# EEXCESS

## Enhancing Europe's eXchange in Cultural Educational and Scientific reSources

## Deliverable D5.4

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

| | |
|---|---|
| Identifier: | EEXCESS-D5.4-Final-Prototype-on-User-Profile-and-Context-Detection-Usage-Analysis-Methods-and-Services-final.pdf |
| Deliverable number: | D5.4 |
| Author(s) and company: | Jörg Schlötterer (Uni Passau), Christin Seifert (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 | 2016-05-30 |

**History**

| Version | Date | Reason of change |
|---|---|---|
| 1 | 2016-04-30 | First draft and structure created |
| 2 | 2016-05-11 | Structure revised |
| 3 | 2016-05-17 | Included entity disambiguation parts |
| 4 | 2016-05-18 | Integrated usage analysis |
| 5 | 2016-05-19 | Updated context detection |
| 6 | 2016-05-20 | Integrated resource mining |
| 7 | 2016-05-23 | Finalized overview, introduction and summary |
| 7 | 2016-05-23 | Version for internal review (INSA) |
| 8 | 2016-05-30 | 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.

# Contents

# 1 Executive Summary

This deliverable describes the development and research related to user and usage mining. 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).

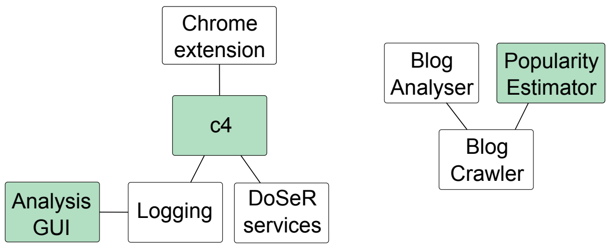


Figure 1: Overview of the main components developed in this work package. The green color indicates the development focus since the last deliverable.

In terms of components, the libraries, services and prototypes shown in figure 1 have been developed. 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 and focus paragraph identification. For those paragraphs (but not limited to them), C4 provides capabilities to construct personalized queries. C4 exhibits wrappers for logging and disambiguation services (DoSeR). C4 is currently used by other clients, namely the Wordpress plugin, the Moodle Plugin and the Wikipedia Reference Butler. Due to the restriction of the Google Add-on store, the Google Docs add-on 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 an entity knowledge base, thus it is a server-side component with the service calls wrapped in C4. 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 query pre-selection from a set of possible queries, based on previous queries. Mechanisms for learning on which page the extension should be switched on/off are also included. 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**. 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. Based on the logging data, the **Analysis GUI** web interface provides means for privacy-preserving usage analysis.

**The Blog Crawler** enables the data collection for external resource mining. The **Blog Analyzer** for linking Blogs to EEXCESS enables the analysis of the collected data. The **Popularity Estimator** estimates the popularity of scientific papers or blogs solely on the basis of their textual content. The source code of the crawler and the analyzer is available from `http://purl.org/eexcess/components/research/blogcrawler` and `http://purl.org/eexcess/components/research/bloganalyzer` respectively. The source code of the popularity estimator is available from `https://github.com/n-witt/econstorModelling/blob/master/classifier.ipynb`.

# 2 Introduction

## 2.1 Purpose of this Document

This deliverable describes the final 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 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 D6.4 [Mok+16]. 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 D7.6 [Dop16b].

## 2.3 Status of this Document

This is the final version of D5.4.

## 2.4 Related Documents

**D7.6** Final Prototype Integration and Deployment [Dop16b]
In detail the following section is of interest for the reader:

- Refer to section 3.2 for a guided tour of the prototype for context detection, the Chrome extension.

**D6.4** Final Security Proxy Prototype and Reputation Protocols [Mok+16]
In detail the following section is of interest for the reader:

- Refer to section 6.2.2 for the logging server component.

**D7.5** Second Evaluation Report Test Beds [Dop15]
In detail the following section is of interest for the reader:

- Refer to section 7 for the evaluation setup of the Chrome Extension Test Bed, including first results.

# 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 alignemt 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. The following figure gives an overview of the structure of this deliverable:

| Context Detection & Query Construction | Mining External Resources | Privacy-preserving Usage Analysis of Internal Resources |
|---|---|---|
| • concept in section 4 | • concept in section 7.2 | • concept in section 8 |
| • libraries in section 5 | • libraries in section 7.3 | • libraries in section 9 |
| • prototype in section 6 | • prototype in section 7.2 | • prototype in section 10 |

## 3.1 Publications

In WP5 the following publications have been accepted since the write-up of deliverable D5.3 [Sei+15] or are under submission. A reference to the corresponding section in this document is provided for the papers under submission. Accepted papers are referenced again in the respective sections of this deliverable.

**Under submission**

- We submitted a paper presenting our query construction approach on paragraph level (c.f. section 4) together with an evaluation and possible enhancements. In particular, we evaluated the performance of automatically generated queries against the best possibly achievable performance and the performance of user generated queries. The paper has been accepted as a poster (instead of a full paper) at TPDL '16, but at the time of writing this deliverable, we had not decided whether to take this opportunity, as the notification arrived during the finalization of this deliverable. The work is summarized in section 4.5.

- A paper was submitted to the TIR16 workshop dealing with the exploration of the document embedding space. This is described in more detail in section 7.2

- A paper, describing the EEXCESS framework as a whole has been submitted. This paper includes a description of the main components of context detection and query construction. The concept is described in detail in section 4.

**Accepted for publication**

- In this paper, we present an approach to create interest profiles from Twitter followees (the accounts a user follows) to eleviate the cold-start problem of personalization. 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. We accepted the invitation to publish an extended version in the ACM Applied Computing Review (to appear in fall issue).

  Christoph Besel, Jörg Schlötterer, and Granizer Michael. "Inferring Semantic Interest Profiles from Twitter Followees". In: *Proceedings of the 31th Annual ACM Symposium on Applied Computing*. SAC '16. New York, NY, USA: ACM, 2016. DOI: 10.1145/2851613.2851819

- In this paper, we propose DoSeR (Disambiguation of Semantic Resources), a (named) entity disambiguation framework that is knowledge-base-agnostic in terms of RDF (e.g. DBpedia) and entity-annotated document knowledge bases (e.g. Wikipedia). Our framework automatically generates semantic entity embeddings given one or multiple knowledge bases. In the following, DoSeR accepts documents with a given set of surface forms as input and collectively links them to an entity in a knowledge base with a graph-based approach. More details are presented in section 4.2.1.

  Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. "DoSeR - A Knowledge-Base-Agnostic Framework for Disambiguating Entities Using Semantic Embeddings". In: *The Semantic Web. Latest Advances and New Domains - 13th European Semantic Web Conference, ESWC 2016, Heraklion, Kreta, to appear*. 2016

- In this paper, we propose a new collective, graph-based disambiguation algorithm utilizing semantic entity and document embeddings for robust entity disambiguation. Our approach is also able to abstain if no appropriate entity can be found for a specific surface form. Our evaluation shows, that our approach achieves significantly (>5%) better results than all other publicly available disambiguation algorithms on 7 of 9 datasets without data set specific tuning. More details are presented in section 4.2.1.

  Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. "Robust and Collective Entity Disambiguation through Semantic Embeddings". In: *Proceeding of the 39rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, to appear*. 2016

- In this paper, we 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.

  Johannes Jurgovsky, Michael Granitzer, and Christin Seifert. "Evaluating Memory Efficiency and Robustness of Word Embeddings". In: *Advances in Information Retrieval, ECIR 2016, Padova, Italy*. Springer International Publishing, 2016, pp. 200–211

# 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]. In [Sei+15], a revised version was presented, which is again revised in this document. Briefly summarized, the major revisions comprise the following:

- In addition to the session level (which now acts as a filter), the page level also serves as input for the paragraph level.
- The information need on page level is expressed as the topic of the page.
- The information need detection on page level can be explicitly controlled by the user.
- The information need detection on paragraph level builds on the information need detection on page level. It no longer accounts for the topical overlap with the user profile.

In the web environment, observable context dimensions encompass first of all the web pages visited and in addition information like the user's location. As a proof of concept, we developed a mobile application[Sch+15], which integrates multiple context dimensions. In the general concept, 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*. Due to different characteristics, each of these levels requires its own treatment. We focus on the paragraph level and utilize more coarse-grained levels as supportive input to this level. Further, the concepts and techniques for the paragraph level are also applicable to the more fine-grained levels (terms and phrases). It is to note, that not all clients implement each level and some clients require adaptations of individual steps, but this section provides a comprehensive overview on 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.

## 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. As already mentioned, we focus on the *paragraph* level, which is able to cover the *phrase* level as well and utilize the more coarse-grained levels as supportive input. Even though the *page* and *session* level serve as supportive features in the deployment, the presented concept also allows to construct queries on these levels.

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 | input from page level + user interaction | entity disambiguation and selection keyword detection | terms + entities |
| **pages** | NONE | url & title classification | main topic extraction | entity |
| **sessions** (sequence of pages) | topic similarity navigation patterns | session clusters | main topic extraction entity disambiguation and selection | entities categories (filter) |

### 4.1.1 Phrase Level

The concept for the phrase level has not been modified since the last deliverable. Hence we simply repeat its description here.

**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 deployment, but instead apply the same approach to text selections as for the paragraph level. This is due to the fact, that, though we can identify the terms of the selection a user would utilize for a query with a high accuracy, we have no guarantee for a good query: Terms not contained in the selection might be missing and terms incorrectly classified as relevant add noise. Also, utilizing the same approach for paragraphs and user defined text selection makes the query construction process consistent. Last but not least, the user generated query is not necessarily the optimal query as we will show in section 4.5.

### 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, i.e. 84% of the paragraphs are extracted correctly (c.f. 4.5). The heuristic does not depend on a particular page structure and 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, 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. Still this simple approach provides reasonable detection performance, i.e. 65% of the focus paragraphs are detected correctly (c.f. 4.5).

