Faceted Token-Based Exploration of Large Corpora Using Wikidata

Studienarbeit von Hanish Goel
May 25, 2016

TECHNISCHE
UNIVERSITÄT
DARMSTADT

UBIQUITOUS
KNOWLEDGE
PROCESSING

Faceted Token-Based Exploration of Large Corpora Using Wikidata

vorgelegte Studienarbeit von Hanish Goel

Supervisor: Prof. Dr. Iryna Gurevych
Coordinators: Dr. Hatem Mousselly-Sergieh, Mohamed Khemakhem

Tag der Einreichung:

Erklärung zur Studienarbeit

Hiermit versichere ich, die vorliegende Studienarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den May 25, 2016

(Hanish Goel)

Abstract

The huge increase in the amount of available documents on the World Wide Web makes finding relevant ones a challenging task. The quality of results that traditional full-text search engines deliver is still not optimal for many types of user queries. Because the current practice in information retrieval mostly relies on keyword-based search over full text data, which is modeled with bag-of-words. Such a model misses the actual semantic and factual information in text. To deal with this issue, knowledge bases are proposed, which are nowadays the backbone of many search web applications.

Knowledge bases form the basic infrastructure for the Semantic Search. KB provides a sophisticated representation of world knowledge on the semantic level, i.e., they contain semantic entities and their relations instead of simple words. Moreover, they store well-known facts about a knowledge domain. This ability of KB allows the identification of the validity context of specific relations. For example, while in the context "Neil Armstrong educated at University of Southern California" the relation "Neil Armstrong – University of Southern California" is valid where as it does not hold in general. Both the information extraction and retrieval processes can benefit from such knowledge, which gives semantics to plain text.

In this project, we developed an application that uses Wikidata to enrich text corpus with factual knowledge. The application enhances the search term with wikidata item and property-value pair to find more precise and relevant results. Once results are returned and evaluated, the same wikidata item and property-value pair is used to annotate the text and the annotated text is stored in separate database. The annotated terms that gives information about facts and relations between those facts in the enriched version of text corpus can be used by search applications for building their semantic index.

Contents

1 Motivation **4**

2 Background Information **5**

2.1 Information Retrieval 5

2.2 Knowledge Base 6

2.3 Text Corpus 7

2.4 Related Work 7

3 Our Solution **8**

4 Possible Applications **11**

5 Implementation **13**

5.1 Architecture of the system 13

5.2 Use Case View 15

5.3 Tools used for the implementation 16

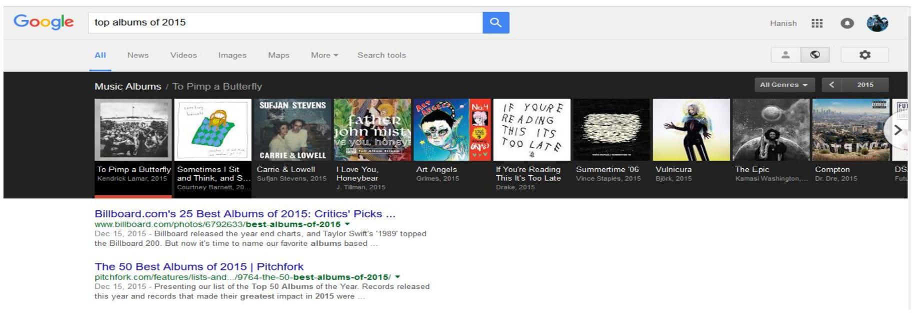
6 Conclusion and Future Work **18**

List of Figures **19**

Bibliography **20**

1 Motivation

A common problem that has been around for a long time is that Information Retrieval systems do not provide accurate and relevant results based on the keywords being searched for. Because by just matching query to keywords, the system misses those documents that have the potential to be more relevant to the user's intent. With the enormous amount of structured and unstructured information emerging on the web, the gap between vocabularies used in indexed documents and user queries has been increased. And the user is also moving from former queries like "restaurants in Frankfurt", to more specific queries such as "where to eat Indian food in Frankfurt". Therefore, there is a growing need to consider factual information and understand search intent to deliver more relevant results. For example, when user searches for "top albums of 2015", Google shows results (see Figure 1.1) based on their knowledge graph. The Knowledge Graph is a knowledge base used by Google to enhance its search engine's search results with semantic information gathered from a wide variety of sources.



Billboard.com's 25 Best Albums of 2015: Critics' Picks ...
www.billboard.com/photos/6792633/**best-albums-of-2015**
Dec 15, 2015 - Billboard released the year end charts, and Taylor Swift's '1989' topped the Billboard 200. But now it's time to name our favorite **albums** based ...

The 50 Best Albums of 2015 | Pitchfork
pitchfork.com/features/lists-and.../9764-the-50-**best-albums-of-2015**/
Dec 15, 2015 - Presenting our list of the **Top** 50 **Albums** of the Year. Records released this year and records that made their **greatest** impact in **2015** were ...

