Keenan Venuti Assignment 4 MSDS: 459

Abstract

The most powerful knowledge bases are capable of answering questions across a sundry of topics with specificity, breadth, and scale (Peng, 2019). Google search engine alone processes 3.5 billion searches per day; some of those queries are natural language questions, searching for a specific answer (2020). Results to those queries are *facts* and can be represented by the semantic triple. In this project, I developed a domain-specific knowledge base consisting of 221,129 triples, loaded into a Neo4j graph database (GraphDB). The GraphDB held the discography, albumography, associated musical group, and relevant artist relations of hip hop artists, originally scraped from the Hip Hop Lyrics Archive (ohhla, 2020). With a well connected and dense knowledge base, I proceeded to develop a Natural Language Understanding (NLU) layer capable of receiving questions, extracting relevant entities and relationships, and querying the GraphDB. The results showed a promising system capable of complex information retrieval and flexibility when introducing new nodes and relationship types.

Introduction

Within a knowledge base, data are organized into 3 key structures: object, predicate, and subject. While these structures are conceptually simple, they are extremely powerful when paired with computer processing capabilities and massive data stores. The 3 structures in unison are named a “triple”. Triples can store any fact within the universe, but using them in mass for logic based reasoning and natural language understanding quickly fails at scale. Trying to find relationships between two predicates based on a shared object relationship results in a string matching operation of complexity O(n) also known as linear time. Graph Databases, like Neo4j, are *a best of both worlds* solution to this problem. GraphDBs use two key data structures, nodes and edges. Each node can have 1 to n potential edges against all other n nodes; in addition, nodes and edges can have properties specific to each instance. By eliminating the subject, and repurposing the object as a new node, the triple is now, node, relationship, node. As a result, the GraphDB is still able to store triples while quickly finding relationships between objects with shared subjects. With the knowledge graph established, there exists the flexibility to link multiple datasets without complex SQL joins; instead, GraphDB ingestion is entity-centric.

This flexibility also comes at a cost, namely it requires more discipline in schema enforcement and entity definition. Using the GraphDB I built as an example, we can fine tune the proper definition of an entity. The data type *Artist* represents a real human with relationships to others, as a result, it is the core node type of my GraphDB. An artist can belong to a musical group, develop albums, sing on songs, and have multiple other properties. Herein lies the challenge of GraphDBs, what necessitates another node and what is better suited as a property? The answer should be inspired by the final business use case. In my project, the end goal was a NLU system capable of basic question answering using Neo4j as a knowledge base. Based on that goal, I chose to make musical groups, albums, and songs as node types rather than node properties. Other details, such as the artist's birthday, were relegated to properties. This decision was purely based on my anticipation of how the question answering system would be used. The question, "What album did Kanye West and Jay-Z develop together", is much more likely than "Who shares the same birth month as P. Diddy?". To escape linear search time, the later query would require 12 linked *birth month* nodes. However, given the rarity of the question, adding dates in this context would only complicate the GraphDB schema. Ultimately, a GraphDB should be structured for the final use case, rather than adapting the final use case to how the data are structured. These types of GraphDB schema design choices are necessary but not without prior research.

Literature review

The core components of this project, accumulating/processing natural language data, structuring a GraphDB, ingesting data from RDF triplestores, and developing a Question Answering NLU layer, each necessitate their own areas of research. In preparation for this assignment, I developed a potential solution for each component after reviewing relevant academic and applied materials. Starting with triple extraction from raw text, I followed examples from researchers who describe their, "system is based on a pipeline of text processing modules that includes a semantic parser and a coreference solver. By using coreference chains, we group entity actions and properties described in different sentences and convert them into entity triples." (Exner & Nugues, 2012). With entity heavy musical lyrics text data, proper entity extraction, using the tool Spacy, was key in creating a complete and cohesive GraphDB. With entities extracted and relationships aggregated to a predefined set of potential intents, I studied research on best practices regarding GraphDB schema creation. Following Neo4j's guide on creating social GraphDBs, nodes should be, "used to represent entities", whereas properties should be, "name-value pairs that are used to add qualities to nodes and relationships" (Neo4j docs, 2020). Neo4j defines GraphDB schemas as "whiteboard-friendly", developers should be able to, "draw example data on the whiteboard and connect it to other data drawn to show how different items connect." (Neo4j developer, 2020). Neo4j best practices inspired the use of an entity first model representing Artist, Album, Song, Musical Group as entities.

