FUSEPOOL

STREP

FP7 – 296192

D3.2 SUPERVISED AUTOMATION API FOR DATA CURATION

COORDINATOR: THOMAS GEHRIG

LEAD AUTHOR: JULIEN PEREZ

CO-AUTHOR(S):

ADRIEN CZERNY

GUILLAUME BOUCHARD

STEPHANE GAMARD

1ST QUALITY REVIEWER: ANTON HEIJS (TREPAREL)

2ND QUALITY REVIEWER: SARVEN CAPADISLI (BUAS)

Deliverable nature:	Report (R)
Dissemination level: (Confidentiality)	Public (PU)
Contractual delivery date:	01/04/2014
Actual delivery date:	10/08/2014 – Upgrade 30/11/2014
Version:	8.0
Total number of pages:	18
Keywords:	Fusepool, document labelling, software architecture, prediction, machine learning

Abstract

This document describes the software architecture deployed on Fusepool to perform the document multi-labelling task. Its goal is to be able to automatically assigned labels, preliminary chosen by users, to document present on a research result list. The general pipelined software architecture of prediction is detailed. It is built on top of the Fusepool machine learning framework and makes use of a component called “Listen-Update-Predict” (LUP) that enables a quick and safe integration of adaptive components in the platform. The OSGi architecture design on the Fusepool platform and the server-side computation operations responsible for the statistical models of prediction are sequentially presented. The precise implementation of the document labelling task is described.

Executive summary

This deliverables summarizes the work that has been done in the task T3.4 to automate the document labelling task based on user interaction through the Fusepool Graphical User Interface (GUI). The goal of T3.2 is to enable the Fusepool platform to automatically assign user-specified labels to document displayed on research hitlists. The result of this task is used to quickly assign labels to document with respect to their content. In order to benefit from the user interactions of the Fusepool information retrieval system, annotations are produced by storing the labels of each document provided by the users.

In addition to the machine learning framework specified in Task 4.1, we contribute a *PredictionHub* OSGi bundle that is in charge of providing the learning framework infrastructure of the Fusepool platform . The *PredictionHub* OSGi component centralizes the access to any *Listen-Update-Predict* (LUP) module available on the platform. These modules represent the three main functionalities that are needed to define a new prediction engine in the Fusepool platform:

- Listener (L): this functionality defines under which event the learning must be triggered. (e.g. user clicks or hovers over a search result).
- Updating (U): this functionality defines the list of actions that must be done when the listener triggers an event.
- Prediction (P): this functionality defines a process that performs the prediction based on the current state of the prediction engine.

These three functionalities have been implemented for the source refinement task.

Finally, for the updating functionality of document labelling, a low rank linear decomposition of the generated dataset of user activities for this task, i.e the labelling of document by users, is proposed. It enables us to infer the most relevant labels associated to each document when displayed on the hitlist of a user.

Document information

FP7 Project Number	296192	Acronym	Fusepool
Full Title	Fusepool – Information pooling for product/service development and research		
Project URL	http://www.fusepool.eu		
Document URL			
EU Project Officer	Martina Eydner		

Deliverable	No.	D2.3	Title	Data sourcing and component integration
Work Package	No.	WP2	Title	Adaptive data sourcing API for source refinement

Date of Delivery	Contractual	M18	Actual	M18
Status	Version 8.0		Final □	
Nature	Prototype □ Report X Dissemination □			
Dissemination level	Public X Consortium □			

Authors (Partner)	Xerox, Salsa			
Lead Author (WP Lead)	Julien	Perez	Julien.Perez@xerox.com	
	Xerox			

Version Log			
Issue Date	Rev. No.	Author	Change
14 Feb 2014	1.0	Julien Perez	First version
17 Feb 2014	2.0	Guillaume Bouchard	Updated version
20 Feb 2014	3.0	Adrien Czerny	Detailed explanation of the software
17 April 2014	4.0	Guillaume Bouchard	Corrections
19 April 2014	4.1	Guillaume Bouchard	Minor Corrections
27 July 2014	4.2	Guillaume Bouchard	Took reviewer feedback into account
15 Nov 2014	7.0	Guillaume Bouchard	Corrections after Final Review Feedback
30 Nov 2014	8.0	Thomas Gehrig	Formalization and final control

