SSSW2016 hackathon

July 21, 2016

Hack day!

WHAT YOU ARE DOING TODAY: implement functionalities around semantic scholarly datasets
OBJECTIVES: build tools to provide enrichment, querying, and browsing for the provided datasets
SPECIFIC TASKS: Knowledge Extraction, Data Transformation, Sentiment Analysis, Linked Dataset Exploration
DATASETS: ScholarlyData dataset, Semantic Web Journal dataset, SentiWordNet.

Objectives

Your goal is to implement functionalities around semantic scholarly datasets, i.e. data about academic artefacts, events, researchers etc.

Your application will enable analytics over aggregated data, such as:

- considering paper topics (knowledge extraction, data integration, machine learning, etc.), what is the trend of reviews (positive/negative) in each topics?
- what are the authors that have the highest rate of publication at a certain venue?
- what are the main topics published at a certain venue?
- is a paper review positive or negative according to sentiment analysis? Does the overall score (assigned by a reviewer) correlates with the extracted sentiment?

These are only examples, you are free to think of any additional application exploiting these data. Be creative! But also be practical and share the work in the team wisely... remember you have 3 hours.

Datasets

ScholarlyData

Scholarlydata provides Linked Data for conferences and workshops. The dataset contains data about papers, people with their affiliations, organisations, events (both academic and social), roles held by people in organising and programme committee. These data are organised according to the conference-ontology whose OWL source code and documentation are available at `http://www.scholarlydata.org/ontology/conference-ontology.owl` and `http://www.essepuntato.it/lode/http://www.scholarlydata.org/ontology/conference-ontology.owl`, respectively.

The Linked Data are available both as SPARQL and as RDF dumps. During the hackathon you can use the SPARQL endpoint of Scholarlydata that is installed locally on the virtual machines of the laboratory. The local SPARQL endpoint can be accessed at `http://localhost:8894/sparql`.

In order to query Scholarlydata over the local endpoint it is mandatory to set the `FROM` clause to `<scholarlydata>`. The following is an example of a SPARQL SELECT query performed to the local endpoint:

```
SELECT *
FROM <scholarlydata>
WHERE{
    ?s ?p ?o
}
LIMIT 1000
```

Semantic Web Journal

The Semantic Web Journal publishes metadata about submitted and accepted papers as Linked Open Data. They include papers, editors, reviewers, decisions, acceptance rates, paper types, the text of the reviews, and much more.

The dataset can be accessed via SPARQL at `http://localhost:8894/sparql`. In order to query it, it is mandatory to set the `FROM` clause to `<semantic-web-journal>`. The following is an example of a SPARQL SELECT query performed to the local endpoint:

```
SELECT *
FROM <semantic-web-journal>
WHERE{
    ?s ?p ?o
}
LIMIT 1000
```

WordNet RDF

WordNet is a large dataset of English lexicon. Nouns, verbs, adjectives and adverbs are grouped into sets of synonyms called *synsets*, each expressing a distinct concept. Synsets are interlinked by means of semantic and lexical relations.

We have an RDF version of WordNet available on the local installation of virtuoso on the virtual machines. This dataset can be accessd by setting the `FROM` clause to `<wordnet>`. The following is an example of a SPARQL SELECT query performed to the local endpoint:

```
SELECT *
FROM <wordnet>
WHERE{
    ?s ?p ?o
}
LIMIT 1000
```

