# Linked Open Data

Using SPARQL to Retrieve Non-Trivial Information Related to Movies
Primary Topic: SEMI, Secondary Topic: TS[1] & IENLP[2]
Course: 2021-1B – Group: 126 – Submission Date: 2022-02-06

Ajay Mathew Joseph[1]
University of Twente
The Netherlands
a.m.joseph@student.utwente.nl

Nishchit Mahajan[2]
University of Twente
The Netherlands
n.mahajan@student.utwente.nl

## ABSTRACT

Everyday new data is being produced and it is often stored in multiple places. Increasing data is an ongoing issue and everyday new approaches are being suggested to make use of it. Everyone has some information about a concept but often nobody has the complete information. If the data could be combined from multiple sources, the existing datasets could be enriched and could provide added value. Linked Open data is one of the interesting solutions to such a problem. In this project we understand the concept of Linked Open Data and how it could be useful to link existing datasets with it.

## KEYWORDS

Linked Open Data, RDF, SPARQL, DBPedia

## 1 INTRODUCTION

Data is everywhere and everything is data. Everyday data is being produced at an exponential speed and handling such huge amount of data is becoming troublesome. Storing such huge amounts of data can make data pipelines slow and can be very expensive. As a major resort, the concept of Linked Data can be very helpful. Coined in 2006 by Tim Berners Lee [1], the concept of linking data can be referenced as far as back to 1999. This concept of linking helps to share, reuse, and make data machine more readable. This technique helps us to create a small database and link the same topics to multiple different databases. A recent tool to work on such problems is Resource Desciption Framework (RDF). This framework is a standard model for data interchanges on the Web. This platform forms a directed graph and interlinks the data using named links. In this project, we make use of the concept of RDF in Linked data and try to connect a small movie dataset to DBPedia, a publicly available free and open database and run some complex queries to test if interlinked data can provide desired results.

The goal of this project is to understand the concept of Linked Open Data by linking an existing movie dataset with DBPedia, an established Linked Open Dataset and thus enriching the initial dataset. We also demonstrate the added value by executing five complex queries and retrieving non-trivial information.

This report has been organized in the following way. Section 2 talks about some related work by other people in similar areas and discusses the dataset used. Section 3 discusses the implementation of the project, including the functions and SPARQL queries used in this project. Section 4 talks about the results obtained from this project. Section 5 discusses about the results and also, lists the limitations of this approach. Section 6 provides a conclusion to the project report which is followed by the references. Finally appendix A provides the link to the code repository of the complete project.

## 2 BACKGROUND

This project essentially deals with three concepts which are, RDF, DBPedia and SPARQL. In the following paragraphs, we would describe what they are.

As mentioned in the introduction, RDF is a model for data interchange on the web. In RDF datasets, the data is often stored in the format of triples. The format of a triple is `subject->predicate->object`. Here, `subject` is the primary entity and it is connected to the `object` which is the secondary entity with a property/verb which is the `predicate`

DBPedia is a project which extracts structured content from the Wikipedia project. Almost all Wikipedia pages contain structured information, which is displayed on the top right of the Wikipedia pages as 'infobox'es. The DBPedia project extracts this information and a dataset is created out of it which can be queried. The entries in the RDF dataset would be linked to the corresponding DBPedia entry using the `owl:sameAs` predicate. The links would be retrieved using the DBPedia Spotlight API which is a tool for automatically annotating DBPedia resources.

SPARQL is the standard query language and protocol for Linked Open Data and RDF Databases. Information from any data source that can be mapped to and RDF can be queried using SPARQL. The added advantage of SPARQL is that it can be used to query from multiple sources. In this project, we use SPARQL to query information from DBPedia.

## 3 APPROACH

There are mainly two steps in this project, namely, (a) Constructing an RDF dataset from a base dataset, and (b) running SPARQL queries and retrieving the results for the purpose of demonstration.

