Google Summer of Code

Towards a Neural Extraction Framework

Mentors: *Tommaso Soru, Ziwei XU*

Personal information

Name: Abdulsobur Oyewale

Preferred name: Abdulsobur

Email: Abdulsoburoyewale@gmail.com

University: Federal University of Technology Akure

Discipline: Computer Science

LinkedIn

Project

● Problem Description

The DBpedia property *dbo:wikiPageWikiLink* helps us to see for the articles which point to the Berlin_Wall article such that their text contains hyperlink for the Berlin_Wall article.

As of March 2024, when we execute the SPARQL query given below in the DBpedia endpoint, we get 363 links as the result.

```
SELECT (COUNT(?wikiLink) as ?count)
WHERE {
<http://dbpedia.org/resource/Berlin_Wall> dbo:wikiPageWikiLink ?wikiLink. }
```

The result we get is the one below. So, we have 363 base entities related to the Berlin_Wall article via the predicate *dbo:wikiPageWikiLink.*

SPARQL | HTML5 table

count
363

But only a few of these base entities are linked from Berlin_Wall via another predicate as well. To check this, we run the following query.

```
select ?link ?pred ?pred_inv where {
<http://dbpedia.org/resource/Berlin_Wall>
<http://dbpedia.org/ontology/wikiPageWikiLink> ?link .
optional {
<http://dbpedia.org/resource/Berlin_Wall> ?pred ?link
filter(?pred != <http://dbpedia.org/ontology/wikiPageWikiLink>) }
optional {
?link ?pred_inv <http://dbpedia.org/resource/Berlin_Wall>
filter(?pred_inv != <http://dbpedia.org/ontology/wikiPageWikiLink>) }
} order by ?link
```

And we get the result given as below in the html file:

Entities linked to Berlin_Wall by predicate other than dbo:wikiPageWikiLink.

So, there are 10 entities that are also linked via some other predicates as well to Berlin_Wall.

The relationship of the remaining articles is not much clear and to extract such relations, tables and infoboxes are currently used. In the previous iteration of the project, attempt was

to extract these relationships using the unstructured or textual content of the wikipedia page using techniques like **entity extraction**, **relation extraction** and **entity resolution**.

Last year the first end-2-end framework was developed for this project, while this year I would be improving the project by devising several methods to improve the current pipeline.

Basic Ideology

I have completed the warm-up tasks in the discussion forum. I have explored some recent research in Ontology Enrichment, ML pipeline scaling, and also checked with the implementations in the previous iterations of this project. Below are some resources I referred when exploring about this project:

1. **DBpedia** ontology enrichment for **inconsistency** detection
2. Mechanism for inconsistency correction in the DBPedia Live
3. Simple algorithms for predicate suggestions using similarity and co-occurrence
4. Measuring accuracy of triples in knowledge graphs
5. **UniRel**: Unified representation and interaction for joint relational triple extraction
6. **T2kg**: An end-to-end system for creating knowledge graph from unstructured text
7. Learning to Map Natural Language Statements into Knowledge Base Representations for Knowledge Base Construction
8. Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources (Short Paper)