Table of Contents

Abbreviations

Acronym	Full meaning
Annostore	Repository containing annotations
API	Application programming interface
BUAS	Bern University of Applied Sciences
ECS	Enhanced Content Store
DoW	Description of Work (part of the Grant Agreement)
ENOLL	European Network of Living Labs
GEOX	Geox KFT
GUI	Graphical User Interface
Hitlist	List of documents output from the search result
LL	Living Lab
LUP	Listen-Update-Predict
MS	Milestone
OSGi	Open Services Gateway initiative
RAD	Rapid application development
RDF	Resource Description Framework
SEARCH	Searchbox SA
SME	Small and Medium Enterprise
TREPA	Treparel Information Solutions BV
WP	Work package
XEROX	Xerox SAS
XSLT	eXtensible Stylesheet Language Transformations
XML	Extensible Markup Language

I Introduction

One of the main innovations introduced in Fusepool is the ability of the information retrieval platform to adapt to the user needs and preferences. To enable it, a *learning framework* has been defined in D4.1. In this framework, the data are divided into the original content stored in the Enhanced Content Store (ECS) and, on the other hand, into all the feedbacks created by the users (implicit or explicit feedbacks), which are stored in a separate repository called the Annotation Store (Annostore). This enables an easy integration of user-created data, as well as the update of every functional component of the Fusepool platform that is impacted by the user feedback. In this document, we describe the *data sourcing component* that automatically adapts the influence of a data source on the composition of result sets based on how often a user chooses this data source when browsing the output of a search. In the following of this section, the implementation of the learning framework proposed in D4.1 is detailed for the specific task at hand (T2.5). Then, we present the design and implementation of the adaptive data sourcing component.

The goal of T3.5 is to enable the Fusepool platform to automatically detect relevant data sources with respect to queries and users. This task will naturally be used to optimize information retrieval results.

Figure 1 presents the workflow and dependencies implied by the machine-learning framework introduced in D.4.1 as a general approach to account for user feedback in the Fusepool platform.

In the current implementation, the learning framework models the data source boosting prediction engines using a general bi-linear regression model[1] and has been detailed in the previous deliverable D2.3.

[1] Ajit P. Singh and Geoffrey J. Gordon, “Relational Learning via Collective Matrix Factorization”

II Listen-Update-Predict (LUP) component for Labelling task

The *Listen-Update-Predict* (LUP) implemented for the Labelling T3.4 task is the OSGi component responsible for listening, updating labelling prediction models by retrieving new annotations and sending them to the prediction algorithm. At prediction time, when a list a documents is fetched by the Firstswim GUI, the list of predicted labels for each document is given by the LUP 3.4 (the LUP Module responsible for the Labelling task) through the PredictionHub (see section II.).

1. Overview of the component

The LUP 3.4 is responsible of fetching the proper data for the learning task from the Annostore and provides a service that allows predictions on the labels of a document using the predict method below:

```
public String predict(HashMap<String,String> params);
```

The HashMap *params* must contain the following keys:

1. *docURI*

The returned value of this method produces a String object containing the following formatted statements:

```
"<label1>;<label2>;...;<labelN>"
```

For instance, the code corresponding to the prediction of the boosting parameters for the query "a query" would be:

```
HashMap<String,String> params = new HashMap<String,String>();
String docURI = "<http://fusepool.info/doc/pmc/2751467>";
/**
 * We build the parameters to give it to the L3.4via the predictionHub
 */
params.put("docURI", docURI);
/**
 * We call the LUP34.predict(...) method via the predictionHub.predict(...)
method
 */
String predictedLabels = predictionHub.predict("LUP34", params);
/**
 * We dump the result of the prediction
 */
log.info(predictedLabels);
/**
 * "tissue;sodium;English"
 */
```

