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

DBpedia

ProbKB

yago
select knowledge

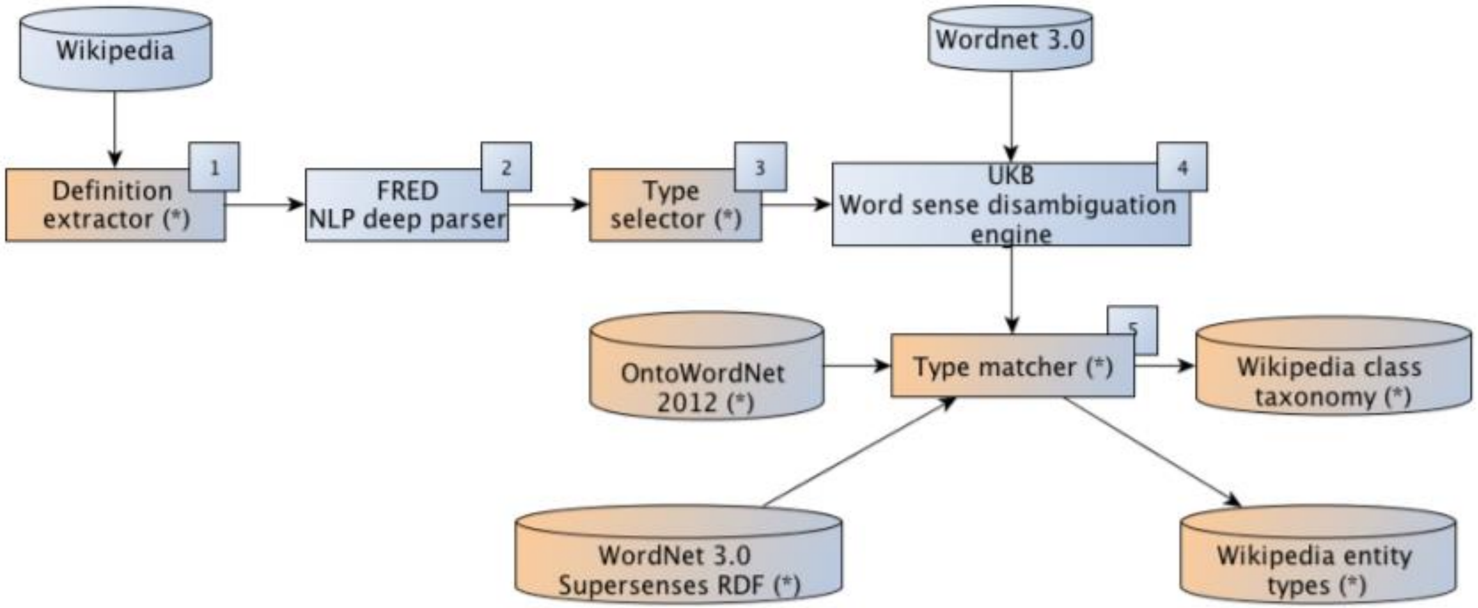
Freebase™

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

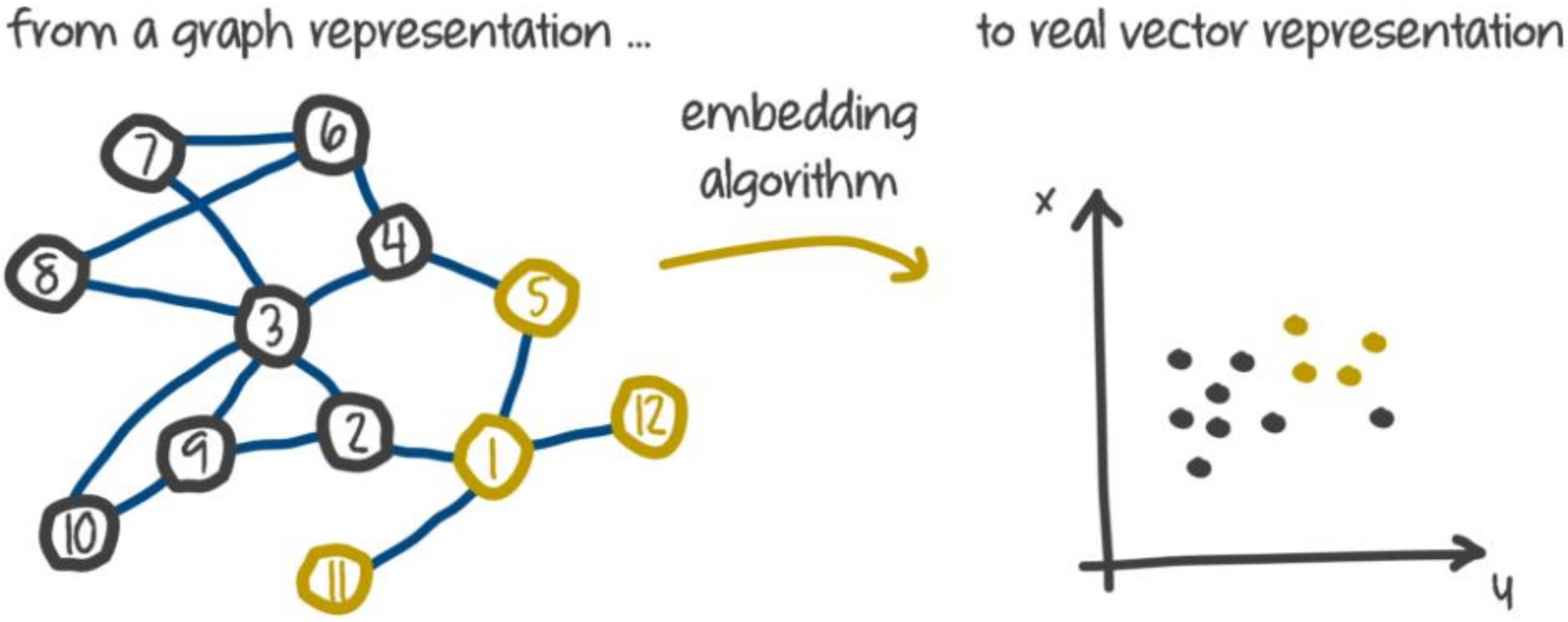
ID	graph pattern (GP)	inferred axioms
gp_1	e owl:sameAs x && x domain:aliasOf y && y owl:sameAs z && z rdf:type C	e rdf:type C
gp_2	e rdf:type x && x owl:sameAs y && y domain:aliasOf z && w owl:sameAs z && w rdf:type C	e rdf:type C
gp_3	e owl:sameAs x && x [r] y && y rdf:type C	e rdf:type C