Figure 1.1: Search based on Knowledge Graph

The above example suggests that if we create a factually enriched corpus of music albums where albums are annotated with facts such as "user rating", "year of release" and so on, the search system can provide better results and informative summaries to its users. The current day IR systems which look for enriched corpus in order to provide better information exploration and retrieval services is the motivation for this endeavor.

2 Background Information

The purpose of this chapter is to provide related work and some background information about the terms and technologies used throughout this project. We begin with the core discipline of this project, i.e. the Information Retrieval. Then, we continue with the most popular knowledge bases such as DBpedia, Wikidata, and then related work.

2.1 Information Retrieval

Information retrieval (IR) is one of the oldest research areas in the information science. The goal of the IR discipline is to search and retrieve the most relevant documents to the information needs of the user. Therefore, a good IR system should retrieve only those documents that satisfy the user needs, not a bunch of unnecessary data.

General IR Process

Information retrieval systems have been evolved and improved a lot since their first emergence in 1950s. However, the core process hardly changes. The important aspects of the IR process are described below.

- **User Interface**: The user interface is one of the most important aspects in IR. The design of the user interface brings the tradeoff between user-friendliness and performance. Simple and relaxed interfaces are easier to use at the cost of resulting ambiguous queries. Complex and powerful interfaces, however, provide more detailed and precise query formulation, while they are cumbersome and time-consuming for the end-user.

- **Query Processor**: The raw query submitted by the user should be processed before searching. Usually, the query is transformed into an internal form that the system can interpret. Furthermore, several processing tasks can be involved such as stopword elimination, stemming and other application specific tasks. These tasks should also be done in the indexing phase for consistent matching.

- **Indexing**: Indexing is an essential part of the IR systems for two reasons. First, it optimizes the query performance and improves the respond times considerably by storing terms in an inverted file structure. Basically, it stores the text positions for occurrences of each term. Secondly, a number of processing tasks are carried out during the indexing phase similar to the query processing phase, which further improves the performance. In our project, we use state of the art technology Solr[1] for indexing text documents.

- **Searching**: In this phase, the query terms are searched against the inverted index. All the documents that contain the occurrences of the query terms are retrieved. Depending on the application, the retrieval can be done even for the partially matched documents.

- **Ranking**: The documents retrieved in the previous step are given scores according to the matching quality between the query terms and the documents. The documents are sorted according to this score, so that the most relevant documents are presented to the user on top of the retrieval list. The raking of documents in our application is done by Solr itself.

[1] http://lucene.apache.org/solr/

- **IR Models**: We want to mention briefly about the IR model that is used in Apache Lucene[2], an open-source indexer and searcher. Our application uses Solr[1] which is built on Apache Lucene. Lucene uses a combination of VSM and Boolean model for the retrieval. Basically, VSM is used to score the relevant documents and Boolean model is used to select the matching documents according to the user query, so it is essentially a VSM with Boolean capabilities.

2.2 Knowledge Base

In this section, we discuss three most popular knowledge bases. A knowledge base (KB) is a system that stores complex structured and unstructured information about the world. The term "knowledge-base" was coined around 1970s to distinguish this form of knowledge store from the more common and widely-used term database.

DBpedia

The DBpedia project[6] was started in 2006 by people at the Free University of Berlin and Leipzig University. It extracts factual information from Wikipedia and makes it freely available using Semantic Web and Linked Data standards. The extracted knowledge, comprising more than 1.8 billion facts, is structured according to an ontology maintained by the community [6]. The knowledge is obtained from different Wikipedia language editions, thus covering more than 100 languages, and mapped to the community ontology. Data is accessed using an SQL-like query language for RDF called SPARQL. DBpedia is popular because of its support for RDF standards, simple naming conventions, and straightforward vocabulary. However, applications that use DBpedia often encounter problems with more complex queries and data quality.

Freebase

Freebase [7] is a database system designed to be a public repository of the world's knowledge. It was announced in 2007 by Metaweb and was inspired by broadly used information communities such as The Semantic Web and Wikipedia. It was an online collection of structured data gathered from many sources, including individual, user-submitted wiki contributions. Freebase data was available for commercial and non-commercial use under a Creative Commons Attribution License. Currently, Freebase website and API has been shutdown.

Wikidata

Wikidata [5] is a free, collaborative, multilingual knowledge base operated by the Wikimedia Foundation. Wikimedia launched Wikidata in October 2012. Wikidata currently contains 17,521,379 items and is continuously growing in both size and quality. Following are the key insights of Wikidata.

- **Free**: The data in Wikidata is published under the Creative Commons license, allowing anyone to reuse and republish the data in any way. The data is available free of charge and without any conditions or requirements.

- **Multilingual**: Editing, consuming, browsing, and reusing the data is fully multilingual. Data entered in any language is available immediately in all other available languages.

[2] https://lucene.apache.org/core/

- **A secondary database**: Wikidata not only stores statements, it can also store their sources. It can deal with contradictory statements, including their sources, and thus reflect the diversity of knowledge in the sources.
- **Support for external applications**: Everyone can use Wikidata for many different services and applications. External applications can easily access the data using standard formats such as JSON and RDF.

