Semantic Web Technologies and Knowledge Graphs: Adding Context and Meaning to Tabular Data

Jakob Brown

Abstract

The present paper documents efforts to develop software components that provide a semantic enrichment of a typical tabular data set, in the hopes of extending the possibilities of what can be done using said data, by either deriving more nuanced and meaningful information out of the data on its own, or enabling the harmonious linkage with other data sets, by providing the contextual information necessary to place the insight they respectively hold in relation to each other and the surrounding domain.

About the Data set

The data set used in the present study was the 'Pizza restaurants and Pizzas on their Menus' data set, curated by *Datafiniti*. This data set consists of over 3,500 records of different pizzas sold in the USA and different restaurants. For the most part, save for some missing values, the information provided with each pizza was the restaurant it was served at, the pizzas name as presented on the menu, the description of the pizza as presented on the menu, the price of the item, the city, address, and postcode of the restaurant, the State of the city, and the Country (always the 'us').

Ontology Modelling

The first task was to create an ontology in Protégé, based upon the information in the data set, to neatly classify and relate the types of pizza, their restaurants, locations in terms of city, state, and country, as well as extraneous details such as price and name. The design of this ontology was motivated greatly by the tasks ahead, which included aligning the created ontology with the famous pizza ontology made at Manchester University. SPARQL querying was also necessary to query particular information, and so these specific information needs were reflected in the created ontology's structure.

Classes

Classes comprised of *Pizza, Pizza toppings, Pizza base, Establishment, City, State,* and *Country*. Within the base class there were *Deep dish* and *Thin crust*. Within the toppings class, there was *Vegetarian friendly toppings, meat toppings, and seafood toppings.* A key distinction between types of pizza is whether they are vegetarian or not, and so this information would be key in many lines of investigation using this data and ontology. Therefore, it was deemed necessary to distinguish between these types of pizzas, which were defined by their toppings. Various toppings were assigned to each subclass of topping for the purpose of colour and detail; however, these specific toppings were not actually assigned to the dataset pizzas later on due to the significant effect on computational workload later on. The vegetarian-status of a pizza was determined instead by the subclasses of the *Pizza* class: *Meat pizza, Seafood pizza, Vegetarian pizza,* and *Named pizza.* Specific pizzas such as *Hawaiian, Bianca,* and *Americana* were made subclasses of *Named pizza*. Vegetarian pizza class was made disjoint with meat and seafood class, for obvious logical reasons.

Object Properties

The object properties specified to associate the different class entities together were as follows: *Has ingredient (subclasses: Has base, Has topping), Is ingredient of (subclasses: Is base of, Is topping of), Has location, Is located in, Serves, Served at. Has base* was made functional – as a pizza could only have

one base - and thus *Is base of* was made inverse functional. This was also the case for *Served at* and *Serves*. This was decided as it was clear that each pizza, regardless of being the exact same type of pizza, was going to need to be able to be distinguished, and so that was done by treating each pizza as unique, thus a pizza could only be served at the individual restaurant it is associated with in the data set.

The location object properties allowed for the Russian-doll-like envelopment of geographic information with the same object property, keeping things clean and simple while also reflecting the nature in which smaller locations are related to/subsumed by the larger locations they inhabit and vice versa, e.g. *restaurant x Is located in City x Is located in State x Is located in Country x.* These two inverse object properties could then also be made transitive, making reasoning do the work for linking together locations rather than extra programmatical workload. This decision also allowed for a great degree of flexibility when querying the subsequent knowledge graph in terms of finding locations of establishments or pizzas etc. The *Served at* and *Serves* object properties linked the pizzas themselves to the restaurant, which then could be extrapolated to any of the geo-location data classes. Finally, the location object properties were thought to likely mimic other ontologies that specified geo-location in a similar way, and so could enable easy integration between them and the created ontology, at any level of location, be it city or country.

Data Properties

Data properties were used to attach any extraneous information about the pizzas that were not already captured by the classes or object properties e.g. *Has name, Has address, Has postcode, Has price*. These were all functional properties. Having postcode and price as data properties was a clear choice, but also it conveniently limited any consequences of missing values to nothing more than an extra line of code for each. There were no consequences therefore on reasoning, and so no rows had to be omitted from the data set before transferring the data to a knowledge graph. The *Has name* data property may have been seen as unnecessary, however part of the data pre-processing was linking together the name of the pizza with the restaurant name to create a new entity name e.g. *Bianca Pizza at Sloppy Joe's.* This was done in order to prevent ambiguities between different instances which may have the same pizza name, and to be able to discern the individual observations of a concept from the concept itself, regardless of how common the pizza name was. The *Has name* property therefore allows for just the pizza menu name to be identified should it be needed.

