

EEXCESS

Enhancing Europe's eXchange in Cultural Educational and Scientific reSources

Deliverable D5.3

Second Prototype on User Profile and Context Detection, Usage Analysis Methods and Services

Identifier:	EEXCESS-D5.3-Second-Prototype-on-User-Profile-and-Context-Detection-Usage-Analysis-Methods-and-Services-final.pdf
Deliverable number:	D5.3
Author(s) and company:	Christin Seifert (Uni Passau), Jörg Schlötterer (Uni Passau), Nils Witt (ZBW), Johannes Jurgovsky (Uni Passau), Stefan Zwicklbauer (Uni Passau)
Internal reviewers:	INSA
Work package / task:	WP5, Task 5.1, 5.2 and 5.3
Document status:	Final
Confidentiality:	Public
Version	2015-10-30

History

Version	Date	Reason of change
1	2015-09-29	First draft and structure created
2	2015-10-12	Included contribution ZWB (usage analysis)
3	2015-10-13	Included entity disambiguation parts, added appendix structure
4	2015-10-14	Overview and summary drafts
5	2015-10-15	Added context detection and internal usage mining
6	2015-10-15	Finalized overview, introduction and summary
7	2015-10-19	Version for internal review (INSA)
7	2015-10-26	Final Version

Impressum

Full project title: Enhancing Europe's eXchange in Cultural Educational and Scientific reSources
Grant Agreement No: 600601
Workpackage Leader: Christin Seifert, Uni Passau
Project Co-ordinator: Silvia Russegger, JR-DIG
Scientific Project Leader: Michael Granitzer, Uni Passau

Acknowledgement: The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° 600601.

Disclaimer: This document does not represent the opinion of the European Community, and the European Community is not responsible for any use that might be made of its content.

This document contains material, which is the copyright of certain EEXCESS consortium parties, and may not be reproduced or copied without permission. All EEXCESS consortium parties have agreed to full publication of this document. The commercial use of any information contained in this document may require a license from the proprietor of that information.

Neither the EEXCESS consortium as a whole, nor a certain party of the EEXCESS consortium warrant that the information contained in this document is capable of use, nor that use of the information is free from risk, and does not accept any liability for loss or damage suffered by any person using this information

Contents

1	Executive Summary	5
1.1	Potential Risks	6
1.2	Next Steps	6
2	Introduction	7
2.1	Purpose of this Document	7
2.2	Scope of this Document	7
2.3	Status of this Document	7
2.4	Related Documents	7
3	Overview	8
3.1	Publications	8
4	Context Detection & Query Construction Concept	10
4.1	Detailed Context Detection and Query Construction per Granularity Level	10
4.1.1	Phrase Level	11
4.1.2	Paragraph Level	11
4.1.3	Page Level	12
4.1.4	Session Level	13
4.2	Entity and Category Detection	13
4.2.1	Named Entity Annotation	14
4.2.2	Category Annotation	14
4.2.3	Main Topic Detection	14
4.3	Keyword Extraction and Filtering	15
4.3.1	Keyword Extraction	15
4.3.2	Filtering	16
4.4	Embedded Context Detection	16
4.5	Summary and Future Work	17
5	Context Detection Library and Services	18
5.1	Software	19
5.1.1	Source Code and License	19
5.1.2	Installation and Usage	19
6	Context Detection Prototype: Browser Extension	19
6.1	Source Code and License	21
6.2	Installation and Usage	21
7	Resource Mining Concept	22
7.1	Upcoming work	22
7.2	Research questions	23
7.3	Datasets	23
7.4	Metadata augmentation	23
7.5	Network architecture	24
8	Resource Mining Prototype	24
8.1	Source Code and License	25
8.2	Summary	26

9 Privacy-Preserving Usage Analysis Concept	27
9.1 Intended Purpose	27
9.2 Usage Data	28
9.3 Privacy-Preserving Usage Mining	28
10 Privacy-Preserving Usage Analysis Libraries	29
11 Privacy-Preserving Usage Analysis Prototype	29
12 Summary and Future Work	30
13 Glossary	31
14 References	33
A Appendix: Source Code Documentation	36

1 Executive Summary

This deliverable describes the development and research related to user and usage mining in Y3. The work can be grouped in 4 tasks:

- Develop client-side user mining libraries that can be re-used by clients using Web technologies (JavaScript, HTML, CSS) (corresponds to Task 5.1 in DoW).
- Develop a feature-rich prototype using the context detection libraries, serving as test case for end user testing and example for developers of other clients (corresponds to Task 5.1 in DoW).
- Develop a prototype for analyzing usage of EEXCESS resources. Due to the client-server architecture of EEXCESS, this requires proper logging on the server as prerequisite (corresponds to Task 5.2 in DoW).
- Develop a prototype for mining of external resources (corresponds to Task 5.3 in DoW).

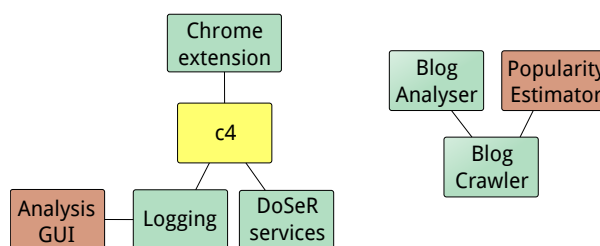


Figure 1: Overview of the main components developed in this work package indicating the status of development (green - finished, yellow - stable, but will be extended, red - development started)

In terms of components, the libraries, services and prototypes shown in figure 1 have been developed or are under development. The source code of all components is available from Github <https://github.com/EEXCESS/>, the README.md of each component describes its purpose, usage and the API if applicable.

C4 (SCientific and Cultural Content in Context) is a library encompassing all the context detection modules. As previous experiments had shown that the information need in Web context is better represented by a paragraph of a web site than the whole site, modules include paragraph detection, focus paragraph identification and noun phrase detection. Personalization is realized as follows: for each manual search the categories of named entities are stored comprising the user profile. This profile is then used to filter the keywords generated for the current paragraph leading to a more narrow search query. Moreover, a wrapper for the logging and disambiguation service is available there. C4 is currently used by other clients, namely the Wordpress plugin and the Moodle Plugin. Due to the restriction of the Google Add-on store, the Google Docs plugin implements its own context detection functions. C4 is available from Github <http://purl.org/eexcess/components/c4>.

DoSeR (Disambiguation of Semantic Resources) is a component for named entity disambiguation and category assignment. It is the basis for the client-side query generation. DoSeR requires a entity knowledge base, thus it is a server-side component with the service calls wrapped in C4. DoSeR is finished on the algorithmic level, additional languages will be added in the form of new knowledge bases (the language detection itself is located on the client). The source code can be found at <http://purl.org/eexcess/components/research/doser>.

Chrome Extension The Chrome extension is one of the main EEXCESS end-user prototypes and the most feature-rich w.r.t. context detection and personalization. The extension detects the paragraph of interest on a web page, extracts the keywords, enables user adaptations to the query constructed from these keywords, sends the query and presents the results. Personalization is implemented as keyword filtering based on previous queries. Mechanisms for learning on which page the extension should be switched on/off are also included. The Chrome Extension has reached a stable state, updates will be part of the C4 library. The extension is available in the Chrome Web store <http://purl.org/eexcess/clients/chrome-extension>, and the source code can be found on Github <http://purl.org/eexcess/components/chrome-extension>.

Logging has been implemented as a crucial prerequisite for the analysis of internal resource usage, and the development of the **Analysis GUI**. The logging concept is privacy-preserving, and a client-side logging library has been developed for the usage of all clients and is part of the C4 library. Concept and API definition have been done in cooperation with work package 6, work package 5 has then focused on the implementation of the client side logging.

The Blog Crawler has been further developed and finalized to improve the data collection, also for subsequent experiments. The **Blog Analyser** for linking Blogs to EEXCESS resources has been finalized. The development of the **Popularity Estimator**, a component for finding features that predict the popularity of scientific papers and blogs, has started. The source code of the crawler and the analyser is available from <http://purl.org/eexcess/components/research/blogcrawler> and <http://purl.org/eexcess/components/research/bloganalyzer> respectively.

1.1 Potential Risks

We investigated potential risks of the current prototypes and deployments and developed fall-back strategies, accordingly.

Server-side components not reachable: Depending on the number of clients simultaneously requesting services, it might happen that the server-side keyword detection (DoSeR service) is not available, and thus no automatic query can be constructed. In this case, the client will rely on client-side keyword detection methods, such as TextRank or TF-IDF based word filtering, where the TF-IDF statistics have been calculated from the browsing history. These methods are implemented already in C4, data collection for deciding which one would serve the purpose best is currently ongoing.

Restricted privacy-settings: External services such as DoSeR might not be allowed with restricted privacy settings. The fall-back solution in this case is the same as above.

1.2 Next Steps

The next steps in this project, apart from fixing bugs in finished components, are the following:

1. Finish the Analysis GUI to make EEXCESS usage statistics available to internal and prospective data providers.
2. Finish the Popularity Estimator to find external resources that relate to EEXCESS content.
3. Finish development of the context detection library. To this extend a user study is planned in Q4/2015 in which training data for model improvements will be collected.

2 Introduction

2.1 Purpose of this Document

This deliverable describes the second prototype for the functionality described in Task 5.1, 5.2 and 5.3. The source code for the prototype software components is available from open source repositories, URLs for the repositories are given in the executive summary (section 1) and in the respective section for each component.

2.2 Scope of this Document

This deliverable describes the software components for user and usage mining developed within work package 5. The focus of this deliverable are the the context-detection functionalities used in various clients, and the usage mining components. A prerequisite for analysis of internal usage mining is a proper logging. Logging comprises of a client and a server part. The logging client libraries are described in this deliverable, the server component is part of deliverable D6.3 [Mok+14]. The context detection concept has been implemented in various prototypes, with a focus on the Chrome extension. The Chrome extension GUI is not described here, a guided tour can be found in deliverable D2.4 [Sei+15a].

2.3 Status of this Document

This is a the final version of D5.3.

2.4 Related Documents

D2.4 Second Software Components for Presentation and Augmentation Interfaces [Sei+15a]

In detail the following section is of interest for the reader:

- Refer to section 3.1 for a guided tour of the prototype for context detection, the Chrome extension.

D6.3 Second Security Proxy Prototype and Reputation Protocols [Mok+14]

In detail the following section is of interest for the reader:

- Refer to section 5.2.2 for the logging server component.

D7.4 Second Prototype Integration and Deployment [Dop15]

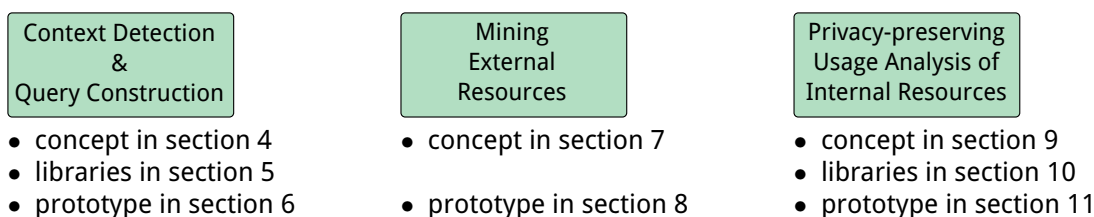
In detail the following section is of interest for the reader:

- Refer to section 7.1 for the Test Bed evaluation plan of the Chrome Extension Test Bed, including goals to collect data for evaluation and improvement of context detection algorithms.

3 Overview

The work in this work package can be structured into work on (i) context detection and query construction for personalization, (ii) usage mining of external resources, and (iii) privacy-preserving usage-mining of project-internal resources. This is in alignment with the three tasks in the DoW.

This deliverable presents the conceptual idea for each of the tasks, as well as developed libraries and prototypes (if applicable). The following figure gives an overview of the structure of this deliverable:



3.1 Publications

In WP5 the following publications have been accepted since the write-up of deliverable D5.2 [Sei+14] or are under submission. Some papers refer to prototypes that have been already presented in the last deliverable (but the paper had not been accepted at this time) and will not be referenced further in this deliverable. Papers relevant to the content of this deliverable are referenced again in the respective sections.

Under submission

- We submitted a paper that investigated the memory efficiency and robustness of word embeddings. In particular, we explored three methods for post-processing Skip-Gram word representations in order to reduce their required memory while still representing words accurately. The work is summarized in section 4.4.
- We submitted a paper presenting an approach to create interest profiles from external sources to alleviate the cold-start problem of personalization. In particular, we utilize the followees (the accounts a user follows) of a Twitter user, which does not require the user to tweet actively (i.e. she does not need to write tweets). We found that 7 out of 10 predicted interests are indeed relevant interests of the test users. An overview is presented in section 4.3.2.

User Mining and Personalization

- In this paper, we present a concept for creating search queries from the current user context. A revised version of this concept is presented in section 4 of this deliverable. The work received the **First Place at the ACM Student Challenge, 2015**.

Jörg Schlötterer. "From Context to Query". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. SAC '15. Salamanca, Spain: ACM, 2015, pp. 1108–1109. ISBN: 978-1-4503-3196-8. DOI: 10.1145/2695664.2696061. URL: <http://doi.acm.org/10.1145/2695664.2696061>

- In this paper, we present a procedure for collecting a personalization data set for long-tail content (using Europeana as search backend). The data set is used for first experiments on learning user queries from user selections. The data set was also used in work package 3 for query diversification experiments.

Christin Seifert, Jörg Schlötterer, and Michael Granitzer. "Towards a Feature-Rich Data Set for Personalized Access to Long-Tail Content". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, Apr. 2015

- In this paper, we investigate the three crucial properties of disambiguation systems (i) entity context (KB), i.e. the way entities are described, (ii) user data, i.e. quantity and quality of externally disambiguated entities, and (iii) quantity and heterogeneity of entities to disambiguate are investigated. More details are presented in section 4.2.1.

Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. "From General to Specialized Domain: Analyzing Three Crucial Problems of Biomedical Entity Disambiguation". In: *Proceedings of 26th International Conference on Database and Expert Systems Applications (DEXA)*. Springer, 2015

- In this paper, we investigate how the quantity of annotated entities within documents and the document count used for entity classification influence disambiguation results. More details are presented in section 4.2.1

Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. "Search-based Entity Disambiguation with Document-Centric Knowledge Bases". In: *Proceedings of the 14th International Conference on Knowledge Management and Knowledge Technologies (I-Know)*. Oct. 2015

- In this paper, we reviewed state-of-the art in entity disambiguation, with the focus on the biomedical domain. More details are presented in section 4.2.1.

Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. "Linking Biomedical Data to the Cloud". English. In: *Smart Health*. Ed. by Andreas Holzinger, Carsten Röcker, and Martina Zieffle. Vol. 8700. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 209–235. ISBN: 978-3-319-16225-6. DOI: 10.1007/978-3-319-16226-3_9. URL: http://dx.doi.org/10.1007/978-3-319-16226-3_9

- In this paper the usage of mobile sensors for search-based recommendations was investigated. The resulting prototype was presented as a demo.

Jörg Schlötterer et al. "From Context-Aware to Context-Based: Mobile Just-In-Time Retrieval of Cultural Heritage Objects". In: *Proc. European Conference on IR Research (ECIR 2015)*. Ed. by Allan Hanbury et al. LNCS 9022. Vienna, Austria: Springer, Mar. 2015, pp. 805–808. DOI: 10.1007/978-3-319-16354-3_90

Usage Mining

- This paper describes the usage mining experiments and results from the previous year on Blogs and Twitter data that lead to the new research direction with respect to usage mining.

