries* ***@param** start, the start date of the period over which to calculate the statistic* ***@param** end, the end date of the period over which to calculate the statistic* ***@returns** the distinct number of queries* ***float getNumberOfDistintQueries (Date start, Date end)*** *Returns the average length of a query in number of terms;* ***@param** start, the start date of the period over which to calculate the statistic* ***@param** end, the end date of the period over which to calculate the statistic* ***@returns** the average length of a query in number of terms;*

	Map<String,Integer> getTopQueriesWithFrequencies(Date start, Date end) *Returns the most frequent queries;* ***@param*** *start, the start date of the period over which to calculate the statistic* ***@param*** *end, the end date of the period over which to calculate the statistic* ***@returns*** *A map containing the most frequent queries with their frequencies.* ***Map<Integer, Integer> getNumberOfSessionsForDay (Date start, Date end)*** *Returns the frequency distribution of the sessions over the days of a month.* ***@param*** *start, the start date of the period over which to calculate the statistic* ***@param*** *end, the end date of the period over which to calculate the statistic* ***@returns*** *A map containing for each day of a month, the frequency of the queries performed during that day* ***Map<Integer, Integer> getNumberOfSessionsForHour (Date start, Date end)*** *Returns The frequency distribution of the sessions over the hours of a day* ***@param*** *start, the start date of the period over which to calculate the statistic* ***@param*** *end, the end date of the period over which to calculate the statistic* ***@returns*** *A map containing for each hour of a day, the frequency of the queries performed during that hour* ***Session getSession (String sessionId)*** *Given the id, it retrieves a particular session, and the records within it;* ***@param*** *sessionId the id of the session to retrive* ***@returns*** *the session id for which to retrieve the session;*

4.3.4 Software Packaging

The metadata based ranking module is 100% java code and it has been developed using Maven for build automation.

The backend and the client depend on

- common-data-model 1.0
- common-api-server 1.0
- gson 1.7.1 – Google library for parsing json

4.3.5 Installation and configuration

The Europeana query logs come as raw apache logs, e.g.,

```
66.249.66.144 - - [12/Jan/2011:04:02:02 +0100] "GET /portal/europeana-sitemap-hashed.xml?prefix=6090&images=false HTTP/1.1" 404 3246 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
66.249.66.167 - - [12/Jan/2011:04:02:02 +0100] "GET /portal/europeana-sitemap-hashed.xml?prefix=0C4B&images=false HTTP/1.1" 404 3246 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)"
1.1.1.1- - [12/Jan/2011:04:06:05 +0100] "GET /portal/record/03903/255A899BE7E9146D28D88C3D57A8E265B0ADD4E3.html?query=leonardo+da+vinci&start=954&startPage=937&pageId=brd&view=text_only HTTP/1.1" 200 1542 " Mozilla/5.0 (Windows; U; Windows NT 5.1; ja; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8 ( .NET CLR 3.5.30729"
1.1.1.1- - [12/Jan/2011:04:06:15 +0100] "GET /portal/brief-doc.html?query=leonardo+da+vinci+florence&start=954&startPage=937&pageId=brd&view=text_only HTTP/1.1" 200 1542 " Mozilla/5.0 (Windows; U; Windows NT 5.1; ja; rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8 ( .NET CLR 3.5.30729"
```

The Query Log Indexing service provides a command to put the logs in a well-structured format:

```
java –cp ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar
eu.europeana.assets.service.ir.text.cli.SessionizeQueryLogFileCLI –input rawQueryLog -output parsedQueryLog.json
```

SessionizeQueryLogFileCLI accepts a raw apache log from Europeana (rawQueryLog) as input, filters the bots and produces a json file containing the log records “sessionized” on the users, for example for the logs previously shown , the file will contain:

```
{"start"   : "Jan 12, 2011 4:06:05 AM"
  "end"    : "Jan 12, 2011 4:06:15 AM ",
  "country":"Italy",
  "successful":false,
  "ip":"1.1.1.1",
  "userAgent":  "Mozilla/5.0  (Windows;  U;  Windows  NT  5.1;  ja;  rv:1.9.2.8) Gecko/20100722 Firefox/3.6.8 ( .NET CLR 3.5.30729)",
   "queries":["leonardo da vinci","leonardo da vinci florence"],
   "session":[
                    {"query":"leonardo+da+vinci",
                     "cleanQuery":"leonardo da vinci",
                     "date":"Jan 12, 2011 4:06:05 AM",
                    "hasQuery":false,"hasClick":true}]},
                    {"query":"leonardo+da+vinci-florence",
                     "cleanQuery":"leonardo da vinci",
                     "date":"Jan 12, 2011 4:06:15 AM",
                    "hasQuery":true,"hasClick":false}]}
]}
```