The above characteristics of Wikidata make it a suitable candidate for our application.

Data model of Wikidata: The wikidata is a collection of items and statements about those items. Items are basically topics which are linked to Wikipedia articles. Factual Information is added to items by creating statements which is just a property-value pair. For example, Figure 2.1 shows the item "London" has a property "population" with value "8173900".

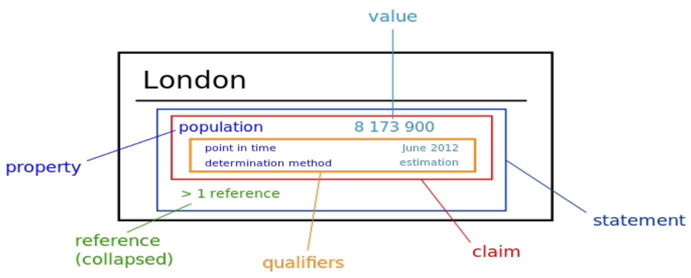


Figure 2.1: Wikidata Data Model

2.3 Text Corpus

A corpus (plural corpora) or text corpus is a large and structured set of texts that is usually electronically stored and processed. Our application uses Wikipedia database[3] for text corpus. Originally conceived in 2001, Wikipedia is a free, open content online encyclopedia created through the collaborative effort of a community of users. It collected well over five million articles in English and more than that number in all other languages combined. Wikipedia has become a resource of enormous value, with potential applications across all areas of science, technology, and culture[5].

2.4 Related Work

There has been lot of research done in the area of query expansion techniques to improve the information retrieval. The main idea is expanding both the indices and queries with the synonyms and semantic of the words to achieve better recall and precision. Many query expansion methods such as dictionary-based query expansion [9] and knowledge-based [8] query expansion, have been studied. The query expansion task can be defined by semantic enrichment of a query with its semantically related concepts. With these complementary concepts, additional relevant documents, which may not contain the keywords provided in that query, can be retrieved. For instance, a given query "Hiroshima" can retrieve documents where the keyword Hiroshima directly appears but not the documents that only contain related concepts such as atomic bomb, Nagasaki or Etajima. The semantic enrichment of a query can be achieved by using the knowledge systems such as Wikipedia or DBpedia [2]. However, these techniques have not been successful in improving search engines.

[3] https://en.wikipedia.org/wiki/Wikipedia:Database_download

3 Our Solution

Our solution to improve information exploration and retrieval is enriching text corpus with lexical knowledge resources such as Wikidata. As discussed in section 2.2, Wikidata provides world knowledge as a collection of items and statements about those items. Within the scope of this project, we have developed an application to manually annotate text documents with the factual information from Wikidata. The following sections describe the important aspects of the approach starting with a summary of the overall project.

The application indexes sentences extracted from Wikipedia corpus and provide user interface for browsing and annotating sentences with Wikidata. Users can search for specific keywords and obtain a result set that shows the matching items in the corpus. User can apply different kinds of filtering based on Wikidata properties or items and evaluate the relevance of result sentences. Finally, the user will be able to extract annotated sentences corresponding to his/her initial query and the applied filtering.

Extracting Corpus Data

We use Wikipedia database for text corpus in our project. We turn the contents of Wikipedia into a set of sentences. The Wikipedia dump file[1] is available in XML format. The XML file contains lot of irrelevant information such as the Wikimedia markup of the articles, formatting information, etc. So, first we cleaned and removed such information from each of the article text before extracting sentences from it.

To process the XML file, we use the gwtwiki toolkit (bliki-core-3.0.16.jar)[2]. There is a wide range of toolkits available on the web for processing Wikipedia content in different languages of varying quality. But gwtwiki appears to be one of the most functional and robust, handling both the parsing of the XML file and converting each article from markup into a plain text format.

By invoking the gwtwiki parser and doing text tokenizing, we obtain a set of good sentences. Here are the first few sentences from the Anarchism article:

> "*Anarchism is a political philosophy which considers the state undesirable, unnecessary, and harmful, and instead promotes a stateless society, or anarchy*."
>
> "*The Concise Oxford Dictionary of Politics*."
>
> "*It seeks to diminish or even abolish authority in the conduct of human relations*."

Currently, our application supports text corpus in English language. The next section discusses how these sentences are indexed.

Indexing Sentences

Given the necessity to use a very large corpus to separate noise from relevant context correlations, Solr[3] as a platform for indexing is used. Solr is able to achieve fast search responses because, instead of searching the text directly, it searches an inverted index instead. Inverted index inverts a page-centric

[1] https://dumps.wikimedia.org/enwiki/latest/
[2] https://code.google.com/archive/p/gwtwiki/
[3] http://lucene.apache.org/solr/

data structure (page->words) to a keyword-centric data structure (word->pages). We constructed a Solr index such that each entry represents a sentence from Wikipedia article. Each sentence has its own metadata associated with it, such as page id, page title, etc. That information is also included with each sentence. For example, Table 3.1 shows the fields and values of index structure. We use SolrJ API[4] for interacting and indexing sentences on Solr Server.