Christin Seifert et al. "Digital Library Content in the Social Web: Resource Usage and Content Injection". In: *IEEE STCN Newsletter*. Vol. 3. 1. 2015. URL: <https://sites.google.com/a/ieee.net/stc-social-networking/e-letter/stcsn-e-letter-vol-3-no-1/>

- This paper describes the experiments for identifying economists on Twitter. It concludes that this is feasible but requires a well-curated author database that is not available for the cultural domain. The results contributed to the change of research direction in the usage mining task.

Alexander Böhm et al. "Identifying Tweets from the Economic Domain". In: *NLDB ISKO Workshop*. 2015

4 Context Detection & Query Construction Concept

In this section, we describe the general concept of context detection and the extraction of the relevant parts from the context in order to generate a search query profile. The concept was first described in [Sch15] and also in [Pas15]. Here we present a revised version with the focus on the implementation. In the web environment, observable context dimensions encompass first of all the web pages visited and in addition information like the user's location. We focus on the textual content of web pages as the primary source of contextual information. Further contextual dimensions (like for example mouse position) are used as additional cues to identify, select and filter relevant parts of the textual context.

We conduct a subdivision of the textual context into five levels of granularity (from fine-grained to coarse): *terms*, *phrases*, *paragraphs*, *pages* and *sessions*. We conduct this subdivision to cover scenarios demanding either narrow or broad focus. For example, when searching for information about the French Revolution, a timeline with the general course of events may be desirable in the beginning, whereas when reading a paragraph about the Marseillaise on a particular page, a sound clip of it is more likely to be relevant. Due to different characteristics, each of these levels requires its own treatment. Furthermore, a more coarse-grained level might influence the treatment of a more fine-grained level. Not all clients implement each level and some clients require adaptations of individual steps, but this section provides a comprehensive overview of the basic principles.

The goal of our context detection and query construction concept is to transform the user's context into a query profile for the federated recommender, in order to retrieve additional resources relevant to the task at hand. Therefore, in accordance with the user-based information seeking model of Marchionini and White [MW07], our steps towards a query profile consist of:

1. **identifying the relevant context**
2. **recognizing an information need**
3. **expressing this information need** (in terms of a query profile to the federated recommender)

Table 1 provides an overview of these three steps for the different levels of context granularity, which will be detailed in the next section. We omitted the term level, since it is covered by our approach for the phrase level by regarding terms as single term phrases.

Table 1: Context Detection & Query Construction Overview

context granularity	level detection method	information need detection	information need formulation	information need representation
phrases	text selection	TRUE (for selection)	Conditional Random Field model	terms
paragraph	web browser focus area	topic overlap with user profile	entity disambiguation and selection keyword detection	terms + entities (who, what, where)
pages	NONE	url & title classification	entity disambiguation and selection keyword detection	terms + entities (who, what, where)
sessions (sequence of pages)	topic similarity navigation patterns	session clusters	entity disambiguation and selection	terms + entities (who, what, where)

4.1 Detailed Context Detection and Query Construction per Granularity Level

This section provides a detailed description of the **relevant context identification**, **information need recognition** and **information need expression** for each context granularity level. The granularity levels

covered by the current deployment comprise the *phrase* and *paragraph* level. For the sake of completeness, we describe all levels and steps and explain what has been implemented already and how we plan to integrate the missing steps.

4.1.1 Phrase Level

Relevant context identification The most accurate way to identify the phrase currently read by the user is eye tracking - browser events, such as mouse movements or scroll position yield only limited accuracy [HPW11]. Thus, we rely on explicit user interaction in this case, i.e. a text selection, which is a strong indicator for reading focus [HPW11].

Information need recognition Given a text selection, we assume an information need implicitly.

Information need expression To gather ground truth data, we conducted an experiment, in which we had users select arbitrary pieces of text in web pages and issue queries to find resources relevant to that selection. It turned out, that most terms in the users' queries were already contained in the corresponding text selection. Given these results, we trained a conditional random field model (CRF), to determine which terms of the selection should be used as query, achieving almost 90% accuracy with 10-fold cross validation [SSG15].

We did not integrate the CRF into the current deployment, but instead apply the same approach to text selections as for the paragraph level. This is due to the fact, that an automatically extracted paragraph might not perfectly match what the user is currently reading. Hence, with a text selection, the user has the control to explicitly select a piece of text and set it as the current context. However, we plan to integrate the CRF and utilize it when the length of the selected text piece is below a certain threshold.

4.1.2 Paragraph Level

Relevant context identification In order to find the relevant paragraph in a web page (i.e. the paragraph, the user is currently looking at), it is first necessary to distinguish between paragraphs which convey actual information and irrelevant ones, such as navigational menus, advertisements, etc. For this very first step, we favor a simple approach, in order to keep the computational effort low and the response times high. A heuristic, based on a fixed length threshold of DOM text nodes already provided sufficient recognition performance and does not depend on a particular page structure. Hence, this heuristic is applicable to arbitrary web pages without influencing the user experience in a negative way.

Once the paragraphs are extracted, the next step is to determine the paragraph which is currently in the user's focus. We developed an approach, that takes into account the size of the paragraph, its position on the screen, the scroll position and the mouse position, as those have been shown to be able to serve as indicators for reading focus [HPW11]. However, we discovered that the selection process of this approach is not easily interpretable for users and prone to unintentional focus switches. Therefore, in the current deployment, we regard the topmost left paragraph as focused unless the user explicitly changes the focused paragraph by clicking on it, which gives the user more control and provides consistency. Nevertheless, we aim to further investigate a fully automatic approach.

Information need recognition In order to determine whether an information need exists for the paragraph, we need to distinguish between paragraphs that are relevant to the user (information need) and those that are not (no information need). Since there is no dataset available, that allows optimizing towards exactly this task, we use similar data obtained from Wikipedia: We crawled edits of registered authors and based on the edit history of a user, we aim to predict which paragraph the user will modify when she edits a previously unseen page. Preliminary results showed rather poor performance of standard classifiers on this dataset (F1 scores between 0.05 and 0.33, depending on the classifier), due to the highly unbalanced class distribution (only

15% of the paragraphs have been edited). This shows that determining the information need is a hard task, which cannot be solved with readily available classifiers and we are investigating the problem in detail, looking for appropriate solutions.

Since we are not able to determine whether an information need exists with a satisfying performance until now, we opted for a simpler solution for the current deployment: When the user looks at a paragraph for a certain amount of time or explicitly selects the paragraph, we assume an information need. In addition, the notification about the availability of additional results for the paragraph causes only a subtle change in the peripheral area of the display. Such changes are not recognized by the user when concentrating on a task and are automatically recognized when not concentrating on a task [YMK13]. With this approach, the results are presented in an unobtrusive manner.

Information need expression With about 71% of search queries containing named entities [Guo+09] and named entities providing semantic meaning, named entities naturally render themselves as good candidates for query profile construction. The extraction of named entities from a paragraph is described in detail in section 4.2. In the current deployment, the query consists of all of the extracted named entities.

However, the amount of entities extracted can be quite large, especially for long paragraphs. Therefore, it needs filtering, in order to provide precise results. We plan to filter the entities based on their frequency and the overlap of categories associated to those entities with interests in the user profile (also represented as categories). The interests in the user profile will be determined by the categories associated with named entities in queries for which the user had a look at the result set. See section 4.3 for details.

Furthermore, the named entity extraction may fail to find all relevant keywords in the paragraph or even be unable to extract any entity. In particular, the named entity extraction is optimized towards English text. While we plan to integrate a German version as well, we will not be able to cover all languages. Also, the computational effort makes it necessary to perform the extraction server-side, raising privacy issues. Therefore, we plan to integrate other techniques that can be implemented client-side as well. See section 4.3 for details.

4.1.3 Page Level

Relevant context identification By design, only a single page can be the *active* page in a browser window. We consider the *active* page relevant, even though this may not be true in the (rare) situation of a split screen with several browser windows or tabs.

Information need recognition In the current state, we assume an information need for all pages for the same reasons as for the paragraph level (result notifications are presented in the peripheral area of the display and can be easily ignored). In the near future, we will provide the user with the ability to temporarily deactivate the EEXCESS application for the current page and session and to completely deactivate EEXCESS for a particular page (and of course also the possibility to re-activate). We will collect statistics about the pages where EEXCESS has been (de-)activated and aim to train a classifier based on page URL and title, predicting whether EEXCESS should be active or not. For example, users will probably never have a need for additional resources when visiting their banking website and hence will prefer EEXCESS being inactive.

Information need expression The information need could in principle be expressed through the same mechanisms as used for the paragraph level. However, we will not create queries on page level, but only on levels below (paragraph, phrase). The levels below provide a more narrow focus, leading to more precise results and as a page always consists of at least one paragraph, we see this as the better option.

4.1.4 Session Level

As just mentioned, we only construct queries below the page level, i.e. for paragraphs and phrases. We describe the concept for the paragraph level though, since some parts still provide useful input for individual steps on the lower level. Those parts and their relation to steps on other levels will be outlined after the concept description at the end of this section.

Relevant context identification The relevant context of a session is a set of pages, which belong to the session. The first indicator for session boundaries is the topical coherence of subsequently visited pages. In some cases, this is not sufficient. For example the pages of a "reading online news" session may have diverse topics, but still belong to the same session. Preliminary experiments indicated that a small set of recurring sessions (such as the just mentioned "reading online news") constitute the main part of a user's browsing behaviour. This hypothesis is supported by the finding that few sites account for the majority of visits in a user's browsing history [Obe+07]. Therefore, we plan to cluster the user's visited pages by their frequency, in order to identify features of recurring sessions.

Information need recognition Recurring sessions typically do not exhibit an information need per se, as those are sessions such as "visiting institutional pages". Nevertheless, they still can exhibit an information need on page level or below. Consider again the "reading online news" example, in which an information need in online news itself does not exist, but certainly in the topics of the particular news articles. Hence, we neglect recurring sessions and focus on rare ones. Indicators for an information need in the latter case are visits of a search engine in between other pages or textual input to a search form field on an arbitrary page.

Information need expression Expressing the information need on session level could be achieved with the same approach as on page level (and hence paragraph level), with an aggregation of all the entities and keywords extracted from the pages that belong to the session.

Even though we do not construct queries on session level, the first two steps (**session identification** and **information need recognition**) still provide useful input for some steps in the levels below. For the construction of queries on paragraph level (c.f. 4.1.3), we aim to filter the query term candidates by interests in the user profile. This interest profile can be seen as a long term profile, while a session provides a short term profile, containing the associated categories of entities extracted within this session. Hence, the query term candidates can be filtered by a combination of long and short term profile. The information need detection on session level might be used as an additional feature for the information need detection on page level (among page url and title).

4.2 Entity and Category Detection

To detect entities and categories of paragraphs we use our **DoSeR**-framework¹. DoSeR offers a JSON REST interface which performs the following tasks to a given text snippet:

- Named Entity Annotation
- Category Annotation
- Main topic Detection

Our algorithm processes the tasks in the given order and returns the response in JSON format. In the following we briefly describe these tasks.

¹<http://purl.org/eexcess/components/research/doser>

4.2.1 Named Entity Annotation

Named Entity Annotation relies on two important subtasks: (Named) Entity Recognition and (Named) Entity Disambiguation. Entity recognition forms the first step of creating entity annotations. It identifies proper nouns (in the following denoted as **surface forms**) that can be linked to a semantic meaning. The task of entity disambiguation establishes links between identified surface forms and entities within a knowledge base (KB) and faces the problem of semantic ambiguity [ZSG15b].

In our work we focused on entity disambiguation with different KBs: Entity-centric KBs and Document-centric KBs. In this context we identified three crucial and well-known properties of (specialized) disambiguation systems [ZSG15a]. These are (i) entity context, i.e. the way entities are described, (ii) user data, i.e. quantity and quality of externally disambiguated entities, and (iii) quantity and heterogeneity of entities to disambiguate, i.e. the number and size of different domains in a knowledge base. We analyzed these properties with our ranking-based (Learning To Rank), publicly available disambiguation system **DoSeR** (Disambiguation of Semantic Resources). Our evaluation reveals that the choice of entity context that is used to attain the best disambiguation results strongly depends on the amount of available user data. Additionally, we show that disambiguation accuracy decreases with large-scale and heterogeneous KBs. Overall, we suggest to use a federated approach of different entity contexts to maintain the advantages of both approaches [ZSG15a].

While search-based, document-centric KBs perform excellent in specialized domains (i.e. biomedical domain), the question remains how the quantity of annotated entities within documents and the document count used for entity classification influence disambiguation results. Another open question is whether disambiguation results hold true on more general knowledge data sets (e.g. Wikipedia) [ZSG15c]. Our results indicate that search-based entity disambiguation with document-centric (KB) performs poorly on general domains (i.e. Wikipedia). Additionally, the results show that disambiguation accuracy increases when using short documents (e.g. Wikipedia paragraphs) instead of long article pages.

To provide robustness in terms of reliability and performance we apply the Named Entity Recognition and Named Entity Disambiguation system DBpedia Spotlight² instead of DoSeR. DBpediaSpotlight is one of the first semantic approaches (2011) and constitutes an entity-centric approach which is based upon DBpedia. Based on a vector-space representation of entities and using the cosine similarity, this approach has a public available web service. The service is able to recognize and disambiguate English and German language entities as determined in the request. Furthermore, we detect dates in documents and treat them like normal entities.

4.2.2 Category Annotation

Since we exclusively annotate Wikipedia/DBpedia entities, we are able to create statistics of categories that are associated with the entities extracted in Section 4.2.1. Given an entity, we extract a set of Categories³. After extracting all categories we create a category distribution given all categories of the identified entities.

4.2.3 Main Topic Detection

We provide the following two possibilities to extract the main topic of a paragraph given the set of extracted entities of Section 4.2.1: (i) PageRank with Word2Vec, and (ii) Doc2Vec. In the current implementation we use Doc2Vec as standard topic detection method. We note that the outcome might differ after some iterations since the inference step produces slightly different vectors after each step.

PageRank with Word2Vec In the PageRank with Word2Vec approach, we create a fully-connected, undirected weighted graph with entities extracted from a paragraph being the nodes. Each edge

²<https://github.com/dbpedia-spotlight/dbpedia-spotlight/wiki>

³<http://purl.org/dc/terms/subject>

describes the semantic similarity between two nodes. In our work the semantic similarity is the cosine similarity between the entities n-dimensional vectors. To create these vectors we use word2vec which generally takes a text corpus as input and produces word vectors as output. In our special case we use a corpus comprising entities only. When using the PageRank algorithm on the given graph we simulate a random walk on the graph. The node with the highest PageRank score represents the entity with the highest importance within the paragraph. Hence, the highest ranked entity represents our main topic.

Doc2Vec Generally based on Word2Vec, Doc2Vec produces a vector given a sentence or document. Hence, we use the entire input paragraph and infer a representative vector given a Doc2Vec model created on the Wikipedia corpus. We compare this vector with the vectors of the Wikipedia pages (entities) by computing the cosine similarity. The Wikipedia page (entity) with the highest similarity to the input paragraph represents the main topic. To significantly improve the performance we reduce the target entity set to those entities which have been annotated in the given paragraph (cf. Section 4.2.1).