gp_4	e owl:sameAs x && x rdf:type C	e rdf:type C
gp_5	e dul:associatedWith x && x rdf:type C	e rdf:type C
gp_6	(e owl:sameAs x && x anyP y && y rdf:type C) || (e anyP x && x rdf:type C)	e rdf:type C

Potential **Data-driven** Applications

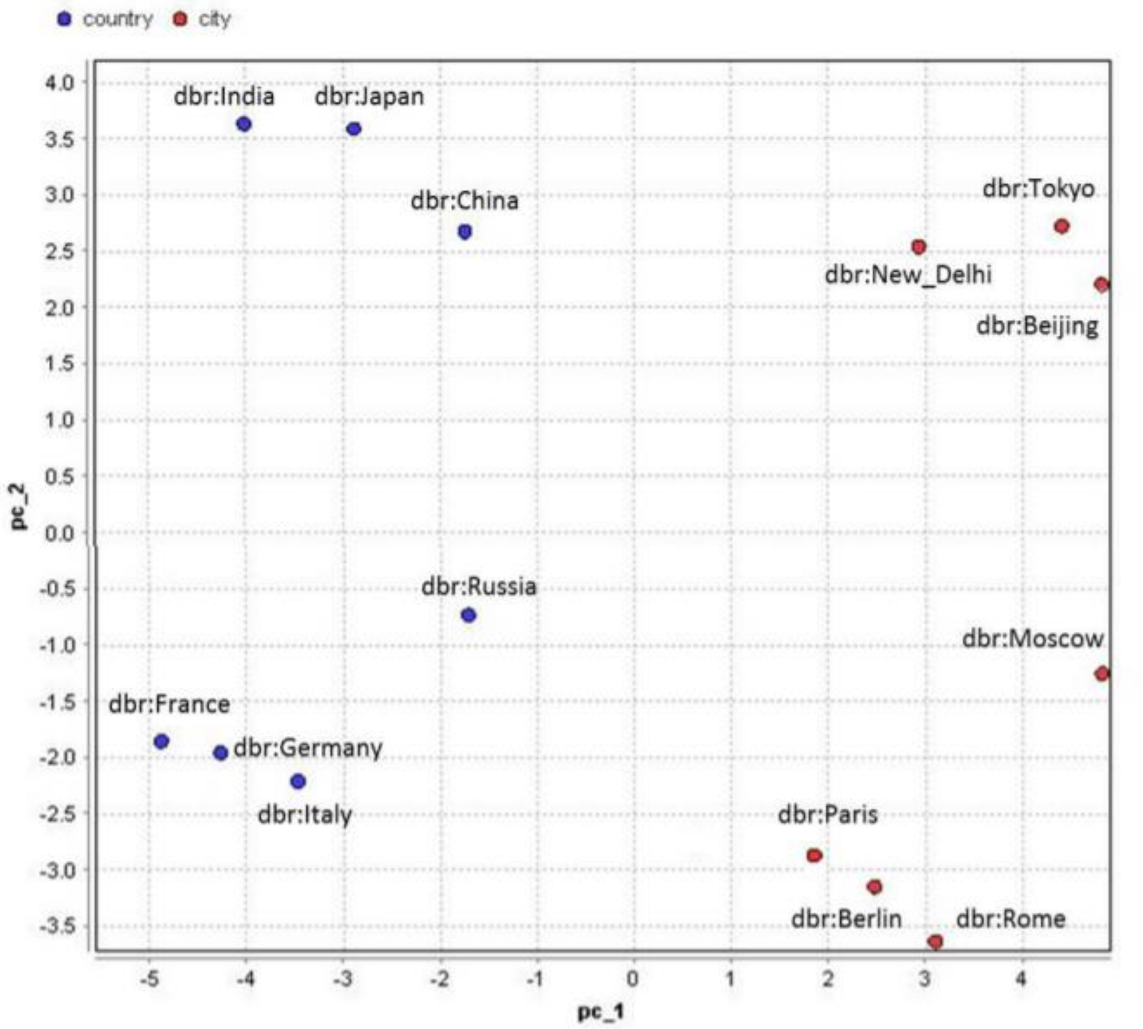
- Fuzzy reasoning
  - What is the probability of an entity being a politician, given that they are also actors?
- Type Recommendation
- Profiling ontology coherence
  - How closely does the data conform to the declaratives?

