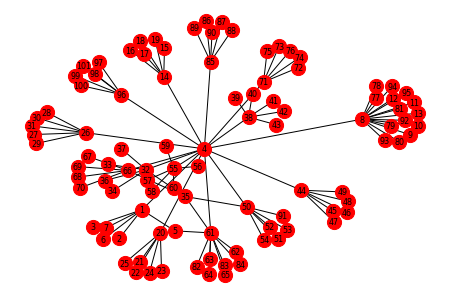
## 



From Unstructured Text to Knowledge Graphs: Using NLP tools and Google KG APIs

05.03.2019

**─**

Anusha Manur, Nandita Srinivasan, Urmil Parikh

# Overview

Data in the world grows by leaps and bounds on a daily basis. At this scale, it becomes impossible to subject all data to human processing. The more we automate natural language text processing, the faster data processing could become. A lot of work has been done in this field, out of which two of the sub-tasks are those of Named Entity Recognition(NER) and Relation Extraction. These can help data processing go a long way with respect to context recognition and greatly enhance question-answering systems. Google is one of the world’s most famous search engines that processes numerous natural language queries on a daily basis. It has an underlying Knowledge Graph of entities and relations that it leverages. In this project, we attempt to implement our own version of a knowledge graph, and test our implementation of NER and Relation Extraction in order to query this graph. Lastly, we compare its results to Google’s Knowledge Graph API for the corresponding entities and see how it matches up.

# Goals

1. To construct our version of a knowledge graph from a given dataset that encompasses a domain.
2. To perform named entity and relation extraction from a set of possible queries and compare the results of our graph to Google’s Knowledge Graph API.

# Workflow

We have two main datasets:

* movie\_metadata, which contains over 5000 movies along with information about them such as the three main actors, genre and language.
* queries.txt, which contains over 250 queries regarding movie information, such as, “Who directed I am legend?”

We build a knowledge graph from our knowledge base of movies in the format described in the sections below. We then parse the queries from the queries.txt dataset and extract the entities and relations from them. The queries are written in natural language, just like most people would type out a query in Google’s search bar. The same extracted entities are fed to Google’s Knowledge Graph API to observe the results. The various steps in the workflow are summarized in the below sections.

## Building our knowledge graph

We use the networkx module to build our version of a knowledge graph. We read the movie\_metadata.csv file line by line, to get the information for each of the movies. We create the following types of nodes:

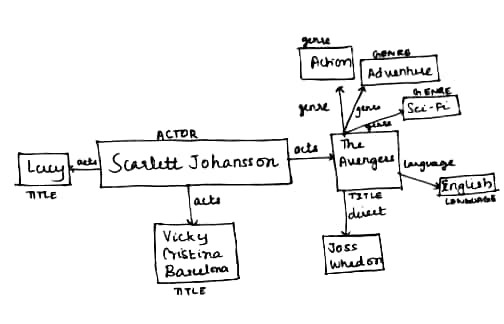
* Actor
* Director
* Genre
* Title
* Language

We create a dictionary that contains the mappings of each created node, and its unique node ID. This is needed to avoid the creation of duplicate nodes. For example, We come across the movie “The Avengers”. As we read the line pertaining to this movie, we create a node for each of the above listed entities like actor and director. We make sure that there is no duplication of nodes in the graph. Now that we have created the annotated nodes, we create their corresponding attributed edges.

To do this, we add an edge from each of the other nodes towards the movie title node of that particular row as shown below



This leads to the construction of a graph of named nodes and annotated edges as below-



## Named Entity Relation Extraction

Here, our main goal is to parse a number of natural language queries, and be able to successfully extract the required entities and relations in a way in which we can query our generated graph.

We first start by tokenizing the sentence to get the constituent words and punctuations. We then put it through NLTK’s POS-tagger to extract the various parts of speech. We require the parts of speech in order to create a context from which we can recognize the presence of a movie's name, the name of the person, or the relation that the query is about. For example:

* What did Scarlett Johansson act in?
* Who directed I am legend?

The relations required are those marked in orange. The subject of the “Who” and “What” are implicit. We use NLTK’s Ne\_chunk in order to identify the presence of person names in the query string. The function has more powerful capabilities of being able to identify locations and organizations, but we do not leverage those and limit it only to the recognition of “PERSON” entities. The function continuous\_chunks obtains “PERSON” tagged words that occur consecutively - such as “Scarlett Johansson” instead of “Scarlett, Johansson”. We then stem each word to its root form using NLTK’s stemmer feature.

* Who acted in I am legend?
* Who is the director of avengers?

These words give us a clue for what relation is being queried from the recognized entity. Our technique is to locate the node in the graph corresponding to the entity- in the second example, the “title” node corresponding to “The Avengers” and follow the edge leading out of it to a “director” node- a “directs” edge.