Detailed approach

Predicate Suggestion

Simple algorithms for predicate suggestions using similarity and co-occurrence

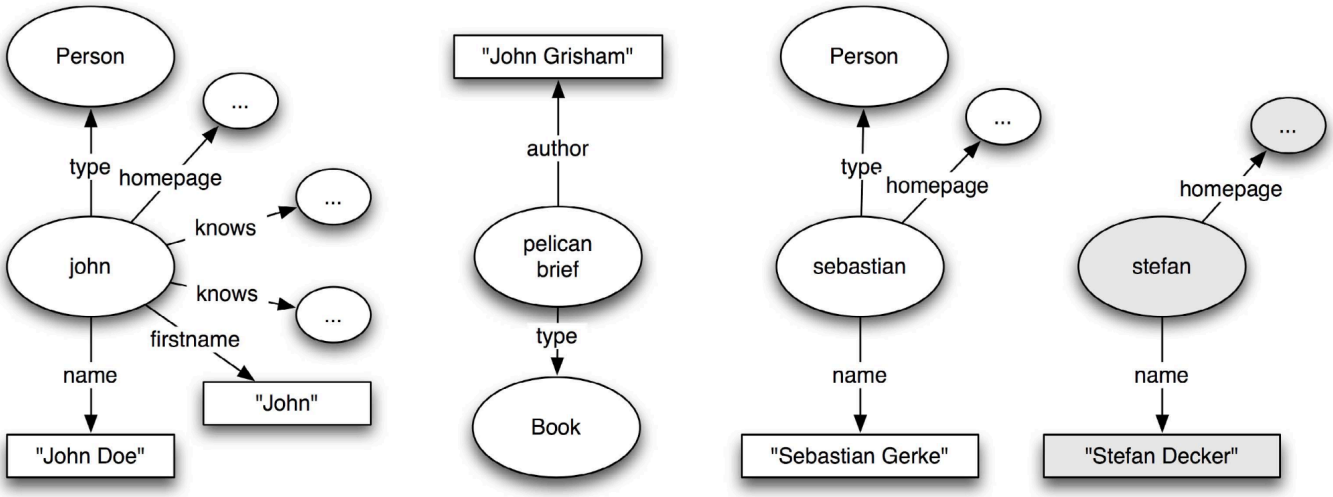
Co-occurrence-Based Algorithm

The general idea of the co-occurrence-based algorithm is to approximate resource similarity through the co-occurrence of predicates. We consider the pair-wise occurrences of predicates, suggest predicate candidates for each pairwise occurrence, and combine these candidates through intersection.

The algorithm is based on association rule mining used for recommendations in e.g. online stores: when buying one book, other books that are often bought together with this book are recommended. In our case, books are replaced by predicates and shopping transactions by resources.

The example below will give better details of the algorithm:

For example, the figure below shows a simple knowledge base: the person “John”, with his name, some friends, and homepage, the book “The Pelican Brief”, with its title and author, and the person “Stefan”, with his name. Adding the person “Sebastian” and some more statements about John, as shown. We want to suggest relevant predicates and further predicates for “Stefan” based only on the given graph.



Simple algorithms for predicate suggestions using similarity and co-occurrence
Eyal Oren, Sebastian Gerke, Stefan Decker

In the first step we will calculate usage statistics of predicates in the DBpedia knowledge base. We count for each predicate, the resources that use this predicate, as defined below:

$$occ(p) = |\{v \in V | p \in Lo(v)\}|$$

Secondly, we count for each pair of predicates, the number of times they co-occur together in the same resource:

$$coocc(p_1, p_2) = |\{v \in V | p_1 \in Lo(v) \wedge p_2 \in Lo(v)\}|$$

Predicate occurrence and co-occurrence frequency

predicate	*freq.*
type	3
name	2
knows	1
homepage	2
firstname	1
author	1

(a) occurrence

	type	name	knows	homepage	firstname	author
type	3	2	1	2	1	1
name	2	2	1	2	1	0
knows	1	1	1	1	1	0
homepage	2	2	1	2	1	0
firstname	1	1	1	1	1	0
author	1	0	0	0	0	0

(b) co-occurrence

To compute suggestions for a given resource. We consider all predicates in the DBpedia knowledge base that occur more than once with each of the predicates from “Stefan” as suggestion candidates. In our example, the predicates “type”, “knows”, and “firstname” are candidates for the resource “Stefan”. For each candidate we calculate our confidence in suggesting it.

Each constituent is computed as shown in equation below: the confidence for suggesting any p2 based on the existence of a p1 is given as the co-occurrence frequency of p1 and p2 relative to the occurrence frequency of p1 by itself. In the example, p2 , the candidate, would be “type”, “knows”, or “firstname”, and p1 , the existing predicates, would be “name” and “homepage”. Intuitively, we consider a relatively frequent co-occurrence as evidence for predicting p2 .

$$confidence(p_1 \Rightarrow p_2) = coocc(p_1, p_2) / occ(p_1)$$

In our example, as shown in the table below, “type” co-occurs with both predicates of “Stefan” 100% of the time, whereas the two other candidates (“knows” and “firstname”) co-occur only 50% of the time with each of the predicates of “Stefan”. We rank each candidate by the combined (unweighted) confidence: in this example, “type” will be ranked first, with a combined confidence of 100%, and the other two second, with a combined confidence of 25%.

