Semantic Web Technologies: An Algorithmic Approach for Tabular Data Transformation, SPARQL Querying, and Ontology Alignment

Khalid Kadri, Alexander Heckmann

I. Abstract

The undertaking of this project has involved a series of interconnected tasks centered around the conversion of tabular data into a more versatile and sophisticated knowledge graph. The project has seen the implementation of SPARQL for queries and reasoning and the alignment of ontologies. To accomplish these tasks, we have made use of Python libraries like Pandas for data manipulation, rdflib for the creation and manipulation of RDF graphs, SPARQLWrapper for the execution of SPARQL queries, and Owlready2 for ontology management and reasoning.

II. About the dataset

The dataset, is a robust collection of data drawn from a comprehensive Kaggle dataset, which focuses on pizza restaurants and the variety of pizzas they sell. This version of the dataset, specifically curated for this project, comprises 500 rows of data. It is a rich repository of information, painting a detailed picture of the pizza restaurant landscape.

Each row in the dataset represents a unique data point, capturing information about a specific pizza restaurant. The attributes contained within these data points are multifaceted and informative. They may include the name of the restaurant, its geographical location (city, state, country), the type of pizzas on the menu, specific pizza names, descriptions, and potentially, additional information like price or customer rating. This assortment of information provides a wide canvas for data analysis and knowledge extraction.

III. Tabular Data to Knowledge Graph:

The transformation of tabular data to be able to insert it into a knowledge graph is a critical step in facilitating semantic queries and more profound data analysis. It involved multiple preprocessing steps with mapping of the tabular data to that ontology.

First, we applied basic text processing to the dataset. In this step we just keep ASCII characters to make sure string values in the dataset are compatible with the ontology. We replaced whitespaces and most other special characters with underscores to have a simple way of generating URIs.

For URI generation of instance names, these simple underscore strings were used, although for restaurants, menu items, addresses, and item values these URIs got a bit more sophisticated. For restaurants, we concatenated the name with the city the restaurant is in. Menu item names were then concatenated to this output to give unique item URIs. Addresses got unique URIs by concatenating the actual address with the city it is located in. Item values have the most sophisticated, although still simple approach: if the amount or the currency are not found, then nothing is added as a URI, if there is, then the concatenation of the two is used as URI.

For the ingredients, we wrote down all classes in the given ontology as a dictionary and looped through all key values in the dict, using the key as the URI for the instance, and an upper camel case version of the key as the class value. Therefore, the URI for the class was generated on the fly instead of beforehand, effectively reducing computational efforts. The dict was chosen as a way of listing ingredients so that at a later point a menu item can be assigned to a given type based on the ingredients it has.

For inserting currencies, we looped through all unique values in the dataset that are valid instead of the whole dataset to further reduce computational efforts.

Then we created the sophisticated portion of our algorithm that iterates through the dataset and maps the data to the ontology. There we insert triples of menu item instances, ingredients, associated types, values, associated restaurants, their locations, and their categories as the type. We found a simple way of finding matching elements for the complex task of adding ingredients, pizza types, and restaurant categories. For restaurant categories we created an array to compare the current row's category with each element in the array. If none is found, then it's assigned the abstract Restaurant class. The same approach applies for special pizza names, although slightly different. We created a dict containing special names that maps that name to an appropriate type. This is used to give items the correct type, but also to filter out menu items that are not actually pizza. Then we also search for ingredients in both the description and the title and use that previously mentioned ingredients dict to map ingredients to pizza types. If no match is found, the abstract Pizza class will be used.

Once the basic mapping was established, the next phase involved making our ontology interoperable and reusable, we utilized existing vocabularies wherever possible. This strategy helps to promote interoperability and the integration and linking of our data with other data available on the web. We used DBPedia lookups for locations where results were applicable, in the same loop as inserting all the other information. To reduce the multiple lookups of the same items, we implemented a simple flag for this to not happen - as DBPedia lookups take quite some time.

In the end, we created a simple, yet computationally efficient algorithm for transforming data into a compatible format for our ontology.

The output was an RDF graph that represented the same information as the original tabular data but in a semantically rich, linked format with example instances where possible. The RDF data was then serialized into a Turtle (ttl) file.

Following the generation of the RDF data, we proceeded to perform reasoning on the ontology and the RDF data [1]. The reasoning process allowed us to infer new information based on the semantics defined in our ontology. The output of this process was an extended graph that contained both the original and inferred information, saved again in Turtle format.