**Information need recognition** As a first indicator, we utilize the information need detection on page level. If this detection neglects an information need, we neglect the information need on paragraph level as well. This way, we follow a conservative approach, sacrificing to miss an existing information need for unobtrusiveness: Even though there is no information need on page level, there still might be an information need on paragraph level. However, the absence of an information need on page level might be user-defined and hence acts as a veto. Consequently, the automatic information need detection on page level should only neglect an information need if the probability that no information need exists is really high. More details are provided in the next section 4.1.3.

If a (potential) information need exists on page level and the user looks at a paragraph for a certain amount of time or explicitly selects the paragraph, we assume an information need on paragraph level. 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. In order to avoid under- or over-specified queries, we introduce the concept of a main topic for the query, which is defined as the overall topic of the paragraph. Utilizing this main topic, queries in conjunctive normal form of the following structure can be constructed:

("main topic") AND ("entity 1" OR "entity 2" OR ...)

The extraction of named entities and the main topic from a paragraph is described in detail in section 4.2. The evaluation of the suitability of the main topic revealed, that using the topic of the whole page instead of the paragraph's main topic is the better choice for query construction. Details of the evaluation will be presented in section 4.5 and on the page topic in the next section 4.1.3.

The amount of extracted entities extracted can still be large, especially for long paragraphs. Therefore, it needs filtering, in order to provide precise results. A filtering mechanism, which splits long paragraphs into sub-paragraphs and selects the sub-paragraph (and corresponding query) with the highest overlap with the user profile is already in place. See section 4.3 for details. A second mechanism, which filters the entities based on their frequency, position(s) in the paragraph, similarity to the main topic and among each other, has been evaluated as a research prototype (c.f. 4.5).

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 or German text and 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 integrated a client-side fallback solution. 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** The information need recognition on page level is mainly controlled by the user: She can decide whether to activate or deactivate EEXCESS on a particular. The latter is a veto for an information need, while the former indicates a potential information need, which is then used as first indicator for the more fine-grained levels. The user can decide between an opt-in and an opt-out approach, i.e. she can de-activate EEXCESS in general and define exceptions or activate EEXCESS in general and also define exceptions. In the latter case, the automatic information need recognition is active on pages which are not defined as inactive exceptions. On these pages, it is the task of the recognition mechanism to decide whether a (potential) information need exists or not. Currently, we assume a potential information need on all pages, except for those, on which EEXCESS has been de-activated commonly by a large user group. Examples for such cases are the Web-interface for Google Mail or Stackoverflow. We plan to enhance this blacklisting approach by predicting a potential information need, or more precisely, its rejection, by a classifier, based on page URL and title. Still, we do not have sufficient training data for such a classifier.

**Information need expression** The topic of the page provides the main topic for the query structure defined on paragraph level. In principle, this topic could be extracted with the same mechanism as for a paragraph, with details provided in section 4.2. However, this extraction mechanism raises performance issues: the extraction of named entities from the whole page takes too long. Therefore, we utilize the paragraph's main topic in the deployed version and are investigating means to extract the page topic in a more efficient way. In this line, we see the page title as the primary source of information. Unfortunately, this title often contains additional information, not related to the page, which can be considered as noise, that has to be removed. For example, in the title of the Wikipedia page about looms - "Loom - Wikipedia, the free encyclopedia", only the term in front of the hyphen is relevant. Often this part is related to the provider of the page. Therefore, in a first approach, we followed several links in the page and compared their titles to the original title, removing equal parts from the original title. While this approach provided good performance in extracting the relevant parts, it caused problems with the (browser) session management on some pages and hence is not applicable.

### 4.1.4 Session Level

We describe the session level from a conceptual point of view for the sake of completeness, while it has been implemented in the deployment to a minor extent. The implemented part comprises the information need expression, used as a filter for the paragraph level (indicated in gray color in Table 1). In the implementation, the whole browsing history is treated as a single large session.

**Relevant context identification** The relevant context of a session is a set of pages, which belong to this 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** The main topics from previously visited pages within a session can be used to expand the query on paragraph level, i.e. they provide additional keyword candidates for the right part of the query structure described in the paragraph level. As the topic extraction for a page already happens on page level, those topics are readily available on the session level.

The information need expression on session level is used for filtering the queries on paragraph level (indicated gray in Table 1). To this end, the categories associated with the entities of a query, for which a user has viewed the results are stored in a user profile. On paragraph level, the query with the highest overlap with this profile is selected as the query that is issued to the federated recommender. Details are presented in section 4.3.2.

As mentioned above, in the deployment the whole browsing (or query) history is treated as a single large session, while a more sophisticated approach could distinguish between a long- and short-term profile. With a session detection approach in place, the categories of queries within a session would make up the short-term profile and the categories of the whole query history the long-term profile.

## 4.2 Entity and Category Detection

As detailed above, entities and categories are used for query generation and personalization. To extract them from paragraphs we use our **DoSeR**-framework[1]. 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.

### 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 robust entity disambiguation, where robustness is defined as achieving high accuracy on different KBs and over a large set of different domains. We first distinguished between entity-centric KBs (i.e. DBpedia, YAGO3) and document-centric KBs (i.e. Wikipedia, CalbC). 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

[1] `http://purl.org/eexcess/components/research/doser`

**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.

In addition to KB properties that affect disambiguation accuracy in general, we focused on entity disambiguation algorithms that provide high accuracy on different kinds of KBs (i.e. entity-centric KBs, document-centric KBs and a combination between them) on various domains. These domains are for example on short Twitter messages, web pages, news documents, encyclopedias, RSS-Feeds etc. While most authors report to outperform other entity disambiguation algorithms on their KBs/-domain/data set, they do not achieve comparable accuracy on other domains. So their approaches could be considered as not being very robust against different types of data sets.

To tackle this problem, we first proposed how to easily generate semantic entity embeddings to compute a state-of-the-art semantic entity similarity measurement between entities regardless of the type of KBs available [ZSG16a]. Embeddings are n-dimensional vectors of concepts which describe the similarities between these concepts via cosine similarity. To create these embeddings, we make use of Word2Vec, a group of state-of-the-art, unsupervised algorithms to create word embeddings from (textual) documents initially presented by Mikolov et al [Mik+13]. However, we presented two algorithms to create appropriate input corpora for Word2Vec given an entity-centric or document-centric KB. The output corpora can be easily combined to leverage the information of multiple KBs. A simple, collective, graph-based entity disambiguation approach (in the following denoted as DoSeR_simple) reveals that the proposed similarity measurement based on semantic embeddings achieves state-of-the-art accuracy on entity-centric KBs (i.e. DBpedia and YAGO3) and document-centric KBs (i.e. Wikipedia), despite ignoring the surface form surrounding context [ZSG16a]. Table 2 shows an overview of disambiguation results of DoSeR_simple and publicly-available state-of-the-art disambiguation approaches. We evaluated the approach on various data sets from different domains, like tweets (e.g. Microposts-2014 Test), news documents (e.g. MSNBC) and web (e.g. AIDA/CONLL-TestB).

Based on the DoSeR framework and the entity embeddings used in DoSeR_simple [ZSG16a], we additionally proposed a new robust, collective and open-source state-of-the-art disambiguation system that also leverages the surrounding context of the respective surface forms [ZSG16b] (in the following denoted as DoSeR_advanced). Besides the entity embeddings created with Word2Vec [ZSG16a], we also create entity-context embeddings with Doc2Vec, a modification of Word2Vec. Doc2Vec learns fixed-length embeddings from variable-length pieces of texts like documents [LM14]. It addresses some of the key weaknesses of bag-of-word models by incorporating more semantics and considering the word order within a small context. Our Doc2Vec model is trained on an arbitrary entity-context corpus (i.e. Wikipedia) yielding the entity-context embeddings, and the same model is later used to generate the surface-form-context embeddings. The cosine similarity between both, entity-context embeddings and surface-form-context embeddings denotes the contextual matching of the respective entity to the context of a surface form.

Given the semantic embeddings (i.e. entity embeddings generated with Word2Vec and entity-context embeddings with Doc2Vec) and a disambiguation index containing entity definitions (e.g. labels, typical surface forms etc), we create a disambiguation graph. This graph consists of nodes for all entity candidates per surface form and one node, the topic node, that represents the current predominant topic of already disambiguated entities. This topic node allows us to include a-priori information

Table 2: Micro-averaged F1 values of DoSeR_simple, DBpedia Spotlight, AIDA, WAT and Wikifier on seven data sets.

| Data set | DoSeR | DoSeR (+Wiki) | Wikifier | Spotlight | AIDA | WAT |
|---|---|---|---|---|---|---|
| ACE2004 | 0.681 | **0.864** | 0.824 | 0.713 | 0.741 | 0.800 |
| AIDA/CONLL-TestB | 0.597 | 0.722 | 0.776 | 0.593 | 0.806 | **0.843** |
| AQUAINT | 0.638 | 0.820 | **0.862** | 0.713 | 0.534 | 0.768 |
| MSNBC | 0.719 | **0.881** | 0.851 | 0.511 | 0.796 | 0.777 |
| N3-Reuters | 0.700 | **0.727** | 0.694 | 0.577 | 0.571 | 0.644 |
| IITB | 0.497 | 0.713 | **0.755** | 0.447 | 0.277 | 0.611 |
| Microposts-2014 Test | 0.469 | **0.639** | 0.586 | 0.623 | 0.412 | 0.595 |
| **Average** | 0.614 | **0.767** | 0.764 | 0.597 | 0.591 | 0.720 |

from previous steps into the structure of the graph. The edge weights are based on similarities between entity embeddings as well as similarities between entity-context embeddings and the surface forms' surrounding context [ZSG16b]. To compute the most likely entity (i.e. disambiguation result) for each surface form based on the created disambiguation graph, we apply the PageRank algorithm. PageRank is a well-researched, link-based ranking algorithm simulating a random walk on graphs and reflecting the importance of each node. In the special case that no entity fits a query surface form, our algorithm is able to abstain resulting in returning the pseudo-entity *NIL*. To significantly optimize the performance we integrated a *Semantic Embedding Candidate Filter* that filters those entity candidates that fit to the general topic described by the already disambiguated entities requiring at least 3 already assigned entities. The underlying assumption is, that all entities in a paragraph are somehow topically related.

