Fast Entity Linking via Graph Embeddings

Alberto Parravicini
Rhicheek Patra
Davide Bartolini
Marco Santambrogio

2019-06-30, GRADES-NDA
Entity Linking (EL): connecting words of interest to unique identities (e.g. Wikipedia Page)

“The Indiana Pacers and Miami Heat [...] meet at Miami's American Airlines Arena”

- en.wikipedia.org/wiki/Indiana_Pacers
- en.wikipedia.org/wiki/Miami
- en.wikipedia.org/wiki/Miami_Heat
- .../wiki/American_Airlines_Arena
Use Cases

Component of applications that require high-level representations of text:

1. **Search Engines**, for semantic search

2. **Recommender Systems**, to retrieve documents similar to each other

3. **Chat bots**, to understand intents and entities
The EL Pipeline (1/2)

An EL system requires 2 steps:

1. **Named Entity Recognition (NER):** spot mentions (a.k.a. Named Entities)
   - High-accuracy in the state-of-the-art\(^1\)

```
"Trump will answer Clinton's claims about the Wall"
```

<table>
<thead>
<tr>
<th>Text</th>
<th>Trump, Clinton, Wall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>PER, PER, LOC</td>
</tr>
<tr>
<td>Offset</td>
<td>0, 3, 7</td>
</tr>
</tbody>
</table>

\(^1\) Huang, Zhiheng, Wei Xu, and Kai Yu. "Bidirectional LSTM-CRF models for sequence tagging."
The EL Pipeline (2/2)

An EL system requires 2 steps:

2. **Entity Linking**: connect mentions to entities

```
"Trump will answer Clinton's claims about the Wall"
```

<table>
<thead>
<tr>
<th>Text</th>
<th>Type</th>
<th>Entity</th>
<th>Wikipedia Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trump</td>
<td>PER</td>
<td>.../wiki/Donald_Trump</td>
<td></td>
</tr>
<tr>
<td>Clinton</td>
<td>PER</td>
<td>.../wiki/Hillary_Clinton</td>
<td></td>
</tr>
<tr>
<td>Wall</td>
<td>LOC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The EL Pipeline (2/2)

An EL system requires 2 steps:

2. **Entity Linking**: connect mentions to entities

"Trump will answer Clinton's claims about the Wall"

Text: Trump, Clinton, Wall
Type: PER, PER, LOC

…/wiki/Donald_Trump …/wiki/Hillary_Clinton …/wiki/Defensive_wall …/wiki/Berlin_wall …/wiki/Mexico–United_States_barrier
The EL Pipeline (2/2)

An EL system requires 2 steps:

2. **Entity Linking**: connect mentions to entities

```
"Trump will answer Clinton's claims about the Wall"
```

<table>
<thead>
<tr>
<th>Text</th>
<th>Type</th>
<th>Wiki Link</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trump</td>
<td>PER</td>
<td>../wiki/Donald_Trump</td>
</tr>
<tr>
<td>Clinton</td>
<td>PER</td>
<td>../wiki/Hillary_Clinton</td>
</tr>
<tr>
<td>Wall</td>
<td>LOC</td>
<td>../wiki/Mexico–United_States_barrier</td>
</tr>
</tbody>
</table>
Our contributions

- Novel unsupervised framework for EL
  - No dependency on NLP
- First EL algorithm to use graph embeddings
  - Accuracy similar to supervised SoA techniques
- Highly scalable and real-time execution time
  - $< 1$ sec to process text with 30+ mentions
We obtain a large graph from DBpedia

- All the information of Wikipedia
- Stored as triples
- 12M entities
- 170M links
Embeddings Creation (1/2)

- Graph embeddings encode vertices as vectors
  - “Similar” vertices have “similar” embeddings
- Idea: entities with the same context should have low embedding distance

News: Trump will answer Clinton’s claims regarding the Wall.
In our work, we use **DeepWalk**\(^1\) like **word2vec**\(^2\), it leverages random walks (i.e. vertex sequences) to create embeddings.

- Embedding size 170, walk length 8

DeepWalk uses only the graph topology.

- Simple baseline, we can use better algorithms and leverage graph features.