Field	**Value**
id	109589119093376
page_id	109589
page_title	Golf, Florida
rev_id	10539714
rev_text_id	10539714
text	Golf was founded in the 1950s as a planned community on golf course

Table 3.1: Structure of Indexed Corpus

Analyzers and Tokenizers When a document is indexed, its individual fields are subject to analyzing and tokenizing filters that can transform and normalize the data in the fields. For example, removing blank spaces, stemming and removing stop words from the document. Doing so can increase recall, for example, "ram", "Ram" and "RAM" would all match a query for "ram". We configured field analyzers and tokenizer filters(such as StandardTokenizerFactory, EnglishMinimalStemFilterFactory) in the Solr Schema file. The ranking of documents is done by Solr by default.

User Interface

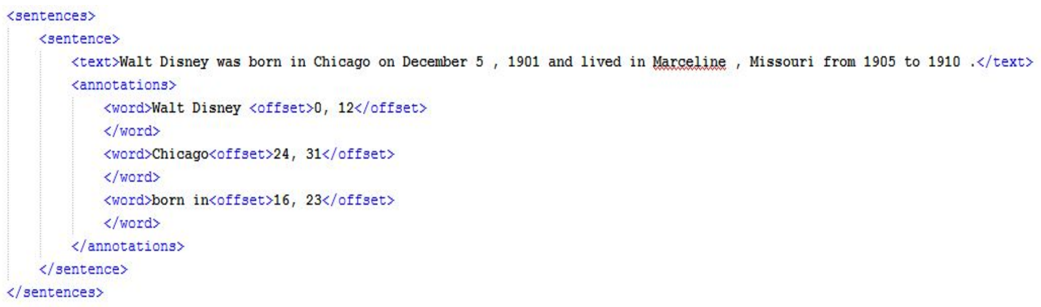
We developed user interface for querying and browsing sentences from the indexed corpus. The interface provides user the ability to apply filtering to search results based on items in wikidata, and to evaluate the result sentences based on their relevance to the search criteria. The system highlights the selected items in the result sentences. The Wikidata items are displayed in tree structure that eases the visualization and selection. The System poses some restrictions on selection of wikidata items and evaluation of sentences.

- The system restricts the user to only select one triple of item, property, value from Wikidata tree.
- On Selection of child node, all parent nodes will be selected automatically.
- Once a sentence is evaluated by the user, it will not be further shown in the search results.

Annotated Corpus

On user interface, when user evaluates a sentence, the sentence is stored in database along with the offsets of words (selected item and property-value pair) in the sentence. Figure 3.1 shows the structure of the annotated sentence that is saved in the database. User can export the annotated corpus in XML format from UI.

[4] https://wiki.apache.org/solr/Solrj



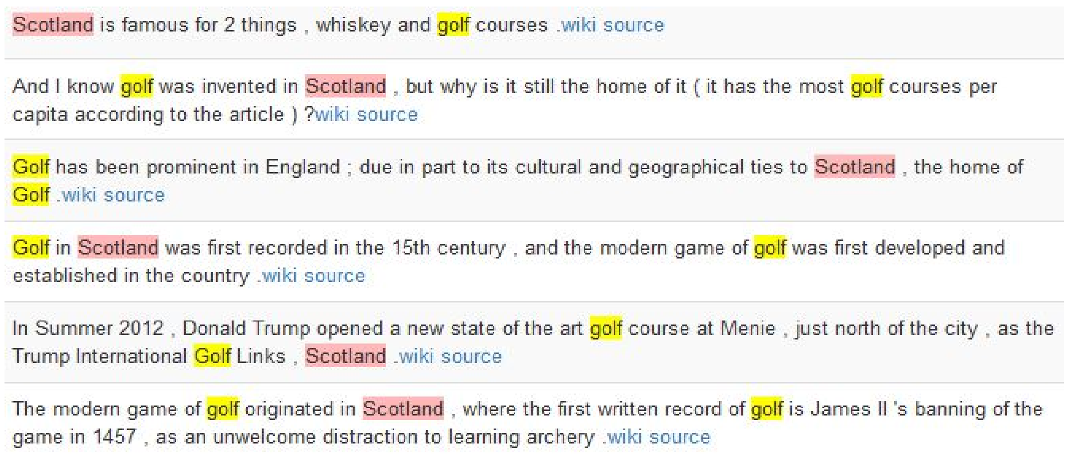
```
<sentences>
    <sentence>
        <text>Walt Disney was born in Chicago on December 5 , 1901 and lived in Marceline , Missouri from 1905 to 1910 .</text>
        <annotations>
            <word>Walt Disney <offset>0, 12</offset>
            </word>
            <word>Chicago<offset>24, 31</offset>
            </word>
            <word>born in<offset>16, 23</offset>
            </word>
        </annotations>
    </sentence>
</sentences>
```

Figure 3.1: Structure of Annotated Sentence

4 Possible Applications

This section discusses two of the main possible uses of this application.

