From Unstructured Text to Knowledge Graphs using NLP Tools and Google Knowledge Graph APIs

Vidhyalakshimi Sreenivasan, vsreeni
Sai Kiran Mayee Maddi, smaddi
Priyankha Bhalasubbramanian, pbhalas

May 4, 2019

1 Objective

- The aim of this project is to construct a Knowledge Graph (KG) from unstructured text like web data using NLP tasks like **Named Entity Recognition** and **Relationship Extraction**.
- We then query our Knowledge Graph to generate useful insights about the structure and data present in the KG.
- Finally, to benchmark our results we use Google's Knowledge Graph Search API to query and compare the nature of results

2 Architecture

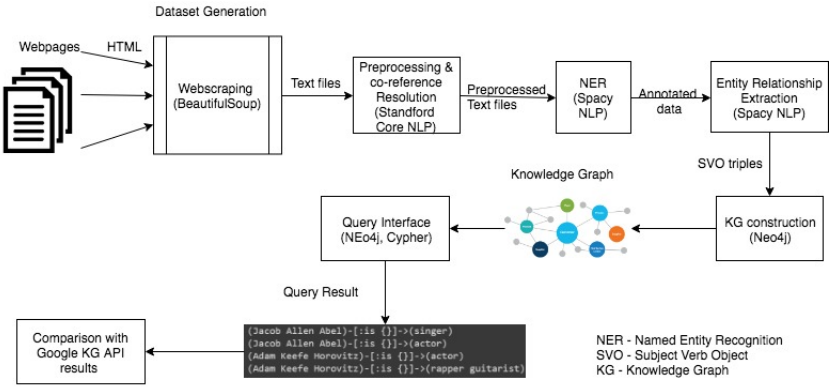

Figure 1: Architecture of our project

The project involves four phases as summarized in the Table 1 The following

Table 1: Phases involved in the project

Phase 1	Unstructured Data Extraction and pre-processing
Phase 2	Named Entity Recognition and Relationship Extraction
Phase 3	Knowledge Graph Construction and Querying
Phase 4	Interfacing with Google Knowledge Graph Search API

sub-sections elaborate the techniques that we used in each of the sections.

2.1 Phase 1

Unstructured Data Extraction: Unstructured data includes text, images, videos etc., which are sources of data that do not have a pre-defined data model and hence cannot be represented on a relational database.
In this project we worked on unstructured text data scraped from the web. We have focused our category to PERSON and specifically to those pertaining to Movies like popular actors, directors, producers etc. For this purpose we scraped the unstructured text from the Wikipedia pages of 25 popular movie actors/directors/producers/singers etc as text files.

Pre-processing: Since the raw text data in this case is a sequence of sentences, it is important to understand the relationships across multiple sentences. One relevant task here is coreference resolution which is used to disambiguate the pronouns present in a sentence and bind them to the entities/nouns present in the context.

2.2 Phase 2

Named Entity Recognition (NER): It is a very useful NLP task that has a variety of applications like in efficient search algorithms, content classification, recommendations etc. In our project we have used NER to construct Knowledge Graph for unstructured text. NER locates and classifies named entities into predefined categories like person, organization, location etc. **Conditional Random Field** (CRF) is a state-of-the algorithm used in the process of Named Entity Recognition. As a precursor to NER it is important to perform tokenization, Part-of-speech (POS tagging) and noun chunking.

Relationship Extraction: The next step before knowledge graph construction is identification of relationships that exist between named entities. We incorporated SpaCy's POS tag annotations to find relationships that exist among those tags using dependency parsing. These dependencies are translated into Subject Verb Object (SVO) triples which capture the relationships between named entities present in our unstructured text.

2.2.1 Conditional Random Field

Conditional Random Field (CRF) is a **supervised** ML algorithm used to classify whether a given token is a named entity and identify its annotations. The tagging scheme used by the algorithm is BIO which describes the entity boundaries.

Table 2: Phases involved in the project

B EGIN	First token of a named entity
I NSIDE	Inner token of a named entity
O UT	Non-entity token

Table 3: Annotations of our dataset

Type	**Description**
PERSON	People; including fictional
ORG	Companies, agencies, institutions
NORP	Nationalities, religious or political groups
GPE	Geo-political (countries, cities, states)
LOC	Non geo-political- mountain ranges, water bodies
EVENT	Named hurricanes, wars, plotitical or sports events
CARDINAL	Numerals
OBJECT	General Things

The annotations used in a CRF model are summarized in the Table 3.