Approach

- Embed instances in knowledge graph in vector space
  - Used existing algorithm (RDF2Vec)



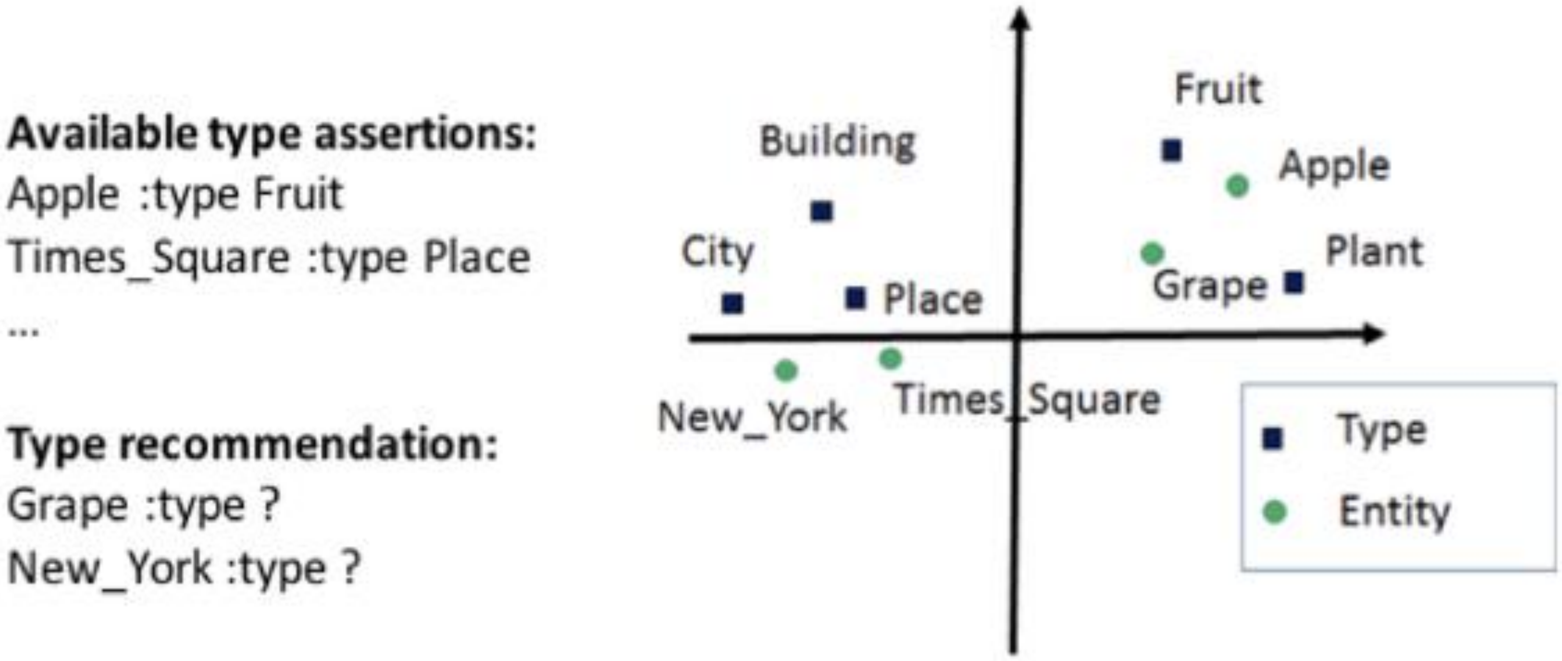
RDF2Vec: Some visualizations



- Based on DeepWalk algorithm
- Results are fairly intuitive

Approach: intuition

- Construct type embeddings in the same vector space as pre-computed entity embeddings



Algorithm

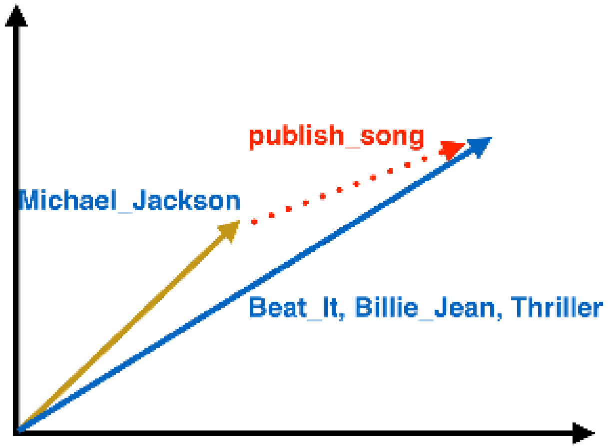
Algorithm 1 Generate Type Embeddings

Input: Sets S and $\vec{S}$ of entities and entity embeddings, type-only Knowledge Base $\mathcal{T}'$
Output: Type embedding $\vec{t}$ for each type t in $\mathcal{T}'$

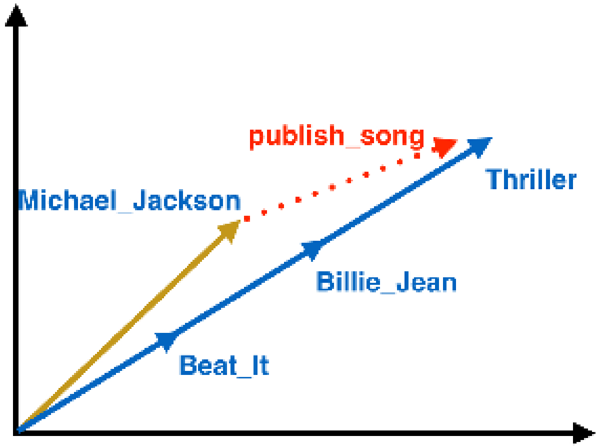
1. Initialize empty dictionary T_S where keys are entities and values are type-sets
2. Initialize type-set T of $\mathcal{T}'$ to the empty set
   // First pass through $\mathcal{T}'$: collect entity-type statistics
3. **for all** triples $(s, : type, t) \in \mathcal{T}'$ such that $\vec{s} \in \vec{S}$ **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 **end for**
   //Second pass through $\mathcal{T}'$ to derive type embeddings
6. Initialize Mean parameter dictionary M such that $keys(M) = T$, and each value in M is $\vec{0}$
7. **for all** triples $(s, : type, t) \in \mathcal{T}'$ such that $s \in S$ **do**
   Update M[t] using Equation 1, using $T(s) = T_S[s]$
8. **end for**
   //Derive type embedding from $\vec{\mu_t}$
9. **for all** types $t \in keys(M)$ **do**
   Let type embedding $\vec{t}$ 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…