Other facets detailed in the created ontology include annotation properties for each created entity. This includes a novel *Created by* property, used to assign the present author as the creator of the whole ontology. Domains and ranges were specified as necessary. This was done in order to instruct the reasoner where to expect certain objects, predicates and subjects to come from, thus creating a framework of clear logical rules which would prevent unsatisfiable events in the ontology. Price information range was specified as type *xsd:float*, in order to later ascertain averages, and capture the exact values of prices not rounded up to the nearest whole US dollar.

Tabular Data to knowledge Graph

As previously mentioned, item name was transformed by combining the pizza name with the establishment name in order to guarantee individual entity resolution with respect to each of the 3,500 data points. Furthermore, other pre-processing measures taken included the removal of punctuation, capitalisation, or non-ascii characters from the relevant string columns which would find themselves input in to the knowledge graph.

Prior to creating the knowledge graph, any rows that had the official state abbreviations as their state were

transformed to have the full state name as their state. This would later aid greatly in adding triples using dbpedia entity URIs [1] for state to enrich the quality of the data. Finally, several new columns were created (*Vegetarian, Meat, Seafood, Bianca, Margherita, Americana, Hawaiian, Dessert pizza, Thin crust, Deep dish*), which were then one-hot encoded via iterating through each row's item name and item description to identify the presence of a list of buzzwords, each list with a corresponding column e.g. *Seafood* column was fired if observation contained words like *fish, crab etc.* These columns were then iterated over one more time to ensure no disjoint columns like vegetarian and meat were both fired in the same instance, to prevent failure to reason.

An empty graph was initialised via python's *rdfblib*. Iterating through each row of the data set, the prepared information was extracted from the relevant columns combined with the created ontology namespace, and fed into the knowledge graph in the form of triples. Classes such as pizza subclasses and geographical classes were assigned using *RDF:type* predication. Custom object and data properties were then assigned between entities and between entities and their respective row's available information.

With the geographical information, it was possible to enrich the present knowledge graph by using well-established ontologies that already had URIs for geographical locations and using their corresponding entities to entwine with the novel ones made in the created ontology. It was elected to use those of dbpedia [1]. A dbpedia lookup function was adapted and built upon to match locations in the data set with dbpedia URIs if they bore a substantial lexical similarity, and if they were of the same 'type' of location as the one it was being compared to e.g. city. This function was used to once again iterate over the data set: if a match was returned on the city or the state of a given row, this dbpedia URI would be used. If not, then the URI would be manually constructed using the created ontology namespace plus the string of the city or state in the data. For country, as every observation in the data was in the US, the 'http://dbpedia.org/resource/United_States ' URI was used as the country for every observation. Just as with the food classes, these locations were classed as rdf:type jb:city/state etc., as well as linked to each other through the aforementioned object properties *Is located in,* and *Has location*. The data property *Has name* was also provided for the locations, as *rdfs:literals* of the original name given in the data set.

Reasoning and SPARQL Queries

Reasoning was successfully performed on the created ontology and the generated RDF data using *owlrl.OWLRL_Semantics*. Entailments generated were checked in the process, i.e. "*jb:Bianca rdfs:subClassOf jb:Specific_pizza* ." returned true, as would be desired. The subsequent graph was serialized in turtle format. Using this graph, SPARQL queries were then written in order to return particular information, ranging from average price of a particular type of pizza across the data, to the details of all of the restaurants that sold a Bianca pizza. The queries, along with the information need they were written to satisfy, are documented below. The results of the queries can be found in csv format in the coursework submission zip file.

```
qres = g.query(

"""
PREFIX jb: <http://www.semanticweb.org/jake/ontologies/2021/2/jbrown/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?establishment ?address ?postcode (SAMPLE(?city) AS ?city) (SAMPLE(?state) AS ?state) ?country
WHERE {
?establishment jb:Serves ?pizza .
?pizza rdf:type jb:Bianca .
?establishment jb:Has_address ?address .
?establishment jb:Has_postcode ?postcode .
?establishment jb:Is_located_in ?city .
?city rdf:type jb:City .
?city jb:Is_located_in ?state .
?state rdf:type jb:State .
?state jb:Is_located_in ?country .
?country rdf:type jb:Country .

}
GROUP BY ?establishment

""")
```

Figure 1. Subtask SPARQL.2 - Return all the details of the restaurants that sell pizzas without tomate (i.e. pizza bianca).

```
qres = g.query(

"""
PREFIX jb: <http://www.semanticweb.org/jake/ontologies/2021/2
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT (AVG(?price) AS ?avgprice)
WHERE
{
?pizza rdf:type jb:Margherita .
?pizza jb:Has_price ?price . }
```