Extracting Wikidata annotated text: NLP researchers can use this application to create a training set of annotated text about a given event (e.g. world cup), a person, or a field or research e.g. NLP. To achieve this task, the user can query the indexed corpus to obtain a collection of documents matching his query. For each document of the result set, relevant terms are highlighted. Furthermore, the collection of Wikidata properties associated with that terms are shown. For a given term, the user can choose one of these properties and the target terms that are linked to the source terms (via the selected property) are highlighted. For example, if the user is interested in labeling terms that are connected to "golf" and "Scotland", he can select the item "golf" and a property whose value is "Scotland" from the property list. Figure 4.1 shows selected items are highlighted in the result text. Finally, the user can extract the annotated sentences that meet his needs and use them to build his training corpus.



Scotland is famous for 2 things , whiskey and golf courses .wiki source

And I know golf was invented in Scotland , but why is it still the home of it (it has the most golf courses per capita according to the article) ?wiki source

Golf has been prominent in England ; due in part to its cultural and geographical ties to Scotland , the home of Golf .wiki source

Golf in Scotland was first recorded in the 15th century , and the modern game of golf was first developed and established in the country .wiki source

In Summer 2012 , Donald Trump opened a new state of the art golf course at Menie , just north of the city , as the Trump International Golf Links , Scotland .wiki source

The modern game of golf originated in Scotland , where the first written record of golf is James II 's banning of the game in 1457 , as an unwelcome distraction to learning archery .wiki source

Figure 4.1: Set of annotated sentences that matches the search pattern "golf" and "Scotland"

Collecting Textual Evidences: Our application provides interface for collecting textual evidence for the facts about an entity. Retrieving textual evidence for the facts about an entity is important for the following reasons[10].

- Evidence for fact credibility: By collecting many occurrences of SPO statements in news, social media and other web contents, a user can more easily assess the statement's truth.
- Statistics for semantic search and analytic: Entity search over heterogeneous data often needs to rank results. To this end, it is important to have co-occurrence frequencies for SPO components.
- Multidocument summarization: By knowing which SPO facts are central in a document or an entire corpus of documents, we can automatically generate more informative summaries. The corpus may be a daily news or a month's scholarly publications on a scientist's field of interest.

Knowledge bases (KBs), like wikidata, DBpedia contain hundred millions of facts about entities, in the form of subject-predicate-object (SPO) triples. But for assessing the correctness of a fact, it is important to see the fact in context, by looking at a variety of news, biographies, or postings in online communities[10]. For example, to realize that the SPO statements "NeilArmstrong bornIn Chicago" and

"AlbertEinstein diedOn 25-Aug-2012" are wrong, one needs to explore texts about these claims. Our tool makes this task easier by providing interface that retrieves text documents containing supporting evidence for a given SPO triple.

5 Implementation

In this chapter, we briefly mention about an architecture and use case of the application we implemented. Then, we describe the tools that we used for the implementation.

5.1 Architecture of the system

Figure 5.1 provides the high level details of the system architecture. The system is a JEE based application developed in java programming and deployed on an application server Apache Tomcat[1]. Eclipse Java Development Tool (JDT)[2] is used for the development of the application.