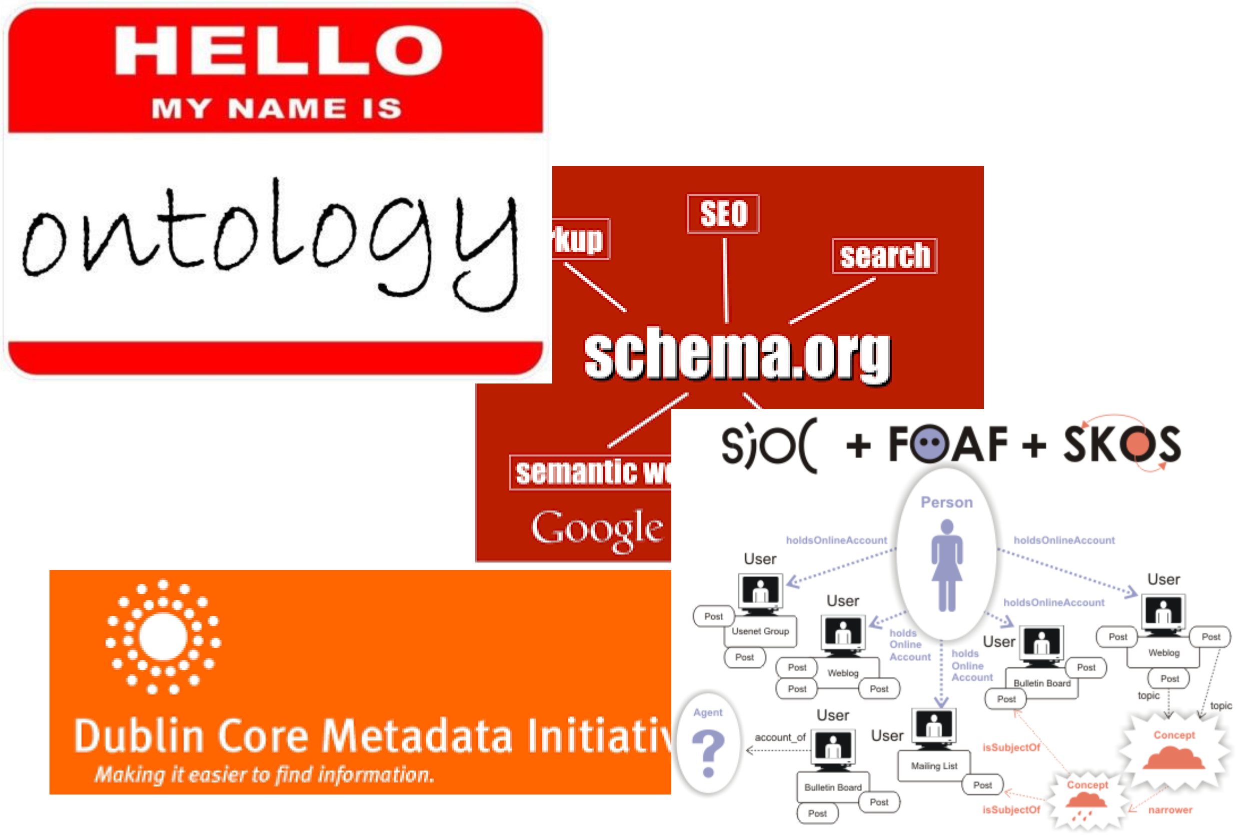


(a) TransE



(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

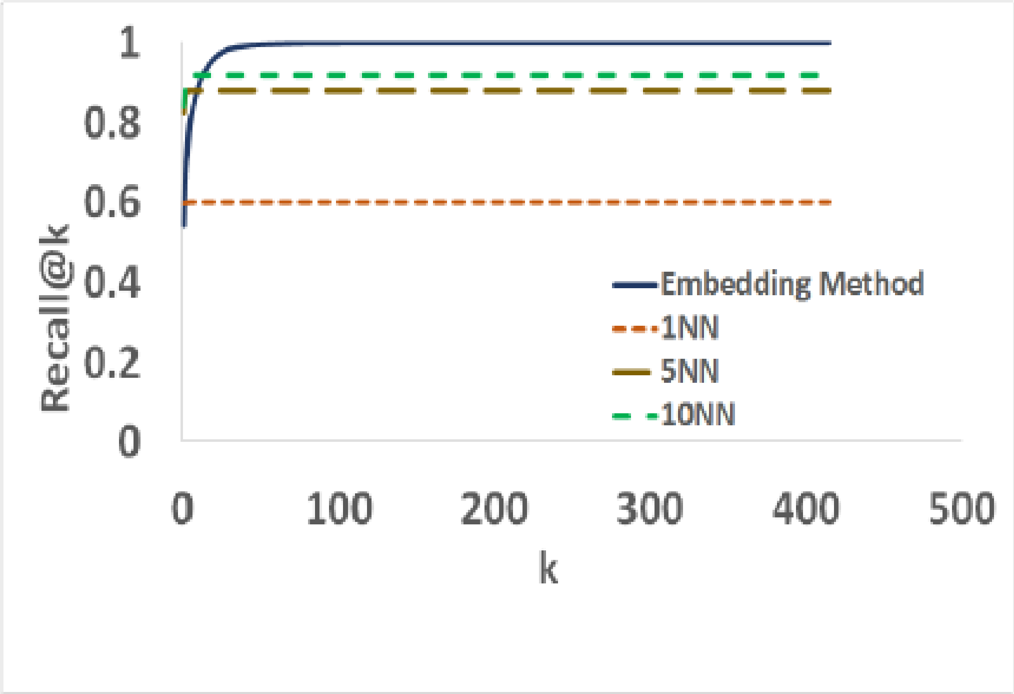
Experiments

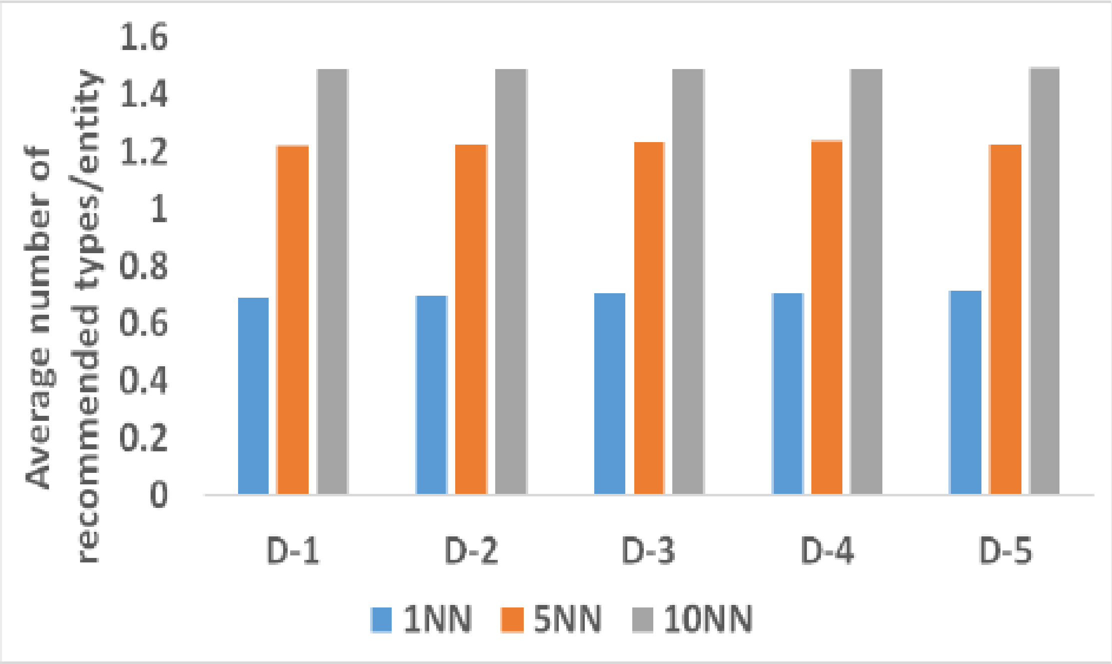
- Partitioned DBpedia knowledge graph into five sets

Dataset	Num. triples	Num. unique instances	Num. unique types	Size on disk (bytes)
D-1	792,835	792,626	410	113,015,667
D-2	793,500	793,326	412	113,124,417
D-3	793,268	793,065	409	113,104,646
D-4	793,720	793,500	410	113,168,488
D-5	792,865	792,646	410	113,031,346

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!