In an in-depth evaluation we compare DoSeR_advanced against other publicly-available state-of-the-art disambiguation approaches (cf. Table 3) and a very strong Prior baseline which links surface forms to the entities with the highest prior probability $p(e_i|m)$. It estimates the probability of seeing an entity $e_i$ with a given surface form $m$. Overall, our approach disambiguates the entities highly accurate and attains state-of-the-art or nearly state-of-the-art results on all nine data sets. Hence, our approach is very well suited for all kinds of documents available in the web (e.g. tweets, news, etc.). Further details to our disambiguation approaches can be found in [ZSG16a] for DoSeR_simple and [ZSG16b] for DoSeR_advanced.

In term of disambiguation performance, our system has the advantage to accept multiple queries in parallel, but is not yet optimized for high-performance disambiguation. For that reason, we apply the Named Entity Recognition and Named Entity Disambiguation system DBpedia Spotlight[2] 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. A comparison of disambiguation accuracy between DoSeR and DBpedia Spotlight is given in Table 2 and 3. 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.

[2]https://github.com/dbpedia-spotlight/dbpedia-spotlight/wiki

Table 3: Comparing micro-averaged F1 values of DoSeR_advanced, the prior probability baseline as well as the publicly available entity disambiguation systems Wikifier, Spotlight, AIDA, Babelfy and WAT on nine data sets.

| Data Set | DoSeR | Prior | Wikifier | Spotlight | AIDA | Babelfy | WAT |
|---|---|---|---|---|---|---|---|
| ACE2004 | **0.907** | 0.831 | 0.834 | 0.713 | 0.815 | 0.561 | 0.800 |
| AIDA/CONLL-TestB | 0.784 | 0.661 | 0.777 | 0.593 | 0.774 | 0.592 | **0.843** |
| AQUAINT | 0.842 | 0.803 | **0.862** | 0.713 | 0.532 | 0.652 | 0.768 |
| DBpedia Spotlight | **0.810** | 0.745 | 0.797 | 0.789 | 0.508 | 0.522 | 0.652 |
| MSNBC | **0.911** | 0.711 | 0.851 | 0.511 | 0.782 | 0.607 | 0.777 |
| N3-Reuters128 | **0.850** | 0.700 | 0.703 | 0.577 | 0.596 | 0.534 | 0.644 |
| IITB | 0.741 | 0.711 | **0.766** | 0.447 | 0.270 | 0.470 | 0.611 |
| Microposts-2014 Test | **0.750** | 0.630 | 0.586 | 0.453 | 0.453 | 0.473 | 0.595 |
| N3 RSS-500 | **0.751** | 0.678 | 0.732 | 0.622 | 0.716 | 0.630 | 0.682 |
| **Average** | **0.816** | 0.718 | 0.768 | 0.602 | 0.605 | 0.560 | 0.708 |

### 4.2.2 Category Annotation

Since we exclusively annotate English or German Wikipedia/DBpedia entities, DoSeR is able to create the respective statistics of categories that are associated with the entities extracted in Section 4.2.1. Given an entity, we extract a set of Categories[3]. 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) Doc2Vec, and (ii) PageRank with Word2Vec. A couple of expert users evaluated both approaches and concluded a superiority of the Doc2Vec approach. Another important reason is that PageRank with Word2Vec relies on extracted entities with DBpedia Spotlight, which might annotate incorrect entities. Thus, a topic extraction with PageRank with WordVec strongly depends on the entity annotation quality. For those two reasons, we employed the Doc2Vec approach as default topic detection method in the current implementation. We emphasize that the outcome might differ after some iterations since the inference step produces slightly different vectors after each step.

**Doc2Vec (Default)** Generally based on Word2Vec, Doc2Vec produces a vector given a sentence or document (cf. Section 4.2.1). Hence, we use the entire input paragraph and infer a representative vector given our 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).

**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

[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.

## 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 and German text. While further languages might be added by integrating the respective knowledge bases, we are not able to cover all languages. Therefore, a language agnostic fallback solution is desirable. We refer to this fallback as keyword extraction and present our approach in the next section 4.3.1.

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). This filtering step is also required for the keyword extraction. The filtering step is described in section 4.3.2.

### 4.3.1 Keyword Extraction

For the keyword extraction, we evaluated standard keyword extraction techniques, like the pure term frequency, tf-idf, BM25 and TextRank [MT04]. To this end, we developed a browser extension[4] 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 than 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, which is language dependent. 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, by either accounting for browsing history until installation of the extension, browsing history from the installation of the extension or the combination of both. For the browsing history part until the installation, the pages in the user's history are crawled to build the document corpus. The crawled pages do not necessarily resemble the pages the user has seen, as we remove the parameter parts from the URL, in order to avoid undesirable repetition of actions. For example, the URL might be the confirmation of a shopping step and we need to avoid placing such an order again. Also session information might not be available anymore. Keywords extracted with different approaches are presented to the user without telling her, which approach has been used.

The results (629 ratings from 26 users) vote for TextRank as the best performing approach with an accuracy around 0.66. The best language independent approach is tf-idf, with an accuracy around 0.60. For these measurements, the accuracy of tf-idf and BM25 has been averaged across different corpus sizes, i.e. the amount of browsing history taken into account. In detail, accounting for the browsing history since installation performs worst (0.57), followed by the browsing history up to the

[4]`http://mics.fim.uni-passau.de/serverREL/RELEVANTICO/intro/en.html`

installation (0.59). Accounting for the whole history performs best (0.64) but still worse than TextRank (0.66). Hence we opted for TextRank.

TextRank achieves the best performance, when the terms are filtered for nouns and adjectives, while filtering for nouns is also possible. Omitting this pre-processing step leads to significantly lower performance [MT04]. For the extraction of noun phrases, we developed **NounPhraseJS**[5] 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 dependance. To overcome the language dependance, we apply a heuristic based on the case sensitivity of terms. Of course, such a heuristic provides lower performance than sophisticated POS-tagging. Nevertheless, we decided for TextRank together with this heuristic, as building the document corpus in the other language independent approaches can raise performance issues, especially when the browsing history grows large. Furthermore, the heuristic in the pre-processing step can be easily replaced by sophisticated POS-tagging for particular languages.

### 4.3.2 Filtering and Personalization

All of the presented approaches 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. Also, the quality of the filtering step depends on the parameter k, which is set to 10 for the TextRank approach.

Regarding the named entities, we filter the query candidates based on a user profile. This user profile consists of categories assigned to entities in past queries. The categories provide a more abstract level than the entities themselves. With this more abstract representation, the user profile is a representation of the user's interests. Since queries are created and sent automatically, we need to make sure, that the user is indeed interested in the categories in the profile. Therefore, we only add those categories to the profile, where the user has viewed the result set of the corresponding query. The filtering step is applied in the following way: We subdivide large paragraphs into smaller sub-paragraphs and extract the entities of each sub-paragraph separately. Hence we have different query sets for each sub-paragraph. The query set with the highest overlap in terms of categories in the user profile is then pre-selected and issued. Still the user can select another sub-query or issue the query for the whole paragraph.

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 follow two strategies to overcome this limitation: On the one hand, we, 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 the second strategy, 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. The possibility to explicitly state interests has already been integrated into the deployment, while the second strategy is available as standalone prototype.

[5] `https://github.com/EEXCESS/NounPhraseJS`

## 4.4 Embedded Context Detection

The embedded context detection has not been modified since the last deliverable. Hence, we simply repeat its description in this section in order for this document to be self-contained.

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 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. More details on the compression methods and evaluation results can be found in the publication [JGS16].

## 4.5 Perfomance Evaluation

We evaluated the performance of the context identification and information need expression step on paragraph level as described in section 4.1.2 with a user study. In particular, this evaluation comprised the extraction and detection of the focused paragraph, the suitability of the main topic (see also section 4.2.3) and the performance of the constructed queries. Also, we investigated additional methods to filter the extracted named entities based on the evaluation data.

### 4.5.1 Study Setup and Participants

We used a modified version of the chrome extension for the study. The modifications comprised the inclusion of appropriate logging mechanisms and the removal of components, which were not relevant to the evaluation (e.g. additional visualizations). Participants were instructed to choose from a set of Wikipedia pages and navigate to a particular section on these pages. The focused paragraph was selected and highlighted automatically. If the participant disagreed with this selection, she was asked to modify it accordingly. Afterwards, an automatic query was generated and issued and participants had to rate the results. Then they were asked to adapt the query and again rate the results until they either arrived at perfect results, did not deem the search engine able to deliver appropriate results or a timeout was met. More details of the study setup can be found in deliverable [Dop16a]. In that deliverable, we reported first results of the study for 27 participants. Meanwhile, 77 university students took part in the study and the results reported here are based on the data of all those participants. The evaluation was carried out on a cleaned dataset, e.g. we removed queries, were no rating has been given or paragraphs and associated queries from pages other than the predefined ones.

### 4.5.2 Paragraph Detection and Extraction

The evaluation of the focused paragraph comprises two steps: First, the extraction of paragraphs (separating actual paragraphs from navigational menus, advertisements, etc.) and second the detection of the focused paragraph on the set of extracted paragraphs. Whenever a participant did not modify the extracted paragraphs, we considered them as correctly extracted. This was the case for 84% of the paragraphs, i.e. those paragraphs were extracted correctly or meaningful from a user perspective. From the set of correctly extracted paragraphs, the focused paragraph was detected correctly in 65% of the cases, i.e. participants did not modify its pre-selection.