4.3 Keyword Extraction and Filtering

Named entities extracted by the approach presented in the previous section 4.2 build the basis for our query term candidates. However, two problems can occur when using named entities:

1. The named entity extraction may fail to extract all relevant entities or even fail to extract relevant entities
2. The named entity extraction may extract too many entities

The first problem can be attributed to the coverage of the underlying knowledge base: If an entity is not represented in the knowledge base, it can obviously not be extracted. Moreover, the entity extraction is optimized towards a specific language. Currently, we provide support for English text. A misclassification occurs, when feeding German text to the entity extraction is the German preposition "mit" being recognized as "Massachusetts Institute of Technology (MIT)". Although we plan to integrate a German version as well, we are not able to cover other languages yet. Therefore, a language agnostic fallback solution is desirable. In the further course, we will refer to the extraction of query term candidates that are no named entities as keyword extraction. Section 4.3.1 presents our intended approaches to extract keywords.

Regarding the second problem of extracting too many entities, we need to apply filtering, in order to restrict the query terms to the most relevant (for the user). The filtering step might also be required for the keyword extraction (depending on the the particular approach). The filtering step is described in section 4.3.2.

4.3.1 Keyword Extraction

Besides named entities, noun phrases often contain the relevant bits of information in a sentence. In order to avoid privacy issues, we developed **NounPhraseJS**⁴ a JavaScript noun phrase detection, which can be executed on the client-side. **NounPhraseJS** achieves a classification rate of 94.8% on the CoNLL-2000 shared task dataset [TB00] with a training/test split of 80/20. In addition, we applied **NounPhraseJS** to date extraction, as alternative for the date extraction provided by **DoSer** (c.f. section 4.2.1). For this task, the WikiWars dataset [MD10] has been used and up to 98.9% of the terms have been correctly classified as temporal or non-temporal expression. While this approach can be executed client-side, it requires labeled data for the training. Hence, it also faces the problem of language dependence, namely the language of the labeled training data. Also, the list of extracted noun phrase may grow quite large, which makes filtering necessary.

⁴<https://github.com/EEXCESS/NounPhraseJS>

As another option, we are currently evaluating standard keyword extraction techniques, like the pure term frequency, tf-idf, BM25 and TextRank [MT04]. To this end, we developed a browser extension⁵ for Google Chrome and Mozilla Firefox. This browser extension extracts keywords with the aforementioned techniques from web pages and presents the top 5 to the user, who then can evaluate, whether those keywords are relevant to her on that page or not. The pure frequency of terms in a document is the most simple measure, but requires stopword filtering. TextRank requires a pre-processing step, assigning Part-of-Speech (POS) tags. Hence it faces the same problems as the named entity and noun phrase extraction. Tf-idf and BM25 can be used language independent and by design filter stopwords automatically (words that occur in every document in the corpus get assigned less weight). However, the two last mentioned techniques require a document corpus, which we obtain from the browsing history. We vary the amount of browsing history taken into account and present keywords extracted with different approaches to the user (without telling the user, which approach has been used). Preliminary results (530 ratings, obtained from 24 users) vote for TextRank as the best performing approach with an accuracy around 0.67. The best language independent approach is tf-idf with an accuracy around 0.61. For this measurements, the accuracy of tf-idf and BM25 has been averaged across different corpus sizes (the amount of browsing history taken into account) and may slightly improve, when only accounting for the optimal corpus size. However, up to now, the sample size is not yet large enough to draw definitive conclusions for the corpus size.

4.3.2 Filtering

As already mentioned, it might be necessary to filter the query candidates in order to narrow the focus. All of the approaches presented provide the ability to filter the query candidates by their weight and use only the top-k for the query. For those approaches, that do not assign a weight directly to the terms, the term frequency can be used as weight as a simple approach. However, the term frequency does not account for user specific needs, i.e. the query candidates are not personalized. Regarding tf-idf and BM25, the query terms are personalized by the user specific document corpus (the browsing history) as it influences their weights. Regarding the named entities, we plan to filter the query candidates based on a user profile. This user profile will be constructed of categories assigned to entities used in past queries. Since queries are created and sent automatically, we need to make sure, that the user is indeed interested in those categories. Therefore, in the construction of the user profile, we account only for those entities, where the user has viewed the corresponding result set.

With using only categories for the user profile, where the user has viewed the result set of a query constructed of the corresponding entities, a certain set of queries needs to be executed to fill the user profile: The approach suffers from a coldstart problem. We envisage two strategies to overcome this limitation: On the one hand, we will provide the ability for the user to explicitly state her interest in Wikipedia top-level categories and will spread this interest two lower categories. On the other hand, we aim to provide the possibility to fill the user interest profile by utilizing a user's already existing social media network account. We investigated this using Twitter as external source, and found that 7 out of 10 detected interests were indeed relevant for the test users. This approach also provides the possibility to just provide a Twitter account name, without the need for credentials. Hence, it could also be used by users that do not maintain a Twitter account themselves, but know a person with similar interests, who is on Twitter: They can simple use the Twitter name of this account.

4.4 Embedded Context Detection

Achieving high classification accuracy on Natural Language Processing (NLP) tasks (e.g., POS-tagging, noun phrase detection) often relies on expressive representations for words. It has been shown that unsupervisedly learned Word2Vec word representations (word vectors), estimated from large text corpora, improve the accuracy on many NLP tasks through their high-quality features. However, these word vectors must be computed in advance, i.e. before they can be used within a NLP classification

⁵<http://mics.fim.uni-passau.de/serverREL/RELEVANTICO/intro/en.html>

task. Once computed they provide a certain degree of accuracy boost but also require a lot of memory (60-150 MB) to be stored.

We were curious if we could exploit the benefits of these word vectors also for memory limited applications. Since there is little known about the robustness of word vectors against parameter perturbations and about their efficiency in preserving word similarities under memory constraints. In our work, we investigate three post-processing methods for word vectors to study their robustness and memory efficiency. We employ a dimensionality-based, a parameter-based and a resolution-based method to obtain parameter-reduced vectors and we provided a concept that connects the three approaches. We contrasted these methods with the relative accuracy loss on six intrinsic evaluation tasks and compared them with regard to the memory efficiency of the reduced vectors. The evaluation showed that the quality of PCA-reduced word vectors is, for some tasks, superior to vectors of equivalent size and that low Bit-resolution word vectors offer great potential for memory savings by alleviating the risk of accuracy loss

In particular, we could reduce the total memory requirement of these word representations by 75% without significant accuracy loss on several evaluation tasks. The results indicate that post-processed word vectors could also enhance applications on resource limited devices with valuable word features.

4.5 Summary and Future Work

Table 2 summarizes, which parts we already implemented, which parts we partially implemented, which parts we still plan to implement.

Table 2: Implementation status overview.

Already implemented and deployed	
Paragraph extraction	Available in the context detection library (c.f. section 5)
Focused paragraph detection	Available in the context detection library (c.f. section 5)
Named entity extraction	Available server-side (c.f. section 4.2)
Implemented in prototypes and planned to be integrated in the deployment	
Keyword extraction (tf, tf-idf, BM25, TextRank)	implemented for the keyword extraction experiment, need to integrate the best performing technique into the context detection library
Noun phrase extraction	Implemented as stand-alone prototype (NounPhraseJS), need to integrate into the keyword extraction process
Embedded context detection	Implemented as stand-alone prototype, need to integrate into NounPhraseJS
Twitter interest profile construction	Implemented as stand-alone prototype, need to deploy and make accessible via REST-API
Planned to implement	
Client-side CRF	Optimize query generation on phrase level
Page classification	Determining an information need on page level
Query candidates filtering	Limit the set of query candidates and personalize
Session detection	Provide a short-term profile for optimized filtering

The implementations in the deployment will be evaluated with a large scale user study in Q4 2015. More details can be found in deliverable D7.4 [Dop15] on page 28.

5 Context Detection Library and Services

Research and development in the context detection task lead to i) a client-side context detection library, and ii) a service for annotating entities and categories.

Client-Side Modules In this section, we start with an overview on the organization of the different client-side modules, in order for the reader to get the whole picture and then provide details for the context detection software. Figure 2 depicts the main components and their interplay. Basically, we

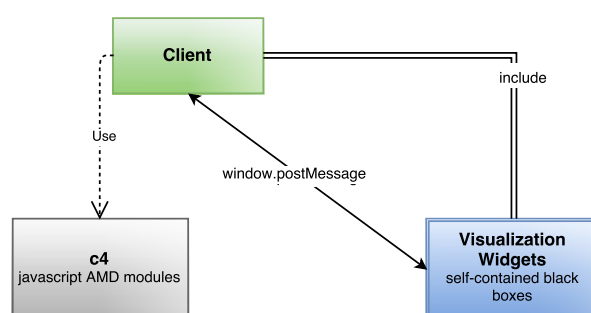


Figure 2: Overview of the client-side modules

have two module types: the components in C4 (Cultural and sScientific Content in Context) and Visualization Widgets. The latter comprise components, which do not need to be aware of the web page context. They are provided as self-contained web sites, which communicate with their environment via the *Web Messaging API*⁶. The main advantage of providing those widgets as self-contained pages and including them as iframes is that they do not inherit any layout definitions of the including page. Further, they do not add any elements to the including page (except the iframe itself) and hence are not prone to be affected by element modifications in that page. Also, developers who include these widgets do not need to care about their internals, but only send (and listen for) a well-defined set of messages. This set of messages is provided in the Appendix on page 54.

C4 comprises components, which either provide functions without any display elements (or at least only a small amount) or need a tight connection to the including web page. The parts relevant to this deliverable all reside in C4. In addition to the existing and planned context detection modules mentioned in the previous section 4.5, C4 features utility modules for server connections (for query requests, named entity extraction, logging) and window messaging, a module to create ready-to-use citations from JSON metadata and a module for adding a search bar to the bottom of a page, that allows interaction with the query. Details for the available modules and how to use them are provided in the Appendix on page 41.

Entity-Services To detect entities and categories of paragraphs we use the **DoSeR** framework (Disambiguation of Semantic Resources). DoSeR offers rest interfaces to annotate textual or tabular data with semantic annotations. The EEXCESS project uses the entity and category annotations interface to perform the following tasks: Named Entity Annotation, Category Annotation and Main topic Detection. In order to detect the main topics of paragraphs, we apply word2vec/doc2vec which relies on a Word2Vec Rest server. The REST Server is an important component in the DoSeR framework. Details for the available services and how to use them are provided in the appendix on pages 39 and 37.

⁶<https://w3c.github.io/webmessaging/>

5.1 Software

- Client-side context detection features such as the extraction of paragraphs and detection of the active paragraph are available in the C4 package. This package also includes utility (server connections) and augmentation tools (search bar, citation tool).
⇒ API description and usage details in the appendix on page 41.
- Server-side semantic enrichment of paragraphs with entities and categories is performed in DoSeR. DoSeR also provides several tools to investigate the underlying data (e.g. tables).
⇒ API description and usage details in the appendix on pages 37 and 39.

5.1.1 Source Code and License

- The source code of the C4 libraries is available on GitHub <http://purl.org/eexcess/components/c4>. The libraries are published under MIT license⁷.
- The source code of DoSeR is available on GitHub <http://purl.org/eexcess/components/research/doser>. The DoSeR library is published under GNU GENERAL PUBLIC license 2⁸.

5.1.2 Installation and Usage

- C4 is available as bower⁹ package. Hence it can conveniently be installed via “bower install c4”, which will load all the necessary files and dependencies. After installation, the desired modules can be included by providing “c4/<module_name>” to the require statement of RequireJS¹⁰.
⇒ API description and usage details in the appendix on page 41.
- DoSeR is a stand-alone Java library which starts an Apache Tomcat webserver. In order to work correctly it is necessary to download the doc2vec model as well as DBpediaSpotlight.
⇒ The download links and in-detail installation guides are given in the respective readme files in the appendix on pages 39 and 37.

6 Context Detection Prototype: Browser Extension

As an example for the usage of the context detection library, we describe the feature-richest prototype, which is the Chrome browser extension. In other clients, the modules are used analogous.

A screenshot of the extension on the Wikipedia page about Ada Lovelace is shown in figure 3. The paragraphs which are surrounded by a dotted gray border have been extracted from the page by the extension. The green border indicates the focused paragraph. The query term candidates which have been extracted from this paragraph via named entity detection are shown at the bottom of the page. They have been used to send a query to the EEXCESS federated recommender (via the privacy proxy). The number of returned results is indicated in the lower right corner. A guided tour of the Chrome extension is provided in Deliverable D2.4 [Sei+15a, section 3.1].

Figure 4 provides an overview of the extension architecture. It is composed of three main parts (web page environment, local extension and global extension environment), which will be detailed in the following.

⁷<http://opensource.org/licenses/MIT>

⁸<http://opensource.org/licenses/GPL-2.0>

⁹<http://bower.io/>

¹⁰<http://requirejs.org/>

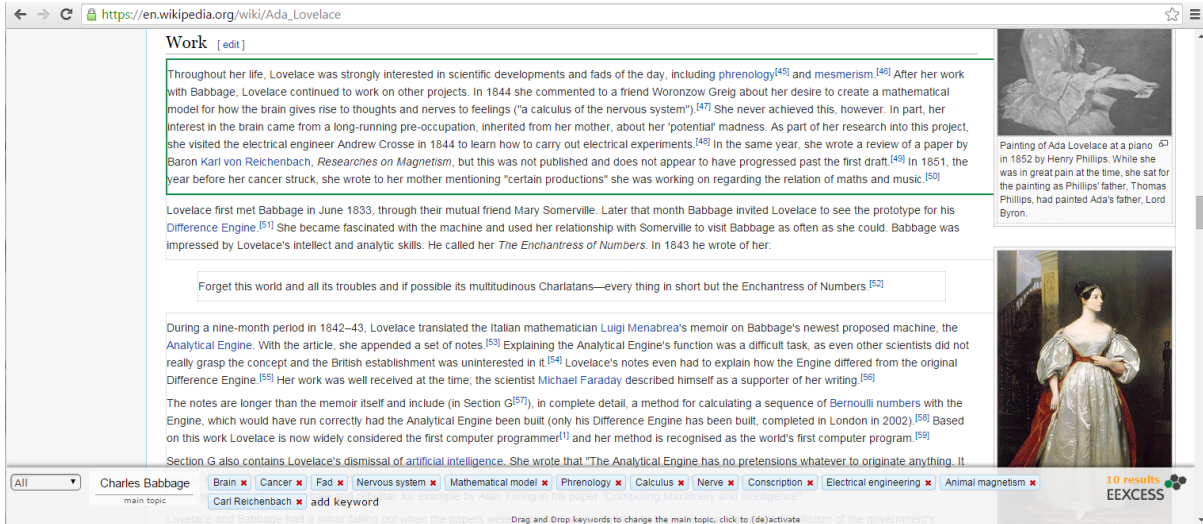


Figure 3: Screenshot of the Chrome extension with extracted paragraphs (outlined in light gray), focused paragraph (outlined in green), extracted keywords (bottom) and result indicator (bottom right above the EEXCESS icon).