Let's say that the input sequence of words is denoted as $x = (x_1, x_2, .., x_m)$, where each term x_i is a word in the sequence and let $s = (s_1, s_2, .., s_m)$ be the corresponding sequence of named entity tags [2]. For CRF we need to model the conditional probability

$$p(s_1, s_2, .., s_m | x_1, x_2, .., x_m)$$

We also define a feature map that translates the input sequence paired with the sequence of states into a d-dimensional feature vector.

$$\phi(x_1, x_2, .., x_m, s_1, s_2, ..s_m) \in \mathbb{R}^d$$

The conditional probability defined above can be modelled as a log linear function that takes in a parameter vector $w \in \mathbb{R}^d$

$$p(s|x; w) = \frac{exp(w.\phi(x, s))}{\sum_{s'} exp(w.\phi(x, s'))}$$

From the input examples $\{(x^i, s^i)\}_{i=1}^n$ we can define the regularized log likelihood function L

$$L(w) = \sum_{i=1}^{n} \log p(s^i|x^i; w) - \frac{\lambda_2}{2}||w||_2^2 - \lambda_1||w||_1$$

The terms $\frac{\lambda_2}{2}||w||_2^2$ and $\lambda_1||w||_1$ are regularization terms that penalizes our model complexity. We then try to optimize the parameter w and estimate it as,

$$w^* = arg\ max_{w \in \mathbb{R}^d} L(w)$$

Using this we can predict the most likely tag sequence s^* for a given sentence x by

$$s^* = arg\ max_s p(s|x; w^*)$$

For implementing this model we tried Sklearn's CRF_Suite of functions [3]. The GMB(Groningen Meaning Bank) corpus which is annotated using using the scheme [[((WORD, TAG), IOB), ...]...] was used for building our model. Apart from the annotations present in the corpus we extract other features like word parts, POS tags and some nearby words to render context. We then split the dataset into training and testing to validate the performance of the constructed model based on ground truth values. After splitting the data and extracting relevant features, we train our CRF model. We finally, predict the output tag sequence using the test data and evaluate the result.

Table 4: Evaluation metrics - Sklearn CRF_Suite

Average mode	**Precision**	**Recall**	**F1 measure**
Micro average	0.80	0.78	0.79
Macro average	0.50	0.50	0.48
Weighted Average	0.78	0.78	0.78

One issue that we faced with the Sklearn's CRF suite is that some obvious entities were misclassified. For example Google was predicted as a named entity belonging to the class PERSON rather than ORGANIZATION.
To overcome this issue, we tried using SpaCy NLP's pretrained model which and compute the performance by comparing it's predictions with ground truth.

Table 5: Evaluation metrics - SpaCy NLP

Average mode	**Precision**	**Recall**	**F1 measure**
Micro average	0.78	0.78	0.78
Macro average	0.05	0.04	0.05
Weighted Average	0.83	0.78	0.81

As observed the weighted average F1 measure for SpaCy NLP is significantly higher than that of Sklearn's CRF. Additionally SpaCy NLP achieved a good accuracy of 0.788.

2.3 Phase 3

2.3.1 Knowledge Graph Construction

Once the named entities and relationships are extracted we are now good to construct the **Knowledge Graph (KG)** whose **nodes** comprise of the named entities and the directed **edges** correspond to the relationships that exist among them. As elucidated in Phase 2, the SVO triples that are extracted from text indicate the existence of relationships between the subject and object that are labelled by the verb which connects them.

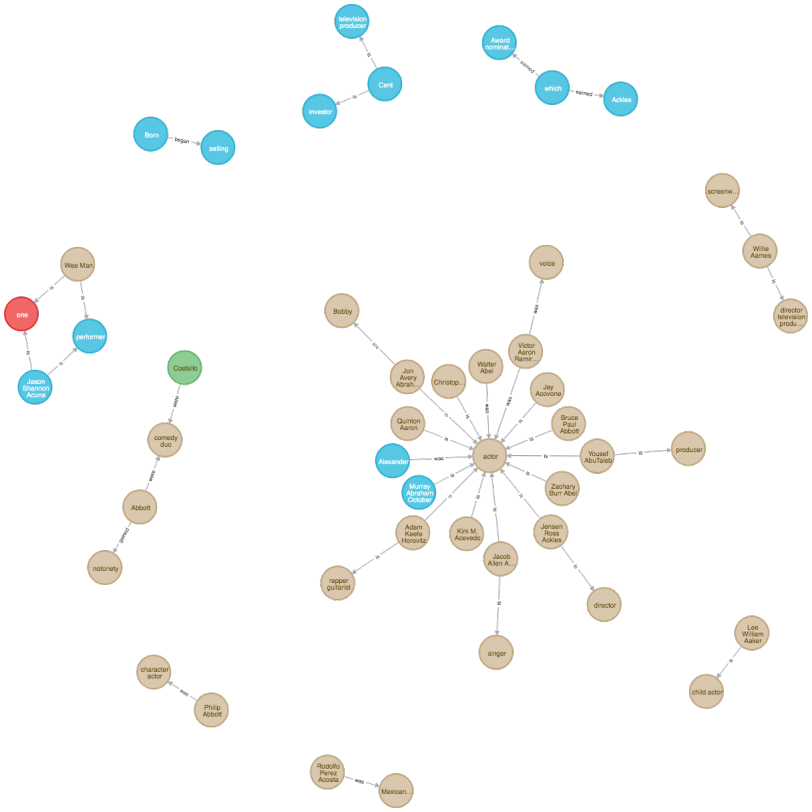

Figure 2: Knowledge Graph

The Knowledge Graph in Figure 2 was built on Neo4j Graph Database.