### 4.5.3 Suitability of Main Topic

We consider the suggested main topic as suitable, if it was not modified by the user or a modification did not lead to an improved result set (in terms of relevance ratings). This results in a main topic fit of 83%. However, in some cases, the query has not been modified at all. This can be due to the user being already perfectly satisfied with the results of the automatic query or not being able to modify the query due to time constraints for example. After removing the cases, where we cannot make a definitive statement about the main topic quality (i.e., we do not know the reason, why it has not been modified), the remaining fit is still at 79%.

As the evaluation was carried out on Wikipedia pages and the extracted main topic is represented by a Wikipedia article title, we can easily compare the extracted main topic of the paragraph with the topic of the page. In 103 queries, both were the same. Moreover, the suggested main topic was different from the page topic in 81% of the queries where the main topic has been changed and the change resulted in an improvement in 63%. Also, the page topic was set as new main topic of the query in 25% of the main topic modifications. These findings suggest that the topic of the page is better suited for the query than the (extracted) topic of the paragraph.

### 4.5.4 Query Performance

We compared the performance of the automatically constructed queries with the best queries, users were able to formulate. Further, we evaluated the performance of the best achievable queries. To this end, we set up an own search index with all the results retrieved during the study. To find the best queries, in principle, we issued queries with all possible combinations of the extracted entities against our own index. We also evaluated the performance of the automatic and user queries on this index. Based on the best achievable queries, we trained a decision tree classifier, to filter the extracted named entities of automatic queries. The F1-scores of the different approaches are depicted in Table 4. The reason for the performance drop of the filtered and best queries on the original index

Table 4: Comparison of automatic, user, best and filtered queries on our own and the original index.

| | own index | | | | original index | | | |
|---|---|---|---|---|---|---|---|---|
| | best | automatic | filtered | user | best | automatic | filtered | user |
| F1-score | 0.49 | 0.31 | 0.33 | 0.35 | 0.21 | 0.24 | 0.15 | 0.31 |

is, that those queries yield new results, which were not obtained during the study. These results might actually be relevant, but as we do not have ratings available, and hence cannot judge the relevance, we treat all of them as not relevant. The evaluation shows a low performance in general, but this a factor, we cannot influence directly (for the evaluation, we obtained results from the search APIs of Mendeley and Europeana directly, without using the federated recommender). However, the performance of the automatic queries is not far below the best queries users were able to formulate. When accounting for all queries issued by the users, the performance of the user queries also drops. By taking the best query, a user was able to formulate, this query could in fact also be an automatic query, in case the user was not able to formulate a better performing query (in terms of F1-score).

## 4.6 Summary

Table 5 summarizes, which components we implemented and integrated into the EEXCESS framework, which components we implemented and made available as standalone prototype, and which components we implemented as a research prototype.

Table 5: Overview of implemented prototypes.

| **Implemented and integrated into the EEXCESS framework** | |
|---|---|
| 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) |
| Main topic extraction | Available server-side (c.f. section 4.2) |
| Keyword extraction via TextRank | Available in the context detection library (c.f. section 5) |
| Query personalization | Available in the context detection library (c.f. section 5) |
| **Implemented and available as standalone prototype** | |
| NounPhraseJS | Client-side extraction of noun phrases |
| Embedded context detection | Compression of Word2Vec vectors |
| Twitter interest profile construction | Construction of interest profiles from Twitter followees |
| **Implemented as research prototype** | |
| Client-side CRF | Query generation on phrase level |
| Query candidates filtering | Limit the set of query candidates |

# 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. Although several sub-modules and additional features were added, there have not been any changes on the architectural level, nor on the source code locations or installation procedures. Hence, this section is a repetition from the last deliverable D5.3 [Sei+15].

**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 have two module types: the components in C4 (Cultural and sCientific 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

[6]https://w3c.github.io/webmessaging/

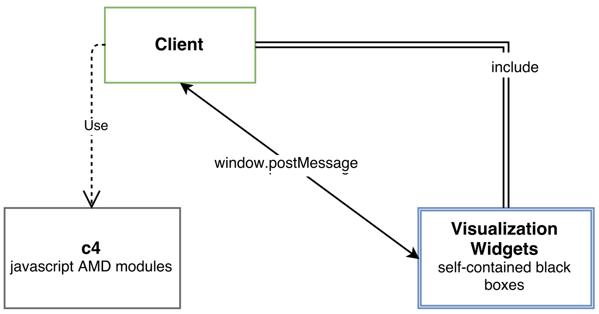


Figure 2: Overview of the client-side modules

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 63.

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 4.6, 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 50.

**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 48 and 46.

## 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).
  $\Rightarrow$ API description and usage details in the appendix on page 50.
- 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).
  $\Rightarrow$ API description and usage details in the appendix on pages 46 and 48.

### 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 license[7].
- 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 2[8].

[7] `http://opensource.org/licenses/MIT`
[8] `http://opensource.org/licenses/GPL-2.0`

### 5.1.2 Installation and Usage

- C4 is available as bower[9] 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 RequireJS[10].
  ⇒ API description and usage details in the appendix on page 50.

- 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 48 and 46.

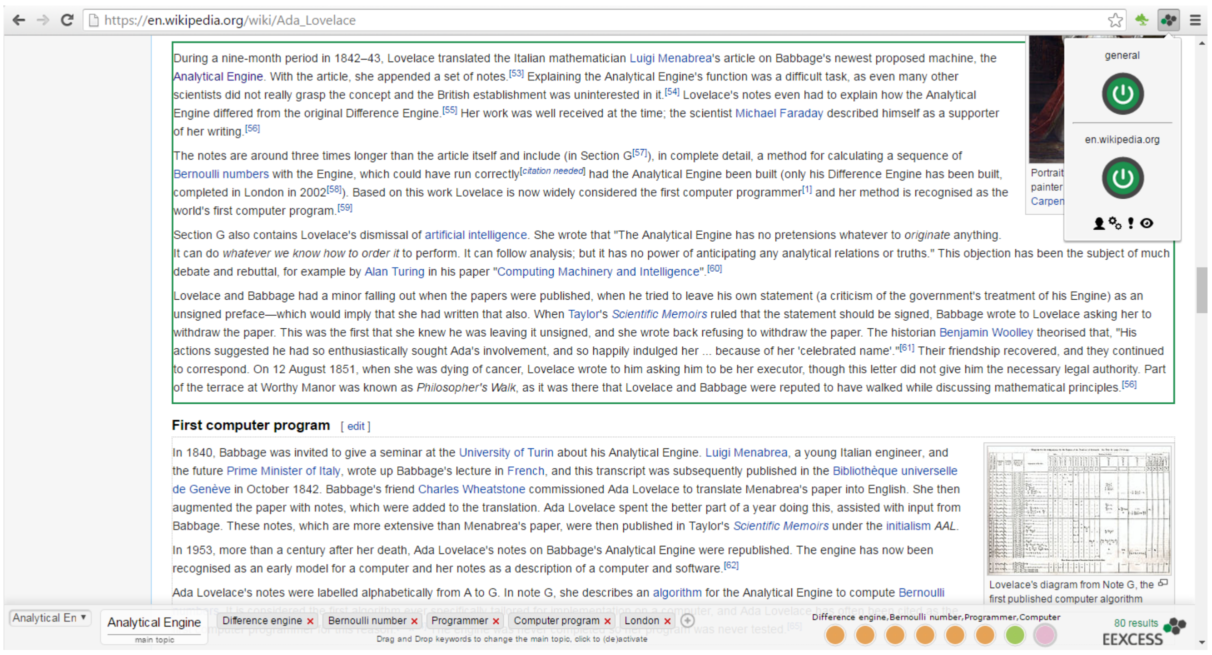
# 6 Context Detection Prototype: Browser Extension

As an example for the usage of the context detection library, we describe the feature-richest prototype, which is the Chrome browser extension. The chrome extension implements the context detection and query construction as described in section 4 to the full extent. In other clients, the modules are used analogous, but partially to a lesser extent. We start with an updated description of the interface, while we repeat the architecture, source code location and installation from the last deliverable D5.3 [Sei+15], as the latter did not change since then.

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. This paragraph contains four sub-paragraphs and hence, four sub-queries have been created via named entity detection from those four sub-paragraphs. The query term candidates (i.e. the sub-query) with the highest overlap of corresponding categories with the user profile 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. The possibility to change the sub-query is by the selection menu at the bottom left: Further sub-queries are indicated by their main-topic and selectable by the user. Further, the possibility to use all extracted entities from the paragraph is provided, as well as filtering the query for persons or/and locations. The popup in the upper right contains the de-/activation mechanisms and links to a feedback form and the option, profile and help page. A guided tour of the Chrome extension is provided in Deliverable D7.6 [Dop16b, section 3.2].

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.