Figure 2. Subtask SPARQL.3 – Return the average price of a Margherita pizza.

```
qres = g.query(

"""
PREFIX jb: <http://www.semanticweb.org/jake/ontologies/2021/2/jbrown/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT (COUNT(DISTINCT ?restaurant) AS ?count) ?city ?state
WHERE {
?restaurant jb:Is_located_in ?city .
?city rdf:type jb:City .
?city jb:Is_located_in ?state .
?state rdf:type jb:State .
}
GROUP BY ?city
ORDER BY ?state DESC(?count)

""")
```

Figure 3. Subtask SPARQL.4 Return number of restaurants by city, sorted by state and number of restaurants.

```
qres = g.query(

"""
PREFIX jb: <http://www.semanticweb.org/jake/ontologies/2021/2/jbrown/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?res
WHERE {
?res rdf:type jb:Establishment .
FILTER NOT EXISTS {?res jb:Has_postcode ?postcode}
}


""")
```

Figure 4. Subtask SPARQL.5 Return the list of restaurants with missing postcode.

Ontology Alignment

In order to demonstrate the ability to link together different ontology and subsequently knowledge-graph-based data with ease, a simple, element-level, syntactic alignment was then performed between the created ontology and the aforementioned pizza ontology on which the created one was based. This was done by parsing both ontologies in to a knowledge graph, before adding logical equivalence triples using *OWL: OWL:equivalent_class* and *OWL:equivalent_property*.

In order to locate equivalent classes and properties, the classes and properties of both ontologies were both extracted to lists to first look over and see what kind of equivalences could be drawn and by what means. Following that, the syntax of the created ontology was transformed to match that of the *pizza.owl* ontology. This way, when looking for lexical similarity, perfect matches would be easy to find. Iterating over the list of classes and properties (separately), each entity was checked against all entities of the converse ontology. Classes and properties were checked against each other separately, in order to prevent heterogenous matches which would not have been appropriate for this kind of ontology, as well as reducing computational expense by reducing the number of comparisons needing to be made (although these ontologies were not large enough to cause any workload/time problems anyway).

Perfect lexical matches were immediately written as equivalent classes/properties, with the exception of some filters which prevented false matches being drawn (e.g. the word ‘topping’ was stripped from the pizza.owl ontology toppings to allow matches with the created ontology toppings which did not have the word ‘topping’ after them. This would then have created a false match between pizza.owl’s ‘pizzaTopping’ and the created ontology’s ‘Pizza’ class.) The remainder of the entities were put through a lexical matcher (Levenstein’s Jaro_Winkler), producing a numerical value for how well-matched any two entities were. This measure was preferred to the acutal Levenshtein distance, as it provided a normalised score between 0 and 1, which is much easier to apply a consistent threshold to than the Levenshtein distance, which calculates the number of edits (insertions, substitutions, deletions) necessary to convert one string to another. A relatively high threshold was applied to this value, which were then displayed manually to the programmer in any case to ensure no false matches were allowed to slip through the alignment process. The correct alignments were added as triples, totalling 26 equivalences found between the ontologies. More matches were available to be equated, were it not for the shortcomings of the syntactic approach, which only takes lexical characteristics into account. An external approach, making use of outside information, as well

as employment of formal semantics may well have yielded more matches. The alignments were then saved.

Reasoning was performed for i. the created ontology, ii. The *pizza.owl* ontology, and iii. the computed alignment (without the data). The number of unsatisfiable classes (*OWL:Nothing)* was counted for each. It transpired that there were unsatisfiable classes in all 3 (113, 397, and 491 respectively). However, this did not prevent reasoning from being successfully performed, programmatically or in Protégé. It is not known why this is the case, only that whatever facts/triples were including *owl:nothing* entities, they did not cause an unsatisfiable clause in the ontology as a whole. A new graph was initialised, and into it were parsed the created ontology, the pizza.owl ontology, the generated RDF data, and the ontology alignments. Reasoning was performed successfully, allowing the querying of the data generated using the created ontology, using the pizza.owl ontology terminology. The query, stated and illustrated in SPARQL below, returned the ostensibly correct results, which can be verified in the csv file *"OA_meatypizza_query_results.csv".*

```
g = Graph()
g.parse('aligned_ontology_with_data_post_reasoning_owlrl.ttl', format

qres = g.query(
"""
PREFIX jb: <http://www.semanticweb.org/jake/ontologies/2021/2/jbrown/
PREFIX pizza: <http://www.co-ode.org/ontologies/pizza#>

SELECT DISTINCT ?pizzas
WHERE {

?pizzas rdf:type pizza:MeatyPizza .

}


""")
```

Figure 5. Subtask OA.2.b - Create a query to return the pizzas with type pizza:MeatyPizza.

This successful query indicates a successful cohesion between the two ontologies, in particular an accurate equivalence drawn between the created ontology's subclass of pizza *'http://www.semanticweb.org/jake/ontologies/2021/2/jbrown/Meat_pizza',* and pizza.owl's *'http://www.co-ode.org/ontologies/pizza#MeatyPizza'.* With more detail added to the created ontology in terms of ingredients/toppings, even more detailed queries could be made, mixing together the two ontologies as needed. Or further yet, more ontologies could be aligned, such as the dbpedia ontology which was used earlier for geo-location terms.