2. The LUPEngine service

The LUPEngine service is the interface to implement in order to build a LUP service. In fact, the main idea of this design is to abstract and to formalize the three needed elements of any process of user adaptability in the Fusepool platform. As explained earlier, the task T3.4 follows the same architecture than the tasks 2.5, 4.5 and 5.5. As a consequence, this design defines the core sequence of procedures to implement in the Fusepool platform in order to produce a learnable component. For recall, these three fundamental tasks are (1) Listening to a particular event from the Annostore (2) Updating the learning model corresponding to the handling task (3) Providing a prediction service according the purpose of the given LUP module. The following listing details the set of Java methods to implement.

```
public interface LUPEngine {

    String LUP_NAME = "DefaultName";

    /**
     * @return the name of the class implementing this interface.
     */
    public String getName();

    /**
     * @return a short description of the LUP module so that it can be
displayed.
     */
    public String getDescription();

    /**
     * Basically used in the OSGi prediction component to plug a listener
to the
     * Annostore.
     * @return the reference of the engine's graphListener implementation.
     */
    public GraphListener getListener();
```

```
    /**
     * @return the filter that will be used when plugging the listener to the
     * Annostore.
     */
    public FilterTriple getFilter();

    /**
     * @return the listener's delay, which represents the period it remains
     * inactive.
     */
    public long getDelay();

    /**
     * Method used to communicate with learning servers (basically openXerox).
     */
    public void updateModels(HashMap<String, String> params);

    /**
     * These methods should be used at activation and deactivation respectively
     * to ensure the consistence of the learning model when platform reboot.
     */
    public void save();
    public void load();

    /**
     * Prediction method.
     */
    public String predict(HashMap<String,String> params);
```

The following methods should also be implemented in order to make a LUP module deployable on the Fusepool platform. Below are some examples in the case when the bundle uses the *Felix SCR Annotations*:

```
    /**
     * Activation method : called when the bundle goes from a RESOLVED
     * state to an ACTIVE state e.g. when deployed on the platform
     */
    @Activate
    private void activate() {...}

    /**
     * Dectivation method : called when the bundle goes from an ACTIVE
     * state to a RESOLVED stage e.g. when stopped
     */
    @Deactivate
    private void deactivate() {...}
```

3. Functional description

The LUP 3.4 OSGi component is based on the Learning Framework, and is used in two different ways: Listening (updating) and Prediction

Listening – updating use case

This is the automated way of working of the LUPs. For every machine learning task an important job is to build continuously models which represent the data we observe. Building and updating these models is called the learning. The automation of this learning task is built upon the listening of some specific modifications of the Annotation Store. Indeed, this is the purpose of the Annotation Store to host the data generated by the users interactions with the GUI (see Fig.2):