Hence, using the function, we construct a tuple as below-

[Entity, Entity\_Name, Relation]

For the above example, it would be-

[Title, The Avengers, Director]

This is because, the name of the movie is known to us; we know it is a node of type “title” and have observed the relation token “direct” in the query.

## Querying the graph

On obtaining the tuple from the above function, we query our graph using the same. We first obtain all the nodes of type entity. In our example, we retrieve all the “title” type nodes. In these, we look for a value of Entity\_name. In our example, we look for a value of “The Avengers” among the “title” nodes. On finding the matched node, we inspect the edges leading out of it for one that points to a “Relation” node. Here, a “director” node. There could be one, as in language, or there could be many, as in genre or actor. All such nodes are retrieved and placed in a list.

## Testing over the queries.txt dataset

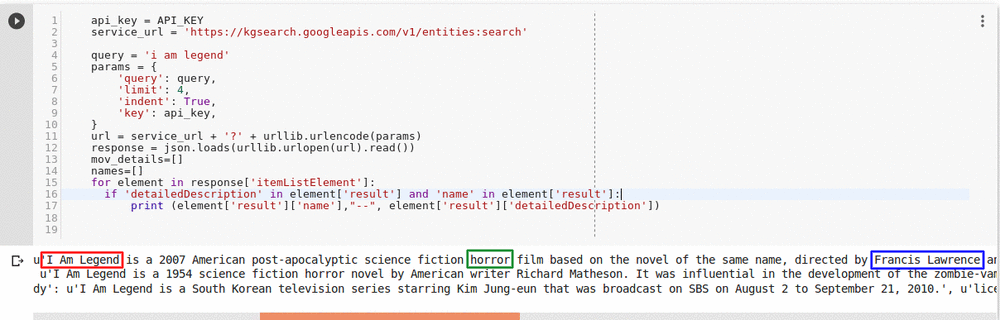
We read the queries.txt dataset line by line and perform NER on them using the functions we wrote. The functions must be able to correctly identify the context of the query and extract the right entities and relations. Only if this is successful, will the query to our graph also be successful. If, for example, the right entity is not extracted, and we were able to get only “Avengers” and not “The Avengers”, the query to the graph would fail and return an empty set, implying the NER was not successful. We run the functions on over 250 such queries and calculate accuracy as the fraction of queries that were successful to the overall number of queries. The results are discussed in further sections below.

## Comparing to Google’s Knowledge Graph API

We use the google knowledge graph api to query details about the main entity as the API takes a single entity as input and not an entire sentence.

Before we send requests to Google Knowledge Graph Search API, we use the Google Developers Console to create an API key and enable access to our application.

Example : The results of the query “i am legend” is as shown below



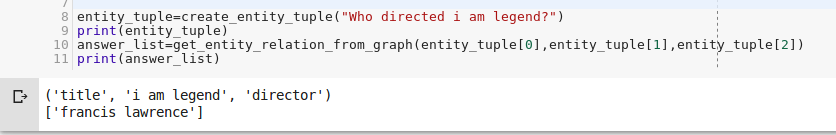
# Results

We created the knowledge graph for over 5000 movies in the movie dataset. We then ran the queries over both Google’s Knowledge Graph API and our own Knowledge Graph and obtained the results as summarized below:

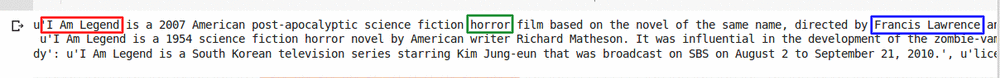
|  |  |
| --- | --- |
| Knowledge Graph | Accuracy |
| Our Implementation | 60.3% |

Then we compare the results of the queries obtained from both our constructed graph as well as Google’s knowledge graph as seen below.

Results from our constructed graph



Results from Google’s knowledge graph



# Assumptions

We make the following assumptions for the queries-

* No pre-trained model like spacy or StanfordNLP has been used for named entity recognition and relation extraction.
* We have tested our model on the queries we manually generated.
* To perform NER, we define a pattern-based context for the queries in the generated dataset. Hence, the queries we use to search the graph should conform to the same.

# Conclusion

In conclusion, we can see that our NER function and knowledge graph perform similarly to that of Google’s Knowledge Graph API, with respect to the context of Movies. This goes to show how the concept of NER and knowledge graph creation can have diverse applications in being able to classify and summarize content, creating effective search engines and also in chat based applications. Knowledge graphs prove to be very powerful in offering semantic context to otherwise unstructured, natural language text.

# References

[1] Code for Continuous Chunking: <https://stackoverflow.com/questions/48660547/how-can-i-extract-gpelocation-using-nltk-ne-chunk>

[2] Google Knowledge Graph Search API <https://developers.google.com/knowledge-graph/>