Ontology Embedding

Using the created ontology (the created ontology with the generated data was too computationally expensive), each of the entities were vectorised using Owl2Vec* [2]. This process entails random walks over the words of the data, word embedding, and producing a vectorisation of each of the words embedded by training a *gensim word2vec* language model. Once these vectors were produced, the cosine similarity between 2 vectors could be calculated using *gensim.keyedVectors.* Similarity scores were examined for 5 pairs of entities (*table 1*). As can be seen, the cosine similarity between the vectors of these pairs of words vary in a way that is to be expected. In pair 1, little is similar about the two words: they are different in the letters they contain and their length. Moving forward, 'salami' and 'data' are more similar in length and share some of the same vowel structure. With pair 3 onwards, words are deemed by *gensim* to be very similar indeed: 'Pizza_base' and 'vegetarian_pizza' have a word in common, as well as the linking underscore, so follow the same rare structure. The highest score is unsurprisingly that of 'Vegetarian_pizza', and 'vegetarian_pizza': a score of over 99 indicates the smallest possible difference between the two words, which is indeed the case.

Word 1	Word 2	Cosine Similarity
‘Beef’	‘Miscellaneous’	0.664
‘data’	‘salami’	0.776
‘Pizza_base’	‘Vegetarian_pizza’	0.989
‘Vegetarian_pizza’	‘vegetarian_pizza’	0.9902
‘vegetable’	‘vegetarian’	0.944

Table 1. Cosine similarity scores for 5 pairs of words from created ontology, vectorised through Owl2Vec.*

Subsequently, a K-means clustering algorithm was applied to entire vectorised ontology, in the hope of inferring some semantic differentiation. A visual silhouette Analysis and principal component analysis (PCA) (*figure 6 (see appendix)*) allowed it to be seen that 6 was the optimal K-value for discerning the words, with a silhouette score of 0.5417. Some vague syntactical patterns could be argued, though nothing too certain: Words of a certain length and structure were often clustered together (e.g. ‘crab and ‘chicken’; and all much longer entities due to the namespace being part of the word were clustered together as well (all localised to the bottom left area of the PCA space). Although not much semantic insight was yielded, this is an exciting possibility: that with some semantic information layered on top of the increasingly sophisticated lexical vectorisation techniques, the data plugged in to a clustering algorithm such as K-means could return clusters perfectly segmented by their meaning or associations to one another.

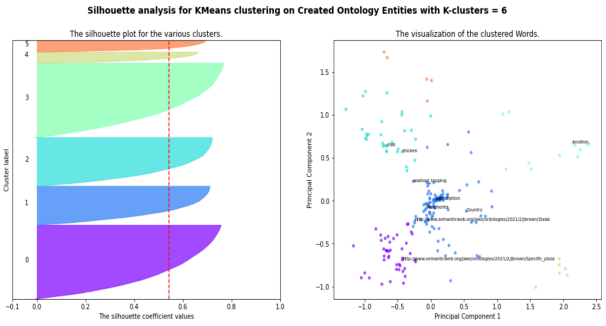

Figure 6. Silhouette analysis of K-means clustering of word vectors (k = 6) (left), and Partially labelled PCA visualisation of word vectors in normalised 2-D space (right).

References

[4] Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., Cyganiak, R. and Ives, Z., 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web* (pp. 722-735). Springer, Berlin, Heidelberg.

[5] Chen, J., Hu, P., Jimenez-Ruiz, E., Holter, O.M., Antonyrajah, D. and Horrocks, I., 2020. OWL2Vec*: Embedding of OWL ontologies. *arXiv preprint arXiv:2009.14654*.

Code References

[1] Christen, P. 2002. Stringcmp.py. Lab 6, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

[2] Jimenez-Ruiz, E. 2021. LoadEmbeddings.py. Lab 9, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

[3] Jimenez-Ruiz, E. 2021. AccessEntityLabels.py. Lab 8, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

[4] Jimenez-Ruiz, E. 2021. OWLReasoning.py. Lab 7, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

[5] Jimenez-Ruiz, E. 2021. LoadOntology.py. Lab 5, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

[6] Jimenez-Ruiz, E. 2020. OWL2Vec_Standalone.py. Lab 9, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

[7] Jimenez-Ruiz, E. 2019. Entity.py. Lab 6, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

[8] Jimenez-Ruiz, E. 2019. Lookup.py. Lab 6, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

[9] Jimenez-Ruiz, E. 2019. Onto_access.py. Lab 5, INM713 Semantic Web Technologies & Knowledge Graphs. City, University of London

Appendix

Figure 6 (enlarged)