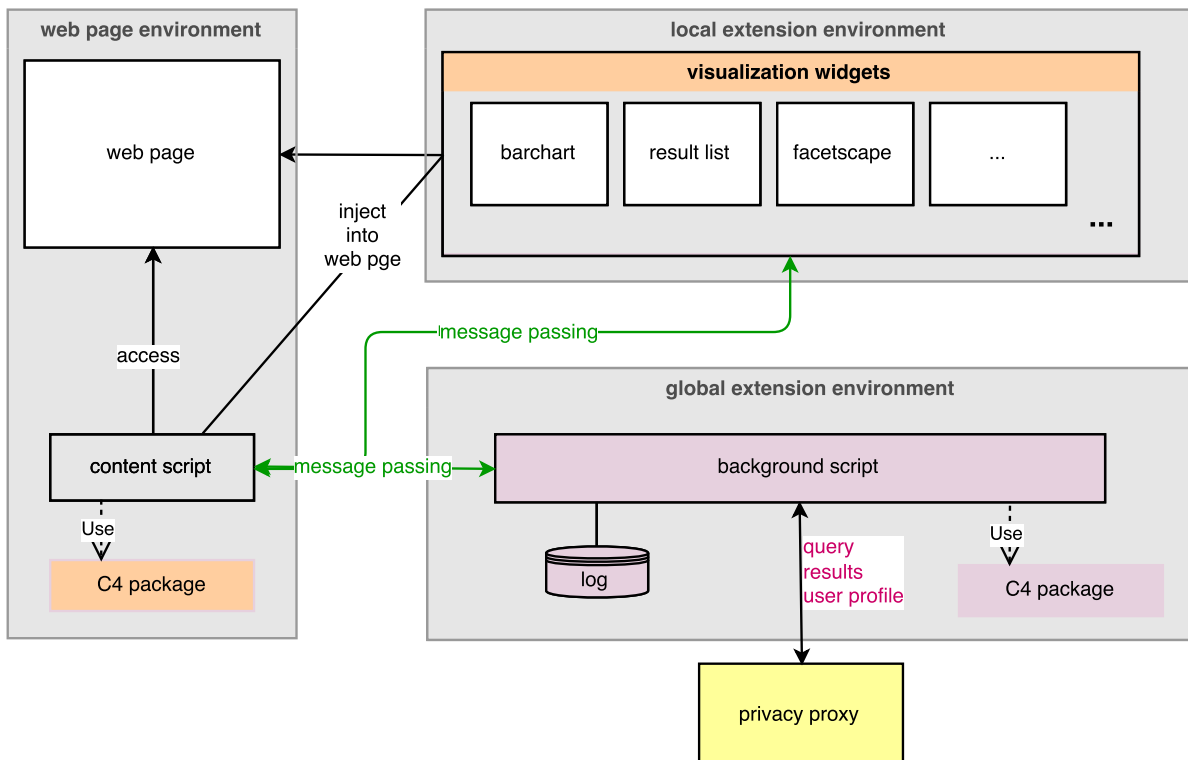


Figure 4: Browser extension architecture.

Local extension environment The visualization widgets reside in the local extension environment. This is similar to a regular web page environment, but instead of being accessible under some url like `http://<domain_name>`, the widgets are accessible at `chrome-extension://<extension_ID>`. This means, they behave like a regular web page. For example, when the same widget would be added to a web page in two different iframes, each frame would have a separate execution environment. Hence, the execution environment for the widgets is temporary. The widgets communicate with the content script via the web messaging API, e.g. the content script might send a message about new results and the widgets would display those results.

Global extension environment The background script is a single, permanent script in global execution environment. Hence it can store and share information across tabs, like past queries for example. The background script is responsible for the connection to the privacy proxy. Therefore, it makes use of *APIconnector* module of the C4 package. Before it sends a query request to the privacy proxy, it enriches the query profile provided by the content script by additional long term user profile information.

Web page environment The content script in the web page environment has two tasks: context detection within the page and augmentation of the page. For the latter, it adds the *searchBar* module of C4 to the page. This module displays a bar at the bottom of the page, which informs about new results and allows interaction with the query and corresponding results (via visualization widgets). By default, the *searchBar* directly queries the privacy proxy via the *APIconnector* module, while the content script provides a custom query function to the *searchBar*, which routes the query through the background script. This is necessary in order to be able to keep track of past queries and enrich the query with additional user profile features.

Regarding the context detection, the content script first extracts the paragraphs of the web page via the *paragraphDetection* module of C4 and tracks the focused paragraph via functionality provided by the same module. Whenever a focused paragraph is detected, an according event will be thrown with the paragraph attached and the content script retrieves the query terms for this paragraph via the *paragraphToQuery* method, also provided in the *paragraphDetection* module. This methods in turn retrieves the query terms via a REST call to the **DoSer** framework. Once the query terms are available, the content scripts instructs the *searchBar* to display them and the *searchBar* in turn will trigger the provided query function.

6.1 Source Code and License

The source code of the EEXCESS Chrome browser extension is available from github <http://purl.org/eexcess/components/chrome-extension>. The extension is published under MIT license¹¹.

6.2 Installation and Usage

The ready-to-use version of the Chrome extension can be installed from the Chrome webstore by visiting <http://purl.org/eexcess/clients/chrome-extension> with a supported browser (Chrome or Chromium).

To setup the Chrome extension for development, you first need to checkout the source code. Afterwards, you need to run “npm install” to load the required node modules (requires node.js¹²). The final step to load all dependencies is to run “bower install”. Once you have completed these three steps, navigate to “chrome://extensions” in your Chrome browser, activate the developer mode and then you will be able to add the extension via “load an unpacked extension”.

¹¹<http://opensource.org/licenses/MIT>

¹²<https://nodejs.org/en/>

7 Resource Mining Concept

This section describes the current state and the subsequent work of the resource mining component. As already depicted in D5.2 [Sei+14], the purpose of this component is to identify publicly available resources (like blogs or research papers) and utilize them in a way that increases the value of EEXCESS items. The first approach was to find blog posts that dealt with economic topics, which mentioned or linked to EconBiz resources. The benefit of such a mapping is, that it enlarges the amount of available resources that can be subject to a recommendation. This means, that not just items that can be found in partner systems can be recommended, but also items beyond that scope.

Unfortunately, the results of the first experiments were sobering [Sei+15b], which led to a modification of the goals of this component. What we are currently working on is a framework that is able to predict the relevance of a text merely based on the text itself. This means, given some text of adequate length, the algorithm can estimate the likelihood that this text will gain significant attention within the typical target group. We start with a set of research papers and then switch to the set of blog posts that the blog crawler has already fetched (see section 8).

In order to achieve this goal, state-of-the-art machine learning methods will be used. Based on self-taught learning [Rai+07], a deep neural network will learn a model of the English language which in turn will act as the basis for a model that is able to predict the relevance of a text document. A similar approach is described in [ZZL15].

Estimating the relevance of text is potentially useful for recommendation systems. Such a system could be used to penalize text items with a low relevance score and boost those with a high relevance score. Beyond the scope of this project, it could for example also play a role in the publishing process of research papers.

7.1 Upcoming work

Estimating properties from text documents by means of machine learning is a well-known approach that was able to yield reasonable results. For example Lipka and Stein have been able to identify featured Wikipedia articles from text [LS10], Ashok et al. gained promising results in discriminating good from very successful novels [AFC13] and Louis and Nenkova did research on the question how specific the content of a sentence is (i.e. they were able to distinguish specific from more general sentences) [LN12]. The usual procedure for tasks like this was to identify features that have strong correlation with the classes (or categories) that one wants to classify documents into. After this was done, a program was written to extract these features from training documents in order to train a classifier (say a Support Vector Machine [BGV92]). Subsequently, this trained classifier was used to classify new documents with respect to the features at hand. In this workflow a lot of effort was spent for the process of feature engineering [MM93; Gri07; Luy11] which needs to be repeated whenever other features are to be examined.

In contrast, in deep learning the feature engineering is done by machines instead of humans [Rai+07]. Hinton [Hin07] introduced an efficient method to pre-train neural networks which thereafter became part of several state-of-the-art machine learning systems (e.g. in speech recognition [Den+13] [Hin+12] and object recognition in images [Le13]).

Based on this approach this experiment will leverage deep learning techniques aiming at understanding text structures and thereby gaining the ability to estimate the relevance that a text document might have in the future. During pre-training the deep neural network (DNN) will learn to some degree the fundamental structures of the English language (the concept of words, syntax, punctuation etc.), similarly to the approach of Sutskever et al. [SMH11]. In a subsequent step the DNN will learn to discriminate relevant documents from irrelevant documents (on the basis of the knowledge learned in the previous step).

7.2 Research questions

The main research question we want to address is:

Can the relevance of a text document be estimated properly by a DNN, merely through the text itself?

Other research questions that we want to study include:

- Which features does a DNN learn from text corpus? Can they be visualized and understood by humans?
- Which input format is appropriate to learn the required features?
- How does the feature learning dataset affect the classification results and the ability to generalize? Can a model trained on a dataset containing only papers from a specific domain be applied to a papers from several domains (cross-domain learning).

7.3 Datasets

The learning process comprises two phases:

1. **Unsupervised pre-training.** During this phase the network is initialized by learning fundamental principles of the pre-training data (i.e. the language model).
2. **Supervised training.** During this phase the network learns to map the previously learned concepts to labels. Backpropagation [Wer74] is then used to refine the weights in order to achieve reasonable classification results.

Both training phases require separate dataset with different characteristics:

- **Pre-training corpus (feature learning):**
This corpus is currently undetermined. Candidates include open access repositories that contribute to RePEc and arXiv.org. It is required to have a dataset that is vastly greater than the training set.
- **Training, validation and test corpus:**
ZBW's open access repository EconStor¹³ contains 100k economic research papers. After augmenting the dataset with relevance labels, the dataset will be split randomly into three chunks:
 - training data (70%),
 - validation data (20%),
 - test data (10%).

The training set will be used for training, the validation set will be used to prevent overfitting and the test set will be used to assess the overall performance of the DNN.

7.4 Metadata augmentation

RePEc¹⁴ is among the largest providers for academic material (over 1,200,000 papers) and the largest for the economic domain, which makes it representative for this specific domain. What makes RePEc particularly interesting for the proposed research is its citation graph, that counts the citations between the RePEc papers. It should be noted that RePEc does not host articles itself. It only provides the metadata, whereas the hosting is done by several open access repositories (like EconStor). Figure 6 and 5 show the same paper hosted at EconStor and RePEc. The import thing to note here is that

¹³<http://econstor.eu>

¹⁴<http://repec.org>

Institute	# of articles in RePEc
EconStor	28,600
Deutsches Institut für Wirtschaftsforschung (DIW)	4,500
European Regional Science Association (ERSA)	4,300
Fondazione Eni Enrico Mattei (FEEM)	1,500
Forschungsinstitut zur Zukunft der Arbeit (IZA)	9,650
Humboldt-Universität zu Berlin	800
ifo Institut - Leibniz-Institut für Wirtschaftsforschung, München	5,500
Institut für Arbeitsmarkt- und Berufsforschung (IAB), Nürnberg	500
Institut für Makroökonomie und Konjunkturforschung (IMK)	270
Institut für Weltwirtschaft (IfW), Kiel	2,000
Inter-American Development Bank, Washington, DC	800
SFB/TR 15 Governance and the Efficiency of Economic Systems	500
Volkswirtschaftliche Fakultät, Ludwig-Maximilians-Universität München	400
Tinbergen Institute, Amsterdam and Rotterdam	2,300
University of California	500
Sum	62,120

Table 3: Number of EconStor articles that are on RePEc

Econstor provides a PDF containing the full text whereas RePEc provides citation information. Both services provide an API that allows to easily merge these information.

But not all EconStor article can be found on RePEc. Therefore, we estimated the expected coverage. Table 3 shows in the first row the number of EconStor articles that are directly referenced by RePEc. The rest of the table shows the number of articles that can be found on RePEc as well as on EconStor but in those cases RePEc does not link to EconStor. This means that at least 62k citation counts can be found. Presumably the number is even higher, because only the 15 largest institutes were taken into account.

7.5 Network architecture

Convolutional network [LeC+98] with (max or mean) pooling followed by fully connected layers on top will be used. In case overfitting is observed, regularization (e.g. dropout) will be used between the fully connected layers. Data in the input layer will be represented on character level, as this has shown interesting results in similar tasks [ZZL15]. The output layer will only indicate whether a text is estimated to be relevant or not, instead of indicating a continuous value. It will not indicate how popular it might be. The depth and the number of neurons per layer will be determined during the experiment, starting with 3-4 convolutional layers and 3 fully connected layers with 1024 neurons per layer.

8 Resource Mining Prototype

In this section the two existing components, the Blog Crawler and the Blog analyzer) are described. The question, that led to the development of the blog analyzer was:

Can we establish backlinks from manually selected economic blogs to EconBiz resources?

That is to say can we implement a mechanism, that, in a first step, identifies resources (like topics, persons, publications) in blog posts and in a second step searches EconBiz to detect if these resources are included in EconBiz.

The Weirdest People in the World?

Joseph Henrich, Steve J. Heine and Ara Norenzayan

No 139, [Working Paper Series of the German Council for Social and Economic Data](#) from [German Council for Social and Economic Data \(RatSWD\)](#)

Abstract: Behavioral scientists routinely publish broad claims about human psychology and behavior in the world's top journals based on samples drawn entirely from Western, Educated, Industrialized, Rich and Democratic (WEIRD) societies. Researchers—often implicitly—assume that either there is little variation across human populations, or that these “standard subjects” are as representative of the species as any other population. Are these assumptions justified? Here, our review of the comparative database from across the behavioral sciences suggests both that there is substantial variability in experimental results across populations and that WEIRD subjects are particularly unusual compared with the rest of the species—frequent outliers. The domains reviewed include visual perception, fairness, cooperation, spatial reasoning, categorization and inferential induction, moral reasoning, reasoning styles, selfconcepts and related motivations, and the heritability of IQ. The findings suggest that members of WEIRD societies, including young children, are among the least representative populations one could find for generalizing about humans. Many of these findings involve domains that are associated with fundamental aspects of psychology, motivation, and behavior—hence, there are no obvious a priori grounds for claiming that a particular behavioral phenomenon is universal based on sampling from a single subpopulation. Overall, these empirical patterns suggests that we need to be less cavalier in addressing questions of human nature on the basis of data drawn from this particularly thin, and rather unusual, slice of humanity. We close by proposing ways to structurally re-organize the behavioral sciences to best tackle these challenges.

Keywords: [external validity](#); [population variability](#); [experiments](#); [cross-cultural research](#); [culture](#); [human universals](#); [generalizability](#); [evolutionary psychology](#); [cultural psychology](#); [behavioral economics](#) (search for similar items in EconPapers)

New Economics Papers: this item is included in [nep-cbe](#), [nep-cwa](#), [nep-evo](#), [nep-exp](#), [nep-hpe](#) and [nep-neu](#)

Date: 2010

References: [Add references at Site](#)

Citations [View citations in EconPapers \(63\)](#) [Track citations by RSS feed](#)

Figure 5: Working paper hosted at RePec with citation information highlighted.

In case of success this approach could enhance the federated recommender with links to related social media content. For instance, users could receive recommendations not only including the EconBiz resource, but also a reference to the blog post that deals with that resource.