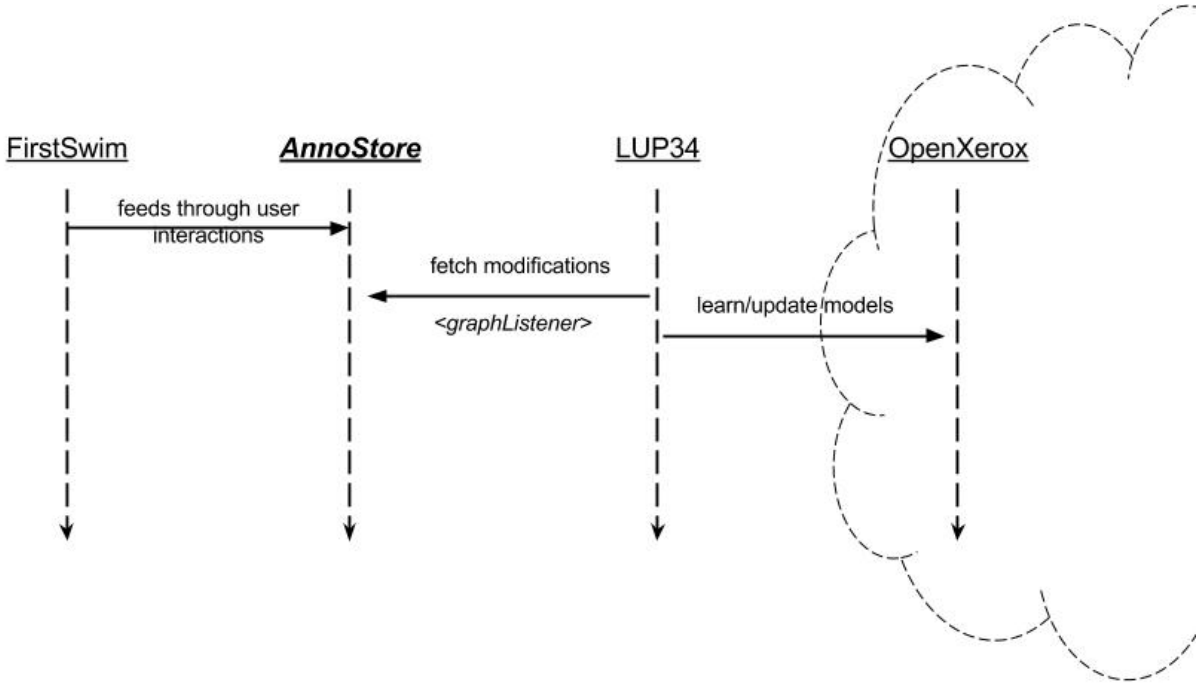

Figure 1. Listening Annostore, Updating models use-case

Fetching the modifications is done with the method:

```
public void graphChanged(List<GraphEvent> list);
```

which is triggered when a given triple is inserted into the Annostore. This method accesses the Annostore to get the important informations linked to the new annotation. Then, this method calls:

```
public void updateModels(HashMap<String,String> params);
```

which will use the OpenXeroxClient bundle to pass the proper parameters in order to update the learning models on the OpenXerox server.

Prediction

The other way of using the LUP modules is to invoke the prediction methods they are providing. This is the actual purpose of these modules: based on the learning process, it is possible for other Fusepool components to access predictions for different tasks. On Figure 6, the diagram sequence illustrates the Prediction use-case.

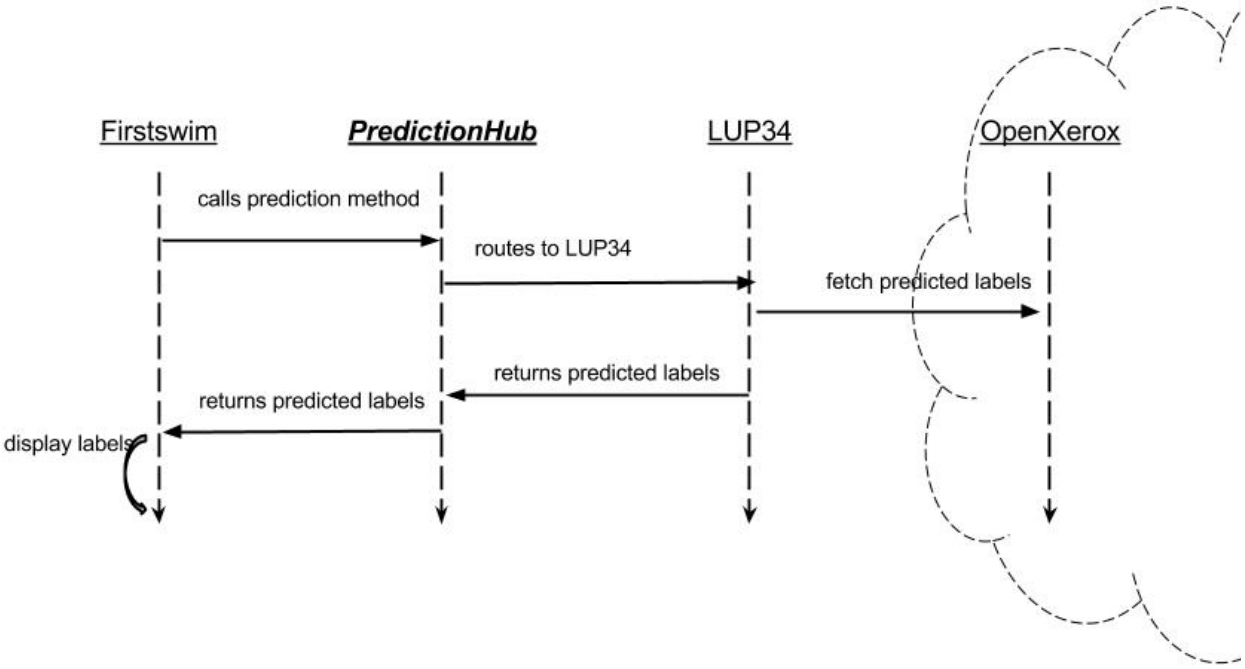

Figure 2. Prediction use-case

This sequence use one single method:

```
public String predict(HashMap<String,String> params);
```

…which will parse the *params* key-value pairs. From params we should get the document URI for which we need predicted labels. Then this method calls the OpenXeroxClient bundle to make the proper GET request to the OpenXerox service. This service will return the needed informations e.g. a list of predicted labels.