The json query logs files are then indexed on mongoDB using the command:

```
java –cp ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar
eu.europeana.assets.service.ir.text.cli.StoreJsonLogsOnMongoDbCLI –input queryLog.json
```

Every time the admin indexes new logs, he will have to optimize the mongoDB index by using the following mongoDB commands:

```
db.QueryLogRecordImpl.ensureIndex( { Date : 1, IsClick : 1 } );
db.SessionImpl.ensureIndex( { Date : 1, IsSuccessful:1 } );
```

In order to generate the advanced statistics over the query log, the admin will have to run the following command:

```
java –cp ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar
eu.europeana.assets.service.ir.text.cli.GetStatsFromMongoDbCLI –from MM/YYYY -to
MM/YYYY -folder statsOutputFolder
```

where:

- **from, to** are respectively the starting date (format MM/YYYY) and the ending date of the time range for which the statistics have to be generated,
- **statsOutputFolder** is the folder where the produced statistics will be stored (one json file for each month).

Finally the statistics will be stored in the MongoDB database using the command:

```
java –cp ir-text-0.0.1-SNAPSHOT-jar-with-dependencies.jar -input statsFolder/statsFile
```

Where:

- **input** is a file of statistics extracted by using the `GetStatsFromMongoDbCLI`, or a folder containing files of statistics.

5. Scientific Foundations

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.

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.

Boldi, P., Bonchi, F., Castill2o0, C., Donato, D., Vigna, S.. Query suggestions using query-flow graphs. In: Proc. WSCD'09. ACM. 2009.

Boldi, P., Bonchi, F., Castil1lo0, C., Vigna, S. From 'dango' to 'japanese cakes': Query reformulation models and patterns. In: Proc. WI'09. IEEE. 2009.

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.

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.

Baeza-Yates, R., Tiberi, A. Extracting semantic relations from query logs. In: Proc. KDD'07. ACM. 2007.

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.

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.

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.

D. Ceccarelli, S. Gordea, C. Lucchese, F. M. Nardini and G. Tolomei. Improving Europeana Search Experience Using Query Logs **.** In Proceedings of «TPDL '11: International Conference on Theory and Practice of Digital Libraries», Berlin, Germany, September 2011.

D. Ceccarelli, S. Gordea, C. Lucchese, F.M. Nardini, R. Perego, G. Tolomei. Discovering Europeana users' search behavior. In « ERCIM News», No. 86, 2011.

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.

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.

R. Herbrich, T. Graepel, and K. Obermayer. Large Margin Rank Boundaries for Ordinal Regression. Advances in Large Margin Classifiers, pages 115-132, 2000.

Jarvelin, A., Jarvelin, A., Jarvelin, K. s-grams: Defining generalized n-grams for information retrieval. IPM 43 (4), 1005– 1019. 2007.

Jones, R., Klinkner, K.L.. Beyond the session timeout: automatic hierarchical seg- mentation of search topics in query logs. In: CIKM ’08. pp. 699–708. ACM 2008.

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.

Ma, H., Lyu, M. R., King, I. Diversifying query suggestion results. In: Proc. AAAI’10. AAAI. 2010.

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)

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.

Robertson, S., Zaragoza, H. The probabilistic relevance framework: Bm25 and beyond. Found. Trends Inf. Retr. 3 (4), 333–389. 2009.

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.

Silvestri, F. Mining query logs: Turning search usage data into knowledge. Foundations and Trends in Information Retrieval 1 (1-2), 1–174. 2010.

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).

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.

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.

6. Concluding Remarks

This deliverable describes the software components developed within the activities conducted for the tasks "T2.2.1 Post Querying Processing", "T2.2.2 Metadata based ranking" and "T2.2.3 Text Indexing and Retrieval".

The provided services are, respectively, query recommendation, metadata based ranking and query log processing and indexing.

The document focuses on the technical aspects of the developed services, including: UML diagrams, services description and API documentation, software packaging and installation, and the user manual.