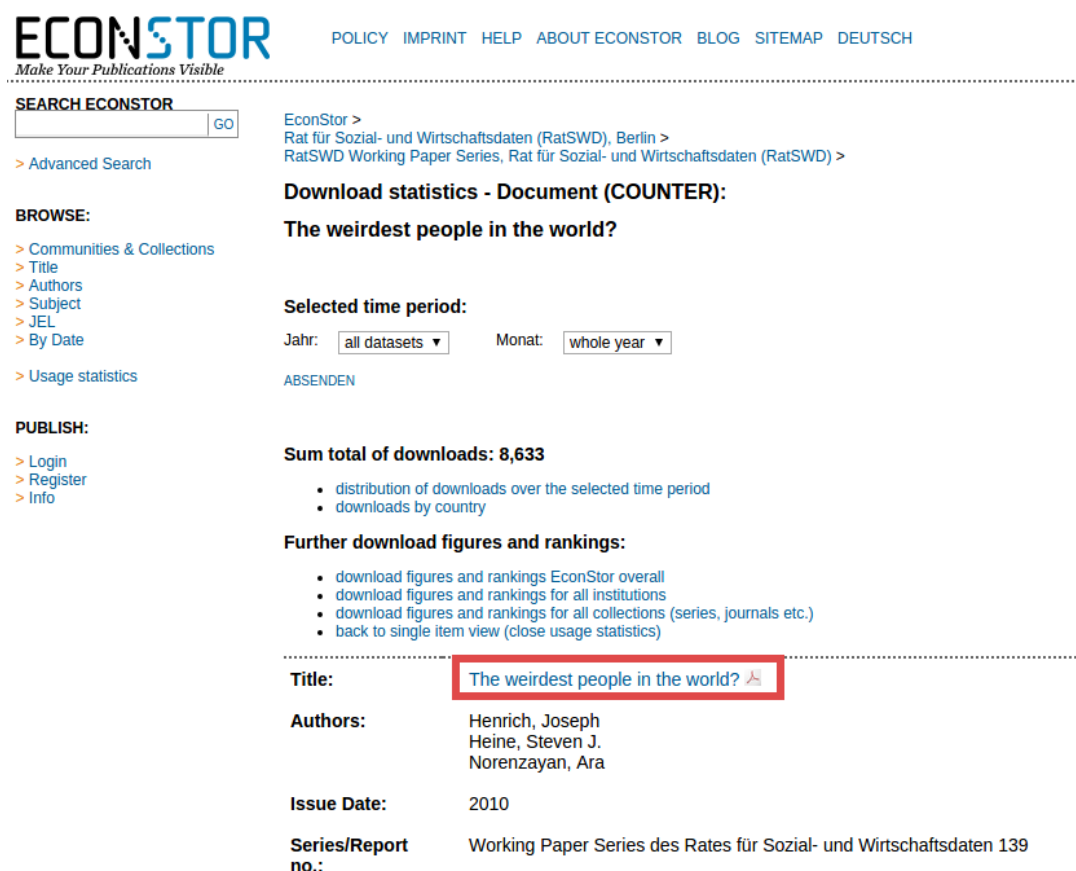
8.1 Source Code and License

The source code of the components described in this section is available via the following URLs:

- Blog Crawler <http://purl.org/eexcess/components/research/blogcrawler>
- Blog Analyser <http://purl.org/eexcess/components/research/bloganalyzer>

This code is released under the conditions of the Apache 2.0¹⁵ license. The API documentation can be found in the appendix on pages 57 and 59.

¹⁵<http://www.apache.org/licenses/LICENSE-2.0>



ECONSTOR Make Your Publications Visible [POLICY](#) [IMPRINT](#) [HELP](#) [ABOUT ECONSTOR](#) [BLOG](#) [SITEMAP](#) [DEUTSCH](#)

SEARCH ECONSTOR

> Advanced Search

BROWSE:

- > Communities & Collections
- > Title
- > Authors
- > Subject
- > JEL
- > By Date
- > Usage statistics

PUBLISH:

- > Login
- > Register
- > Info

EconStor >
Rat für Sozial- und Wirtschaftsdaten (RatSWD), Berlin >
RatSWD Working Paper Series, Rat für Sozial- und Wirtschaftsdaten (RatSWD) >

Download statistics - Document (COUNTER):

The weirdest people in the world?

Selected time period:

Jahr: Monat:

ABSENDEN

Sum total of downloads: 8,633

- distribution of downloads over the selected time period
- downloads by country

Further download figures and rankings:

- download figures and rankings EconStor overall
- download figures and rankings for all institutions
- download figures and rankings for all collections (series, journals etc.)
- back to single item view (close usage statistics)

Title: [The weirdest people in the world?](#)

Authors: Henrich, Joseph
Heine, Steven J.
Norenzayan, Ara

Issue Date: 2010

Series/Report no.: Working Paper Series des Rates für Sozial- und Wirtschaftsdaten 139

Figure 6: Working paper hosted at EconStor with link to full text highlighted.

8.2 Summary

While the blog analyzer will not be developed further, the blog crawler will see some new features in the last year of the project that will allow it to fetch a larger amount of blog posts. This is required in order to make the relevance estimator work reasonably. as DNNs need large data sets to perform well. But, prior to that, we will first investigate the potential of this technique on a data set made up of research papers, which can be acquired much more easily than blog posts.

9 Privacy-Preserving Usage Analysis Concept

In this section we describe means to conduct usage analysis of the EEXCESS framework while preserving the users' privacy. In order to analyze how users interact with EEXCESS components and services, we need to collect usage data. In EEXCESS, usage data is both a valuable resource for improving our services and a sensitive resource in terms of privacy. Therefore, we developed the usage mining concept with both these requirements in mind. For acquiring usage data we log user interactions and for privacy-preserving usage mining we adopt a k-Anonymity approach, that hides individual users in a group of similar users. In the following, we first review the purpose of usage analysis, then we describe the data we collect and finally we present a concept that permits usage analysis without disclosing private information about individual users.

9.1 Intended Purpose

Usage Analysis in EEXCESS serves the following purposes:

- **Organizational:** Provide a mechanism that makes the uptake of the EEXCESS framework transparent.
 - Reporting: In order to compile an in-time assessment of the quality of the services of EEXCESS, we require a well-defined process that gathers information from all components in a consistent format.
 - Planning: For strategical decisions, regarding the integration of new partners and the exploitation of additional content injection scenarios, we need to keep track of temporal usage behavior and trends.
- **Technical:** Identify potential improvements in individual EEXCESS components.
 - Automatic query generation: Usage statistics can be used to improve the query generation process. Since EEXCESS offers the capability of automatically inferring queries from the page, paragraph or sentence level context of a user, knowledge about interest groups can be helpful in guiding this inference process.
 - User interfaces: EEXCESS offers a variety of different types of search interfaces and result visualizations. In order to best support users in finding the resources they are interested in, we need measures to analyze how users interact with the components.
 - Federated Recommender: Usage data can provide valuable input to the Federated Recommender to perform source selection, i.e. to decide to which content providers the query should be forwarded to retrieve the best results.
- **Promotional:** Provide live feedback to various interest groups. In particular, these are analysts working for digital library on the content provider side, and individual user groups on the content consumption side.
 - Integrated content provider: Digital libraries which are already integrated into the EEXCESS ecosystem shall be kept informed about the usage of their resources.
 - Prospective content provider: For future content providers we want to provide information about the current coverage of topical domains and the thematic interests of user groups.
 - Users: We provide the same information as aggregated statistics for users to view their usage of EEXCESS in the context of the interests of other user groups.

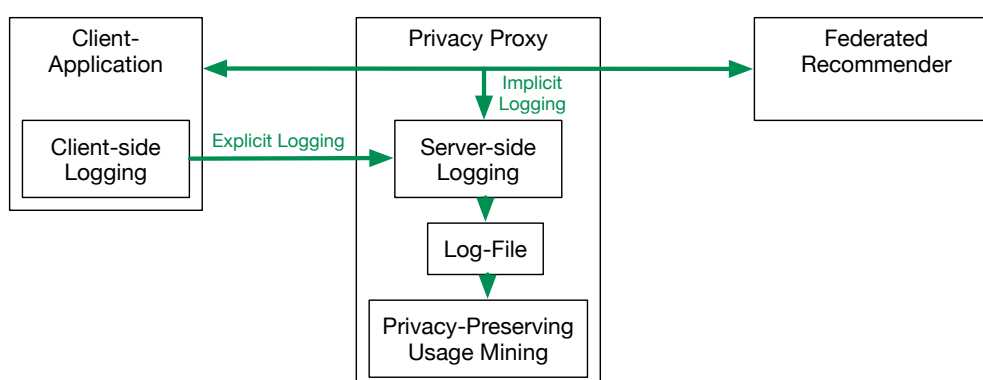


Figure 7: Client-side and server-side logging of usage data.

9.2 Usage Data

To provide reliable information for the purposes mentioned above, we need to collect usage data from the client-applications of EEXCESS. The data collection is handled by a client-side and a server-side logging component. The server-side component resides on the Privacy Proxy and implicitly logs all queries and responses that pass through the proxy. In contrast, the client-side component has to be integrated by client-developers into their applications. The client-side component provides the functionality to create pre-defined logging events from user interactions and to send these to the Privacy Proxy (see also Figure 7). All usage data is stored on the Privacy Proxy. In particular, we collect the following data:

- Queries and Responses
- Details Queries and Details Responses
- User interactions with client components:
 - Opening of a visualization
 - Closing of a visualization
 - Usage data of a visualization
- User interactions with resources:
 - Opening of a resource in detailed view
 - Closing of a resource in detailed view
 - Usage of a resource as text citation
 - Usage of a resource as image citation
 - Usage of a resource as hyperlink citation
 - A user's rating of a resource

The client-side logging component is described in Section 10.

9.3 Privacy-Preserving Usage Mining

This section describes the usage mining concept with regard to privacy-preservation. In contrast to a user's interactions with search interfaces and result visualizations, the usage and rating of resources bears much more sensitive private information about users. Therefore, we focus on preserving the privacy of this kind of information. With usage mining we seek an answer to the question: *What kind*

of resources were actually useful for certain kinds of users? This question suggests that we require a textual description of user interests and a textual description to characterize resources. The usefulness is quantized by the number of explicit interactions with a resource, which we obtain through logging.

A user is represented by his query history. Queries can be issued either manually by searching for particular keywords or generated automatically from the user's context (see also Section 4). Therefore, the query history consists of a list of keywords the user explicitly searched for and keywords that were automatically extracted while navigating through various contexts. We consider this set of keywords as the description of a user.

Similarly, a resource is also represented as a set of keywords that we extract from the resource's annotations. For the purpose of privacy-preserving usage mining we relate user representations to resource representations via a *user-item-matrix*. An entry in this matrix is the user-interest score for an item. We derive each score from the number of times a particular user interacted with a particular item, i.e. we compute a weighted sum of the individual resource interactions listed above (Section 9.2). The result is a user-item matrix in which the rows correspond to users, the columns correspond to items and the entries correspond to the degree of interest of a user for an item.

To enforce privacy guarantees on the resource consumption behavior of an individual user, we hide her in a group of at least k users that have a similar set of keywords. An individual user is thus indistinguishable from the $k - 1$ other users that share the same keywords (k -anonymity). The grouping can be performed by applying a clustering algorithm (e.g., hierarchical, autoencoder, expectation maximization, k-means, etc.) on the keyword sets of all users. Therein, the research question is how the clustering methods can be modified to guarantee a minimum group size k . After having obtained a user group, we can compile a set of user-group keywords from the keyword sets of all users in that group.

For all users within a group, we collect the items that have high user-interest scores in the user-item matrix. Similar to before, we compile a set of item-group keywords from the item descriptions. Finally, this procedure enables us to derive a textual description for items, which a group of users is interested in. Here, the research question is how to build abstract topic descriptions from very specific keywords. Usage data, which is not related to user interests, can be analyzed directly and provided as aggregated statistics without privacy disclosure.

10 Privacy-Preserving Usage Analysis Libraries

The acquisition of usage data is implemented in the client-side logging library. Client-developers are asked to include this library in their applications in order to enforce consistency of the logged data. This library is available as a self-contained module from the C4-package¹⁶. A detailed description and the source code documentation of the module is given in the Appendix on page 41. The server-side logging component is described in deliverable D6.3 [Mok+14].

11 Privacy-Preserving Usage Analysis Prototype

The usage mining prototype development has been started. Currently, we are developing a web application which consists of a usage mining component as backend and a publicly accessible web interface as frontend. The usage mining component regularly crawls the log-files and updates the user-item matrix along with various aggregated statistics (i.e. number of users, number of resources consumed per content provider, etc.). The web interface displays these statistics by means of comprehensive standard visualizations. Similar visualizations can potentially be included in client-applications as well.

¹⁶<http://www.purl.org/eexcess/components/c4>

12 Summary and Future Work

In this deliverable we presented the current status of the user and usage mining prototypes. The core component for user mining is the context detection library C4, which is modularized and can be used by any web-based client. C4 also wrapped the logging functionality and the server-side component for detecting entities and entity categories for a given text (DoSeR). Resource mining development continued along two lines: analysis of project-internal resources with the prerequisite of privacy-preserving logging, and popularity estimation of external resources with the focus on blogs.

Future work on user and usage mining will focus on these three components. Our goal is to

1. Finish the Analysis GUI to make EEXCESS usage statistics available to internal and prospective data providers.
2. Finish the Popularity Estimator to find external resources that relate to EEXCESS content.
3. Finish development of the context detection library. To this extend a user study is planned in Q4/2015 in which training data for model improvements will be collected.

13 Glossary

BITM

BitMedia, Austria

C4

Scientific and Cultural Content in Context

CT

Collection Trust, United Kingdom

DoSeR

Disambiguation of Semantic Resources

DoW

Description of Work

EEXCESS

Enhancing Europe's eXchange in Cultural Educational and Scientific Resources

INSA

Institut National des Sciences Appliquées (INSA) de Lyon, France

JR-DIG

JOANNEUM RESEARCH Forschungsgesellschaft mbH, Austria

KB

Knowledge Base

KBL-AMBL

Kanton Basel Land, Suisse

Know-Center

Kompetenzzentrum für Wissenschaftsbasierte Anwendungen und Systeme Forschungs- und Entwicklungs Center GmbH, Austria

MEN

Mendeley Ltd., United Kingdom

Uni Passau

University of Passau, Germany

WM

wissenmedia, Germany

ZBW

German National Library of Economics, Germany

Acknowledgement

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement nr 600601.