candidate	*name*	*homepage*	*confidence*
type	1.0	1.0	1.0
knows	0.5	0.5	0.25
firstname	0.5	0.5	0.25

Alternatively we can also solve this task by implementing the Learning to Map Natural Language Statements into Knowledge Base Representations for Knowledge Base Construction paper where a relational mapping that covers 7,361 of 9,171 frequent ReVerb patterns with 629 of 634 frequent DBpedia predicates.

Therefore we have two different options in solving this task:

- Using the Co-occurrence-Based Algorithm
- Using the relational mapping covered in the above paper

Scale current models to be efficient enough to millions of entities.

Currently the pipeline is slow. Even on a GPU it's not lightning fast, the model takes time for inference. It takes around **2 seconds for one triple**. Therefore it would take an unreasonable amount of time for an entire wiki page. Therefore I propose we apply **streaming and message-brokering techniques** to improve the efficiency and scalability of our entire pipeline.

By introducing streaming and message-brokering components, we can process data in transit, allowing different stages of the pipeline to operate concurrently and asynchronously. This approach can help reduce overall processing latency and enable better utilization of resources.

To implement this, we can make use of any of the various streaming data ingestion frameworks. However, due to my own experience I would like to make use of **Apache Kafka,** also Kafka provides robust support for building scalable, fault-tolerant, and real-time data pipelines.. This will allow us to continuously receive and process data as it becomes available, rather than waiting for entire batches to accumulate before processing.

To integrate kafka into our pipeline, we can:

- Use the kafka-python library, a pure python client for Apache Kafka, to interact with Kafka from our pipeline.
- Set up the Kafka cluster by keeping the ZooKeeper and Kafka brokers running.
- Write a python script to act as our Kafka producer. This script will be what will publish the text data to a Kafka topic.
- We then create a Kafka consumer script to subscribe to the kafka topic and process the data. This script will consume the text data published to the kafka topic and perform processing tasks(entity extraction, relation extraction, e.t.c).
- We can also scale kafka by adding more brokers and partitions to handle increased data throughput and also optimize Kafka configurations based on our specific requirements for performance, reliability, and resource utilization.

A code example of the above explanation is available here in this colab notebook.

Also, we can also make use of the **Spark NLP**. Spark NLP is a state-of-the-art Natural Language Processing library built on top of Apache Spark. It provides simple, performant &

accurate NLP annotations for machine learning pipelines that scale easily in a distributed environment. Spark NLP allows us to process large volumes of text data in parallel, ensuring efficient utilization of resources and improved processing throughput. SparkNLP provides various documentation that can easily allow us to implement it into our current neural extraction framework pipeline.
Therefore to solve the scalability and efficiency problem. We have two options:

- Using Apache Kafka for scaling our pipeline.
- Using the SparkNLP to scale in a distributed environment

Categorize extracted relations with respect to their semantics

Categorizing semantics requires understanding the properties and the relation exhibited. Following how property logic is defined in owl, we can check for a relation property by following the rules below:

- **Symmetric**: if a pair (x, y) is an instance of P, then the pair (y, x) is also an instance of P.
- **Transitive**: If a pair (x, y) is an instance of P, and the pair (y, z) is also an instance of P, then we can infer the pair (x, z) is an instance of P.
- **Reflexive**: if there exist x, then (x, P, x)
- **Antisymmetric**: if a pair (x, y) is an instance of P, then the pair (y, x) is not an instance of P

However, we can focus on solving this by using a language model that helps us to understand the relationship between entities in a statement. This language model will further help us to categorize the extracted relations with respect to their semantics. As it seems there are no available models that classify relationships as reflexive, transitive, e.t.c.

I propose we use a language model of zero shot classifier. Zero shot text classification is a task in natural language processing where a model is trained on a set of labeled examples but is then able to classify new examples from previously unseen classes. Zero shot classifier models allow us to classify our statements based on sequence given to it to classify.