IV. SPARQL and Reasoning

With the RDF data generated, we embarked on the second part of the process—querying and reasoning over the created knowledge graph. This task was accomplished using SPARQL, a powerful, SQL-like query language specifically designed for RDF.

We started with a simple SPARQL query consisting of a triple pattern and a FILTER clause See Figure 1. This query was designed to return all instances of menu items have a value assigned.

```
query_1 = """PREFIX cw: <http://www.semanticweb.org/city/in3067-inm713/2023/restaurants#>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

    SELECT DISTINCT ?menuitem
    WHERE
    {
    ?menuitem rdf:type cw:MenuItem .
    FILTER EXISTS {?menuitem cw:hasValue ?value}
    }

    """

save_sparql_results(query_1, "1")
```

Figure 1: A query with at least a triple pattern and a FILTER

We then expanded the complexity of our queries by incorporating the AVG function, alongside a FILTER clause, see Figure 2. This second query was designed to determine the average price of low-cost pizzas, i.e. pizzas costing less than $10.

```
# give the average price for cheap pizzas (<10$)
query_2 = """PREFIX cw: <http://www.semanticweb.org/city/in3067-inm713/2023/restaurants#>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

    SELECT (AVG(?amount) AS ?avg_pizza_price)
    WHERE
    {
    ?menuitem rdf:type cw:MenuItem .
    ?menuitem cw:hasValue ?value .
    ?value cw:amount ?amount
    FILTER (?amount < 10)
    }

    """

save_sparql_results(query_2, "2")
```

Figure 2: A query that uses at least one triple pattern, a FILTER and AVG function

Next, we built a query that employed grouping, aggregation, and filtering . Specifically, this query was designed to determine the average price of low-cost pizzas, grouped by their type, see Figure 3.

We then created a query that, in addition to grouping and aggregation, ordered the results according to two variables. In our case, this query returns the average price for cheap eats of type Pizza, grouped by the restaurant and sorted by the average price and the restaurant's name, see Figure 4.

```
# give the average price for cheap pizzas (<10$) that are named, grouped by type
query_3 = """PREFIX cw: <http://www.semanticweb.org/city/in3067-inm713/2023/restaurants#>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

    SELECT ?type (AVG(?amount) AS ?avg_pizza_price)
    WHERE
    {
    ?menuitem rdf:type ?type .
    ?type rdfs:subClassOf cw:NamedPizza .
    ?menuitem cw:hasValue ?value .
    ?value cw:amount ?amount .
    FILTER (?amount < 10) .
    }
    GROUP BY ?type

    """

save_sparql_results(query_3, "3")
```

Figure 3: A query that groups results, uses aggregates, and filters the results

```
# give avg price for cheap pizzas (<10$) grouped by restaurants, ordered by avg price and name
query_4 = """PREFIX cw: <http://www.semanticweb.org/city/in3067-inm713/2023/restaurants#>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

    SELECT ?name (AVG(?amount) AS ?avg_pizza_price)
    WHERE
    {
    ?menuitem rdf:type cw:Pizza .
    ?menuitem cw:hasValue ?value .
    ?value cw:amount ?amount .
    ?menuitem cw:servedInRestaurant ?restaurant .
    ?restaurant cw:restaurantName ?name .
    FILTER (?amount < 10) .
    }
    GROUP BY ?restaurant
    ORDER BY ?avg_pizza_price ?name

    """

save_sparql_results(query_4, "4")
```

Figure 4: A different query from SPARQL.3 that group results, uses aggregates, filters the results and orders the results according to two variables

Finally, we constructed a query that utilized the UNION graph pattern and negation. This query returned all named pizzas and pizzas by style that didn't have a price, see Figure 5.

```
# return all named pizzas and pizzas by style that don't have a price
query_5 = """PREFIX cw: <http://www.semanticweb.org/city/in3067-inm713/2023/restaurants#>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

    SELECT ?name
    WHERE
    {
        {
            ?item rdf:type cw:NamedPizza .
            ?item cw:itemName ?name .
        }
        UNION
        {
            ?item rdf:type cw:PizzaByStyle .
            ?item cw:itemName ?name
        }
    FILTER NOT EXISTS {?item cw:hasValue ?value}
    }

    """

save_sparql_results(query_5, "5")
```

Figure 5: A query that uses the Union graph pattern and negation

In summary, we provided several SparQL queries of varying complexity to retrieve instances from our previous task of algorithmic insertion of instances and therefore helps with validation of results.

