Project Report

Student Project Semantic Web Technologies HWS18

Presented by

Nele Ecker
Alex Lütke
Marvin Messenzehl

Submitted to the
Data and Web Science Group
Prof. Dr. Heiko Paulheim & Sven Hertling
University of Mannheim

October - December 2018

Contents

1 Application Domain and Goals

1.1 Problem Statement And User Needs

Everyone who does regular city trips knows the problem. You visit a foreign city and just dont get the same experience as people who live there and know the place well. Typical information sources, like tourist centers, provide a very biased opinion on what one should see and what not. But what if somebody wants to look for something special or has special interests? If somebody really wants to get to know a city and the different districts, this takes a lot of time and manual preparation.

This problem should be solved within this project through an application that uses semantic web technologies.

1.2 Application Goal

The overall goal of the developed application is to give the user a visual help to easily identify the particularities of the different city districts. This should be done with the help of a map as the main user interface where the different districts are marked in different colors in the form of rectangles.
Examples of this would be categories like *shopping, tourists, universities* and a lot more. The detailed categories depend on the districts, building and the offer of the respective city.

In the end, the user should visit the web application where he/she sees a map of a city where different special districts are marked. Therefore trips can be planned more individual and efficient.

2 Datasets

Which datasets does the application use?
How are they accessed (SPARQL, local)?
How do you combine information from different datasets?

3 Techniques

One of the main ideas during development was the decoupling of major steps in Semaps processing pipeline. So the project is split into three smaller part: The user interface, the processing pipeline and REST interfaces. The frontend website is primarily responsible for the rendering of visual elements. The computation of clusters takes place on a separate backend server, which is contacted by the website using a REST-interface on the same backend server. The following sections will be based on this trisection and elaborate in more detail on each of the components.

3.1 User Interface

3.2 REST Interface

When a request is sent to the REST interface, the server reads given parameters and schedules the crawling and clustering of the data as well as the assembly of a response in the GeoJSON format. So the interface does not execute any functional computations; it rather passes on the request to different execution modules using Java dependency injection. The REST interface itself relies on the *Spring Boot* framework and is hosted at the Platform-as-a-Service provider *heroku*.[1] Spring Boot is a framework built on top of the spring framework, which facilitates the development of web applications and requires only very little configuration overhead. So it follows the architectural notion of microservices.

3.3 Processing Pipeline

A core object structure underlies the processing pipeline, which is utilized whenever the backend server is contacted by the UI website. I.e. After crawling the data from respective servers, the results are saved in internal object representations and read when the processing takes place. The core object structure is tuned towards the task of semantically clustering regions and thus facilitates the use of multiple clustering implementations with diverse libraries. The UML class diagram in figure 1 summarizes the object structure:

The figure shows the centrality of the Instance class. All other attributes interact with it, as they rely on the information held in instance objects. Practically speaking, an instance represents a real-world institution like a specific bar or school. Each object of the instance class can have a number of Categories assigned to it. Categories make up the semantic context of an institution. They have a name like museum or bar and a weight. The semap project assumes, that the deeper a category is in its ontology, the more informative it is. That is why a detailed category like Irish pub is meant to get a higher weight than higher-level categories like thing. The clustering algorithm can then prefer to place instances based on the deep-level knowledge into clusters, resulting in more specific clusters. Before clustering takes place, no instance is assigned to a cluster, as no cluster exists. Ideally after clustering each instance is assigned to exactly one cluster. When running a clustering