Tasks

Knowledge Extraction

This task is aimed at extracting meaningful RDF triples from textual content. As target text we will use paper reviews from the Semantic Web Journal (SWJ) linked data. You can retrieve open reviews associated with papers by means of the datatype property `http://semantic-web-journal.com/ontology#reviewComment`.
The following is an example of an RDF triple that represents an open review associated with the paper titled "Scalable Long-term Preservation of Relational Data through SPARQL queries":

```
<http://semantic-web-journal.com/sejp/node/384>
    <http://semantic-web-journal.com/ontology#reviewComment>
        "Gnther Grz: Summary and Main Contributions.
        The authors address a very important problem,
        scalable long-time archival of relational databases
        as RDF triples.  For this purpose, they implemented
        the SAQ (Semantic Archival and Query) system
```

```
which they motivate by a striking example, define
theoretically, give a detailed account of its
implementation and finally provide an evaluation
based on a new benchmark, with a throrough
discussion.  In SAQ, an RDF view of a RDB is
automatically generated, and long-time preservation
as RDF of selected parts of it are specified in an
extended SPARQL dialect. A data archive as well as
a schema archive are generated.  The authors show
that their modular approach is clearly superior to
other approaches which implement compilers that
translate SPARQL directly into SQL.
Strong points of the paper. The organisation and
presentation of the paper are very good, it
linguistic expression is clear and easy to follow.
The original contribution of the paper is very well
motivated and the argumentation is supported by
convincing examples.  Related work is covered in
detail. With its new benchmark derived form the
well-know Berlin benchmark, the authors illustrate
the superior performance of their approach.
Possible improvements None.  I think the paper
is excellent and should be published "as-is""
```

Your task is to generate RDF triples from these texts. You can use FRED [3]. FRED is a tool to transform natural language text to formal structured knowledge. FRED is available as REST HTTP API at `http://wit.istc.cnr.it/stlab-tools/fred`, it takes a text as input and produces an RDF/OWL graph as output, with the following formats:

- RDF/XML (application/rdf+xml);
- RDF/JSON (application/rdf+json);
- Turtle (text/turtle);
- N3 (text/rdf+n3);
- N-Triples (text/rdf+nt).

The following is an example of a cURL request performed against the FRED API:

```
$> curl -G -X GET -H "Accept: text/turtle" --data-urlencode \
     text="The authors address a very important problem." \
     http://wit.istc.cnr.it/stlab-tools/fred
```

The input is the sentence "The authors address a very important problem." that is associated with the parameter `text` and the output format (i.e., text/turlte) is specified within the `Accept` header of the request. Figure 1 shows the RDF output corresponding to the example sentence.

The following snippet reports the same request performed via Java.

```
String fredEndpoint = "http://wit.istc.cnr.it/stlab-tools/fred";
String text = "The authors address a very important problem.";
String request = "?text=" + URLEncoder.encode(text, "UTF-8");

// Create connection
URLConnection conn = new URL(fredEndpoint + request).openConnection();
// Set the output format in the Accept header
conn.setRequestProperty("Accept", "text/turtle");
// Get the RDF output as an InputStream
InputStream is = conn.getInputStream();
```