This process of transforming and querying data enables more complex data exploration, discovery, and analysis. By representing data as a knowledge graph, we can leverage the semantic relationships between data points to infer new knowledge, as well as to answer complex queries that would be challenging or impossible with traditional relational databases. The ability to incorporate existing vocabularies ensures the interoperability and reusability of our data, which will be tackled in the next section.

Moreover, the use of SPARQL for querying allows us to perform sophisticated data manipulations, including filtering, aggregation, grouping, and ordering. SPARQL's flexibility in handling complex queries across different groups and with exclusion criteria was also demonstrated.

In conclusion, this process underscores the power of semantic web technologies in enhancing data analysis and knowledge discovery. The use of ontologies and SPARQL provides a robust, flexible, and powerful toolset for working with data in a semantically rich, linked, and interoperable manner.

V. Onotology Alignment

In this phase of the project, we focused on Ontology Alignment, a process that involves identifying correspondences between two different ontologies. Our task revolved around two datasets: a city-restaurant ontology (represented as cw_onto in the code), and a separate pizza.owl ontology.

In the initial stages of this task, we sought to identify equivalences between the entities of the two input ontologies. To achieve this, we employed a variety of Python libraries, including Owlready2, to extract ontology classes, data properties, and object properties from both datasets. We then used a similarity-checking algorithm to find corresponding entities between the two ontologies based on their names using difflib's similarity score [2]. The discovered correspondences were categorized under classes, data properties, and object properties, and were then saved as triples in Turtle format using owl:equivalentClass and owl:equivalentProperty as predicates for these equivalence triples. [3]

With the equivalences established, our next task was to perform reasoning using a multitude of sources to create a single graph or RDF model. This combined graph included the city-restaurant ontology, the pizza.owl ontology, the computed alignment from our previous step, and the generated data from our city-restaurant dataset. To compile these sources, we used the rdflib library to parse the files and load them into a single graph. We then expanded the graph using Owlready2's deductive closure, which performed a reasoning process based on OWL's semantics, effectively integrating the disparate sources into a unified knowledge graph.

Finally, we aimed to demonstrate the efficacy of our ontology alignment process by creating a SPARQL query that used the pizza.owl vocabulary to retrieve information from our generated city-restaurant data. This query was designed to extract the names of items classified as NamedPizza in the pizza.owl ontology from our combined graph. The successful execution of this query underscored the value of ontology alignment in enabling unified querying across multiple datasets with diverse vocabularies and structures.

To summarize, our Ontology Alignment task involved the identification of equivalent entities across two distinct ontologies, the creation of a single, unified graph using these ontologies, their computed alignment, and generated data, and the use of a SPARQL query to demonstrate the power of this unified approach. These steps enabled us to harmonize the two datasets, expanding the scope of possible queries and enhancing the richness of the information that could be extracted.

VI. Conclusion

This project has served as a practical exploration of semantic web technologies, demonstrating their utility in transforming, querying, and aligning diverse data sets. Through the conversion of tabular data into a knowledge graph, we were able to illustrate the inherent flexibility and expressiveness of ontologies, facilitating more intricate understanding and exploration of the data.

The power of SPARQL in querying such data was also made evident. With its extensive features, such as aggregation, ordering, and filtering, SPARQL provides a powerful toolset for digging into complex semantic data, going beyond the capabilities of traditional SQL-like languages.

However, the project didn't stop at querying a single ontology. We also demonstrated the potential of ontology alignment, linking our ontology with an external one. This step highlighted how semantic technologies can enable interoperability between disparate data sources, providing an avenue for more holistic and integrated analyses.

In the end, the project underscored the significant potential of semantic technologies in managing and interpreting complex data. By transforming tabular data into a knowledge graph, exploiting SPARQL for advanced queries, and aligning different ontologies, we could glean more meaningful insights from our data. These techniques are invaluable in today's data-driven world, where the ability to extract meaningful insights from diverse and complex data sets is key. The lessons learned in this project provide a strong foundation for future endeavors in the realm of semantic data analysis.

References

[1] E. Jimenez-Ruiz, *Reasoning with rdfs semantics and owl 2 rl laboratory6*, INM713 Semantic Web Technologies & Knowledge Graphs, 2023.

[2] Python Software Foundation. "difflib". (2023), [Online]. Available: https://docs.python.org/3/library/difflib.html (visited on 13/05/2023).

[3] E. Jimenez-Ruiz, *Ontology alignment laboratory 7*, INM713 Semantic Web Technologies & Knowledge Graphs, 2023.