14 References

- [Wer74] Paul Werbos. "Beyond regression: New tools for prediction and analysis in the behavioral sciences". In: (1974).
- [BGV92] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. "A training algorithm for optimal margin classifiers". In: *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 144–152.
- [MM93] Robert AJ Matthews and Thomas VN Merriam. "Neural computation in stylometry I: An application to the works of Shakespeare and Fletcher". In: *Literary and Linguistic Computing* 8.4 (1993), pp. 203–209.
- [LeC+98] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [TB00] Erik F. Tjong Kim Sang and Sabine Buchholz. "Introduction to the CoNLL-2000 Shared Task: Chunking". In: *Proceedings of the 2Nd Workshop on Learning Language in Logic and the 4th Conference on Computational Natural Language Learning - Volume 7*. ConLL '00. Lisbon, Portugal: Association for Computational Linguistics, 2000, pp. 127–132. DOI: 10.3115/1117601.1117631. URL: <http://dx.doi.org/10.3115/1117601.1117631>.
- [MT04] Rada Mihalcea and Paul Tarau. "TextRank: Bringing order into texts". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2004.
- [Gri07] Jack Grieve. "Quantitative authorship attribution: An evaluation of techniques". In: *Literary and linguistic computing* 22.3 (2007), pp. 251–270.
- [Hin07] Geoffrey E. Hinton. "Learning multiple layers of representation". In: *Trends in Cognitive Sciences* 11.10 (2007), pp. 428–434. ISSN: 1364-6613. URL: <http://www.sciencedirect.com/science/article/pii/S1364661307002173>.
- [MW07] Gary Marchionini and Ryen White. "Find What You Need, Understand What You Find". In: *Int. J. Hum. Comput. Interaction* 23.3 (2007), pp. 205–237.
- [Obe+07] Hartmut Obendorf et al. "Web Page Revisitation Revisited: Implications of a Long-term Click-stream Study of Browser Usage". In: *CHI '07*. ACM, 2007, pp. 597–606. ISBN: 978-1-59593-593-9. DOI: 10.1145/1240624.1240719. URL: <http://doi.acm.org/10.1145/1240624.1240719>.
- [Rai+07] Rajat Raina et al. "Self-taught Learning: Transfer Learning from Unlabeled Data". In: *Proceedings of the 24th International Conference on Machine Learning*. ICML '07. Corvallis, Oregon, USA: ACM, 2007, pp. 759–766. ISBN: 978-1-59593-793-3. DOI: 10.1145/1273496.1273592. URL: <http://doi.acm.org/10.1145/1273496.1273592>.
- [Guo+09] Jiafeng Guo et al. "Named Entity Recognition in Query". In: *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '09. Boston, MA, USA: ACM, 2009, pp. 267–274. ISBN: 978-1-60558-483-6. DOI: 10.1145/1571941.1571989. URL: <http://doi.acm.org/10.1145/1571941.1571989>.
- [LS10] Nedim Lipka and Benno Stein. "Identifying featured articles in wikipedia: writing style matters". In: *Proceedings of the 19th international conference on World wide web*. ACM, 2010, pp. 1147–1148.
- [MD10] Pawet Mazur and Robert Dale. "WikiWars: A New Corpus for Research on Temporal Expressions". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. EMNLP '10. Cambridge, Massachusetts: Association for Computational Linguistics, 2010, pp. 913–922. URL: <http://dl.acm.org/citation.cfm?id=1870658.1870747>.

- [HPW11] David Hauger, Alexandros Paramythis, and Stephan Weibelzahl. "Using Browser Interaction Data to Determine Page Reading Behavior". In: *UMAP'11*. Springer-Verlag, 2011, pp. 147–158. ISBN: 978-3-642-22361-7. URL: <http://dl.acm.org/citation.cfm?id=2021855.2021869>.
- [Luy11] Kim Luyckx. *Scalability issues in authorship attribution*. ASP/VUBPRESS/UPA, 2011.
- [SMH11] Ilya Sutskever, James Martens, and Geoffrey E Hinton. "Generating text with recurrent neural networks". In: *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*. 2011, pp. 1017–1024.
- [Hin+12] Geoffrey Hinton et al. "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups". In: *Signal Processing Magazine, IEEE* 29.6 (2012), pp. 82–97.
- [LN12] Annie Louis and Ani Nenkova. "A corpus of general and specific sentences from news." In: *LREC*. 2012, pp. 1818–1821.
- [AFC13] Vikas Ganjigunte Ashok, Song Feng, and Yejin Choi. "Success with style: Using writing style to predict the success of novels". In: *Poetry* 580.9 (2013), p. 70.
- [Den+13] *Recent Advances in Deep Learning for Speech Research at Microsoft*. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), May 2013. URL: <http://research.microsoft.com/apps/pubs/default.aspx?id=188864>.
- [Le13] Quoc V Le. "Building high-level features using large scale unsupervised learning". In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 8595–8598.
- [YMK13] Seiji Yamada, Naoki Mori, and Kazuki Kobayashi. "Peripheral Agent: Implementation of Peripheral Cognition Technology". In: *CHI '13 Extended Abstracts on Human Factors in Computing Systems*. CHI EA '13. Paris, France: ACM, 2013, pp. 1701–1706. ISBN: 978-1-4503-1952-2. DOI: 10.1145/2468356.2468661. URL: <http://doi.acm.org/10.1145/2468356.2468661>.
- [Mok+14] Sonia Ben Mokhtar et al. *D6.3 – Second Security Proxy Prototype and Reputation Protocols*. Tech. rep. INSA, 2014.
- [Sei+14] Christin Seifert et al. *D5.2 – First Prototype on User Profile and Context Detection, Usage Analysis Methods and Services*. Tech. rep. University of Passau, 2014.
- [Böh+15] Alexander Böhm et al. "Identifying Tweets from the Economic Domain". In: *NLDB ISKO Workshop*. 2015.
- [Dop15] Gerhard Doppler. *D7.4 – Second Prototype Integration and Deployment*. Tech. rep. BITM, 2015.
- [Pas15] Uni Passau. *D1.2 – Second Conceptual Architecture and Requirements Definition*. Tech. rep. 2015.
- [Sch15] Jörg Schlötterer. "From Context to Query". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. SAC '15. Salamanca, Spain: ACM, 2015, pp. 1108–1109. ISBN: 978-1-4503-3196-8. DOI: 10.1145/2695664.2696061. URL: <http://doi.acm.org/10.1145/2695664.2696061>.
- [Sch+15] Jörg Schlötterer et al. "From Context-Aware to Context-Based: Mobile Just-In-Time Retrieval of Cultural Heritage Objects". In: *Proc. European Conference on IR Research (ECIR 2015)*. Ed. by Allan Hanbury et al. LNCS 9022. Vienna, Austria: Springer, Mar. 2015, pp. 805–808. DOI: 10.1007/978-3-319-16354-3_90.
- [SSG15] Christin Seifert, Jörg Schlötterer, and Michael Granitzer. "Towards a Feature-Rich Data Set for Personalized Access to Long-Tail Content". In: *Proceedings of the 30th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM, Apr. 2015.

- [Sei+15a] Christin Seifert et al. *D2.4 – Second Software Components for Presentation and Augmentation Interfaces*. Tech. rep. Know-Center, 2015.
- [Sei+15b] Christin Seifert et al. “Digital Library Content in the Social Web: Resource Usage and Content Injection”. In: *IEEE STCN Newsletter*. Vol. 3. 1. 2015. URL: <https://sites.google.com/a/ieee.net/stc-social-networking/e-letter/stcsn-e-letter-vol-3-no-1/>.
- [ZZL15] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level Convolutional Networks for Text Classification”. In: *arXiv preprint arXiv:1509.01626* (2015).
- [ZSG15a] Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. “From General to Specialized Domain: Analyzing Three Crucial Problems of Biomedical Entity Disambiguation”. In: *Proceedings of 26th International Conference on Database and Expert Systems Applications (DEXA)*. Springer, 2015.
- [ZSG15b] Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. “Linking Biomedical Data to the Cloud”. English. In: *Smart Health*. Ed. by Andreas Holzinger, Carsten Röcker, and Martina Ziefle. Vol. 8700. Lecture Notes in Computer Science. Springer International Publishing, 2015, pp. 209–235. ISBN: 978-3-319-16225-6. DOI: 10.1007/978-3-319-16226-3_9. URL: http://dx.doi.org/10.1007/978-3-319-16226-3_9.
- [ZSG15c] Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. “Search-based Entity Disambiguation with Document-Centric Knowledge Bases”. In: *Proceedings of the 14th International Conference on Knowledge Management and Knowledge Technologies (I-Know)*. Oct. 2015.

A Appendix: Source Code Documentation

This section contains the detailed source code documentation for the following modules

- Entity and Category Detection service to provide an in-detail summary of entities and categories given one or multiple paragraphs.
⇒ on page 37
- Word2Vec Rest Server to provide word2vec and doc2vec similarities.
⇒ on page 39
- Client-side library for context detection and wrappers for logging and entity disambiguation (Cultural and sScientific Content in Context – **C4**), currently used in the Chrome extension, the Moodle plugin, and the Wordpress plugin.
⇒ on page 41
- Self-contained widgets communicating through a dedicated message set via the Web Messaging API, currently used in the Chrome extension, the Moodle plugin, and the Wordpress plugin.
⇒ on page 54
- Blog Crawler component for crawling and storing blog posts.
⇒ on page 57
- Blog Analyser for linking Blogs to EEXCESS.
⇒ on page 59

Entity and Category Detection

Our Entity and Category Detection service detects entities and categories of paragraphs. We offer a Json rest interface which can be easily used to perform these tasks.

Download

For the Eexcess functions you have to download the following:

- Eexcess Stand-alone jar
- Doc2Vec Model
- DBpedia HDT Files
- Configuration file

The files can be downloaded under this Link

(https://www.dropbox.com/s/qbsbp9zfp7h8sx/eexcess_package.tar?dl=0)

- DBpedia Spotlight & English/German Models DBpedia Spotlight (<https://github.com/dbpedia-spotlight/dbpedia-spotlight/wiki/Run-from-a-JAR>)

Installation

Put Eexcess Stand-Alone jar and the configuration file into the same directory and adapt the configuration file accordingly. Executing the jar file starts an Apache Tomcat 7 server deploying the Eexcess web application.

Additionally, the eexcess application requires our Word2Vec Server Server (<https://github.com/quhfus/DoSeR/wiki/Word2Vec-Rest-Server>) to run properly.

Input Json Format

- Set of Paragraphes: A textual paragraph containing "headline", "id" and "content".
- Paragraph#Headline: The paragraph's headline
- Paragraph#Id: Unique identifier to address the paragraph
- Paragraph#Content: The textual main content, that should be annotated.

Input Example Format

```
{
  "paragraphs" : [{
    "headline" : "Childhood",
    "id" : "0",
    "content" : "Ada Lovelace was born Augusta Ada Byron on 10 December 1815, t
```

```

    he child of the poet George Gordon Byron, 6th Baron Byron, and Anne Isabella \
    "Anna bella\" Milbanke, Baroness Byron. George Byron expected his baby to be a \
    "glorious boy\" and was disappointed when his wife gave birth to a girl. Augusta was named a
    fter Byron's half-sister, Augusta Leigh, and was called \
    "Ada\" by Byron himself."
  }
]
}

```

Output Json Format

The sample code below shows a possible response to the given query above. We omit several entities and categories in the code below due to space constraints.

```

{
  "paragraphs": [
    {
      "id": "0",
      "topic": {
        "text": "Anne Isabella Byron, Baroness Byron",
        "entityUri": "http://dbpedia.org/resource/Anne_Isabella_Byron,_Baroness_Byron",
        "categories": [],
        "type": "",
        "offset": []
      },
      "time": [
        {
          "mention": "10 December 1815",
          "relevantEntities": [
            {
              "text": "Ada Lovelace",
              "entityUri": "http://dbpedia.org/resource/Ada_Lovelace",
              "categories": [
                {
                  "name": "Ada programming language",
                  "uri": "http://dbpedia.org/resource/Category:Ada_programming_language"
                }
              ],
              "type": "Person",
              "offset": 0
            }
          ],
          "statistic": [
            {
              "key": {
                "text": "Lord Byron",
                "entityUri": "http://dbpedia.org/resource/Lord_Byron",
                "categories": [
                  {
                    "name": "People educated at Aberdeen Grammar School",
                    "uri": "http://dbpedia.org/resource/Category:People_educated_at_Aberdeen_Grammar_School"
                  },
                  {
                    "name": "Burials in the East Midlands",
                    "uri": "http://dbpedia.org/resource/Category:Burials_in_the_East_Midlands"
                  }
                ],
                "type": "Person",
                "offset": [83],
                "value": 4
              }
            }
          ]
        }
      ]
    }
  ]
}

```

Request URL

Our Rest service is currently reachable under the following URL

<http://zaire.dimis.fim.uni-passau.de:8999/doser-disambiguationserverstable/webclassify/entityAndCategoryStatistic> (<http://zaire.dimis.fim.uni-passau.de:8999/doser-disambiguationserverstable/webclassify/entityAndCategoryStatistic>)

Word2Vec Rest Server

Our Python Word2Vec Rest Server delivers word2vec similarities between Wikipedia/DBpedia entities. Moreover, it is able to accept sentences/documents and computes a similarity score between the document and one or multiple entities.

Setup

To start the Word2Vec Rest Server, you need the following python packages install:

1. Python 2.7 or later
2. Gensim 0.12.1 or later
3. Gunicorn 19.3.0 or later
4. Flask 0.10 or later

Simply start `> startserver` to start the server. Default settings: Running on `http://0.0.0.0:5000` (`http://0.0.0.0:5000`) Settings can be adapted in `Word2VecRest.py` in the constructor of `GunicornApplication`.

Possible Settings:

- IP/Port Address: The ip address and port the server is binded to
- Workers: The number of parallel requests
- D2WModel: Path to Document to Vec Model
- W2VModel: Path to Word2Vec Model

If the server should be reachable from another host, you should install a proxy server (e.g. nginx) to forward the request since flask and gunicorn do not provide connection outside of localhost by default.

Usage

Word2Vec

To compute the similarities between the entities `Alan_Turing` and `Computer_science` as well as `Ada_Lovelace` and `Lord_Byron` we use the JSON code below. Generally, we can concatenate multiple entity pairs which should be compared.

```
{
  "data": ["Alan_Turing|Computer_science", "Ada_Lovelace|Lord_Byron"]
}
```


We note that the entity names are the same as provided by Wikipedia/DBpedia.

Doc2Vec

With Doc2Vec we are able to infer a vector out of a text snippet. This vector is compared with the given entities vectors. In other words we compute the similarity between the given text and the entity describing texts of the given entities.

```
{
  "document": [{
    "surfaceForm": "Ada Lovelace",
    "qryNr": "0",
    "context": "Lovelace was born 10 December 1815 as the only legitimate child of the poet George, Lord Byron and his wife Anne Isabella Noel. All Byron's other children were born out of wedlock to other women. Byron separated from his wife a month after Ada was born and left England forever four months later, eventually dying of disease in the Greek War of Independence when Ada was eight years old. Ada's mother remained bitter towards Lord Byron and promoted Ada's interest in mathematics and logic in an effort to prevent her from developing what she saw as the insanity seen in her father, but Ada remained interested in him despite this (and was, upon her eventual death, buried next to him at her request).",
    "candidates": ["Ada_Lovelace", "Lord_Byron"]
  }]
}
```

The resulting similarity value is in the range between 0 and 2, with 2 meaning that the documents are identical. Again, the entity names are the same as provided by Wikipedia/DBpedia.

Installation

The simplest way is to use bower (<http://bower.io/>). C4 is available in the package repository, so `bower install c4` will install everything you need.

After c4 is installed, you might need to configure the paths for requirejs. This can be comfortably automated with `grunt-bower-requirejs` (<https://github.com/yeoman/grunt-bower-requirejs>). If you choose to configure the paths manually, your configuration might look similar to this (first part of the script, in which you want to use c4 modules):

```
requirejs.config({
  baseUrl: 'bower_components/',
  paths: {
    jquery: 'jquery/dist/jquery',
    "jquery-ui": 'jquery-ui/jquery-ui',
    graph: 'graph/lib/graph',
    "tag-it": 'tag-it/js/tag-it'
  }
});
```

Once the paths are configured, you need to include your script via requirejs as usual, for example like so:

```
<script data-main="myScript" src="bower_components/requirejs/require.js"></script>
```

where `myScript` is the script you want to execute and in which you use c4 modules.