### 3.1 Constructing RDF dataset

In the first step, we obtain the base dataset from Kaggle. The `movies.csv` file in the dataset contains all the details required for this project. A Python Jupyter notebook was used to construct an RDF dataset from this CSV file. For this, we used the `rdflib` library. The data was initially loaded into a `pandas` dataframe. It was observed that the dataset consists of 45,466 records with 24 attributes. Some of the attributes were `original_language`, `original_title`, `release_date`, etc.. Further, a function was defined to make use of the DBPedia Spotlight API to find the DBPedia link of a movie given it's title. For every movie, it's DBPedia link

was retrieved and alongwith the movie's `original_title` and `release_date`, an RDF dataset was constructed using the `rdflib` library.

## 3.2 Tool for executing SPARQL queries

In the second step, we developed a commandline menu interface which could execute five complex SPARQL queries on this dataset and retrieved their results. The user will be shown a five choice menu and will be asked to enter his choice. Depending on the choice, the corresponding query will be executed using the corresponding functions. These were done using the `SPARQLWrapper` library and Python language. The endpoint for the query was set as "http://dbpedia.org/sparql". The queries were:

(1) Retrieve related movies
(2) Get average of all movies of a particular movie's director
(3) Get other movies where the coactors of a particular movie acted together
(4) Get youngest main crew member of a movie
(5) Get the longest movie of a movie's top actor

We also defined three common functions which is useful in all the queries. They are:

(1) `get_dbpedia_link()`: Given a movie's name, retrieve it's DBPedia link
(2) `get_actor_name()`: Given an actor's URI, retrieve his/her name
(3) `get_movie_name()`: Given a movie's URI, retrieve it's name

The logic for each of the queries will be explained in the following subsections.

*3.2.1 Related movies.* For implementing this query, the input would be a movie's name. The movie's DBPedia link was found using the `get_dbpedia_link()` function and then it was used to find related objects using the `dbo:wikiPageWikiLink` predicate which links to the other Wikipedia pages within this Wikipedia page. The objects found where passed to another function to filter the movies from them and these were returned.

```
SELECT ?remote_value
WHERE {
        <https://dbpedia.org/page/The_Matrix>
            dbo:wikiPageWikiLink ?remote_value
    }
```

**Listing 1: Retrieve `dbo:wikiPageWikiLink` objects**

*3.2.2 Average Budget.* This query reads a particular movie's title and returns the average budget of all the movies which this particular movie's director has directed. We find the director from the movie's data and then pass it to a function which finds all movies which he/she has directed and retrieves the budget of all those movies. The average of budget was returned as a result.

```
SELECT ?director
WHERE {
        <https://dbpedia.org/page/Before_Sunrise>
            dbp:director ?director
    }
```

**Listing 2: Retrieve Director's name**

```
SELECT ?budget
WHERE {
    ?movie dbo:director
        <https://dbpedia.org/page/Richard_Linklater> .
    ?movie dbo:budget ?budget
    }
```

**Listing 3: Retrieve budgets of a director's movies**

*3.2.3 Other movies of coactors.* In this query, we read a particular movie and returns all movies in which the top two actors of this movie acted together. The top two actors were retrieved from the movie's page and then was passed to another function which finds all movies in which these two actors have acted together.

```
SELECT ?actor
WHERE {
        <https://dbpedia.org/page/The_Matrix>
            dbo:starring ?actor .
    }
```

**Listing 4: Retrieve actors of a movie**

```
SELECT ?film
WHERE {
        ?film dbo:starring
            <https://dbpedia.org/page/Keanu_Reeves> .
        ?film dbo:starring
            <https://dbpedia.org/page/Hugo_Weaving> .
    }
```

**Listing 5: Retrieve movies in which two actors acted together**

*3.2.4 Youngest main crew member.* For this query, we retrieve the main crew members' URI (director, actors, cinematographer, composer, music, producer) from a movie's data and then pass them to a function which retrieves the birth dates of all of them in descending order. The first result was returned as an output.

```
SELECT ?dir ?dop ?comp ?music ?screen
    ?prod group_concat(?star; separator=",")
WHERE {
        <https://dbpedia.org/page/The_Matrix>
            dbo:director ?dir .
        <https://dbpedia.org/page/The_Matrix>
            dbo:cinematography ?dop .
        <https://dbpedia.org/page/The_Matrix>
            dbo:musicComposer ?comp .
        <https://dbpedia.org/page/The_Matrix>
            dbp:music ?music .
        <https://dbpedia.org/page/The_Matrix>
            dbo:producer ?prod .
        <https://dbpedia.org/page/The_Matrix>
            dbo:starring ?star .
    }
```