[9] http://bower.io/
[10] http://requirejs.org/


https://en.wikipedia.org/wiki/Ada_Lovelace

During a nine-month period in 1842–43, Lovelace translated the Italian mathematician Luigi Menabrea's article on Babbage's newest proposed machine, the Analytical Engine. With the article, she appended a set of notes.[53] Explaining the Analytical Engine's function was a difficult task, as even many other scientists did not really grasp the concept and the British establishment was uninterested in it.[54] Lovelace's notes even had to explain how the Analytical Engine differed from the original Difference Engine.[55] Her work was well received at the time; the scientist Michael Faraday described himself as a supporter of her writing.[56]

The notes are around three times longer than the article itself and include (in Section G[57]), in complete detail, a method for calculating a sequence of Bernoulli numbers with the Engine, which could have run correctly[citation needed] had the Analytical Engine been built (only his Difference Engine has been built, completed in London in 2002[58]). Based on this work Lovelace is now widely considered the first computer programmer[1] and her method is recognised as the world's first computer program.[59]

Section G also contains Lovelace's dismissal of artificial intelligence. She wrote that "The Analytical Engine has no pretensions whatever to *originate* anything. It can do *whatever we know how to order it* to perform. It can follow analysis; but it has no power of anticipating any analytical relations or truths." This objection has been the subject of much debate and rebuttal, for example by Alan Turing in his paper "Computing Machinery and Intelligence".[60]

Lovelace and Babbage had a minor falling out when the papers were published, when he tried to leave his own statement (a criticism of the government's treatment of his Engine) as an unsigned preface—which would imply that she had written that also. When Taylor's *Scientific Memoirs* ruled that the statement should be signed, Babbage wrote to Lovelace asking her to withdraw the paper. This was the first that she knew he was leaving it unsigned, and she wrote back refusing to withdraw the paper. The historian Benjamin Woolley theorised that, "His actions suggested he had so enthusiastically sought Ada's involvement, and so happily indulged her ... because of her 'celebrated name'."[61] Their friendship recovered, and they continued to correspond. On 12 August 1851, when she was dying of cancer, Lovelace wrote to him asking him to be her executor, though this letter did not give him the necessary legal authority. Part of the terrace at Worthy Manor was known as *Philosopher's Walk*, as it was there that Lovelace and Babbage were reputed to have walked while discussing mathematical principles.[56]

**First computer program** [ edit ]

In 1840, Babbage was invited to give a seminar at the University of Turin about his Analytical Engine. Luigi Menabrea, a young Italian engineer, and the future Prime Minister of Italy, wrote up Babbage's lecture in French, and this transcript was subsequently published in the Bibliothèque universelle de Genève in October 1842. Babbage's friend Charles Wheatstone commissioned Ada Lovelace to translate Menabrea's paper into English. She then augmented the paper with notes, which were added to the translation. Ada Lovelace spent the better part of a year doing this, assisted with input from Babbage. These notes, which are more extensive than Menabrea's paper, were then published in Taylor's *Scientific Memoirs* under the initialism *AAL*.

In 1953, more than a century after her death, Ada Lovelace's notes on Babbage's Analytical Engine were republished. The engine has now been recognised as an early model for a computer and her notes as a description of a computer and software.[62]

Ada Lovelace's notes were labelled alphabetically from A to G. In note G, she describes an algorithm for the Analytical Engine to compute Bernoulli

Lovelace's diagram from Note G, the first published computer algorithm

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

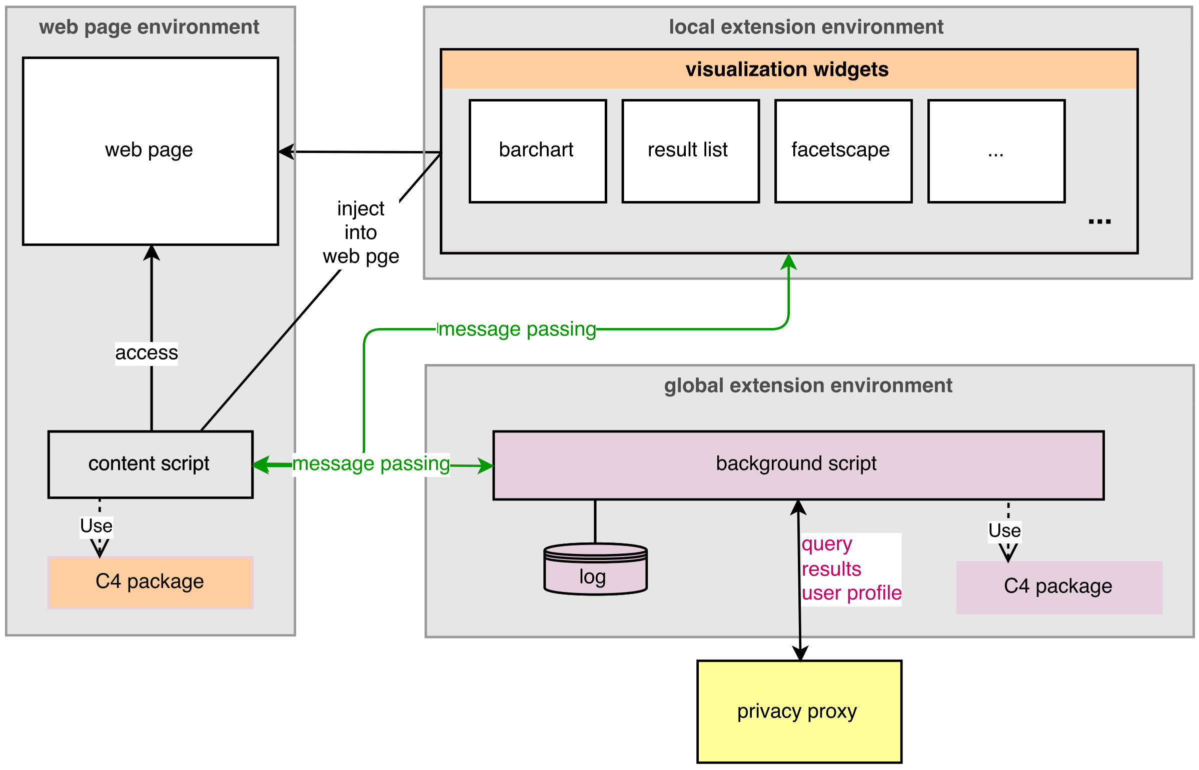


Figure 4: Browser extension architecture.

**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 paragaph 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 license[11].

## 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.js[12]). 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

In this section the resource mining activities will be summarized. Since the research and development was fragmented into two different parts, this section will be structured accordingly. Firstly, the creation of data corpora (i.e. the aggregation of real world data) will be depicted in subsection 7.1. Secondly, the experiments that were conducted using the corpora will be described in subsection 7.2. We will end this section by aggregating the references to the source code repositories and license information in subsection 7.3.

## 7.1 Corpora

Over the course of the project two corpora have been created:

- **Blogs15**. A corpus that consists of over 80.000 blog post, which was introduced in [Sei+14].
- **Econstor16**. A new corpus containing nearly 100.000 research papers from the field of economics, which will be described in this section.

This section recapitulates Blog15 and introduces Econstor16, a new corpus, which hasn't been explained earlier.

### Blogs15

Blogs15 contains 80.000 blog posts from 10 different blogs. Those 10 blogs where selected by their relevance in the field of economics as well as technical and legal aspects (e.g. some blogs disallow web crawlering or suppress such efforts technically). A web crawler (called Blog-Crawler) was developed to visit those websites automatically and download the desired information into a database. Further details are specified in [Sei+14] Section 9.2.1.

### Econstor16

Econstor16 is based on ZBWs[13] open access service Econstor[14] which is among the largest open access repositories in the field economics. Besides the plaintext of the documents and along with the usual meta information like author, title and publication year there are several other useful information like the number of citations a paper received, language, link to the original PDF file, author assigned keyword and keywords assigned by domain experts from a controlled vocabulary (see STW[15]). Moreover, there are Econbiz[16] identifiers as well as RePEc[17] identifiers that allow fetching further information from other services. To date (May 2016), Econstor serves 108,000 documents to the users. But not all of them allow plaintext extraction (using encryption), which reduces the size of the corpus to 96,000. Since the repository is rapidly growing, this number will keep increasing. Figure 5 illustrates the length of the documents.

The bulk has less than 10,000 words. 77% of the documents are written in English, 19% in German. The remaining 4% are composed out of over 20 languages. The code that created this corpus consists of two components, a **downloading component** and a **Corpus Pre-Processor**, which will be described in more detail in the following section.

[13]http://www.zbw.eu
[14]http://econstor.eu/
[15]http://zbw.eu/stw/version/latest/about
[16]http://www.econbiz.de/
[17]http://repec.org/

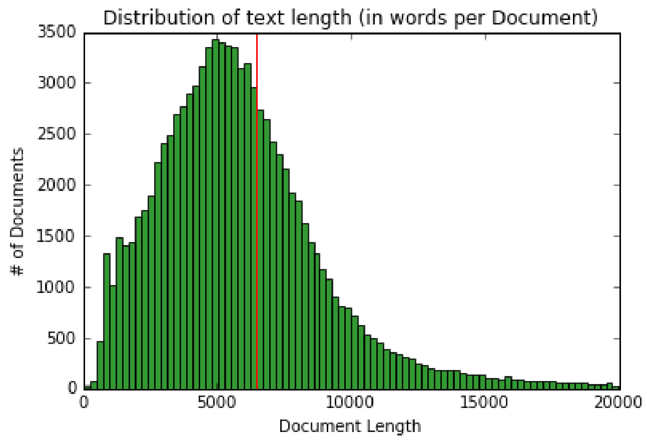


Figure 5: A histogram depicting the length of the documents in Econstor16. The red line marks the mean length of 6476 words per document.

## 7.2 Prototypes

We have developed five prototype applications, which we first summarize briefly and then explain in detail in this section.

- The **Blog-Crawler** was used to create the Blogs15 corpus as already described before. Hence we do not provide an additional description in this section.
- The **Blog-Analyzer** is a fuzzy-matching tool that searches a corpus of text data for PDF links and tries to find them on EconBiz, which allows to retrieve further meta information.
- The **Corpus Pre-Processor** extracts text contained in PDF files, normalizes, filters and stores them in a parallel fashion.
- After doing supervised training the **Popularity Estimator** predicts the whether or not a writing will be popular.
- The **Semantic Relatedness and Explanation Tool** is a means to explore the semantical differences between two documents. It generates a list of words that, when "added" or "subtracted" to one document, approximate the other.

### Blog-Analyzer

The Blog-Analyzer searches the database for links to PDF files and matches those to EconBiz files. This allowed us to find blog posts that mentioned documents from EconBiz. This prototype was already described in more detail in [Sei+14].