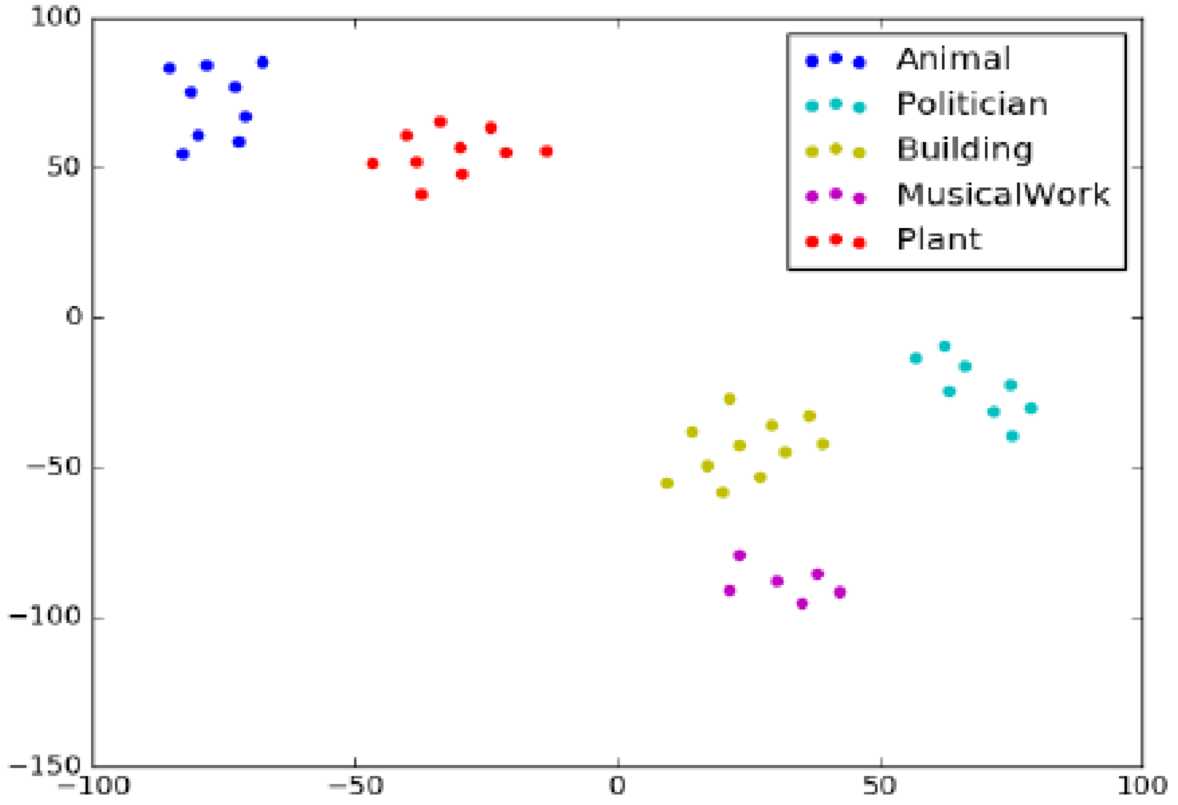


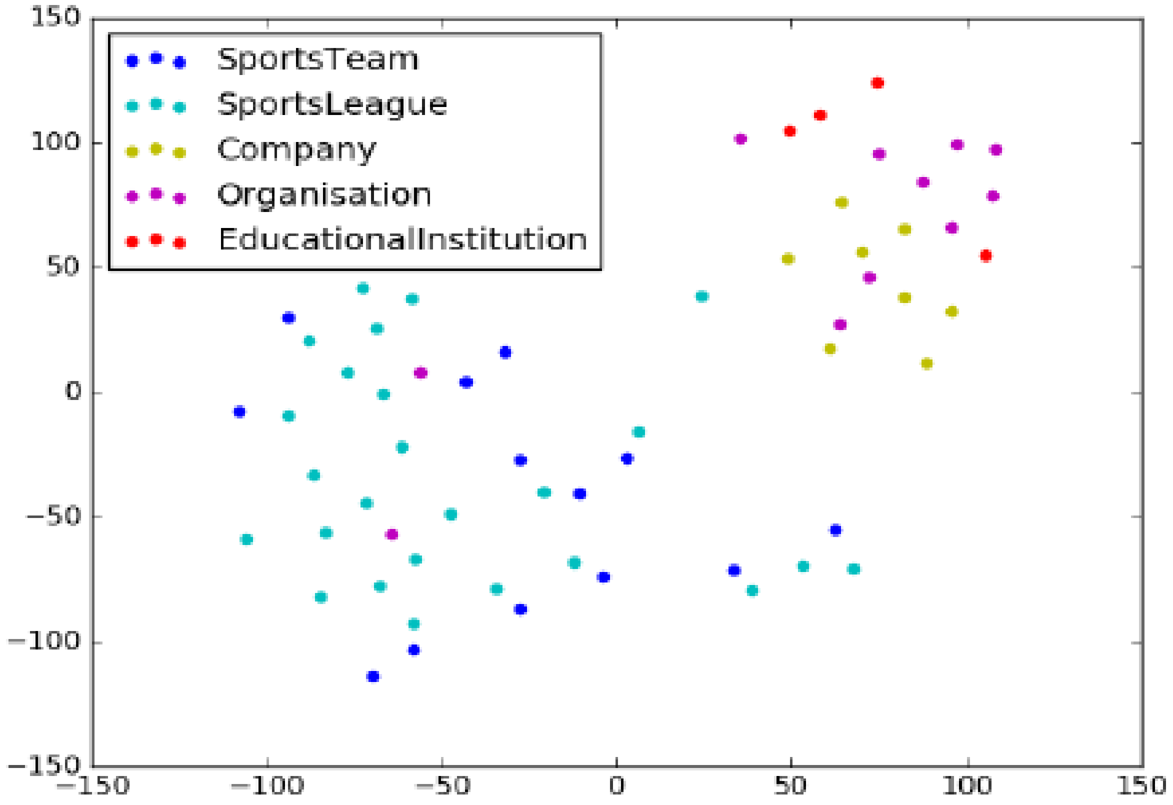
Task 2: Type Recommendation

Entity	Embedding Method Rec.
Shenyang_J-13	Aircraft, Weapon, Rocket
Amtkeli_River	River, BodyOfWater, Lake
Melody_Calling	Album, Single, Band
Esau_(judge_royal)	Monarch, Loyalty, Noble