**Listing 6: Retrieve main crew members of a movie**

```
SELECT ?bd ?name WHERE
{
{SELECT ?bd ?name
WHERE {
        <https://dbpedia.org/page/Keanu_Reeves>
            dbo:birthDate ?bd .
        <https://dbpedia.org/page/Keanu_Reeves>
            dbp:name ?name .
    }}
UNION
{SELECT ?bd ?name
WHERE {
        <https://dbpedia.org/page/Hugo_Weaving>
            dbo:birthDate ?bd .
        <https://dbpedia.org/page/Hugo_Weaving>
            dbp:name ?name .
    }}
    }
ORDER BY DESC(?bd)
```

**Listing 7: Retrieve crew members in order of age**

*3.2.5 Longest movie of an actor.* In this query, we take as input a movie's name and then find it's lead actors URI. This was then used to find all movies in which he/she has acted and the results were sorted (using ORDER BY) in the descending order. The top result was returned as the longest movie of the particular actor.

```
SELECT ?runtime ?name
WHERE
{
    ?movie dbo:starring
        <https://dbpedia.org/page/Robin_Williams> .
    ?movie dbo:runtime ?runtime .
    ?movie dbp:name ?name .
}
ORDER BY DESC(?runtime)
```

**Listing 8: Retrieve movies of an actor in decreasing order of runtime**

# 4 RESULTS

## 4.1 RDF Dataset

The RDF Dataset was constructed with 10,000 of the movie records out of the 45,466 records since the main objective of the project is to demonstrate the added value of Semantic Web Technology with Linked Open Data. While retrieving the DBPedia links, only 6,905 were succesful and hence, the final dataset contained only those many records. The file was saved as `movie.owl`. A snippet of the file can be seen in Figure 1.


**Figure 1: `movie.owl` , RDF Dataset**

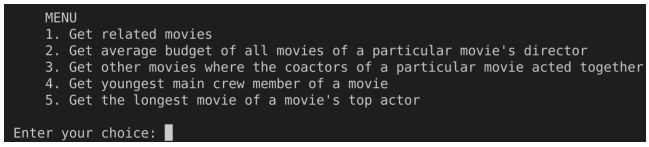


**Figure 2: Menu Interface**

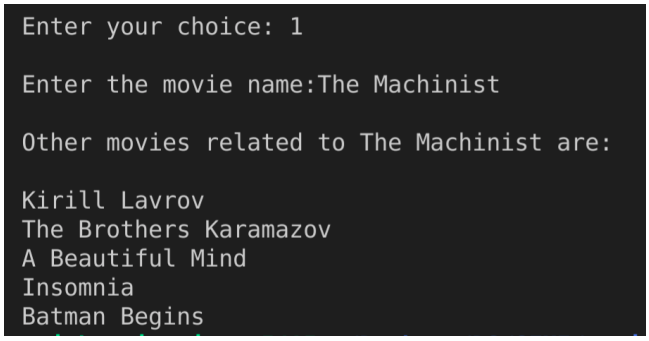


**Figure 3: Related Movies**

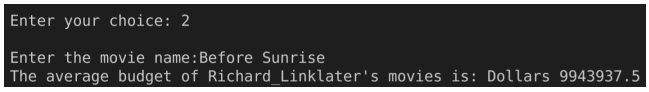


**Figure 4: Average Budget of a Director**

## 4.2 SPARQL Queries

As mentioned in the previous section, a menu interface was shown to the user. The interface can be seen in Figure 2.

The results for each of the queries will be shown in the following subsections.