Methods

As this project was developed *end to end*, there were several key tasks that had to be performed sequentially. First, I scraped the rap lyrics database, "ohhla.com/" using Python, BeautifulSoup, and Regular Expressions. The results were saved into JSON files where they underwent further cleaning and Extract-Transform-Load transformations specific to graph data. To ensure minimal latency of the Question Answering capability and maximize the number of potential results returned from the model, I built the Neo4j graph database using Docker on an Ubuntu Server with a 3.6GHz 4-Core CPU and 32GB of RAM. With the data prepared and the GraphDB running, I created indexes for Artist, Album, and Song nodes and began ingestion. Indexing is key to a performant ingestion with a GraphDB of this scale. In the end, there were 108,301 nodes and 221,129 relationships. Relationships could be of three types, *CONTAINS*, *PRIMARY*, *RAPS_ON*. With the database completed, I moved on to the development of the NLU interaction layer.

Implementation of the NLU Question Answering system necessitated multiple design choices. Ultimately, I chose to implement the NLU capability from *nearly* scratch (I leveraged a pre-trained Word2Vec model, Part of Speech classifier, and spacy Named Entity Recognition classifier). By starting at a very granular level of language processing, triple extraction, I was able to control for the semantic translation between natural language questions and a GraphDB query. This approach was challenging and required multiple iterations of experimentation to both accurately extract triples, and properly classify them as input into a Cypher based search. In the end, the model was capable of receiving a question in natural language like, “How many songs does Beyonce have with Eminem?” and extracting the relevant entities and relationships from the query: “Eminem”, “Beyonce”, “many songs”. From there, the relationship was embedded via a pre-trained Word2Vec model and used to calculate cosine similarity to 4 templated relationship types. In this example the closest match would be the relationship type “count”. With the entities and intent of the queries now extracted, Cypher was used to scour the massive GraphDB and determine if a relationship exists. The custom Python module then returned the results in the form of a list of values, triples, or, in the present example, a boolean value.

Results

The end to end system was able to accurately and quickly extract relevant answers to questions posed to the model. Further, the system was flexible enough to answer questions without a relevant object. For example, when the model read the question, “What songs did Akon sing?”, it correctly inferred that the intent of the query is a list of songs, rather than identifying Akon as the subject, sing as the predicated, and songs as the object. This query returned a list of songs based on outgoing edges from the “Akon” node. The model was also able to answer questions on how two artists are related, how many songs they performed together, and if an artist had any relation to another artist or song. Results are further documented in the Jupyter Notebook PDF attached.

While building the model from scratch made it more flexible, it also lacked in contextual nuance. For example, the spacy named entity recognition model failed to extract entities if the phrasing of the question was structured in alternative ways. The model did not work properly for the query, “How is Drake related to Eminem”, while it correctly worked for, “How Drake related to Eminem.” The triple extraction model did not work on multi-word entities, struggling with names like “Jay-Z” and “Biggie Smalls”. Finally, some rapper names with non-entity components in their name like, “Kanye West"

(west), were not properly extracted. Rappers often have several aliases, this further complicated how the question answering layer can query the Neo4j knowledge graph.

Conclusions

The NLU model and end to end system were capable of answering basic questions in structured formats but failed for complex queries or synonym based entities. This initial work demonstrates both how raw text can be converted into triples and then node/edge GraphDB formats and how triple extraction with named entity recognition creates a powerful question answering system. Similar technology underpins how Google Search functions and how Facebook recommends new friends to their usebase. These models require two things: vast amounts of data and a detailed understanding of user system interactions. Google Search has access to both huge amounts of raw textual data, used to refine their natural language understanding layer, and a powerful A/B experimentation platform to quickly and accurately identify what users like most about their search platform. Using this vast RDF triplestore knowledge base in conjunction with refined user needs will further lead to near human like question answering systems.

References

(2020, November). Google Search Statistics. internet live stats. Retrieved from: https://www.internetlivestats.com/google-search-statistics/

Exner, P., & Nugues, P. (2012, November). Entity Extraction: From Unstructured Text to DBpedia RDF triples. In WoLE@ ISWC (pp. 58-69).

ohhla. (2020, November). The Original Hip-Hop Lyrics Archive. Retrieved from: http://ohhla.com/

Neo4j developer. (2020). Graph Modeling Guidelines. Neo4j developer. Retrieved from: https://neo4j.com/developer/guide-data-modeling/

Neo4j docs. (2020). Graph database concepts. Neo4j docs. Retrieved from: https://neo4j.com/docs/getting-started/current/graphdb-concepts/#graphdb-nodes

Peng, Q. (2019, October). Answering Complex Open-domain Questions at Scale. The Stanford AI Lab Blog. Retrieved from: http://ai.stanford.edu/blog/answering-complex-questions/

Technical/Code References

Forked code for Triple Extraction from:

Borcan, M. (Feb, 2020). Python NLP Tutorial: Building A Knowledge Graph using Python and SpaCy. ./programmerbackpack. Retrieved from: https://programmerbackpack.com/python-nlp-tutorial-information-extraction-and-knowledge-graphs/