Therefore we provide the model with a prompt and a sequence of text that describes what our model should do, in natural language. An example below shows a zero-shot prompt for classifying the property of a sequence of text using the "facebook bart-large-mnli". The default setting for the Zero class classification is to score the best classifier of what the test is about, our goal will be to find the best hypothesis to get the accurate result to achieve our aim.

```
[5] from numpy import argmax

    sequence_to_classify = "The Peaceful Revolution enabled the reunification of Germany in October 1990."
    candidate_labels = ['reflexive', 'transitive', 'irreflexive', 'symmetric']
    result = classifier(sequence_to_classify, candidate_labels)

    scores = result['scores']
    classes = result['labels']

    BEST_INDEX = argmax(scores)
    predicted_class = classes[BEST_INDEX]
    predicted_class

    'transitive'
```

From the example above, The Peaceful Revolution enabled the reunification of Germany, which means it facilitated the reunification process. This implies that if A enables B and B enables C, then A enables C. In this case, if the Peaceful Revolution enabled the reunification of Germany, and the reunification of Germany occurred in October 1990, then it can be inferred that the Peaceful Revolution indirectly enabled the reunification in October 1990. The zero shot classifier was able to deduce that the statement presented to it is transitive.

Presently, due to the short term of completing this proposal I have only been able to make use of the BART model to test out this task. However, I will keep exploring to find better pretrained models or fine-tuned models that can be used to achieve our aim in this specific domain.

Validate generated triples against the DBpedia ontology

Mechanism for inconsistency correction in the DBPedia Live

An RDF triple t = (s, p, o) consists of three elements in which s is the subject of the triple, p is the predicate of the relation between s and o defined as a property; and o is known as the object of the triple. And an Ontology iOntology refers to a collection of concepts and axioms that the triples must obey. Therefore validating a triple against the ontology means to check that every **domain-range** rule that exists is satisfied.

Range - For every property p the ontology defines a set of classes range(p). The set range(p) is a set of classes that the property p is limited to have for an object as defined by the ontology. Given a triple (s, p, o) the ontology requires that o is of the same class of at least one of the classes defined in range(p) for the triple to be consistent.

Domain - For every property p the ontology defines a set of domain(p) classes. The subjects s of every triple (s, p, o) must be of one of the classes defined in domain(p) for the triple to be consistent.

Inconsistency - We characterize an inconsistency in a triplestore as a triple that contains

conflicting information with the underlying ontology used to define the classes in which triples must respect. We define two types of inconsistencies organizing the inconsistency classes in two main groups: range violation and domain violation:

- **Range violation Inconsistency** - A triple is deemed inconsistent in its range if, given a triple t = (s, p, o), o for the given p is such that o 6∈ range(p), such that range(p) the values accepted by p for a relation. Therefore, the triple is range inconsistent when the value of the triple is not part of the range of the predicate.
  Example: Consider the following triple (Lincoln, dbo:birthPlace, University of Alabama). This RDF triple indicates that the birthplace of Lincoln is the University of Alabama. However, the property “dbo:birthPlace " has a defined range that contains only the ontology class “dbo:Place”. Therefore, for this property, the object must be of the type “Place”, which is not the case. The resource “University of Alabama” is of the type ”dbo:University” and none of it’s super classes are subclasses of “dbo:Place”, turning this triple inconsistent with the ontology.

- **Domain violation Inconsistency** - A triple is deemed inconsistent in its domain if, given (s, p, o), s for the given p is such that o 6∈ domain(p), such that domain(p) defines the required subjects for the property p according to the underlying ontology. In this sense, the triple is domain inconsistent when the subject s is not part of the possible values that the predicate allows in its ontology.
  Example: Given the triple (Spock, dbo:birthPlace, California). The property “dbo:birthPlace” contains as domain only the type “dbo:Person”. Since the resource ”Spock” is of the type ”dbo:FictionalCharacter”, which is not the type required of the property’s domain set, the given triple is considered inconsistent.