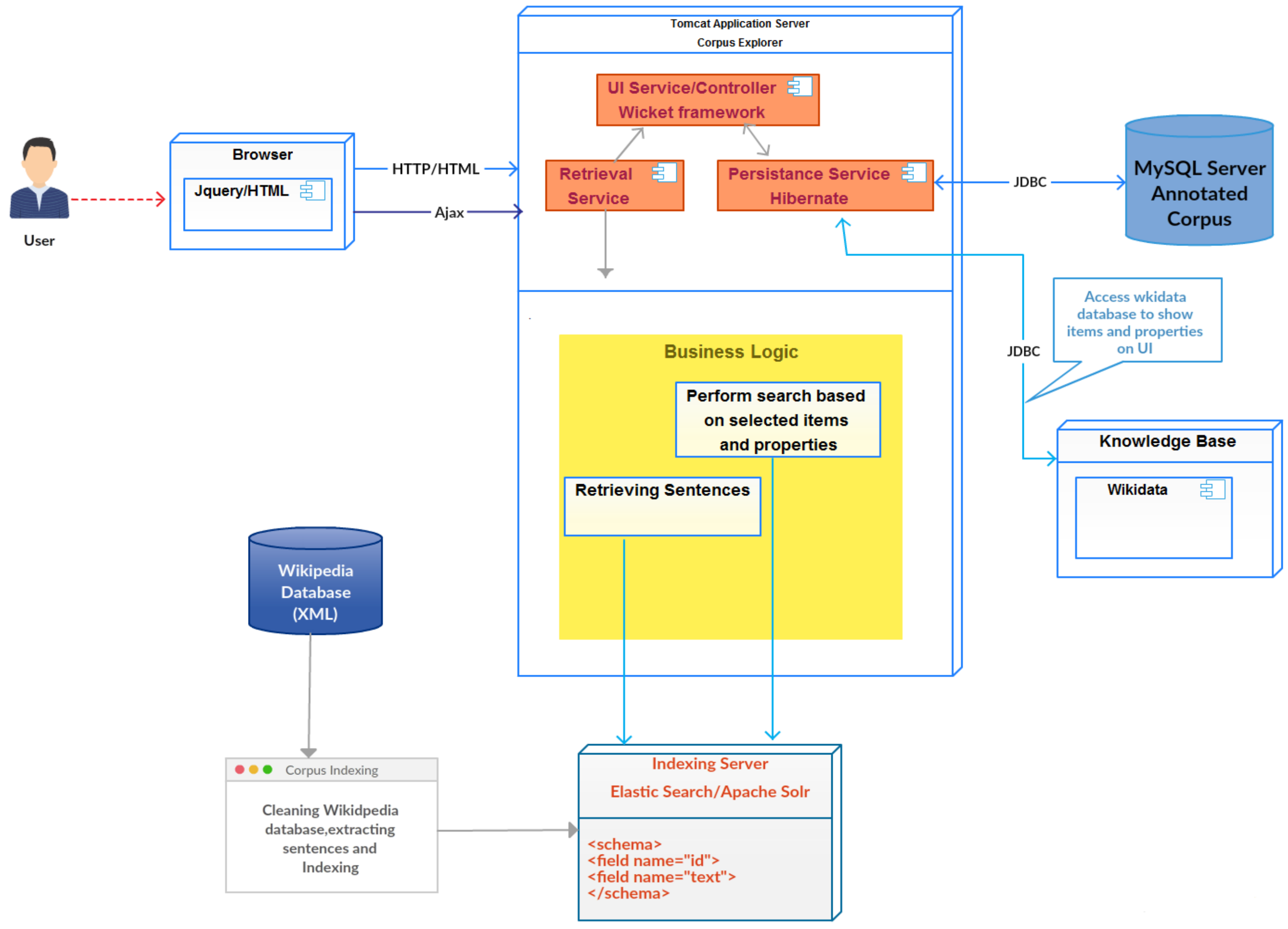


Figure 5.1: Architecture diagram of the system

The application has two modules: *Corpus Explorer* that allows user to browse and evaluate sentences from the corpus, and *Corpus Indexing* that extracts sentences from Wikipedia database and index them on the Solr server.

[1] http://tomcat.apache.org/
[2] http://www.eclipse.org/

Corpus Indexing

This module processes the wikipedia XML database by using gwtwiki toolkit (bliki-core-3.0.16.jar) and indexes the extracted sentences on Solr. We created a Solr core named "wikipedia". Currently, we have more than 6 million sentences indexed on the solr core.

Corpus explorer

The corpus explorer module provides user interface for searching, browsing and evaluating sentences from the indexed corpus. It has the following layers that explain how each component in corpus explorer interacts.

User Interface Layer: This layer represents the front end of the application, and contains the actual GUI elements that users view and click. The user interface is based on a component based Wicket framework[3], hosted on the Apache Tomcat application server.

Wicket provides container application that supplies the web components of the user interface. The container application manages and renders each of the GUI elements that it contains. Each GUI element inside the container application is a self-contained object that can retrieve data for display. Some of the GUI elements implements Ajax to manage communication with the server.

The user interface layer contains components such as tree, data grid, etc. CSS-based design templates of Bootstrap library[4] are applied to forms, buttons and other interface components. The interaction between client browser and server takes place by using Ajax and HTTP requests. This layer interacts with other layers such was DAO layer for Wikidata and user specific data, and Solr for retrieving text sentences.

Data Access Layer: A data access layer provides simplified access to data stored in relational database. By mapping application calls to the persistence layer, data access objects (DAO) provide some specific data operations without exposing details of the database. It separates all the low level data access operations from high level business services. In Java programming language, Data Access Objects can be implemented by using frameworks or commercial products. In this project, we currently use Hibernate[5] as our persistence provider and use its EntityManager. For relational database, we are using MySQL for storing Wikidata, user specific data and annotated text.

Solr Layer: This layer interacts with the Solr server for retrieving the text sentences based on the search criteria. The interaction with Solr server takes place by using SolrJ API. SolrJ is a java client to access Solr. It offers a java interface to add, update, and query the Solr index.

List of System Features

Below is the list of the major system features and functions that is available in this application.

