Supervised Typing of Big Graphs using Semantic Embeddings

Mayank Kejriwal, Pedro Szekely
Information Sciences Institute, USC Viterbi School of Engineering
Big Graphs have become ubiquitous in the Semantic Web
Typing Big Graphs

• DBpedia has over 89,000 entities typed as *owl:thing*

• Hundreds of types in the DBpedia ontology have no *extensional* instances

• Is typing always *absolute*?
  • Should *typeOf(Arnold Schwarzenegger, Politician)* be considered as likely as *typeOf(Barack Obama, Politician)*?
From types to instances to back again...

- Traditional view is that ontology comes first, then data
- Many instances now do not conform ‘closely’ to a specified ontology
- Automatic typing of instances can require a lot of feature engineering
Motivation 1: Automatic, probabilistic typing

• Classify each instance as a type (multi-class classification); use classifier scores as probability
  • What features should be used?
  • What if the ontology changes (e.g., from DBpedia to Freebase)?

• Clustering
  • How should the space be defined?
  • How should the probability be defined?
Motivation 2: No feature engineering

• Use the data itself, not pre-defined graph patterns or features, to deduce types

<table>
<thead>
<tr>
<th>ID</th>
<th>graph pattern (GP)</th>
<th>inferred axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td>gp₁</td>
<td>e owl:sameAs x &amp;&amp; x domain:aliasOf y &amp;&amp; y owl:sameAs z &amp;&amp; z rdf:type C</td>
<td>e rdf:type C</td>
</tr>
<tr>
<td>gp₂</td>
<td>e rdf:type x &amp;&amp; x owl:sameAs y &amp;&amp; y domain:aliasOf z &amp;&amp; w owl:sameAs z &amp;&amp; w rdf:type C</td>
<td>e rdf:type C</td>
</tr>
<tr>
<td>gp₃</td>
<td>e owl:sameAs x &amp;&amp; x [r] y &amp;&amp; y rdf:type C</td>
<td>e rdf:type C</td>
</tr>
<tr>
<td>gp₄</td>
<td>e owl:sameAs x &amp;&amp; x rdf:type C</td>
<td>e rdf:type C</td>
</tr>
<tr>
<td>gp₅</td>
<td>e dul:associatedWith x &amp;&amp; x rdf:type C</td>
<td>e rdf:type C</td>
</tr>
<tr>
<td>gp₆</td>
<td>(e owl:sameAs x &amp;&amp; x anyP y &amp;&amp; y rdf:type C)</td>
<td></td>
</tr>
</tbody>
</table>
Potential **Data-driven** Applications

- Fuzzy reasoning
  - What is the probability of an entity being a politician, given that they are also actors?
- Type Recommendation
- Profiling ontology coherence
  - How closely does the data conform to the declaratives?
Approach

• Embed instances in knowledge graph in vector space
  • Used existing algorithm (RDF2Vec)
RDF2Vec: Some visualizations

- Based on DeepWalk algorithm
- Results are fairly intuitive
Approach: intuition

- Construct type embeddings in the same vector space as pre-computed entity embeddings
Algorithm 1 Generate Type Embeddings

**Input:** Sets $S$ and $\overline{S}$ of entities and entity embeddings, type-only Knowledge Base $\mathcal{T}'$

**Output:** Type embedding $\overrightarrow{t}$ for each type $t$ in $\mathcal{T}'$

1. Initialize empty dictionary $T_S$ where keys are entities and values are type-sets
2. Initialize type-set $T$ of $\mathcal{T}'$ to the empty set
   
   // First pass through $\mathcal{T}'$: collect entity-type statistics
3. for all triples $(s, : type, t) \in \mathcal{T}'$ such that $\overrightarrow{s} \in \overline{S}$
   do
   Add $t$ to $T$
   Add $t$ to $T_S[s]$, if it does not already exist
4. end for
5. for all $s \in keys(T_S)$, set $T_S[s] = |T_S[s]|$ to save memory
6. Second pass through $\mathcal{T}'$ to derive type embeddings
7. for all triples $(s, : type, t) \in \mathcal{T}'$ such that $s \in S$
   Update $M[t]$ using Equation 1, using $T(s) = T_S[s]$
8. end for
   //Derive type embedding from $\overrightarrow{\mu_t}$
9. for all types $t \in keys(M)$
   Let type embedding $\overrightarrow{t}$ be the projection of $M[t]$ on $d$-dimensional hypersphere with unit radius (divide throughout by $||M[t]||_2$)
10. end for
11. return type embeddings derived in last step
Properties of Algorithm

• Only requires two passes through data, very fast!
• Because of incremental nature, can work with dynamic data
• Agnostic to entity embeddings, can work with any set of entity embeddings
  • RDF2Vec, TransE, TransH, NTN...

(a) TransE

(b) TransF
Target ontology vs. original ontology

- Target ontology can be different from source ontology (as long as some training data is available); ontology mapping not required.
Experiments

- Partitioned DBpedia knowledge graph into five sets
Task 1: Type Prediction

- 4 sets used for training, 1 for testing
- Used kNN with voting as baseline
- Found all-or-nothing phenomenon with kNN, not robust!
Task 2: Type Recommendation

<table>
<thead>
<tr>
<th>Entity</th>
<th>Embedding Method Rec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shenyang_J-13</td>
<td>Aircraft, Weapon, Rocket</td>
</tr>
<tr>
<td>Amtkeli_River</td>
<td>River, Body-OfWater, Lake</td>
</tr>
<tr>
<td>Melody_Calling</td>
<td>Album, Single, Band</td>
</tr>
<tr>
<td>Esau_(judge_royal)</td>
<td>Monarch, Loyalty, Noble</td>
</tr>
<tr>
<td>Angus_Deayton</td>
<td>Comedian, ComedyGroup, RadioProgram</td>
</tr>
</tbody>
</table>

• Possible because we get a scored list of types with embedding method
Task 3: Ontology Coherence
Extensions: Generative Type Model (GTM)
Future Work: Instances as probability vectors

• Cast each instance in DBpedia as a probability distribution over ~400+ types
• Full dataset is about 100 GB uncompressed, serialized in JSON lines
• Currently exploring use in large-scale ontology coherence, fuzzy reasoning at scale
Conclusion

• Types, properties (more generally, ontologies) and entities are both important for realizing the Semantic Web vision
• Many ontologies and datasets currently exist on the Semantic Web
• Many overlap in terms of domains, many assertions possible
• We showed a simple method to generate type embeddings at scale without re-running a knowledge graph embedding

http://usc-is-i2.github.io/home/

{kejriwal, pszekely}@isi.edu