Given the definitions below, our algorithm will query the DBpedia Ontology with SPARQL and be in the code format below:

Require: s, o, p

1. *range(p) ← queryDBpedia(p, range)*
2. ***if** range(p) ⊂/ types(o) **then***
   (s, p, o) is range inconsistent
3. ***end if***
4. *domain(p) ← queryDBpedia(p, domain)*
5. ***if** domain(p) ⊂/ types(s) **then***
   (s, p, o) is domain inconsistent
6. ***end if***

With the above solution we can be able to validate if our generated triples align with the DBpedia ontology.

Also, during my research I came across two tools named SHEx and SHACL. I’m exploring the SHACL implementation and also joined their community to verify if this particular task can be achieved using their tool. I have been given a hint and referred to several resources to get me started. However, it seems we will have to explicitly write out the domain and range for all triples we

want to validate. However since i'm still new to the tool i will still explore around and find out if the tool can also help us in achieving the task. Here is the code implementation of what i have experimented with using SHACL

Therefore accomplishing this task we also have two options:

- Using the domain-range algorithm
- Using the SHACL shapes

Adapt algorithm to fit different vocabulary

To make our algorithm to adapt to various vocabularies. We create a parameter that allows mapping of each relation to its corresponding vocabulary property. This mapping parameter will be used during the implementation process to fit the required vocabulary. Modify the framework to include a mechanism for users to specify the target vocabulary. This can be achieved through command-line arguments.

To address this challenge of ontology mapping to other vocabularies I propose we leverage the power of SPARQL to map concepts extracted from our DBpedia ontology to the equivalent terms in other specified ontologies. By implementing some techniques presented here Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources (Short Paper), we can achieve a powerful SPARQL query that performs ontology mapping. An example of how we can leverage the SPARQL query is given below:

```
select distinct ?dbpediaClass ?relation ?externalClass where {
   ?dbpediaClass a owl:Class .
   ?dbpediaClass ?relation ?externalClass
   values(?relation) { (rdfs:subClassOf) (owl:equivalentClass) }
   filter(!regex(?externalClass, "http://dbpedia.org/") && !regex(?externalClass,
"http://www.w3.org/2002/07/owl#"))
 } order by ?dbpediaClass ?relation ?externalClass
```

The result of the above query

In the example above, we used the SPARQL query to select distinct pairs of DBpedia classes, their relations, and external classes where the relation is either "subclass of" or "equivalent class." It filters out external classes that are either from DBpedia or from the OWL namespace. Finally, it orders the results by DBpedia class, relation, and external class. Simply it gets the mappings between DBpedia and external class via the SPARQL query.

We will integrate the SPARQL querying capabilities into our framework to enable seamless

ontology mapping. Mapping rules will be defined and managed within the system (command line arguments or parameters) allowing for flexibility and customization. The outcome of our ontology mapping component will generate output data that conforms to the specified target vocabularies.

I have basic understanding of how the SPARQL query works, however going through more resources online, i hope to easily understand how to implement this and also additionally by leveraging this paper Ontology Mapping and SPARQL Rewriting for Querying Federated RDF Data Sources (Short Paper).

NOTE: - I plan this as a **350 hour project** because of the following reasons:

- Spend time on reading recent research so as to make incremental changes.
- Explore the UniRel code implementation and compare with the REBEL model, as it seems there are certain tasks the REBEL model can't handle and UniRel is proved to be more computationally efficient.
- Understand advanced implementations of SPARQL queries.

TIMELINE

Time	Task to complete
Before May 1	• Explore DBpedia. • Get familiar with the concepts in DBpedia like property, classes, RDF etc. • Try setting up code for the research papers, it needs some refactoring. • Discuss with mentors and community about approaches and get feedback.
May 1 - May 26 (Community Bonding	• Explore SPARQL in detail.