*4.2.1 Related Movies.* This query took as input a movie's name and printed it's related movies. An example is shown in Figure 3

*4.2.2 Average Budget of a Director.* This query took as input a movie's name and printed the average budget of it's director's movies. An example is shown in Figure 4.

*4.2.3 Other movies of two actors.* This query took as input a movie's name and printed all movies in which this particular movie's top two actors acted together.. An example is shown in Figure 5.

```
Enter your choice: 3

Enter the movie name:The Matrix

The top 2 actors are:
Keanu Reeves
Hugo Weaving

The movies where they acted together are:
The Matrix Reloaded
The Matrix Revolutions
The Matrix
```

**Figure 5: Movies of coactors**

```
Enter your choice: 4

Enter the movie name:The Matrix
The youngest main crew member is Keanu Reeves born on 1964-09-02
```

**Figure 6: Youngest Crew Member**

```
Enter your choice: 5

Enter the movie name:Jumanji
The longest movie of Robin Williams is Hamlet running 4.033333333333333 hours
```

**Figure 7: Longest movie**

*4.2.4 Youngest main crew member.* This query took as input a movie's name and prints it's youngest crew member's name and birthdate. An example is shown in Figure 6.

*4.2.5 Longest movie of an actor.* This query took as input a movie's name and prints the name of it's actor's longest movie. An example is shown in Figure 7.

## 5 DISCUSSION

It has been demonstrated that by linking existing datasets with Linked Open Data, the original dataset could be enriched with new and non-trivial information. By linking related entities, data can be retrieved in unforeseen ways. For example, queries like finding a director's average budget is something which cannot be done with a simple Google search. It has been possible only because all data regarding the director and his movies are linked alongwith the movies' budget.

Another example is retrieving related movies from a movie's Wikipedia page. Wikipedia pages usually contain links to the Wikipedia pages of other entities related to the original page. This has huge potential for being very relevant entities because Wikipedia pages often contain almost all information about the particular entity and its background. This could be used to build recommender systems.

Other queries like retrieving the youngest main crew member, the longest movie of an actor, the movies where two actors acted together, etc. are all non-trivial and would be nearly impossible to be retrieved using machine learning models in a deterministic manner.

Search engines like Google typically use machine learning algorithms and page-rank algorithm to predict the relevance of a search result. However, if there is enough links between the entities of a particular domain, data could be retrieved better and faster using Linked Open Data. Machine learning algorithms could incorporated alongwith Linked Open Data tools to retrieve better results.

However, there were some limitations in our approach to this project. First of all, the details regarding an actor or a director cannot be queried by their name alone. The system needs the name of the movie in which they starred or directed. This is a shortcoming since the movie name is irrelevant in the context of this query. Secondly, the DBPedia Spotlight API returned irrelevant results for few ambiguous movie names, for example, the result for the movie title "JFK" was the DBPedia link of the former American president John F. Kennedy. The results of the API are ranked according to a score. This limitation could be overcome by checking if the result is of the type "movie" or not. Thirdly, some DBPedia entries have missing, redundant or inconsistent keys. There is also another concern that Wikipedia entries could be edited by anyone and hence there is a challenge in determining the credibility of the information.

## 6 SUMMARY

It can be seen that with the power of Semantic Web Technology, non-trivial queries can be successfully executed. All the queries implemented in this project cannot be executed with a normal Google search unless there is a specific website or an article dedicated to this specific kind of information. Also, an existing dataset was enriched with more information by linking the entities with their corresponding entries in the Linked Open Data (here, DBPedia). It was also observed that Semantic Web Technology could be used to greatly empower recommender systems and by incorporating them with machine learning models, better results could be obtained.

## A CODE

The source code for this project can be found here: https://github.com/ajktym94/SPARQL_LOD_MOVIES

The RDF dataset was constructed using the Python notebook named `create_movie_rdf.ipynb` and all the SPARQL queries are in the file `main.py`

## REFERENCES

[1] Tim Berners Lee. 2006. Linked Data - Design Issues. https://www.w3.org/DesignIssues/LinkedData.html