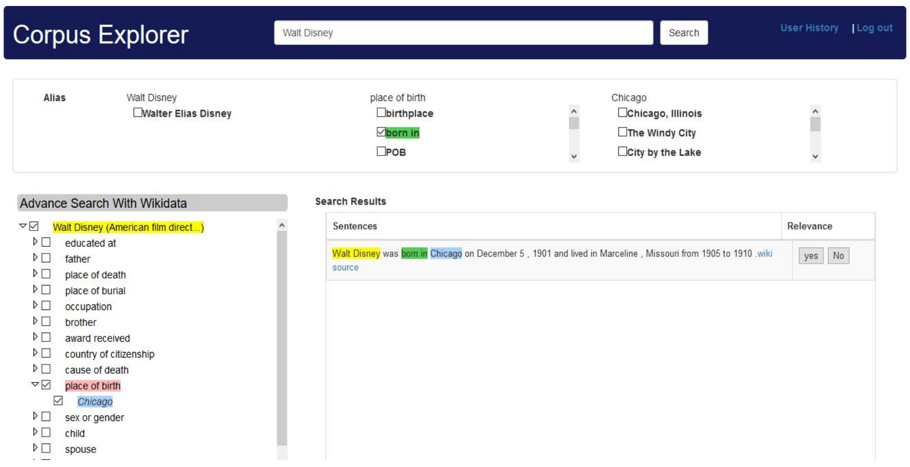
[3] http://wicket.apache.org/
[4] http://getbootstrap.com/
[5] http://hibernate.org/orm/

Area	Functions/Features
Authentication	User login page is provided for maintaining user specific data.
Authentication	Manage User account page is provided and is only accessed by admin account.
UI	When user search for some entity, search result grid displays all the sentences matching the given input. A group of yes and no button will be given to user to evaluate the sentences. On the left side of the page matching Wikidata items will be shown with short description and checkbox based option for selecting them.
UI	Link to Wikipedia article is provided after each sentence that it belongs to.
UI/Search filter	User can filter out the results by selecting a Wikidata item and property value pair. The tool restricts the user to select only one triple of item property and value. When user selects a child node, its parent node will be automatically selected.
UI/Wikidata Properties	The system excludes some of the properties which are not interesting to the user, such as instance of, category. Only the clean and relevant list of properties is shown to the user.
UI/Alias	When user selects item, property or value, the page will display the list of aliases for each of the selected values. The user can select one or more specific aliases for each item to enhance the filter.
UI/Highlight	Once the selected items and aliases are submitted by the user, the system returns the sentences matching the new search criteria and highlights the matching words in the sentences.
UI/Evaluation	The user has the ability to evaluate the relevance of the returned sentences by clicking "yes/no" button.
UI/Export	User history popup is provided that shows the total number of sentences annotated by the user and the last search term. It provides user the ability to download the dump of annotated sentences.

Table 5.1: List of Major System Features

5.2 Use Case View

This section describes the typical use case of the application. The following steps show how user can perform the search and obtain a set of annotated sentences.



Sentences	Relevance
Walt Disney was born in Chicago on December 5 , 1901 and lived in Marceline , Missouri from 1905 to 1910 .wiki source	yes No

Figure 5.2: Usecase of Corpus Explorer

1. The user starts by entering a target word in search box. For example, Figure 5.2 shows the user enters the search term "Walt Disney".
2. The system displays all the items in Wikidata corresponding to the target word with brief description and properties on the left side. The user can select properties as well as their values to construct a target pattern. In Figure 5.2, the target pattern contains 3 key words
   - Walt Disney
   - place of birth
   - Chicago
3. It will be possible to select alias for each word in the pattern to expand more on the query. The system displays list of alias for each of the selected item, property and value. For example, the user selects "born in" as alias for place of birth in Figure 5.2 and the pattern becomes:
   - Walt Disney
   - place of birth| born in
   - Chicago
4. The system returns the sentences matching all the keywords in the expanded query based on the prepared index of the corpus. The user interface highlights the matching words in the sentences and their corresponding Wikidata items.
5. The user can assess the relevance of the returned sentences by answering "yes/no" buttons. When user answers yes, the sentence is stored in database in XML format with annotations. However, if the sentence does not make sense to user for a given pattern, the user can exclude the extraction of that sentence from the corpus by answering no.
6. The user can see his past activity and download the annotated corpus from the user history link provided on top right of the page.

5.3 Tools used for the implementation

Implementation of any application is always preceded by important decisions regarding selection of the platform, the language and frameworks used, etc. These decisions are often influenced by various factors such as the real environment in which the system works, the efficiency that is required, the security concerns, other implementation specific details etc. This section covers the development tools that are used in our project.

Item	Selection
Language	Java 8
Source code repository	SVN
Source code build	Maven 2.1
IDE	Eclipse 4.4
Application Server	Tomcat 8
Object Relational Mapping	Hibernate 4.2.15
Database	MySQL, H2
Web development	JavaScript, Wicket Web Framework, JQuery, Bootstrap
Other libraries	gwtwiki toolkit (bliki-core-3.0.16.jar)

Table 5.2: Matrix of development tools

The following discusses some of the technologies mentioned in Table 5.2 with brief description.