### Corpus Pre-Processor

As a prerequiste for the Corpus Pre-Processor, we developed a **downloading component**. The task of the **downloading component** is to compile the desired information from three different sources and store them into JSON files. Most information is retrieved from the Econbiz-API[18]. This includes, among other, authors, title, hyperlink to the PDF document and the publication year. In order to obtain citation count information we used the CitEc[19]-API which is a service hosted by RePEc. This

[18]https://api.econbiz.de/doc
[19]http://citec.repec.org/

service provides information on how many resources are cited by a paper, how many times was a paper cited and by which papers was it cited. Moreover, we fetched document annotations from the STW service. They were assigned by domain experts and try to reflect the content of a document.
The task of the **Corpus Pre-Processor** is then threefold:

1. **Plaintext extraction.** Based on the pdfminer[20] library, a framework was build, that extracts text from PDF files. This processes is computationally intensive. Therefore, computation is done in a parallel fashion (using Pythons `multiprocessing` infrastructure and the `pypy` interpreter.
2. **Language guessing.** The language of a document is often crucial for textmining tasks. Hence, we employed the Python library `langdetect`[21] to detect the language of the documents.
3. **Text normalization.** Since the resulting plaintexts from `pdfminer` are of varying quality, a pre-processing step detects common errors and fixes them. It also discards the cover page (which is the first page in every document from Econstor).

The Corpus Pre-Processor extracts plaintext from the PDF files and merges it with a corresponding JSON file. Extracting plaintext from PDF files is not a straightforward process. Encoding problems, password-protected files and a large variety of text formattings are only a few of the potential problems. To mitigate some of these problems a filter module is used to correct some of the common errors. Using regular expressions, it can be easily adapt to other scenarios. Further, the Corpus Pre-Processor adds the guessed language to the JSON file as well. The processing can be interrupt at any stage of the process, as the program writes checkpoint files every 30 seconds that allows it to resume the processing.

### Popularity Estimator

Estimating the relevance of documents is often used for recommendation systems. These estimations often rely on citations graphs, author networks or other meta information about the items. We, in contrast, during our work on this prototype focused on the text itself, ignoring meta information. Our approach is based on Document Embeddings [DOL15], which is an algorithm that can be thought of as being a semantic hash algorithm. Likewise conventional hash algorithm, it does assign a fixed length value to an arbitrary long text. But on the contrary, it assigns documents with similar semantic and syntactic properties to nearby locations. This means that the distance between two documents characterizes their similarity.

The purpose of this prototype is to estimate the relevance of a document merely through its Documents Embedding representation, using corpopra like the one described in section 7.1 (The experiment described in this section is based on the Ecostor16 corpus). The setup that is used is outlined in figure 8. The first step transforms the plaintext of the documents into their Document Embeddings (the n-dimensional vector), which is subsequently used to train a classifier (in the figure, a simple feed forward network is used, but other classifiers work likewise) in a supervised fashion. In order to do supervised training, popularity indicating labels are required (e.g. citation counts, Altmetrics, monthly reader and so forth). The labels that we used for the training were derived from the citation count information of the Econstor16 corpus. If a document was cited at least once, it was assigned the label "high impact" and "low impact" otherwise. The threshold of one or more was chosen, because it splits the corpus into two approximately equally-sized parts.

Once the training is completed, the classifier can be used to predict the popularity of documents that haven't been used for training.

For classification we evaluated three different classifiers: Random Forrest, Support Vectors Machine and Feed Forward Neuronal Networks. Both, Random Forrest and Support Vector Machines achieved a prediction accuracy of 66% whereas the Neuronal Network achieved 68%. For all classifiers we explored their respective hyperparameter space. This is exemplified for the Neuronal Network classifier

[20] https://pypi.python.org/pypi/pdfminer/
[21] https://pypi.python.org/pypi/langdetect/1.0.5

Table 6: Hyperparameter space exploration of the classifier

| Input Dimension | Dropout | NN Architecture | Accuracy | Rank |
|---|---|---|---|---|
| 600 | 0.6 | (500, 160) | 68% | 1 |
| 600 | 0.6 | (250, 80) | 67.5% | 2 |
| 600 | 0.6 | (100) | 67.4% | 3 |
| 600 | 0.6 | (400, 150, 40) | 67.3% | 4 |
| ... | ... | ... | ... | ... |
| 300 | 0.4 | (300) | 67.2% | 8 |
| ... | ... | ... | ... | ... |
| 100 | 0.6 | (250, 80) | 66% | 21 |

in table6. The table shows three hyperparameters (Input Dimension, Dropout, NN Architecture), the prediction accuracy as well as the rank of the result. The prediction accuracy obviously benefits from more input dimensions. A Dropout [Hin+12] propability of 0.6 increased the prediction accuracy when compared to 0.4. However, higher values lead to declining results. What the table also show is, that, although Dropout was used, shallow networks performed better than deeper ons. This indicated that the training data contained insufficient variance. Plotting the input vectors (using t-SNE [HS06] for

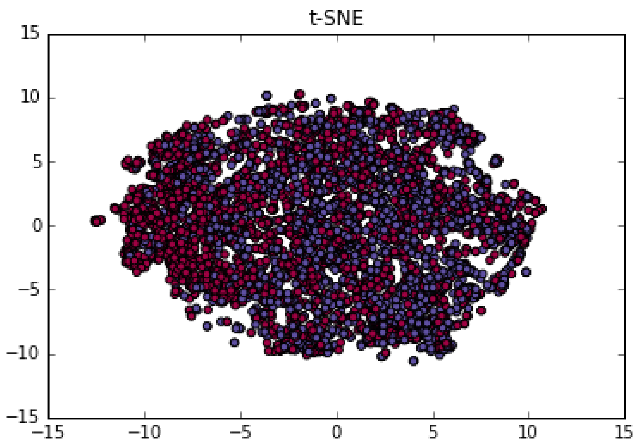


Figure 6: t-SNE visualization of 10.000 documents. Red dots indicate ’High Impact’, blue dots indicate ’Low Impact’

dimensionality reduction), we can also see that it is hard to tell the two classes apart, although a slight tendency towards ’High Impact’ documents can be observed on the on the leftmost part of the figure. This observation is also supported by figure 7 which depicts the train and the validation Cross Entropy loss during training. The crucial observation here is, that train and validation loss start deviating after only a few training iterations. This means, that the classifier’s performance keeps increasing on the training set while stagnating on the validation set. This is a clear sign of overfitting. Although the classifier used only content-features (semantic embeddings of documents), and no network features like citations graphs, an accuracy of 68% could be achieved (trivial classifier 50%). This means, content-features are informative for impact estimation, however we hypothesize that network features could further contribute to the performance.

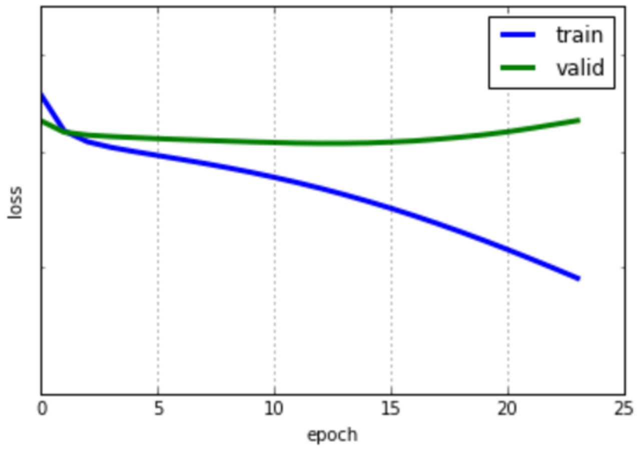


Figure 7: Cross entropy loss of the most accurate classifier

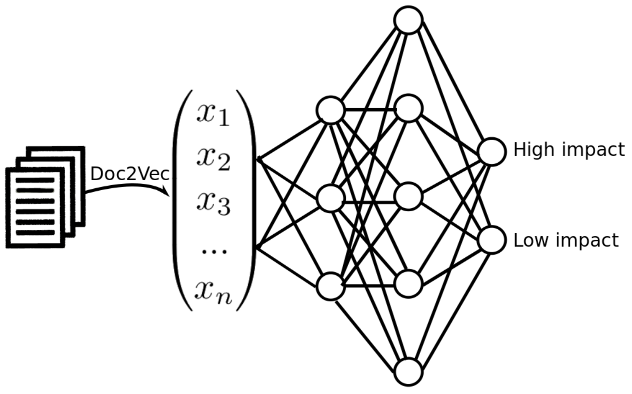


Figure 8: Citation Count Estimator

### Semantic Relatedness and Explanation

The capability of having words that explain the difference between two documents is a rather useful tool. For instance, in a recommendation scenario a user may be interested in documents that are similar to a document at hand but with a specific topical shift. By specifying another document, the user could retrieve the documents of interest. In a scenario where similar documents are to be compared (e.g. annual reports) such an algorithm can give a brief overview. In the context of EEXCESS such a tool may be used as a means to create a semantical path among a document collection.

Word and document embeddings have gained a lot of attention recently, because the tend to work well in text mining tasks. Yet, they elude the humans intuition. This experiment made the attempt to explain the arithmetic difference between two document embeddings by a series of word embeddings. This is possible, because Document and Word Vectors exist in the same space and have the vector length. Hence, mathematical operations are feasible. We developed an algorithm that iteratively picked words from a vocabulary in order to close the topical gap between the documents. Although not all words that were found were great matches, the algorithm is able to find sets of words that are reasonable to a human that read both documents. Remarkably, some of the well-explaining words are mentioned in neither documents.