2.3.2 Query Interface

Once the KG is constructed we can now use it to query the graph to extract useful insights from it. Cypher Query language was used to query the knowledge graph constructed on the Graph database. We used py2neo to query the Knowledge Graph and provided support for basic queries like finding actors, producers, describing an actor and finding multi-faceted actors.

```
Query result for 'names of all actors':
['Jon Avery Abrahams', 'Zachary Burr Abel', 'Victor Aaron Ramirez', 'Quinton Aaron', 'Kirk M. Acevedo', 'Murray Abraham Octobe
r', 'Alexander Bud Abbott', 'Christopher Jacob Abbott', 'Adam Keefe Horovitz', 'Jensen Ross Ackles', 'Yousef AbuTaleb', 'Jay Ac
ovone', 'Walter Abel', 'Jacob Allen Abel', 'Bruce Paul Abbott']

Query result for 'names of all producers':
['Yousef AbuTaleb']

Query result for 'all multi-talented people':
Jon Avery Abrahams
Victor Aaron Ramirez
Adam Keefe Horovitz
Jensen Ross Ackles
Yousef AbuTaleb
Jacob Allen Abel

Query result for description of Jacob Allen Abel
(Jacob Allen Abel)-[:is {}]->(singer)
(Jacob Allen Abel)-[:is {}]->(actor)
```

Figure 3: Query results for a few supported queries

2.4 Phase 4

After extracting queries we are now left with comparing the results that we obtained from Google Knowledge Graph Search API. This API allows us to find named entities and query the knowledge graph constructed by Google developers. It requires us to use our Google account to obtain an API key and query the Knowledge Graph for any topic, person or thing. The response obtained uses standard Schema.org types like JSON-LD.
For the purpose of querying we used an example query of describing about a person called Jacob Allen Abel from Google Knowledge Graph Search API and from our Knowledge Graph's query interface.

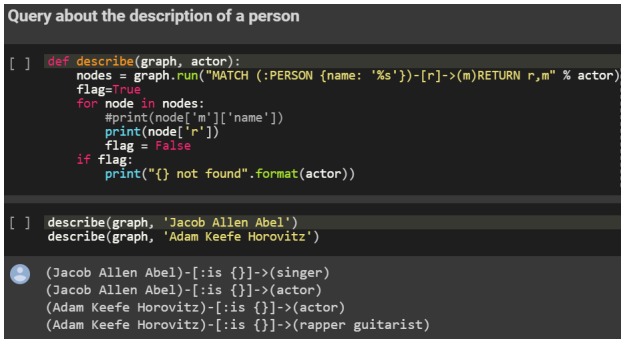

Figure 4: Our query result

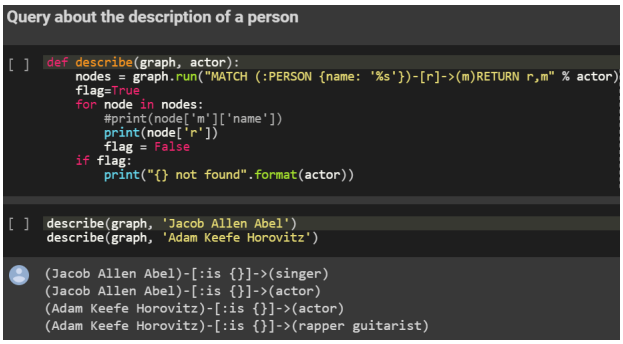

Figure 5: Google API's Query result

As shown in Figure 4 and 5our result aligns well to Google's API in terms of identifying the person's profession.

3 Conclusion

Through this project we were able to implement NLP tasks like Named Entity Recognition and Relationship Extraction that has a plenty of use cases like document classification, question answering etc to construct a Knowledge Graph. We then used the knowledge graph to query information and finally compared our results with Google's Knowledge Graph Search API.

References

[1] Knowledge Graph basics - https://medium.com/vectrconsulting/build-your-own-knowledge-graph-975cf6dde67f

[2] https://www.depends-on-the-definition.com/named-entity-recognition-conditional-random-fields-python/

[3] CRF Implementation https://towardsdatascience.com/named-entity-recognition-and-classification-with-scikit-learn-f05372f07ba2

[4] Spacy Implementation - https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da

[5] Neo4J - https://www.quackit.com/neo4j/tutorial/neo4j_query_language_cypher.cfm

[6] Google Knowledge graph API - https://developers.google.com/knowledge-graph/

[7] https://github.com/dasmith/stanford-corenlp-python

[8] https://github.com/aleenaraj/Coreference_Resolution