Module Overview

- **APIconnector** A module that simplifies requests to the EEXCESS privacy proxy. It allows to send (privacy preserved) queries, obtain details for a set of document badges (<https://github.com/EEXCESS/eexcess/wiki/%5B21.09.2015%5D-Request-and-Response-format#response-format>) and provides a cache of the last queries/result sets.
- **paragraphDetection** A module that allows to extract textual paragraphs from HTML documents (opposed to navigational menus, advertisements, etc.), construct queries in the EEXCESS query profile (<https://github.com/EEXCESS/eexcess/wiki/%5B21.09.2015%5D-Request-and-Response-format#query-format>) format and determine the currently focused paragraph.
- **CitationBuilder** A module to assemble ready-to-use citations from metadata provided as JSON. See the CitationBuilder README.md ([CitationBuilder/README.md](#)) for details.
- **searchBar** A module to add a bar to the bottom of the page, which allows query

interaction and displaying results.

- `iframes` A utility module for communication with iframes, which enables broadcasting messages.
- `namedEntityRecognition` A utility module for communication with the DoSer (<https://github.com/quhfus/DoSeR>) named entity recognition service.
- `logging` A module that simplifies the handling of different types of logging events.

APIconnector

The APIconnector module provides means to communicate with the (EEXCESS) Federated Recommender via the Privacy Proxy.

A working example using the APIconnector can be found in `examples/searchBar_Paragraphs` (`examples/searchBar_Paragraphs`)

- `init(settings)`: allows to initialize the APIconnector with custom parameters. You must call this method and specify the `origin` attribute (see below), before you can send queries. The minimum configuration is shown in the example below. The following parameters can be customized:
 - `base_url` The basic url of the server to call
 - `timeout` The timeout in ms, after which a request to the server is canceled. Default is 10000.
 - `logTimeout` The timeout in ms, after which a logging request to the server is canceled. Default is 5000.
 - `loggingLevel` Flag whether queries/results should be logged on the privacy proxy. Defaults to 0 (logging enabled). If you want to disable the logging on the server you need to set the flag to 1.
 - `cacheSize` The size of the query/result cache. Determines how many queries and corresponding result sets should be cached. Default is 10.
 - `suffix_recommend` The endpoint for the recommender service. Default: "recommend".
 - `suffix_details` The endpoint to get details for result items. Default: "getDetails".
 - `suffix_favicon` The endpoint from which to retrieve the provider favicons. Default: "getPartnerFavIcon?partnerId=".
 - `suffix_log` The endpoint for logging requests. Default: "log/".
 - `origin` The identifier for the requesting client/user. This object must contain the attributes `clientType`, `clientVersion` and `userID`, see the example below.

```
require(['c4/APIconnector'], function(api) {
  api.init({
    origin: {
      clientType: "some client", // the name of the client application
      clientVersion: "42.23", // the version nr of the client applicatio
n
      userID: "E993A29B-A063-426D-896E-131F85193EB7" // UUID of the curr
```

```

ent user
    }
  });
});

```

- `query(profile, callback)`: allows to query the Federated Recommender (through the Privacy Proxy). The expected parameters are a EEXCESS profile (<https://github.com/EEXCESS/eexcess/wiki/%5B21.09.2015%5D-Request-and-Response-format>) and a callback function.

```

require(['c4/APIconnector'], function(api) {
  var profile = {
    contextKeywords: [{
      text: "someKeyword"
    }]
  };
  api.query(profile, function(response) {
    if(response.status === 'success') {
      // do something with the result contained in response.data
    } else {
      // an error occured, details may be in response.data
    }
  });
});

```

- `queryPeas`: allows to query the Federated Recommender (through the Privacy Proxy) in a privacy-preserving way. It returns the exact same result as `query`. It uses the PEAS indistinguishability protocol (<https://github.com/EEXCESS/peas#indistinguishability-protocol>). This example shows how to use it:

```

require(["APIconnector"], function(apiConnector){
  var nbFakeQueries = 2; // The greater the better from a privacy point of view, but the worse from a performance point of view (2 or 3 are acceptable values).
  var query = JSON.parse('{"origin": {"userID": "E993A29B-A063-426D-896E-131F85193EB7", "clientType": "EEXCESS - Google Chrome Extension", "clientVersion": "2beta", "module": "testing"}, "numResults": 3, "contextKeywords": [{"text": "graz", "weight": 0.1}, {"text": "vienna", "weight": 0.3}]}');
  apiConnector.queryPeas(query, nbFakeQueries, function(results){
    var resultsObj = results.data;
  });
});

```

- `getDetails(detailsRequestObj, callback)`: allows to retrieve details for result items from the Federated Recommender (through the Privacy Proxy). The expected parameter is a `detailsRequestObj` (<https://github.com/EEXCESS/eexcess/wiki/%5B21.09.2015%5D-Request-and-Response-format#pp-details-query-format>) object, that has an `origin`, `queryID` and a list of document badges (<https://github.com/EEXCESS/eexcess/wiki/%5B21.09.2015%5D-Request-and-Response-format#response-format>) of the items for which to retrieve details. A callback function can be used to return the results to.

```

require(['c4/APIconnector'], function(api) {
    var detailsRequestObj = {
        origin : {"origin": {"userID": "E993A29B-A063-426D-896E-131F85193EB7", "clientType": "EEXCESS - Google Chrome Extension", "clientVersion": "2beta", "module": "testing"},
        documentBadge: [
            {
                id: "/09003/4A65C4999F4077781A1F9CF2510EE512CD6571B9",
                uri: "http://europeana.eu/resolve/record/09003/4A65C4999F4077781A1F9CF2510EE512CD6571B9",
                provider: "Europeana"
            }
        ],
        queryID: "70342716"
    };
    api.getDetails(detailsRequestObj, function(response) {
        if(response.status === 'success') {
            // do something with the result contained in response.data
        } else {
            // an error occured, details may be in response.data
        }
    });
});

```

- `getCache()`: allows to retrieve the cached queries/result sets.

```

require(['c4/APIconnector'], function(api) {
    api.getCache().forEach(function(){
        console.log(this.profile); // the query
        console.log(this.result); // the result set
    });
});

```

- `getCurrent()`: allows to retrieve the last successfully executed query and corresponding result set. Returns null if no successful query has been executed up to that point.

```

require(['c4/APIconnector'], function(api) {
    var current = api.getCurrent();
    console.log(current.profile); // the query
    console.log(current.result); // the result set
});

```

- `logInteractionType`: Enum for logging interaction types. See `sendLog` for usage.
- `sendLog(interactionType, logEntry)`: allows to send logging requests to the server. The parameter `interactionType` specifies the type of the interaction to log and the parameter `logEntry` the entry to be logged.

```

require(['c4/APIconnector'], function(api) {
    // the log entry normally will be created within a widget, here we define o

```

```

ne explicitly.
// The entry we create logs the citation of a result item as an image.
var logEntry = {
  origin:{
    module:"example widget"
  },
  content:{
    documentBadge:{<documentBadge of the item>}
  },
  queryID:<identifier of the query that provided this result item>
}
api.sendLog(api.logInteractionType.itemCitedAsImage,logEntry);
});

```

paragraphDetection

A module to extract textual paragraphs from arbitrary webpage markup, find the paragraph currently in focus of the user and create a search query from a paragraph.

A working example using the paragraphDetection can be found in [examples/searchBar_Paragraphs](#) ([examples/searchBar_Paragraphs](#))

- `init(settings)`: allows to initialize the paragraph detection with custom parameters. You only need to provide the parameters you want to change. Parameters that can be changed are a `prefix` that is used for the identifiers of newly created HTML elements and the `classname` that will added to those elements (atm a wrapper div with the mentioned parameters is created around the detected paragraph). The example uses the default values, if you are fine with these, you do not need to call the `init` method.

```

require(['c4/paragraphDetection'], function(paragraphDetection) {
  paragraphDetection.init({
    prefix:"eexcess", // default value
    classname:"eexcess_detected_par" // default value
  });
});

```

- `getParagraphs([root])`: allows to detect text paragraphs in arbitrary HTML markup. The detection heuristic tries to extract 'real' paragraphs, opposed to navigation menus, advertisements, etc. The `root` parameter (optional) specifies the root HTML-element from where to start the extraction. If it is not given, the detection will use `document` as root.

Returns an array of the detected paragraphs with the entries in the following format:

```

{
  id: "<prefix>_par_0", // identifier, the prefix can be customized via the i
  nit method
  elements:[], // the HTML-elements spanning the paragrah
  multi:false, // indicator, whether the paragraph consists of e.g. a singe
  <p> element or several <p> siblings

```

```

    content: "Lorem ipsum dolor", // the textual content of the paragraph
    headline: "Sit Amet" // textual content of the corresponding headline of the
    paragraph
  }
}

```

Usage:

```

require(['c4/paragraphDetection'], function(paragraphDetection) {
  var paragraphs = paragraphDetection.getParagraphs();
  paragraphs.forEach(function(entry){
    console.log(entry); // do something with each paragraph
  });
});

```

- `paragraphToQuery(text, callback, [id], [headline])`: creates a query from the given text in the EEXCESS profile (<https://github.com/EEXCESS/eexcess/wiki/%5B21.09.2015%5D-Request-and-Response-format#query-format>) format. Only the attribute `contextKeywords` will be set. The parameters to be set are:

- `text` - The text of the paragraph for which to create a query
- `callback(response)` - The callback function to execute after the query generation. The generated query profile is contained in `response.query` or if an error occurs, error details are provided in `response.error`
- `[id]` - optional identifier of the paragraph
- `[headline]` - optional headline corresponding to the paragraph

```

require(['c4/paragraphDetection'], function(paragraphDetection) {
  var text = 'Lorem ipsum dolor sit amet...';
  paragraphDetection.paragraphToQuery(text, function(response){
    if(typeof response.query !== 'undefined') {
      // query has sucessfully been constructed
      console.log(response.query);
    } else {
      // something went wrong
      console.log(response.error);
    }
  });
});

```

- `findFocusedParagraphSimple([paragraphs])`: tries to determine the paragraph, the user is currently looking at. In this simple version, the topmost left paragraph is accounted as being read, except for the user explicitly clicking on a paragraph. When a change of the focused paragraph occurs, a `paragraphFocused` event is dispatched with the focused paragraph attached. The set of paragraphs to observe can be specified via the optional `paragraphs` parameter. If this parameter is not set, the method will observe paragraphs already detected by the module (if any - e.g. from a previous `getParagraphs` call). The `paragraphFocused` event may be dispatched several times for the same paragraph.

```

require(['jquery', 'c4/paragraphDetection'], function($, paragraphDetection) {

```



```

// detect paragraphs in the document
paragraphDetection.getParagraphs();
// listen for paragraphFoucsed events
$(document).on('paragraphFocused', function(e){
    console.log(evt.originalEvent.detail); // the focused paragraph
});
// set up tracking of focused paragraph
paragraphDetection.findFocusedParagraphSimple();
});

```

- `findFocusedParagraph([paragraphs])`: tries to determine the paragraph, the user is currently looking at.

This method is in principle identical to `findFocusedParagraphSimple`, but accounts for more implicit user interaction. The probability of a focused paragraph is calculated by a weighted combination of its size, position and distance to the mouse position. When mouse movements occur, the distance to the mouse position has a higher weight, while scrolling events render the paragraph position more important.

```

require(['jquery', 'c4/paragraphDetection'], function($, paragraphDetection) {
    // detect paragraphs in the document
    paragraphDetection.getParagraphs();
    // listen for paragraphFoucsed events
    $(document).on('paragraphFocused', function(e){
        console.log(evt.originalEvent.detail); // the focused paragraph
    });
    // set up tracking of focused paragraph
    paragraphDetection.findFocusedParagraphSimple();
});

```

CitationBuilder

Please see the `README.md` (`CitationBuilder/README.md`) of the `CitationBuilder` module for details.

searchBar

A module that adds a search bar to the bottom of the page, which enables to show and modify the query and display the results.

A working example using the `searchBar` can be found in `examples/searchBar_Paragraphs` (`examples/searchBar_Paragraphs`)

- `init(tabs[, config])`: initializes the search bar with a set of visualization widgets (parameter `tabs`) and custom configuration options (optional parameter `config`). The `tabs` parameter specifies the visualization widgets (<https://github.com/EEXCESS/visualization-widgets>) to use for displaying the result in the following format:

```

[-] [{
  name: "widget name" // name of the widget, will be displayed in the tab navigation for selection of the widget
  url: "<path>" // path to the main page of the widget
  icon: "<path>" // path to an icon image for the widget (optional)
},{
  name: "widget2",
  url: "<path2>"
},{
  // ...
}
]

```

The config object allows to customize the following parameters (you only need to specify the ones you would like to change):

- queryFn – a custom function to query a server for results. The function must look like

```

[-] function(profile, function(response){
  console.log(response.status); // should inform about the status, either 'success' or 'error'
  console.log(response.data); // should contain the results on successes and error details on error
});

```

where the `profile` parameter represents an EEXCESS query profile

(<https://github.com/EEXCESS/eexcess/wiki/%5B21.09.2015%5D-Request-and-Response-format#query-format>). By default, the query method of the APIconnector module is used.

- imgPATH – path where images are stored. Defaults to 'img/'
- queryModificationDelay – the delay before a query is executed (in ms) after the user interacted with it (added/removed keywords, changed main topic, etc). Defaults to 500.
- queryDelay – the delay (in ms) before a query is executed due to changes from the parent container. This delay is used after the query has been changed through the setQuery method of this module. Defaults to 2000.
In addition, the delay can also be provided as parameter to the setQuery function, in order to enable different delays for specific interactions.
- storage – an object providing storage capabilities. By default, the search bar will use the browser's local storage to store values. The storage function must exhibit two functions:
 - set(item, callback) The item passed to this function is an object containing key value pairs to store. It looks like this:

```

[-] {
  key1: "value1", // value can be a simple type like String
}

```

```

    key2: {
      // value can also be an JSON-serialiazable objects
    }
  }
}

```

The `callback` parameter is a callback function without parameters to be executed after storing the item.