## Algorithm

The intuition behind the idea is as follows. Assuming we are given two documents ($d_1$ and $d_2$) and their corresponding Document Vectors ($DocVec(d_1)$, $DocVec(d_2)$) as well as the Word Embedding model that was used to generate the Document Embeddings. Then, each document- and each word-vector represents a position in the vector space. Since we can do simple arithmetic operation, we can ask the question: Which Word Vector ($WordVec(w_a)$) maximizes $dist(DocVec(d_1), DocVec(d_2))$? Where $dist$ is some distance measure. Subsequently we can repeat this process by minimizing $dist(DocVec(d_1) + WordVec(w_a), DocVec(d_2))$. This can be repeated until a convergence criteria is satisfied. Finally we have a path that describes the topical distance between two documents. More formally, this algorithm can be described as follows:

**Algorithm 1** Document Vector Approximation

```
1:  function DocVecApprox($model, doc1, doc2$)
2:      path ← list()
3:      $X$ ← dv(doc1)
4:      $Y$ ← dv(doc2)
5:      $Y_{approx}$ ← X
6:      while not converged do
7:          $D \leftarrow Y - Y_{approx}$
8:          $W_c \leftarrow findClosestWord(D, model)$
9:          $W_f \leftarrow findFarthestWord(D, model)$
10:         if $W_c > -W_f$ then
11:             $Y_{approx} \leftarrow Y_{approx} + wv(W_c)$
12:             $append(W_c, path)$
13:         else
14:             $Y_{approx} \leftarrow Y_{approx} - wv(W_f)$
15:             $append(W_f, path)$
16:     return $path$
```

## Implementation and Results

The distance measure that we used in our experiments was cosine similarity (instead of minimizing the distance we maximized the similarity). We also experimented with cosine distance and euclidean distance but gained the best results with the cosine similarity.

We conducted experiments in a semi-automatic fashion, rather than fully-automatic. The reason for this is, that the Econstor16 vocabulary contained too many distorted words (like *atthe*, *a_y* and *yheo*) caused by formatting issues, equations and encoding problems. That was caused by failures during the extraction of the plaintext from the PDF files, which led to non-sense words in the vocabulary. The short term solution was to hand-pick the words that where added to the approximation path, rather than letting an algorithm decide. More precisely, the algorithm came up with a list of candidates and the first meaningful word was selected by hand. This effectively limited the amount of repetitions that could be undertaken. Nevertheless, during the semi-automatic testing, observations were made that will be exemplified using two examples. For the first example we selected two distant documents. The tables 7 and 8 give a brief overview over the content of the documents.

Table 7: Overview Document X, Example 1

| | |
|---|---|
| **Variable** | X |
| **Author** | Hendrik Hagedorn |
| **Title** | In search of the marginal entrepreneur: Benchmarking regulatory frameworks in their effect on entrepreneurship |
| **Keywords** | Benchmarking method, entrepreneurship, incentives, dataset, regulation |

Table 8: Overview Document Y, Example 1

| | |
|---|---|
| **Variable** | Y |
| **Author** | John Hartwick |
| **Title** | Mining Gold for the Currency during the Pax Romana |
| **Keywords** | Gold coinage, Roman money supply, roman empire |

Table 9: Approximation process of two distant documents. The similarity columns indicate the cosine similarity between Y and the current approximation. Note that the approximation was conducted in 100 dimensional space and then replicated in 600 dimensional space.

| **Iteration** | **Vector** | **Similarity @100** | **Similarity @600** |
|---|---|---|---|
| initial | diff = Y - X | -.4368 | -.0327 |
| 1 | diff = diff - "job" | -.1405 | .0044 |
| 2 | diff = diff - "carbon" | .1195 | .0276 |
| 3 | diff = diff + "empire" | .2241 | .1001 |
| 4 | diff = diff + "goldsmith" | .3647 | .0739 |
| 5 | diff = diff - "country" | .5181 | .0748 |
| 6 | diff = diff - "interest" | .6082 | .0745 |

Table 9 presents the results of the approximation algorithm. There are some notable aspects:

1. Although the initial similarity is low (-1 is the maximum value here which denotes a diametrically opposed vector) it takes only a few iterations to achieve a high similarity.
2. Most of the words are actually meaningful to explain the difference between the two documents: "empire" and "goldsmith" account for the document Y and "jobs" and "interest" are specific to document X.
3. There are also words that do not explain the content, like "carbon" and "country"
4. It's worth noting that, although it makes intuitively sense, the word "goldsmith" is not used in either documents.
5. When the results of the approximation in 100 dimensions are replicated in 600 dimensions, we find, that interation 3 produces the best result, and afterwards the process is stuck in a local optimum.

The setting for the second example differs in two aspects: (1) we used a 600 dimensional embedding rather than a 100 dimensional to drive the approximation and (2) we selected documents that were written by the same author, but on different topics. An overview over these documents is given in table 10 and 11. The respective approximation process is depicted in table 12. Again, we find suitable words that explain the difference well ("fuel", "diesel", "debt", "stock") and we find words that are

Table 10: Overview Document X, Example 2

| **Variable** | X |
|---|---|
| **Author** | Hans-Werner Sinn |
| **Title** | Pareto Optimality int the Extraction of Fossile Fuels and the Greenhouse Effect |
| **Keywords** | global warming, resource extraction, Pareto optimality |

Table 11: Overview Document Y, Example 2

| **Variable** | Y |
|---|---|
| **Author** | Hans-Werner Sinn |
| **Title** | EU Enlargement and the Future of the Welfare State |
| **Keywords** | EU expansion, migration, labour market, welfare state |

Table 12: Approximation process of two documents by the same author but on different topics. The similarity column indicates the cosine similarity between Y and the current approximation. Note that the approximation was conducted in 600 dimensional space and then replicated in 100 dimensional space.

| **Iteration** | **Vector** | **Similarity @600** | **Similarity @100** |
|---|---|---|---|
| initial | diff = Y - X | -.0061 | -.4368 |
| 1 | diff = diff - "stock" | .0419 | -.3045 |
| 2 | diff = diff + "industrial" | .1059 | -.2201 |
| 3 | diff = diff - "employee" | .1228 | -.2409 |
| 4 | diff = diff - "fuel" | .1544 | -.1977 |
| 5 | diff = diff - "diesel" | .1876 | -.2231 |
| 6 | diff = diff - "non-statistical" | .1996 | -.1769 |
| 7 | diff = diff + "debt" | .2098 | -.187 |

less suitable ("non-statistical", "industrial"). But in contrast to the first example, the approximation converges much slower. This is due to additional dimensions of the document embedding. Likewise the first example, we find a word that is suitable despite the fact that it's not mentioned in either documents ("diesel"). Moreover, the convergence in 100 dimensions is comparatively slow and does not increase monotonically.

## 7.3 Source Code and License

The source codes of the components described in this section are available via the following URLs:

- Blog Crawler (Blogs15 corpus) `http://purl.org/eexcess/components/research/blogcrawler`
- Blog Analyzer `http://purl.org/eexcess/components/research/bloganalyzer`
- Econstor16 corpus `https://github.com/n-witt/EconstorCorpus` created by the following two components:
  - Downloading Component `https://github.com/n-witt/EconstorCorpus/blob/master/Luke_the_Downloader/EconstorDownloader.ipynb`

  - Corpus Pre-Processor `https://github.com/n-witt/EconstorCorpus/tree/master/Han_the_Converter`

- Popularity Estimator `https://github.com/n-witt/econstorModelling/blob/master/classifier.ipynb`
- Vector Space Explorer `https://github.com/n-witt/econstorModelling/blob/master/ThePaperYouShouldWrite.ipynb`

This code is released under the conditions of the Apache 2.0[22] license. The API documentation can be found in the appendix on pages 66 and 68.

[22]http://www.apache.org/licenses/LICENSE-2.0

# 8 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 analysis concept with both these requirements in mind. For acquiring usage data we log user interactions and for the privacy-preserving analysis we rely on coarse-grained aggregated statistics, that hide the preferences of individual 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. The purpose and collected data were not subject to major changes since the last deliverable, but are necessary to get a holistic view. Hence, we repeat the respective sections from deliverable D5.3 [Sei+15] for completeness.

## 8.1 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.

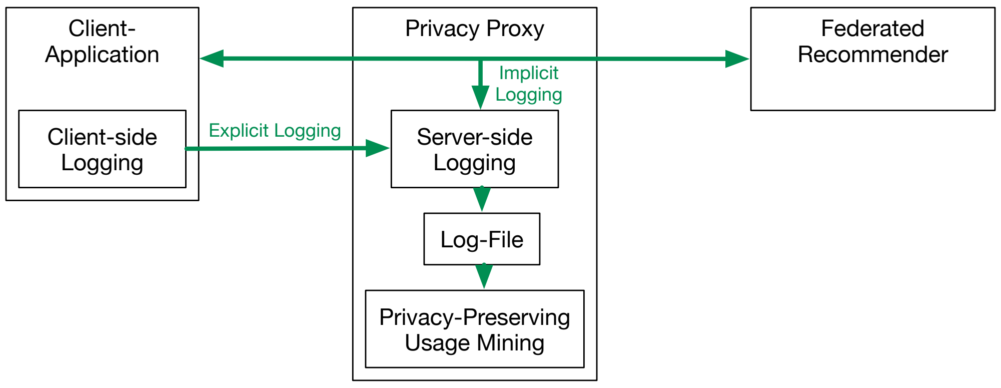


Figure 9: Client-side and server-side logging of usage data.

## 8.2 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 9). 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 9.