Apache Maven: Maven tool[6] is used for build management. Maven is a software tool for project management and build automation. Maven does a similar role to the Apache Ant tool, but it is based on different concepts and works in a profoundly different manner. Maven uses a construct known as a Project Object Model (POM) to describe dependencies of software project on other external modules and libraries, and the build order. It comes with predefined targets for performing certain tasks such as compilation of code and its packaging. Maven dynamically downloads all libraries that project uses and Maven plugins from one or more repositories.

Wicket framework: Wicket framework[3] is used as the application framework which is a component-based web framework. Wicket framework is built on top of java servlet API. However, unlike frameworks based on MVC, Wicket takes away the task of handling request/response objects, which is inbuilt with technologies such as servlets and allow concentrating on the application's business logic. A Wicket developer can focus on building reusable components that are stateful. Instead of building controllers that handle request/response objects, we can create a page, place components on it, and define how each component responds to user input.

Hibernate: Hibernate[5] is an open source Object-Relational persistence and query framework for the Java language. It provides a service for mapping an object-oriented model to a relational database. The primary feature of Hibernate is mapping Java classes to database tables. In addition to this, it provides data query and retrieval facilities. The automatic generation of SQL calls relieves the developer from manual handling and object conversion of the result set. The mapping of Java classes to database tables is defined by using Java Annotations or by configuring an XML file. Hibernates portability across various relational databases and automatic handling of complex and tedious SQL queries makes it a good candidate for any project.

Solr: Solr[7] is an open-source enterprise search server based on Apache Lucene. It provides a highly scalable, distributed and easily manageable searching platform. Its major features include full-text search, faceted search, real-time indexing, dynamic clustering, database integration, and rich document (e.g., Word, PDF) handling. Solr is the second-most popular enterprise search engine after Elasticsearch. It runs as a standalone full-text search server and uses the Lucene Java search library at its core for full-text indexing and search. Solr has REST-like HTTP/XML and JSON APIs that make it usable from most popular programming languages.

[6] https://maven.apache.org/
[7] http://lucene.apache.org/solr/

6 Conclusion and Future Work

We have presented a Wikidata based approach for factual enrichment of text corpus with the aim to improve retrieval system. The input of the system is plain text corpus and the output is factually enriched corpus that can be used by search engines to answer complex queries and provide more informative summaries to its users. In addition to this, there are other possible uses of our tool such as fact spotting, building training set, etc.

The current implementation can be extended and improved in many ways. First of all, we are planning to use paraphrasing for the search term and wikidata items for improving the recall. For example, "Barack Obama studied at Columbia University" might be paraphrased as "Barack Obama graduated from Columbia University" for retrieving more related documents. The performance can be further improved by implementing a word disambiguation module and other NLP techniques. Finally, a mechanism that annotates only relevant words in the text sentences is one of our future goals.

List of Figures

1.1 Search based on Knowledge Graph 4
2.1 Wikidata Data Model 7
3.1 Structure of Annotated Sentence 10
4.1 Set of annotated sentences that matches the search pattern "golf" and "Scotland" 11
5.1 Architecture diagram of the system 13
5.2 Usecase of Corpus Explorer 15

Bibliography

[1] Shekarpour, Saeedeh, et al. "Keyword query expansion on linked data using linguistic and semantic features." Semantic Computing (ICSC), 2013 IEEE Seventh International Conference on. IEEE, 2013.

[2] Aggarwal, Nitish, and Paul Buitelaar. "Query Expansion Using Wikipedia and Dbpedia." CLEF (Online Working Notes/Labs/Workshop). 2012.

[3] Girardi, Christian, et al. "CROMER: a Tool for Cross-Document Event and Entity Coreference." LREC. 2014.

[4] de A Júnior, José Gildo, Ulrich Schiel, and Leandro Balby Marinho. "An approach for building lexical-semantic resources based on heterogeneous information sources." Proceedings of the 30th Annual ACM Symposium on Applied Computing. ACM, 2015.

[5] Vrandečić, Denny, and Markus Krötzsch. "Wikidata: a free collaborative knowledgebase." Communications of the ACM 57.10 (2014): 78-85.

[6] Lehmann, Jens, et al. "DBpedia–a large-scale, multilingual knowledge base extracted from Wikipedia." Semantic Web 6.2 (2015): 167-195.

[7] Bollacker, Kurt, et al. "Freebase: a collaboratively created graph database for structuring human knowledge." Proceedings of the 2008 ACM SIGMOD international conference on Management of data. ACM, 2008.

[8] Bhogal, J., Macfarlane, A., Smith, P.: A review of ontology based query expansion. Inf. Process. Manage. 43(4), 866886 (Jul 2007), http://dx.doi.org/10.1016/j.ipm.2006.09.003

[9] Dania, Carolina, and Michael Clavel. "Modeling social networking privacy." Theoretical Aspects of Software Engineering Conference (TASE), 2014. IEEE, 2014.

[10] Tylenda, Tomasz, Yafang Wang, and Gerhard Weikum. "Spotting facts in the wild." Workshop on Automatic Creation and Curation of Knowledge Bases. 2014.