Angus_Deayton	Comedian, ComedyGroup, RadioProgram

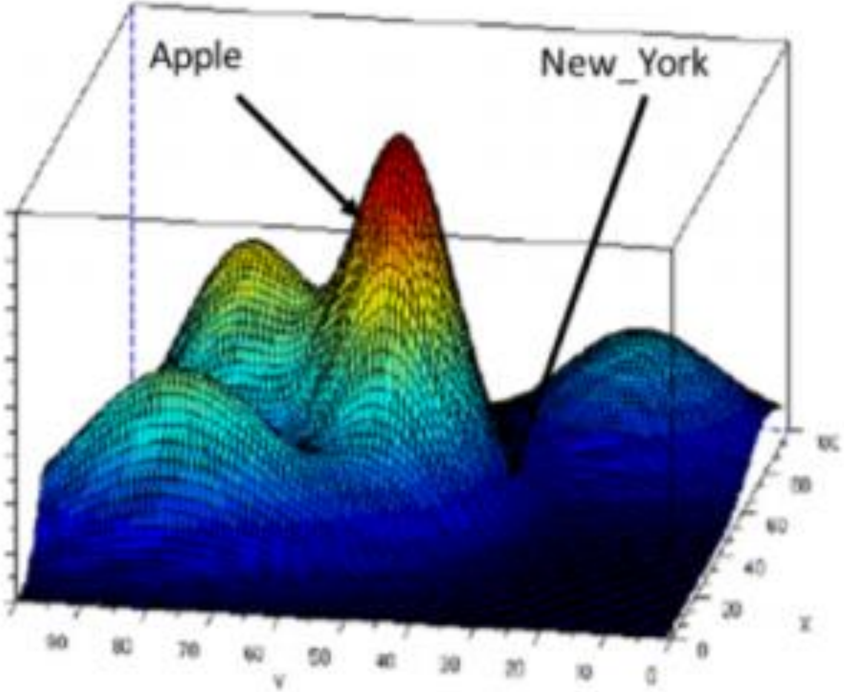
- Possible because we get a scored list of types with embedding method

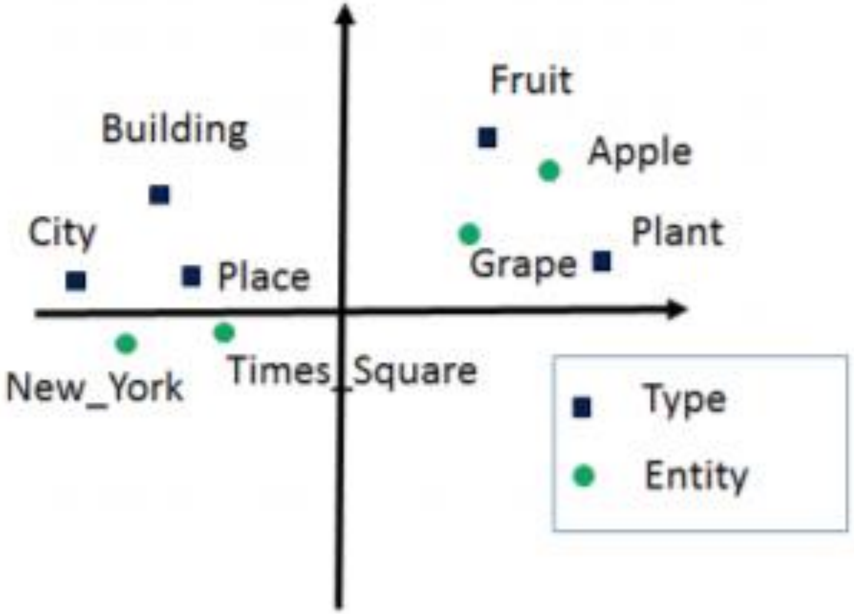
Task 3: Ontology Coherence





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-isi-i2.github.io/home/

{kejriwal, pszekely}@isi.edu