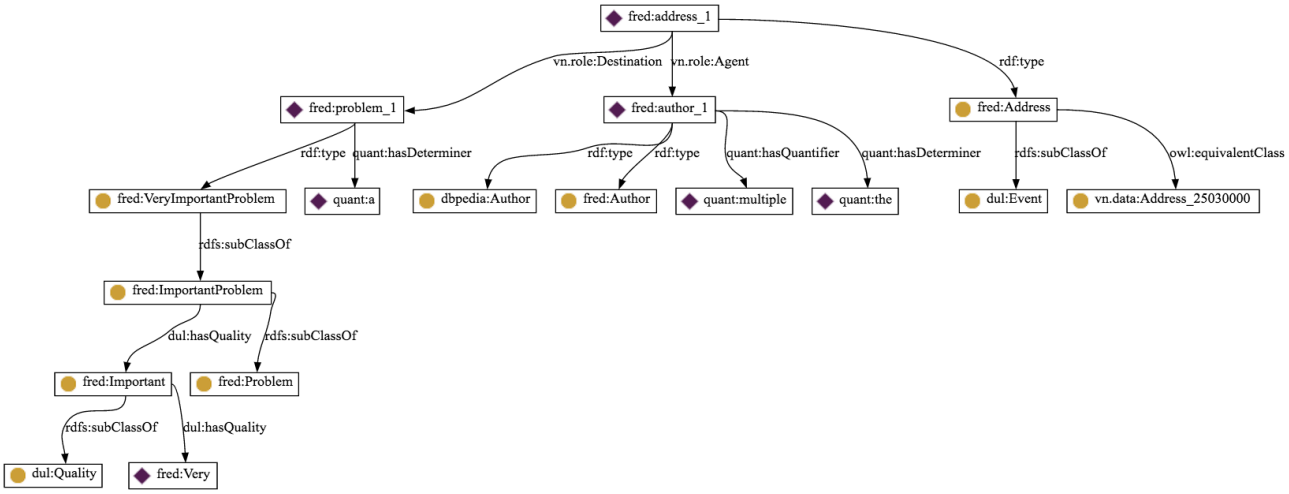

Figure 1: Graphical representation of the RDF graph resulting from FRED given the sentence "The authors address a very important problem.' as input.

```
// Load the RDF output into an Apache Jena in-memory model
Model model = ModelFactory.createDefaultModel();
model.read(is, null, "TURTLE");
```

Hints. FRED performs faster if the size of the text is not too large. Hence, we recommend to split long textual contents into smaller pieces (e.g., sentences).

Store each RDF graph corresponding to an open review as a separate named graph in a triplestore (e.g., Jena TDB or Virtuoso). If you choose Virtuoso as triplestore you can use the local installation on the virtual machine that can be accessed via Apache Jena. An example about how to store an RDF graph to Virtuoso via Jena is reported in the following snippet.

```
String vrtAddress = "jdbc:virtuoso://localhost:1111";
String vrtUser = "dba";
String vrtPass = "dba";

String graphName = "swj-open-review-xxx";

// Create the named-graph on Virtuoso
VirtGraph virtGraph = new VirtGraph(graphName, vrtAddress, vrtUser, vrtPass);

// Add the triples from the Apache Jena model to the Virtuoso named-graph.
Iterator<Statement> it = model.listStatements();
while(it.hasNext()) {
    Statement stmt = it.next();
    virtGraph.add(stmt.asTriple());
}

// close connection
virtGraph.close()
```

Data Transformation

The goal of this task is to convert a relational database to RDF. The relational database is SentiWordNET [2], a lexical resource for opinion mining, built in connection to WornNet. SentiWordNet assigns to each synset of WordNet three sentiment scores: positivity, negativity, objectivity. SentiWordNet is available as a relational