[1]see `https://dashboard.heroku.com/login`

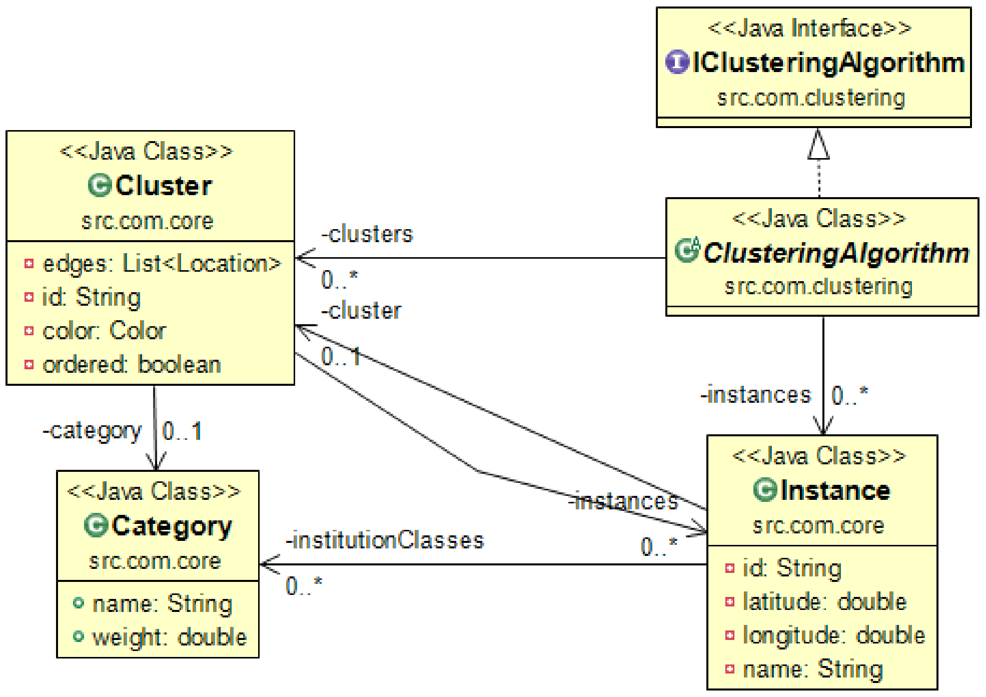

Figure 1: UML class diagram of the core object structure

algorithm (here represented by an abstract class), it will create new objects of type Cluster using a list of instances and assign those instances to a cluster. The different implementations of clustering algorithms are not depicted in the figure for abstraction purposes. But in the end, the actual implementations are of type ClusteringAlgorithm, so they just inherit from the abstract class ClusteringAlgorithm and the interface IClusteringAlgorithm. In the project, a simple square clustering algorithm and a DBScan were implemented.

Simple Square Clustering The simple square clustering algorithm a rather straight-forward implementation of clustering algorithms. It considers each element from the list of instances and creates clusters based on their position. So for each instance, the simple square clustering implementation will check if there is already a cluster, in which the current instance is located and will assign it to this cluster or create a new cluster correspondingly. After having done so for the entire list of instances, each clusters gets a category. The computation is based on a scoring function, which determines the times, a category appears in a cluster and weights each appearance by the categorys weight-property. The category with the highest weight is then allocated to a cluster. Algorithm 1 formalizes the procedures again. In sum, the simple square clustering algorithm does not take into account the semantic context (i.e. the categories) of instances and only uses positional information initially. That is why the results are not guaranteed to be semantically meaningful. However, the algorithm runs very fast, as it is linked to very little computational overhead.

DBScan DBScan is a the sophisticated clustering algorithm in the semap project and relies on the Weka library implementation. The DBScan tries to build clusters based on two pieces of information: The position of instances and their categories. The position is determined by the longitude and latitude coordinates and is normalized on a scale from zero to one; alternatively it can be z-normalized by a parameter setting in the Java coding. Categories are one-hot-encoded, so that each instance gets a vectors consisting of values in the range from one to zero. The vector represents the set of all categories. Each category is represented by a positional value in the vector. All the categories an instance does not belong to, take the value 0 in the vector. The categories, an instance belongs to, take a normalized value greater than zero. The value is determined by the formula:

$$n := number\ of\ categories\ of\ an\ instance \tag{1}$$

$$w_i := weight\ of\ the\ ith\ category\ of\ an\ instance \tag{2}$$

$$w_c := weight\ of\ current\ category \tag{3}$$

Algorithm 1 Sketch of the simple quare clustering algorithm

```
$I \leftarrow$ set of all instances
$C \leftarrow \emptyset$
**for all** $i \in I$ **do**
   **for all** $c \in C$ **do**
      **if** $i\ located\ in\ c$ **then**
         $c.add(i)$
      **else**
         $x \leftarrow createCluster(i.latitude, i.longitude)$
         $C \leftarrow C \cap x$
      **end if**
   **end for**
**end for**
$CAT \leftarrow \emptyset$
**for all** $i \in C.getInstances()$ **do**
   **for all** $category \in i.getCategories()$ **do**
      $CAT_{category} \leftarrow CAT_{category}.getScore() + category.getScore()$
      $CAT \leftarrow CAT \cap CAT_{category}$
   **end for**
**end for**
**return** $max(CAT.getAllScores())$
```

$$val = \frac{w_c}{\sum_{i=0}^{n} w_i} \tag{4}$$

Weight of current category/sum of categories of an instance(weight) So the value depends on the number of instances categories and the weight of the respective categories. This method ensures, that the weighted vector values have the same impact on the clustering as the positional information latitude and longitude. Before the DBScan algorithm can start working in Weka, a preprocessing pipeline transform the internal object representation as described in figure 1 into the object representation needed by the Weka library. It therefore writes an artificial .arff-file to disk and reads it again with tools from the weka library. Afterwards, the DBScan is executed and the results again returned into the core object representation of semap. The entire procedure of the DBScan in the semap project is summarized in algorithm 2 as follows:

Algorithm 2 Sketch of the DBScan clustering algorithm

```
$I \leftarrow$ set of all instances
$C \leftarrow \emptyset$
for all $i \in I$ do
  $x \leftarrow normalize(i.latitude)$
  $i.setLatitude(x)$
  $x \leftarrow normalize(i.longitude)$
  $i.setLongitude(x)$
  $\vec{y} \leftarrow oneHotEncode(i.categories)$
  for all $z \in \vec{y}$ do
    $z \leftarrow normalize\ z\ as\ to\ formula\ (4)$
  end for
  $i.categories \leftarrow \vec{y}$
end for
$dBScan(I)$
$CAT \leftarrow \emptyset$
for all $i \in C.getInstances()$ do
  for all $category \in i.getCategories()$ do
    $CAT_{category} \leftarrow CAT_{category}.getScore() + category.getScore()$
    $CAT \leftarrow CAT \cap CAT_{category}$
  end for
end for
return $max(CAT.getAllScores())$
```

In sum, the value normalization and object transformations are computationally expensive operations. While for example the simple square algorithm clusters multiple millions of instances within few milliseconds, the DBScan requires about 30 seconds to do so with only 15 thousand instances. However, the major part of this timeframe is not spent in the normalization and object transformation implementation. The most time is actually needed by Wekas DBScan algorithm itself. So this runtime behavior is a major limitation of the DBScan algorithm. Nonetheless, the DBScan algorithm is capable of finding much more meaningful results. It considers categories of instances already when performing the clustering and not afterwards. It is also not limited to geographically speaking squared clusters, but has arbitrary decision boundaries. That makes this algorithm a very powerful tool for detailed analyses on regional, public data.

4 Results

What outcomes does the application provide?
How are some user queries answered?

Evaluation, based on Mannheim sample:

5 Known Limitations

In which domains does the application not work?
Are there queries which cannot be answered?
Why?
How could you overcome those limitations, given more time?

6 Lessons Learned

Which challenges did you face?
What were the biggest obstacles?
What would you do different the next time?

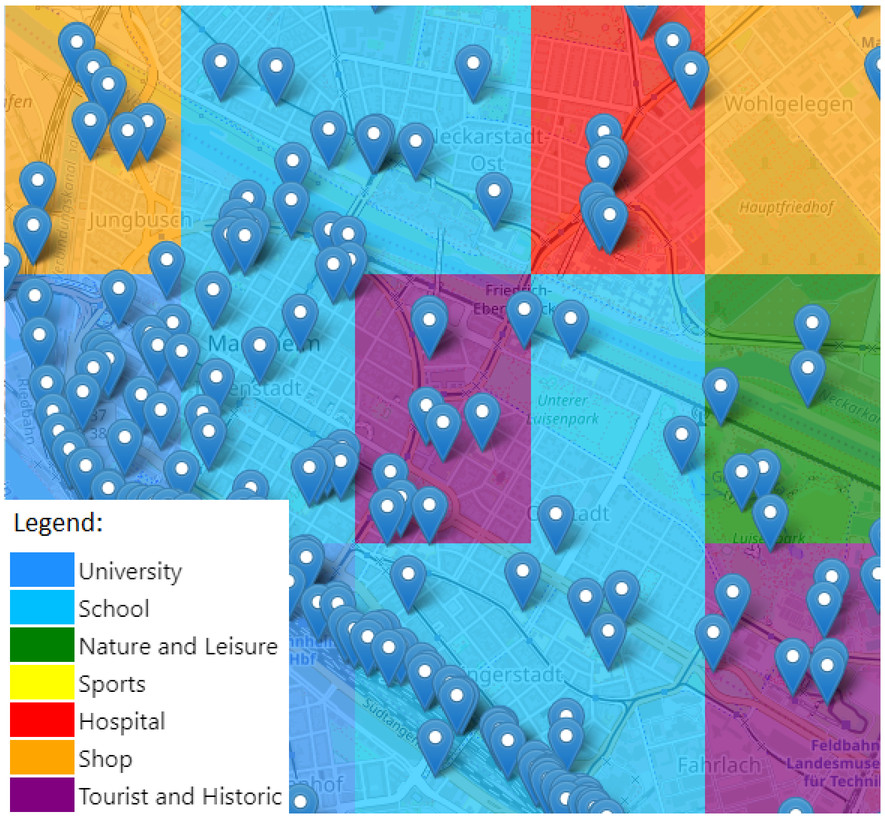

Figure 2: Evaluation based on sample result in Mannheim