Period)	• Get more familiar with the Dbpedia codebase. • Explore work of previous iteration in depth. • Read mentioned research papers. • Keep in touch with mentors and other contributors to learn more.
Week 1 & 2	• Draft out the pseudocode of the co-occurrence algorithm. • Apply Co-occurrence-Based Algorithm for the predicate suggestion. • Try out the relational mapping model and compare it to the co-occurrence algorithm.
Week 3 & 4	• Implement algorithm to validate the triples generated against the DBpedia Ontology • Try out the pySHACL tool to also validate few triples • Select the most efficient method
Week 5 & 6	• Adapt the framework output to fit various vocabularies • Implement some explicit SPARQL queries that performs ontology mappings
Mid Evaluations - July 10 - July 14	
Week 7 & 8	• Explore BART and its paper. • Explore other considerable language models for relation categorizing. • Understand their codebase. • Connect with experienced people in the field to use the code.
Week 9 & 10	Apply the streaming and message-brokering techniques to make our pipeline efficient. • Apply Spark NLP. • Apply Apache Kafka. • Compare both and select the most efficient method.
Week 11 & 12	• Write medium articles on each step of the project. • Cleaning the code. • Writing documentation and Readme files.

Submission Week	
August 19 - August 26 (Submission week)	• Get some feedback from mentors and community, this can help and improve even future iterations.

The timeline given above is subject to change. Given that some parts may take less or more time than expected, then the other parts may need to be rescheduled. But I have created small demos for most of the parts, so that has helped me judge the time that it will take to complete.

I will keep updating the mentors and keep in constant contact with them.

Background & Education

I am Abdulsobur, a resident of Akure, Nigeria. Born and raised in Nigeria.I am currently in my fourth year pursuing a Bachelor of Technology in Computer Science at the Federal University of Technology Akure, Nigeria. I'm expected to graduate in the year 2025.

Following are my educational details:

- Completed my high school at Igbemo Comprehensive High School;
- Currently in 4th year BTech in Computer Science at FUTA
- Grand Prize winner at the Huawei Africa Regional finals

Courses and Research work

I have done the following relevant courses:
Data analysis and Visualization, Machine Learning, Natural Language Processing.

Currently I am doing a course on advanced machine learning in which I am exploring the explainability of the self attention mechanism. I have read several recent research papers in that area and I am trying to do something new using self attention.

I am also exploring RAG conversational chatbot development. I am developing a chatbot that interacts with Graph, JSON, and a NoSQL database to answer questions about a website's information. Previously, I was exploring AI frontend development using Figma files. I couldn't get it into production, but the experience taught me a lot on how my

approach should be in the future.

Skills

- **Programming Languages** (I usually pick up quickly)
    - Python (2+ years)
    - Java (4+ years)
    - Kotlin (4+ years)
- **Data science and Machine Learning**
    - Mathematical Foundations
        - Calculus, linear algebra, probability and statistics.
    - Machine Learning
        - Classical methods, Deep NNs, transformers.
    - Programming
        - Numpy, Pandas.
        - Matplotlib, seaborn
        - Scikit-learn, TensorFlow, PyTorch
        - Networkx
- **Useful Tools**
    - Installation and collaboration
        - Bash, Docker, Git
    - Databases
        - PostgreSQL
        - MongoDB
        - MySQL

Open Source

I am passionate about OSS as it gives an opportunity to everyone to learn and contribute. The power of open source can be understood looking at Python, Linux or other such ecosystems. I also feel that getting to do a GSoC project is very cool as it can help me learn a lot in a short span and I can also earn to support my educational expenses.

Summer Plans

I will be in Nigeria during my summer break. I will be able to put in about 25-30 hours per week. I have no other courses, internships or university commitments this summer. I'll totally commit to working on my DBpedia project and will ensure that I follow the timeline as mentioned. However, if there are some unavoidable circumstances, I'll make sure to put in

extra efforts and ensure that I don’t drift from the timeline. Flexibility with it comes to communication times, being available most of the time (8:00 am UTC to 6:00 pm UTC).

GSoC experience

I had applied to DBpedia for GSoC last year but could not get selected. For me GSoC is very important in the sense that it will boost my confidence that I can do some work that is useful for the world and also that I can earn. If I get selected, I plan to use the stipend for paying my educational and living expenses in the university. This makes me feel independent and confident.

The reason I chose DBpedia was that I am inclined towards NLP. I like to explore NLP related technologies and understand the math and logic beneath them. I wonder how these technologies can power services like conversational AI, question answering systems, summarizing systems etc. Would be very happy and looking forward to contributing to DBpedia and to NLP broadly!