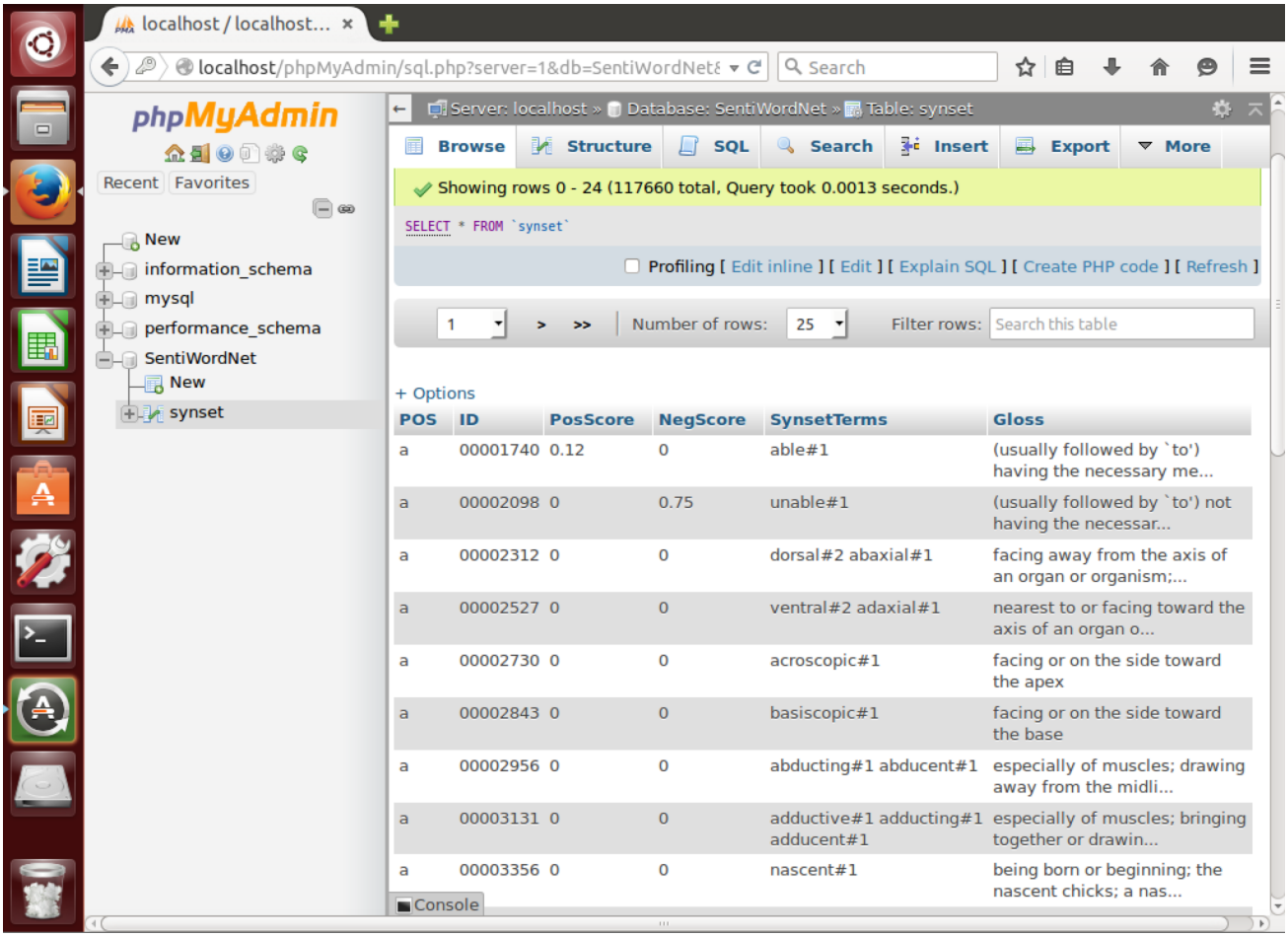

Figure 2: Content of SentiWordNet.

database in the MySQL installation on the virtual machines of the laboratory. MySQL can be accessed via browser with PhpMyAdmin (i.e., http://localhost/phpMyAdmin) providing the username `root` and the password `bertinoro`. Figure 2 shows a sample of the content of SentiWordNet available in MySQL.

The following steps are required to address the task:

1. **Design an ontology that models the concepts and relations available in SentiWordNET**. The ontology should be designed by using a tool for ontology editing, e.g.**Protégé** which is available on the virtual machines.

2. **Design the transformation rules from SentiWordNET DB to RDF**. You address this with the D2RQ Platform [1], a system for accessing relational databases as virtual, read-only RDF graphs (via SPARQL query), without having to replicate it into an RDF store.

   To express the transformation, you need to write a mapping file, expressed with the D2RQ mapping language[1], describing the relation between a relational data model and a target ontology.

. The following is an example of mapping declaration that allows to generate a `swn:Synset` individual for each SentiWordNet synset.

```
# D2RQ Namespace
@prefix d2rq: <http://www.wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#> .
# Namespace of the ontology
@prefix : <http://www.example.org/swn/ontology/> .
```

[1] `http://d2rq.org/d2rq-language`.

```
# Namespace of the mapping file; does not appear in mapped data
@prefix map: <file:///Users/d2r/example.ttl#> .

# Other namespaces
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

map:Database a d2rq:Database;
    d2rq:jdbcDSN "jdbc:mysql://localhost:3306/SentiWordNet";
    d2rq:jdbcDriver "com.mysql.jdbc.Driver";
    d2rq:username "root";
    d2rq:password "bertinoro";
    .
# CREATE individuals of swn:Synset from the table synset of
#the DB SentiWordNet
# The URI is generated by using
# the pattern http://www.example.org/swn/resource/SYNSET.ID

map:Conference a d2rq:ClassMap;
    d2rq:dataStorage map:Database.;
    d2rq:class :Synset;
    d2rq:uriPattern "http://www.example.org/swn/resource/@@synset.ID@@" .
```

D2RQ can be used in two ways: (i) as a standalone application, passing the mapping file as input or (ii) or programmatically with the Apache Jena API.