## 8.3 Privacy-Preserving Usage Analysis

This section describes the usage analysis 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 by computing aggregated statistics that support the different purposes mentioned above (Section 8.1).
These aggregated statistics are computed and regularly updated within the Privacy Proxy. The aggregation is performed across all users and covers queries, responses and the different interaction types. Due to the coarse granularity of the variables, we can make the statistics publicly available via a web-interface without disclosing any private information about users (Section 10). We report general statistics, content provider-specific statistics and client component-specific statistics. The general statistics include:

- Total number of unique users
- Total number of queries
- Total number of responses
- Total number of resource-specific queries
- Total number of resource-specific responses

Besides these numbers, we also report a ranked list of the Top-20 most frequent query terms aggregated over all users. In order to gain more insights into how the different partners contribute to these numbers, we report similar statistics for each content provider separately. These include:

- Name of content provider
- Number of responses
- Number of resource-specific responses
- Number of resources delivered to the client
- Number of resources delivered to the client after issuing a resource-specific query
- Number of interactions of a user with resources of this content provider (aggregated over all users, resources and interaction types)

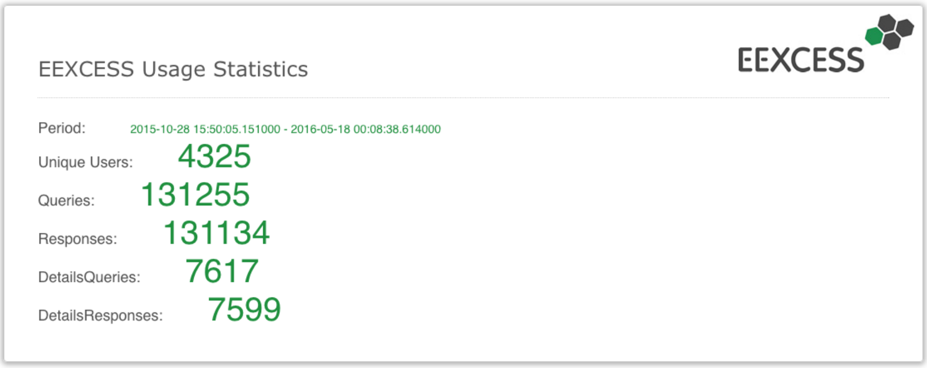
In order to assess the reach of the EEXCESS framework and the usage of individual modules implemented in different client applications, we report client component-specific statistics:

- Name of the client component
- Number of unique users who used the client component
- A list of modules implemented in the client component. Each entry in the list contains:
  - Name of the module
  - Usage of the module; reported as the number of queries and user interactions performed within the module

## 9 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-package[23]. A detailed description and the source code documentation of the module is given in the Appendix on page 50. The server-side logging component is described in deliverable D6.4 [Mok+16], section 6.2.2.

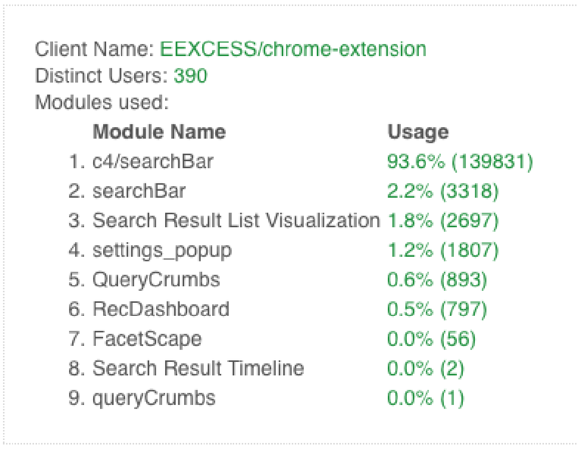
[23] `http://www.purl.org/eexcess/components/c4`



EEXCESS Usage Statistics

| | |
|---|---|
| Period: | 2015-10-28 15:50:05.151000 - 2016-05-18 00:08:38.614000 |
| Unique Users: | 4325 |
| Queries: | 131255 |
| Responses: | 131134 |
| DetailsQueries: | 7617 |
| DetailsResponses: | 7599 |

Figure 10: Screenshot of the usage analysis web-interface showing general statistics of the EEXCESS framework.



Client Name: EEXCESS/chrome-extension
Distinct Users: 390
Modules used:

| Module Name | Usage |
|---|---|
| 1. c4/searchBar | 93.6% (139831) |
| 2. searchBar | 2.2% (3318) |
| 3. Search Result List Visualization | 1.8% (2697) |
| 4. settings_popup | 1.2% (1807) |
| 5. QueryCrumbs | 0.6% (893) |
| 6. RecDashboard | 0.5% (797) |
| 7. FacetScape | 0.0% (56) |
| 8. Search Result Timeline | 0.0% (2) |
| 9. queryCrumbs | 0.0% (1) |

Figure 11: Screenshot of the usage analysis web-interface showing client component-specific statistics of the EEXCESS framework. `Usage` is an aggregation of all queries issued and all interactions performed from within the module. (Module names are not canonical. The same module may appear more than once in the list if it was renamed, e.g. `searchBar` and `c4/searchBar`.)

## 10 Privacy-Preserving Usage Analysis Component

With the collaboration of INSA, we developed and implemented the usage analysis component as a web-service that resides on the Privacy Proxy. It consists of a usage analysis component as backend and a publicly accessible web interface as frontend. The backend component crawls the log-files on a daily basis and updates the aggregated statistics, which are also stored on the Privacy Proxy. The interface is available at `http://eexcess.joanneum.at:4444/eexcess-usage/`, see also Figure 10 and Figure 11.

# 11 Summary

In this deliverable we presented the final status of the user and usage mining prototypes in the frame of EEXCESS. 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 scientific papers. For the analysis of project-internal resources, a web-interface has been developed, which uses the logged data to provide analysis statistics. All of the prototypes have reached a stable state and the current development plan foresees evaluation and bugfixing.

## 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.

## 12 References

[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.

[HS06] Geoffrey E Hinton and Ruslan R Salakhutdinov. "Reducing the dimensionality of data with neural networks". In: *Science* 313.5786 (2006), pp. 504–507.

[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.

[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.

[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.

[Hin+12] Geoffrey E. Hinton et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). URL: http://arxiv.org/abs/1207.0580.

[Mik+13] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *CoRR* abs/1301.3781 (2013).

[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.

[LM14] Quoc V. Le and Tomas Mikolov. "Distributed Representations of Sentences and Documents". In: *Proc. of the 31th Int. Conf. on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. 2014, pp. 1188–1196. URL: http://jmlr.org/proceedings/papers/v32/le14.html.

[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.

[DOL15] Andrew M Dai, Christopher Olah, and Quoc V Le. "Document embedding with paragraph vectors". In: *arXiv preprint arXiv:1507.07998* (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+15] Christin Seifert et al. *D5.3 – Second Prototype on User Profile and Context Detection, Usage Analysis Methods and Services*. Tech. rep. University of Passau, 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.

[BSM16] Christoph Besel, Jörg Schlötterer, and Granizer Michael. "Inferring Semantic Interest Profiles from Twitter Followees". In: *Proceedings of the 31th Annual ACM Symposium on Applied Computing*. SAC '16. New York, NY, USA: ACM, 2016. DOI: 10.1145/2851613.2851819.

[Dop16a] Gerhard Doppler. *D7.5 – Second Evaluation Report Test Beds*. Tech. rep. BITM, 2016.

[Dop16b] Gerhard Doppler. *D7.6 – Final Prototype Integration and Deployment*. Tech. rep. BITM, 2016.

[JGS16] Johannes Jurgovsky, Michael Granitzer, and Christin Seifert. "Evaluating Memory Efficiency and Robustness of Word Embeddings". In: *Advances in Information Retrieval, ECIR 2016, Padova, Italy*. Springer International Publishing, 2016, pp. 200–211.

[Mok+16] Sonia Ben Mokhtar et al. *D6.4 – Final Security Proxy Prototype and Reputation Protocols*. Tech. rep. INSA, 2016.

[ZSG16a] Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. "DoSeR - A Knowledge-Base-Agnostic Framework for Disambiguating Entities Using Semantic Embeddings". In: *The Semantic Web. Latest Advances and New Domains - 13th European Semantic Web Conference, ESWC 2016, Heraklion, Kreta, to appear*. 2016.

[ZSG16b] Stefan Zwicklbauer, Christin Seifert, and Michael Granitzer. “Robust and Collective Entity Disambiguation through Semantic Embeddings”. In: *Proceeding of the 39rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2016, Pisa, Italy, to appear*. 2016.

# 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 46
- Word2Vec Rest Server to provide word2vec and doc2vec similarities.
  ⇒ on page 48
- Client-side library for context detection and wrappers for logging and entity disambiguation (Cultural and sCientific Content in Context – **C4**), currently used in the Chrome extension, the Moodle plugin, and the Wordpress plugin.
  ⇒ on page 50
- 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 63
- Blog Crawler component for crawling and storing blog posts.
  ⇒ on page 66
- Blog Analyser for linking Blogs to EEXCESS.
  ⇒ on page 68

# 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/qbsbp9zfpc7h8sx/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 \"Annabella\" 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 after 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 v
iew, but the worse from a performance point of view (2 or 3 are acceptable va
lues).
  var query = JSON.parse('{"origin": {"userID": "E993A29B-A063-426D-896E-131
F85193EB7", "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-131F85193EB
7", "clientType": "EEXCESS - Google Chrome Extension", "clientVersion": "2bet
a", "module": "testing"},
          documentBadge: [
              {
                  id: "/09003/4A65C4999F4077781A1F9CF2510EE512CD6571B9",
                  uri: "http://europeana.eu/resolve/record/09003/4A65C4999F40
77781A1F9CF2510EE512CD6571B9",
                  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();
    paragrahps.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 navi
gation 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, ei
  ther 'success' or 'error'
    console.log(response.data); // should contain the results on succes
  s 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 `contextKeywors` 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 ent
ities/categories
        console.log(this.topic); // contains the main topic entity
        console.log(this.statistic); // contains the extracted entities/cate
gories
        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, locatio
n, organization, misc)
          console.log(this.value); // number of occurences of the entity in
the paragraph
        });
      });
    } else {
      // an error occured, 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.dat
a.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.dat
a);
```

```
                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 specifiy 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-circ
le",
      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

#### queryTriggerd

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 optionaly 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 optionaly 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-ge`. 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 trigged 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.