4. Ontologies – Labelling task

The ontology specification is crucial for an efficient way of working of the LUP of Task 3.4. This ontology summarizes the information needed to store annotations. The listener of this LUP module depends on particular set of triples that will be extracted from the Annostore knowledge base. The TripleFilter triggers a specific method in the LUP when a proper match is found among the newly inserted triples. On the Figure 3 one can find a graph representing the ontologies for this labelling task.

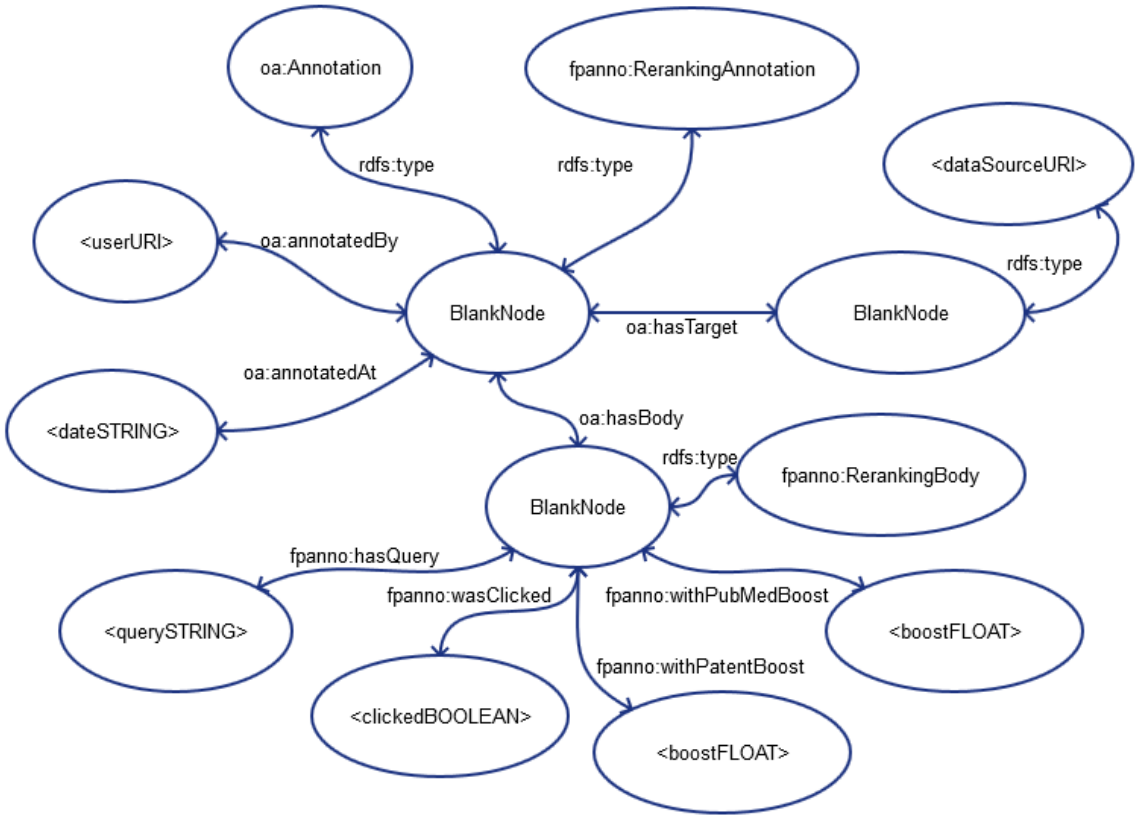

Figure 3. Annotations ontologies for the adaptive data sourcing task

Any triple of the format: `<null, rdfs:type, fpanno:RerankingAnnotation>` triggers the learning process of the LUP 2.5 component.