To use D2RQ as standalone server on the virtual machine you can use the following commands:

```
$> cd /media/sssw/external/software/d2rq-0.8.1/
$> ./d2r-server /home/sssw/workspace/hackathon/files/swn_mapping.ttl
```

and then query the database via the SPARQL endpoint available at `http://localhost:2020`.

Programmatically, you can access the virtual RDF graph as an Apache Jena model. The following is an example, assuming that your mapping file path is `./mapping.ttl`.

```
// Get the Apache Jena model that allows to access SentiWordNet as RDF
Model d2rqModel = new ModelD2RQ("./mapping.ttl");

// Query the virtual RDF graph of SentiWordNet asking all the individuals of the class swn:Synset
String sparql = "SELECT DISTINCT ?synset WHERE{?synset a http://www.example.org/swn/ontology/Synset}"
Query query = QueryFactory.create(sparql);
QueryExecution queryExecution = QueryExecutionFactory.create(query, d2rqModel);
ResultSet resultSet = queryExecution.execSelect();

// Iterate the result set
while(resultSet.hasNext()){
    QuerySolution querySolution = resultSet.next();
    Resource synset = querySolution.getResource("synset");
}
```

Sentiment Analysis

The goal of this task is to enrich the RDF graphs obtained from open reviews with sentiment polarities available from SentiWordNet. This is a sentiment analysis task. However, a word-sense disambiguation (WSD) step is required in order to use those polarities as they are associated with WordNet synsets.

Hints:

- A baseline approach to WSD is to use frequencies of senses available in WordeNet corpus. Words are disambiguated with the most frequent sense associated to them. It is possible to use the WordNet RDF

dataset available in the local installation of virtuoso to perform this baseline WSD approach. For example, the following SPARQL query allows to retrieve all the synsets and sense frequencies related to the term "important":

```
PREFIX wn30: <http://www.w3.org/2006/03/wn/wn30/schema/>
SELECT ?synset (xsd:integer(?frequence) AS ?freq)
FROM<wordnet>
WHERE{
  ?synset wn30:senseLabel "important"@en-us .
  ?synset wn30:containsWordSense ?sense .
  ?sense wn30:tagCount ?frequence
}
ORDER BY DESC(xsd:integer(?frequence))
```

This query can be executed against the SPARQL endpoint of Virtuoso either by its HTML interface available at `http://localhost:8890/sparql` or by using Jena. The following is an example of SPARQL query performed via Jena:

```
Query query = QueryFactory.create(sparql, Syntax.syntaxARQ);
QueryExecution queryExecution = QueryExecutionFactory.create(query, model);
ResultSet resultSet = queryExecution.execSelect();
```

- the terms to use for sentiment analysis can be extracted from FRED graphs. More in detail, FRED gives you *qualities* as triples having `dul:hasQuality`[2] as predicate. The following is an example:

```
fred:ImportantProblem dul:hasQuality fred:Important .

fred:offset_27_36_important semiotics:hasInterpretant fred:Important;
    rdfs:label "important"^^xsd:string .
```

Linked Dataset Exploration

In this task we want you to explore the ScholarlyData dataset and design queries to find interesting facts or possible mistakes. For example, we know that ScholarlyData contains duplicated URIs for the same *persons* and for the same *organisations*. You can explore the dataset to identify good indicators to detect duplicates.

