Automated Construction and Maintenance of Knowledge Graph from Structured Data

Nakul Aggarwal
19CS10044
nnakul.aggarwal@gmail.com

Hritaban Ghosh
19CS30053
ghoshhritaban21@gmail.com

1 Introduction

Most of the internal data and information that is stored in organizations today is in a structured format, stored in databases, unlike the kind of data we see on the web that is mostly unstructured. An equivalent knowledge graph representing an organization's data can go a long way in analysing and making sense of their data, and also making far more complex and granular queries and inferences than a spreadsheet software (like MS Excel, Numbers), using SPARQL-like query language.

An organization generally has multiple databases, each one being handled by different teams or clerks, who might not use a single database schema, i.e., the names and data-types of the columns might differ. Given the current schema that the knowledge graph assumes, we must determine how the data elements from a new data source being integrated into the graph should be added. This is known as the **schema mapping** problem. In addition to relating the schemas of the knowledge graph and the incoming database, we must also deal with the problem of inferring if two entities, one in the knowledge graph and other in the incoming data, are the same real world entity; a problem known as the **record linkage** problem.

In this project, we have implemented a software that constructs and maintains a knowledge graph on the local disk as it receives structured data in real-time, while detecting and solving the problems of schema mapping and record linkage.

2 Problem Statement

Given the location of a knowledge graph on the disk, update the graph as a new database is received as input. The knowledge graph is initially empty (no triples), but the second database onwards, the problems of schema mapping and record linkage must be addressed using suitable techniques. In addition to the triples (set of assertions), the knowledge graph must also store and maintain the schema, i.e., the range of the predicates and any other constraints.

3 Methodology

The incoming structured data is received in the form of a database; that for each table specifies both the schema and the records. The schema, like in an RDBMS, is represented by the column names, their types (entity, string, integer, real, boolean, date) and constraints (primary key, unique value, min value, max value).

3.1 Decomposition

Each table in the database is serially added into the knowledge graph. The integration process into the graph first starts by decomposing all the records in the table into triples. If a row, having value of the primary key as X, has value Y for the column P, then the corresponding piece of information is represented by the triple (X, P, Y). If a table has multiple primary keys, say the columns $C_1, C_2, ..., C_t$ having values $X_1, X_2, ..., X_t$ resp., then the triples would be $(\_ : i, P, Y)$ $(\_ : i, C_1, X_1)$ $(\_ : i, C_2, X_2)$ ... $(\_ : i, C_t, X_t)$.

3.2 Schema Mapping

The mappings between the input schema (table columns) and the schema in the knowledge graph (predicates) is not always simple one-to-one mapping. It becomes a tall order to expect any automatic process to infer such complex mappings without any manual intervention. It may involve complex calculations and may force to make several simplifying assumptions. Many automated mapping solutions rely on machine learning techniques which require large amount of training data to function effectively. As the schema information, by definition, is much smaller than the data itself, it is unrealistic to expect that we will ever have large number of schema mappings available against which a mapping algorithm could be trained.

Due to the numerous practical difficulties faced in fully automating the schema mapping, we use a **bootstrapped-based schema mapping technique** that keep human in the loop. The schema mapper should propose to the human a pair (C, P) as a potential mapping (where C is a table column and P is a predicate) if they pass the following tests.

1. **Linguistic Matching Test** : The predicate name and the column name must have some linguistic similarity. It can be quantified by a combination of several metrics like edit distance, k-gram overlap, longest common substring length, Jaccard's coefficient, KL divergence b/w the frequency distributions etc.

2. **Type Matching Test** : The type of data contained in the column C and the range of the predicate P must be compatible. Like, *integer* and *floating-point* are compatible types (numeric) but *date* and *string* are not.

3. **Constraint Matching Test** : Both of them have the same constraints (unique value, min value, max value).

After the tests are passed, the mapping is successfully inculcated if the human manually gives their consent.

3.3 Record Linking

Only the values that are contained in a table column with *entity* type will be considered for record linking, rest all the values in the table are literals. Efficient record linkage involves two steps: **blocking** and **matching**.

3.3.1 Blocking

Ideally we should treat each possible (E_T, E_G) pair as a candidate for linking (E_T, E_G is an entity in the table and graph resp.), but for time optimization, the blocking process constructs a reduced set of candidates by selecting pairs whose corresponding entity names pass some preliminary linguistic similarity-based tests.

We have achieved this by using several heuristics for linguistic similarity, like edit distance and Jaccard's coefficient (both on the full forms and short forms of the entity names).

3.3.2 Matching

It is a feedback-based semi-supervised strategy to label each pair in the blocked set as YES (refer to same entity) or NO (refer to different entities). This step works by learning a random forest through an active learning process (**RFAL, Random Forest Active Learning**). RFAL involves the following important steps.

1. *Initialize* : Ask the human to manually label a sample of N pairs selected from the blocked set.

2. *Train and Evaluate* : Partition the labelled set into train and test data. Train a random forest classifier on the train data and evaluate it on the test data.

3. *Repeat* : If the performance on test data is satisfactory, stop and predict labels of the remaining unlabelled pairs using the trained random forest classifier. Otherwise, expand the labelled set by asking to manually label another sample of K pairs selected from the unlabelled set. Then repeat the previous step.

Finally, use the labels on all the pairs in the blocked set to decide whether to link the corresponding entities in the knowledge graph or not.

4 Dataset

In this work, a sample dataset was artificially created for the purpose of testing, observation, analysis and proof of concept. It consists of five databases namely - (1) INSTITUTEDB, which describes students, the courses they take and the professor who teach these courses; (2) HALLDB which describes the hall allocation; (3) SOCIETYDB which describes the societies students are a part of; (4) ALUMNIDB which describes various IITs and their alumni; and lastly (5) COMPANYDB which describes the companies the students/alumni intern/work at. The data in these databases are fabricated from real information to demonstrate the results of this work.

5 Links

Find the code, dataset, results & instructions on this GitHub repository. Find the video demonstration & presentation of the project on this YouTube link.

6 References

- Directly mapping Relational Databases to RDF/OWL
- Schema Mapping for Data Transformation & Integration
- Active Learning based on Random Forest
- Record Linkage: Introduction & Recent Advances
- Create a Knowledge Graph from Structured Data