- `get(key, callback)` The `key` parameter is either a single `String` (to get a single value) or an `Array` of `Strings` (to get several values). The `callback` function should be called with an object, containing the provided `key(s)` and their corresponding values like so:

```

var response = {
  key1: value1,
  key2: value2
}
callback(response);

```

- `setQuery(contextKeywords [,delay])`: sets the query in the search bar. The `contextKeywords` must be in the format as the `contextKeywords` in the EEXCESS query profile format (<https://github.com/EEXCESS/eexcess/wiki/%5B21.09.2015%5D-Request-and-Response-format#query-format>). The query will automatically be executed after the delay given by the settings (default: 2000ms, can be customized via `searchBar.init(<tabs>,{queryDelay:<custom value>})`). Alternatively, this setting can be overwritten by providing the optional `delay` parameter, which specifies the delay in ms.

```

require(['jquery', 'c4/searchBar'], function($,searchBar) {
  // searchBar needs to be initialized first, omitted here
  var contextKeywords = [{
    text: "Lorem"
  },{
    text: "ipsum"
  }];
  searchBar.setQuery(contextKeywords, 0); // query is set and will be immedia
tely executed (delay: 0ms)
});

```

iframes

A utility module for communicating between iframes

- `sendMsgAll(message)`: send a message to all iframes embedded in the current window. The expected parameter is the message to send. The example below shows how to inform all included widgets (<https://github.com/EEXCESS/visualization-widgets>) that a new query has been issued.

```

require(['c4/iframes'], function(iframes) {
  var profile = {
    contextKeywords:[{
      text: 'someKeyword'
    }];
  };
  iframes.sendMsgAll({
    event: 'eexces.newQueryTriggered',
    data: profile
  });
});

```

namedEntityRecognition

A utility module to query the EEXCESS recognition and disambiguation service

- `entitiesAndCategories(paragraphs, callback)`: allows to extract Wikipedia entities and associated categories from a given piece of text. In addition, the main topic of the text and time mentions are extracted. The expected parameters are a set of paragraphs and a callback function.

```

require(['c4/namedEntityRecognition'], function(ner) {
  var paragraph = {
    id: 42,
    headline: "I am a headline",
    content: "Lorem ipsum dolor..."
  };
  ner.entitiesAndCategories({paragraphs:[paragraph]}, function(response){
    if(response.status === 'success') {
      // the results are contained in response.data.paragraphs
      response.data.paragraphs.forEach(function(){
        console.log(this.time); // contains time mentions and associated entities/categories
        console.log(this.topic); // contains the main topic entity
        console.log(this.statistic); // contains the extracted entities/categories
        this.statistic.forEach(function(){
          console.log(this.key.text); // label of the entity
          console.log(this.key.categories); // associated categories
          console.log(this.key.type); // type of the entity (person, location, organization, misc)
          console.log(this.value); // number of occurrences of the entity in the paragraph
        });
      });
    } else {
      // an error occurred, details may be in response.data
    }
  });
});

```

```
});
```

logging-API

A module that provides an API for generating logging events in the format specified by Logging Format (<https://github.com/EEXCESS/eexcess/wiki/EEXCESS---Logging>). Logging-Events are passed over to the APIconnector which sends it to the logging endpoints of the Privacy Proxy. A working example on how the logging is to be used, can be found in [examples/loggingExample](#) ([examples/loggingExample](#)).

The logging-API provides methods to create logging events in the correct format and it broadcasts these events as messages via the browser's Messaging-API. Thus, the client application needs to listen to the following messages and forward them to the Privacy Proxy:

```
require(['c4/APIconnector'], function(api) {
  api.init({origin: {
    clientType: "EEXCESS Chrome extension",
    clientVersion: "2.0.1"
    userID: "93939A-8494BE-99ADF2"
  }});

  window.onmessage = function(msg) {
    if (msg.data.event) {
      switch (msg.data.event) {
        case 'eexcess.log.moduleOpened':
          api.sendLog(api.logInteractionType.moduleOpened, msg.data.data);
          break;
        case 'eexcess.log.moduleClosed':
          api.sendLog(api.logInteractionType.moduleClosed, msg.data.data);
          break;
        case 'eexcess.log.moduleStatisticsCollected':
          api.sendLog(api.logInteractionType.moduleStatisticsCollected, msg.data.data);
          break;
        case 'eexcess.log.itemOpened':
          api.sendLog(api.logInteractionType.itemOpened, msg.data.data);
          break;
        case 'eexcess.log.itemClosed':
          api.sendLog(api.logInteractionType.itemClosed, msg.data.data);
          break;
        case 'eexcess.log.itemCitedAsImage':
          api.sendLog(api.logInteractionType.itemCitedAsImage, msg.data.data);
          break;
        case 'eexcess.log.itemCitedAsText':
          api.sendLog(api.logInteractionType.itemCitedAsText, msg.data.data);
          break;
        case 'eexcess.log.itemCitedAsHyperlink':
          api.sendLog(api.logInteractionType.itemCitedAsHyperlink, msg.data.data);
          break;
      }
    }
  };
});
```

```

        break;
    case 'eexcess.log.itemRated':
        api.sendLog(api.logInteractionType.itemRated, msg.data.data);
        break;
    default:
        break;
    }
}
}
});

```

The methods of the logging-API are as follows:

- `init(config)`: allows to initialize the logging-API with custom parameters. You must call this method and specify the `origin` attribute (see below), before you can call methods of the logging-API. The following parameters can be customized:
 - `origin` The identifier for the requesting component. This object must contain a `module` attribute, that specifies the name of the issuing component, see the example below.

```

require(['c4/logging'], function(logging) {
    logging.init({
        origin: {
            module: "Dashboard Visualization"
        }
    });
});

```

The logging-API provides the following methods to create corresponding logging events:

- `moduleOpened(moduleName)`: Create the logging event `moduleOpened`. The expected parameter is the name of the component that has been opened.
- `moduleClosed(moduleName)`: Create the logging event `moduleClosed`. The expected parameter is the name of the component that has been closed.
- `moduleStatisticsCollected(statistics)`: Create the logging event `moduleStatisticsCollected`. The expected parameter can be of any type. `statistics` is logged as-is.
- `itemOpened(documentBadge, queryID)`: Create the logging event `itemOpened`. The expected parameters are a `documentBadge`, which refers to the resource that was opened in detailed view, and a `queryID`, which refers to the query that returned the resource.
- `itemClosed(documentBadge, queryID, duration)`: Create the logging event `itemClosed`. The expected parameters are a `documentBadge`, which refers to the resource that was closed in detailed view, a `queryID`, which refers to the query that returned the resource, and a `duration`. The latter specifies the time in milliseconds, during which the item was opened in detailed view.
- `itemCitedAsImage(documentBadge, queryID)`: Create the logging event `itemCitedAsImage`. The expected parameters are a `documentBadge`, which refers to the resource that was cited as an image, and a `queryID`, which refers to the query that returned the resource.

- `itemCitedAsText(documentBadge, queryID)`: Create the logging event `itemCitedAsText`. The expected parameters are a `documentBadge`, which refers to the resource that was cited as text, and a `queryID`, which refers to the query that returned the resource.
- `itemCitedAsHyperlink(documentBadge, queryID)`: Create the logging event `itemCitedAsHyperlink`. The expected parameters are a `documentBadge`, which refers to the resource that was cited as a hyperlink, and a `queryID`, which refers to the query that returned the resource.
- `itemRated(documentBadge, queryID, minRating, maxRating, rating)`: Create the logging event `itemRated`. The expected parameters are a `documentBadge`, which refers to the resource that was rated and a `queryID`, which referst to the query that returned the rated resource. `minRating` and `maxRating` allow to specify the range a rating can have. `rating` is the actual value that has been assigned to the resource.

The following examples demonstrate the usage of these methods:

```
require(['c4/logging'], function(logging) {
    logging.init({
        origin: {
            module: "Dashboard Visualization"
        }
    });

    var queryID = "483904939"
    var documentBadge = {
        id: "995eb36f-151d-356c-b00c-4ef419bc2124",
        uri: "http://www.mendeley.com/research/hellenism-homoeroticism-shelley-circle",
        provider: "Mendeley"
    };

    logging.moduleOpened("anotherVisualizationName");
    logging.moduleClosed("anotherVisualizationName", 5000);
    logging.moduleStatisticsCollected({usageStatistics: 42});

    logging.itemOpened(documentBadge, queryID);
    logging.itemClosed(documentBadge, queryID, 1500);

    logging.itemCitedAsImage(documentBadge, queryID);
    logging.itemCitedAsText(documentBadge, queryID);
    logging.itemCitedAsHyperlink(documentBadge, queryID);

    logging.itemRated(documentBadge, queryID, 0, 4, 3);
});
```


Widgets

EEXCESS widgets are components like visualizations (BarChart, FacetScape, ...), which are typically included via an iframe. Therefore, they should be self-contained, i.e. include all necessary media, libraries, css-files, etc.

Communication with the EEXCESS-environment is enabled via the `window.postMessage`-API, with the available options described in the following.

Usage

For usage examples see the `examples` folder and the according `readme` file.

Interface - using `window.postMessage`

The data attribute in the transmitted messages adheres to the following pattern:

```
event: eexcess.<event>,  
data: {<event details>}
```

Incoming messages

Available events:

- `queryTriggered`
- `new Results`
- `rating`
- `error`

queryTriggered

This event specifies, that a new query was triggered. The event details contain the profile, that is associated with this query

new Results

This event indicates the arrival of new results. The event details consist of two attributes: *profile* and *results*. *Profile* contains the user profile associated with the results and *results* contains the results retrieved.

rating

Indicates that an item was rated in another component. The widget can then update the item's rating accordingly. The event details contain the *uri* of the item and *score* of the rating.

error

Used to indicate an error. The event details contain an error message as string.

Outgoing Messages

Available events:

- queryTriggered
- eexcess.log.moduleOpened
- eexcess.log.moduleClosed
- eexcess.log.statisticsCollected
- eexcess.log.itemOpened
- eexcess.log.itemClosed
- eexcess.log.itemCitedAsImage
- eexcess.log.itemCitedAsText
- eexcess.log.itemCitedAsHyperlink
- eexcess.log.itemRated
- currentResults

queryTriggered

Indicates a new query. The event details contain the profile associated with that query.

eexcess.log.moduleOpened

Indicates that a module was opened. The event details contain the origin and the name of the module.

eexcess.log.moduleClosed

Indicates that a module was closed. The event details contain the origin, the name of the module and optionally the duration

eexcess.log.statisticsCollected

Indicates that a module wants to log data. The event details contain the origin and the data

eexcess.log.itemOpened

Indicates that an item is opened. The event details contain the origin, the queryID of the original query and the documentBadge.

eexcess.log.itemClosed

Indicates that an item is closed. The event details contain the origin, the queryID of the original query, the documentBadge and optionally the duration.

eexcess.log.itemCitedAsImage

Indicates that an item was cited in document as an image. The event details contain the origin, the queryID of the original query and the documentBadge.

eexcess.log.itemCitedAsText

Indicates that an item was cited in document as an text. The event details contain the origin, the queryID of the original query and the documentBadge.

eexcess.log.itemCitedAsHyperlink

Indicates that an item was cited in document as an hyperlink. The event details contain the origin, the queryID of the original query and the documentBadge.

eexcess.log.itemRated

Indicates that an item was rated. The event details contain the origin, the queryID, the documentBadge and the rating.

currentResults

This event may be used by widgets upon initialization to obtain the current resultset (and associated profile). It triggers the parent window to send a message with a **newResults** event.

BlogCrawler

This prototype is a focus web crawler that allows the visiting pre-defined websites, extract their contents and save them into an elasticsearch (<https://www.elastic.co/products/elasticsearch>) datastore. The blog crawler is based on scrapy (<http://scrapy.org>), a python framework to facilitate the development of webscraping applications.

Requirements

- Linux or Mac OS X
- Python 2.7 or newer
- Scrapy 0.22 or newer
- lxml
- pyOpenSSL
- Elasticsearch 1.2.1 or newer
- Java Runtime Environment 7 or newer

Installation

The easiest way to install the required software is to use the packet manager of the OS. The commands below are tested with Ubuntu 14.04, but they should work on all the distributions that use the apt-get packet manager. For yum-based systems like Fedora and SuSE some modifications might be required.

The following commands will make sure that all requirements are met:

```
❏ sudo apt-get install -y build-essential git python-pip python python-dev libxml2-dev libxslt-dev lib32z1-dev openjdk-7-jdk
sudo pip install pyopenssl lxml scrapy elasticsearch dateutils Twisted service_identity
```

Elasticsearch can not be installed via apt-get. Hence, this has to be done manually. First we download the latest Elasticsearch version:

```
❏ wget https://download.elasticsearch.org/elasticsearch/elasticsearch/elasticsearch-x.y.z.deb
```

where x.y.z. is the current version number and thus needs to be replaced. Then the installation process can be triggered via:

```
[-] sudo dpkg -i elasticsearch-x.y.z.deb
```

Again, x.y.z refers to the latest's version number and has to be replaced. The installation is complete and elasticsearch can be started using the following command:

```
[-] sudo service elasticsearch start
```

It is to note, that the service will not start automatically when the computer boots up. If this is required, the following command has to be used:

```
[-] sudo update-rc.d elasticsearch defaults 95 10
```

The blog crawler can be cloned from Github:

```
[-] git clone purl.org/eexcess/components/research/blogcrawler}}
```

The crawler can be invoked by changing into the just cloned repository-directory and starting the script `crawlall.sh`. That the process can take several hours. To interrupt the process must be pressed.

DataAnalyzer

This prototype is based on the the BlogCrawler that can be found here (<https://github.com/n-witt/BlogCrawler>) . It searches hyperlinks to PDF-files and downloads them. In the next step it tries to find a matching document in EconBiz (<http://www.econbiz.de/>) database. It implements the following strategy:

- The program checks whether the meta data fields author and text of the file contain any information. If so, it sends a query assembled from these strings to EconBiz. After fetching the result list, the length of the list is checked. In case the result list is longer than zero, all results are examined and assessed (details will be described in the next paragraph). When there is no result above a predefined quality threshold, the second stage is executed, otherwise the result is stored and the processing continues with the next document.
- After the text of the first page of the file is retrieved, the text is divided into smaller chunks (using punctuation and newline-symbols for that). The processing of these parts of sentences is similar to the processing of the metadata in the previous section. They are passed to the EconBiz API and the results are examined. If there is no result above a predefined quality threshold, no match was found. Otherwise the list of potential matches is stored.

To assess the quality of the results, a fuzzy string comparison library called `fuzzywuzzy` (<https://pypi.python.org/pypi/fuzzywuzzy/0.2>) is used. It contains a method that is invoked with an arbitrary string (selector) and a list of strings (choices). The method returns the choices sorted by closest match of the selector. Every item of the list also comes with a value from 0 to 100 which is the measure of quality. The following example illustrates that:

```
> choices = ["apple pie", "apples", "spaghetti"]
> process.extract("apple", choices)
[('apples', 91), ('apple pie', 90), ('spaghetti', 36)]
> process.extract("apples", choices)
[('apples', 100), ('apple pie', 74), ('spaghetti', 29)]
```

The quality value is used to decide if a document matches the search query.

The limitation to the first page is due to the fact that extraction of the text is a computationally intensive task that can be mitigated by the limitation. Furthermore we assume that the first page contains the authors name and the title of the document, which is true for many scientific papers. And it is this information that are particularly valuable for descent search results.

Requirements

- Linux or Mac OS X

- Python 2.7 or newer
- fuzzywuzzy
- PyPDF2
- pdfminer

Installation

The recommended installation preliminaries and procedure for the Data Analyzer are the same as for the Blog Crawler. The following was tested with Ubuntu 14.04. To install the dependencies, these commands should be used:

```
❏ sudo apt-get install -y python python-pip git
| sudo pip install fuzzywuzzy PyPDF2 pdfminer
```

Afterwards the repository can be cloned:

```
❏ git clone purl.org/eexcess/components/research/bloganalyzer
```

Finally, the analyzer can started with these commands:

```
❏ cd DataAnalyzer/eu/zbw/
| python pdfMetadataExtractor.py
```

The script will analyze the File in the `samples` directory. The results will be printed when all the computation is done. Every file will be mentioned in the output. The output could look like this:

```
❏ 10. match: True
| quality: 90
| filename: bakken\_fullactivity\_Jan3-2013.pdf
| id: 10004941699
| title: Staff report Research Department of the Federal Reserve Bank of Minnea
| polis
| participant: Minneapolis, Minn. : Federal Reserve Bank of Minneapolis
```

`match: True` denotes that EconBiz found an entry. `quality` refers to the likelihood that the entry found by EconBiz and and file that has been processes correspond. `id`, `title` and `participant` refer to the entry found by EconBiz.