To give you a starting point, you can identify the list of similar URIs with the following SPARQL query:

```
PREFIX person: <https://w3id.org/scholarlydata/person/>
PREFIX conf: <https://w3id.org/scholarlydata/ontology/conference-ontology.owl#>
SELECT distinct ?person ?person2
WHERE{
  ?person conf:givenName ?givenName.
  ?person conf:familyName ?familyName .
  ?person2 conf:givenName ?givenName2.
  ?person2 conf:familyName ?familyName .
  filter(?person != ?person2 AND (regex(?givenName, ?givenName2)
  OR regex(?givenName2, ?givenName)))
}
```

Or you can identify paper co-authors using the following SPARQL query:

```
SELECT DISTINCT ?p ?coauth (COUNT(?coauth) AS ?PAPERCOUNT)
WHERE{
  ?p foaf:made ?paper .
  ?paper <http://purl.org/dc/elements/1.1/creator> ?coauth .
  FILTER(?coauth != ?p)
} GROUP BY ?p ?coauth ORDER BY (?p)
```

[2]The prefix `dul:` is associated with the namespace `http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#`.

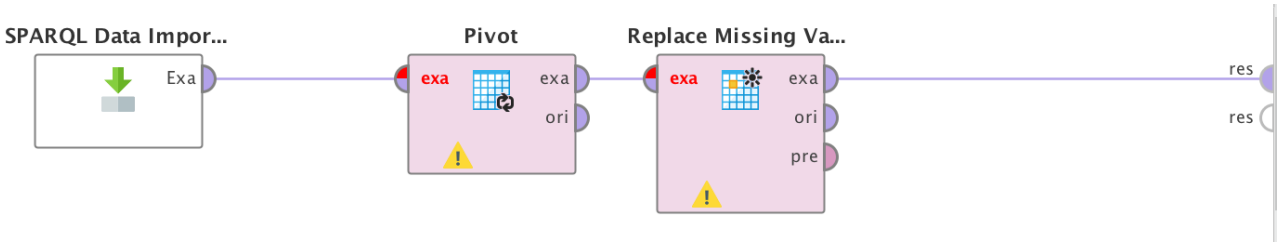

Figure 3: RapidMiner process to query the ScholaryData SPARQL endpoint and construct a co-authorship matrix.

Additionally, you can manipulate data using data analytics tools. In the virtual machine you will find an installation of RapidMiner[3] data analytics platform. Using the *RapidMiner Linked Open Data Extension*[4] you can directly import data via the SPARQL endpoint within RapidMiner.

In your RapidMiner installation[5] you fill find ready made example processes, such as the one showed in Figure 3 to query the ScholaryData SPARQL endpoint and construct a co-authorship matrix.

You can choose to query different information from ScholarlyData, further manipulate obtained data either within RapidMiner or export the data and use it externally (e.g. within your Java code).

Quick link access

FRED : `http://wit.istc.cnr.it/stlab-tools/fred`

D2RQ : `http://d2rq.org/d2rq-language`

LOCAL D2RQ SPARQL (SentiWordNet) endpoint : `http://localhost:2020` (you need to start it first)

LOCAL SPARQL endpoint (scholar data) : `http://localhost:8890/sparql`

RapidMiner Linked Open Data Extension : `http://dws.informatik.uni-mannheim.de/en/research/rapidminerlodextension/`

SPARQL1.1 reference : `https://www.w3.org/TR/sparql11-query`

Apache Jena guide : `https://jena.apache.org/getting_started/index.html`

References

[1] C. Bizer and A. Seaborne. D2RQ - Treating Non-RDF Databases as Virtual RDF Graphs. In *ISWC2004 (posters)*, 2004.

[2] A. Esuli and F. Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. In *In Proceedings of the 5th Conference on Language Resources and Evaluation (LREC 2006)*, pages 417–422, 2006.

[3] A. Gangemi, V. Presutti, D. Reforgiato Recupero, A. G. Nuzzolese, F. Draicchio, and M. Mongiovì. Semantic web machine reading with fred. *Semantic Web*, page to appear, 2016.

[3] `https://rapidminer.com/`

[4] `http://dws.informatik.uni-mannheim.de/en/research/rapidminerlodextension/`

[5] Use the Repository view, you will find a repository called SSSW2016 where the example processes are stored.