III LUP Module for Document Labelling

In order to allow the *Source Refinement* feature on the Fusepool platform we developed the *LUP25* bundle (see Section III.) We integrated the use of this bundle through the *PredictionHub* bundle (see Section II.) in the *EnhancedContentStore* in this way:

```
/**
 * Getting a reference to the predictionHub
 * component
 */
@Reference
private HubEngine predictionHub;
/**
 * Some ECS code...
 */

HashMap<String,String> predictionParams = new HashMap<String, String>();
/**
 * We build the set of parameters needed by the prediction system
 * in a HashMap<String,String>
 */
predictionParams.put("user", userName);
predictionParams.put("query", stuff);
/**
 * We access the predict method of the LUP2.5 through the predictionHub
 */
String predictedBoosts = predictionHub.predict("LUP25",predictionParams);
/**
 * The output is a couple key-value represented in a String where the key
 * is the datasource URI and the value is a boosting parameter (float)
 */
if (predictedBoosts.contains("##")) {
    for (String pair : predictedBoosts.split("##")) {
        String key = pair.split("__")[0];
        String value = pair.split("__")[1];
        value = value.substring(3, value.length()-2);
        log.info("Using boosts " + key + " : " + value);
        /**
         * For each boosting parameter we add a keyword condition
         * to the Lucene query
         */
        conditions.add(new KeywordCondition(RDF.type,
                       key,
                       Float.parseFloat(value)));
    }
} else {
    log.info("Server T2.5 not available");
}
```

Each time a search is made through the Firstswim GUI the ECS is involved. Before querying the index through the `indexService.findResource(…)` method we add some special `KeywordCondition` using the prediction of the *LUP2.5* module.

IV Learning Engine: low-ranked model for document labelling

1. Introduction

The learning engine is in charge of predicting a set of labels, which are user-defined strings, associated to documents displayed on a Hitlist in the Fusepool Graphical User Interface after a given search. On behalf of Listen Update Predict framework, the learning method that we propose consists in constructing a sparse matrix where each row is composed with:

- The bag of words of a labelled document
- A 0-1 encoding of the label assigned to each document

In order to predict the most probable labels for a given document, we propose to apply matrix factorization method in order to estimate the joint probability *p(d, l)* of a label *l* with respect to a document *d*.

2. Architecture

The learning engine associated to the Task 3.4 consists in estimating the joint probability of a label and a document represented as a bag of words. Indeed, the document label task consists in predicting the most probable labels of a text with respect to it bag of word representation when it appears in a hitlist within the set of labels provided by the users of the system. Furthermore, the FirstSwim Graphical User Interface will provide for each document the actual labels given by the user and the one predicted by the system.

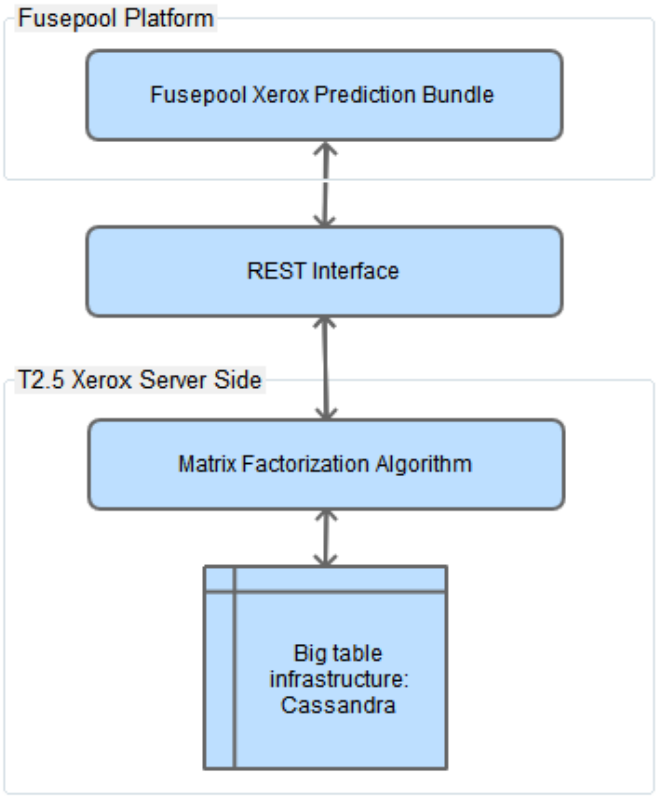

Figure 4. Interaction between components in the prediction engine.

Figure 4 describes the interaction of the prediction engine, also called the T3.4 Xerox Server Side infrastructure with the rest of the Fusepool modules. It communicates with the Fusepool OSGi platform by a HTTP/REST interface that is in charge of three atomic tasks (1) transmitting the documents to the server side (2) the labels associated to a document (3) predicting most probable labels for a given document. Having the documents transmitted to the server side will allow having a potential centralization of different corpora associated to several instances of Fusepool. In this context, it would be possible to fusion knowledge generated from user communities involved in different instances of the Fusepool system.

The actual specification of this Server Side REST interface is the same than the LUP3.4 presented in Section III.1. The T3.4 Server Side is decomposed in two modules : (1) the processed annotations are stored in a state of the art BigTable kind of infrastructure called Apache Cassandra [2] in order to scale up and (2) the matrix factorization algorithm that will be presented below.

3. Formalization

The matrix factorization algorithm used for learning and prediction is based on a stochastic gradient descent of a regularized Frobenius norm of the matrix of the annotations[3][4] as proposed for the T2.5 task of source refinement. In the actual matrix, *Y*, to factorize, a row corresponds to a document, we assume *Na* be the overall number of available annotations at a given time. A row is composed with the following information:

- The counted bag of word of a labelled document
- A 0-1 encoding of the label assigned to each document

For a given language, the words are stemmed and stop words are suppressed in order to reduce what is currently considered as a useless complexity of the model related to meaningless stop-words and various declinations of the same word. The actual goal of a matrix factorization method is to find two matrices, usually called embedding, *A* and *B*, satisfying the following equation:

$$(A, B) = argmin_{A,B} \left|\left|AB - Y\right|\right|$$

Where *A* is a real-valued matrix of size *Na* rows, and *k* columns; and *B* is a real-valued matrix of size *k* rows and *L* columns. Numerous methods have been developed in order to compute approximated solutions to this problem. In our context of big data, Stochastic Gradient Descent is considered the most usable and effective[5] compared to algorithms like least-square minimizations that are known to be costly especially in terms of memory consumption.

Having the embedding matrices, the prediction of the labels to a given document consists of finding a set of available labels maximizing the joint probability *p(d, l)*. So, by computing the dot-product on the actual factorization results of the matrix on the bag of word representation of a document this approach allows to predict the pertinence of a label.

The proposed approach provides several advantages. First, a particular level of flexibility, indeed the representation model of the document can easily be extended. For example, parsing information or natural language type of information can be incorporated to the representation of the document. Secondly, the potential similarities between documents and labels are exploited. Indeed, the proposed methodology is strongly related to the paradigm of collaborative filtering. Finally, the proposed approach is sufficiently non-restrictive in order to not limit the results of a search to a unique label but the list of *n* most probable.

[2] Avinash Lakshman and Avinash Lakshman, “Cassandra - A Decentralized Structured Storage System”

[3] Tamara G. Kolda, Brett W. Baderz, “Tensor Decompositions and Applications”

[4] Yehuda Koren and Robert Bell, “Matrix factorization techniques for recommender systems”

[5] Yong Zhuang and al, “A Fast Parallel SGD for Matrix Factorization in Shared Memory Systems”

4. Experiments

In this last section, we present a set of experimental results obtained in the actual platform. In a first section, we build an artificial corpus where several sets of documents are produced using different mixtures of n-grams. Because of the document representation we propose, the Bag of Word, the ordering of the words is not critical. Furthermore, we are able to control the distribution of words on each set and the overlapping of those distributions over sets. Then, a label is assigned to each documents of the same set and the resulting dataset is used as a learning set. Finally, test documents are produced with the word distribution used before hand and corresponding label are assigned. From that point, a test error can be measured. On a second part, we use a subset of document manually labelled in Fusepool and present the results.

V Conclusion

The predictive labelling task enables each user to classify documents in a way that hitlists become filled with richer kind of informations concerning the documents retrieved. It will be tested with the users in the following months and already provide encouraging results in an experimental setup. There are several ways it could be improved and in particular the way documents are represented. Indeed, representation like TF-IDF or adding of features like the authors can be useful in order to enhance the quality of the prediction. As a matter of fact, the proposed algorithm easily enable a LUP developer to enrich the representation of the documents.

This work also illustrates the generality of the Fusepool learning framework defined in D4.1, which enables us to quickly defined new learning components based on user feedback, without having to redefine the whole data lifecycle pipeline including user feedback, update of the model parameters, version updates and backup functionalities.