---

Candidate Finder

Idea: for each mention, select a few candidate vertices with index-based **string similarity**

- Solve ambiguity following **redirect** and **disambiguation** links

"Trump will answer Clinton's [...]"

```
String Similarity

Trump

String Similarity

Trump,_Donald

disambiguates

Ivanka_Trump

redirects

Trump_Tower

Donald_Trump
```
Step 3/4

Candidate Finder

Idea: for each mention, select a few candidate vertices with index-based **string similarity**

- Solve ambiguity following **redirect** and **disambiguation** links
Disambiguation (1/3)

- We want to pick the “best” candidate for each mention
  - In a “good” solution, candidates are related to each other (e.g. Donald Trump, Hillary Clinton)

- Observation: a good tuple of candidates has embeddings close to each other

- Evaluating all combinations is infeasible
  - 10 mentions with 100 candidates \(100^{10}\)
We use an heuristic state-space search algorithm to maximize:

$$\text{Best Tuple} = \arg\max_T \left( \sum_{t_i \in T} \text{Local}(t_i) + \text{Global}(T) \right)$$

Sum of string similarities

$$\bar{e}(T) = \frac{1}{|T|} \sum_{t_i \in T} e(t_i)$$

Global($T$) = $\sum_{t_i \in T} \frac{\langle e(t_i), \bar{e}(T) \rangle}{\|e(t_i)\|_2 \|\bar{e}(T)\|_2}$

Sum of embedding cosine similarities w.r.t. tuple mean
Disambiguation (3/3)

- Iterative state-space heuristic

```
Function optimizer(candidates):
    T = find_initial_state(candidates)

    while stop condition not met do
        // Create num_children new tuples by modifying
        random elements of T.
        T = new_tuples(T, num_children)
        T = optimize_tuple(T)
        s = compute_score(T)
        if curr_score ≥ best_score then
            best_tuple = T
            best_score = s
        end
    end

    return best_tuple, best_score
```

Greedy iterative procedure

\[
\text{argmax}_T \left( \sum_{t_i \in T} \text{Local}(t_i) + \text{Global}(T) \right)
\]
Results: accuracy

- We compared against 6 SoA EL algorithms, on 5 datasets
- Our Micro-averaged $F_1$ score is comparable with SoA supervised algorithms

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Ours</th>
<th>DoSeR</th>
<th>WK</th>
<th>AIDA</th>
<th>WAT</th>
<th>BB</th>
<th>SL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACE2004</td>
<td>0.84</td>
<td>0.90</td>
<td>0.83</td>
<td>0.81</td>
<td>0.80</td>
<td>0.56</td>
<td>0.71</td>
</tr>
<tr>
<td>AQUAINT</td>
<td>0.86</td>
<td>0.84</td>
<td>0.86</td>
<td>0.53</td>
<td>0.77</td>
<td>0.65</td>
<td>0.71</td>
</tr>
<tr>
<td>MSNBC</td>
<td>0.92</td>
<td>0.91</td>
<td>0.85</td>
<td>0.78</td>
<td>0.78</td>
<td>0.60</td>
<td>0.51</td>
</tr>
<tr>
<td>N3-Reuters</td>
<td>0.82</td>
<td>0.85</td>
<td>0.70</td>
<td>0.60</td>
<td>0.64</td>
<td>0.53</td>
<td>0.58</td>
</tr>
<tr>
<td>N3-RSS-500</td>
<td>0.72</td>
<td>0.75</td>
<td>0.73</td>
<td>0.71</td>
<td>0.68</td>
<td>0.63</td>
<td>0.62</td>
</tr>
</tbody>
</table>

WK is Wikifier, BB is Babelfy, SL is Spotlight
Results: exec. time

- Different settings enable real-time EL, with minimal loss in accuracy
  - E.g. number of iterations, early stop
Conclusion & Future Work

● In short:
  - First EL algorithm to use graph embeddings
  - Accuracy similar to supervised SoA techniques
  - Real-time execution time

● Future works:
  - Better graph embeddings algorithms (use vertex/edge features)
  - Improve disambiguation algorithm for even faster exec. time
Thank you!

Fast Entity Linking via Graph Embeddings

- Novel unsupervised framework for EL
- First EL algorithm to use graph embeddings
  - Accuracy similar to supervised SoA techniques
- Real-time execution time

Alberto Parravicini, alberto.parravicini@polimi.it
Rhicheek Patra
Davide Bartolini
Marco Santambrogio

2019-06-30, GRADES-NDA
Embeddings

- Turn topology and properties of each vertex into a vector
- “Similar” vertices have “similar” embeddings

Graph Creation

We obtain a large graph from DBpedia

- All the information of Wikipedia, stored as triples
- 12M entities, 170M links
Graph Creation

… and join them together

The_Trump_Building

Donald_Trump

Ivanka_Trump

Fred_Trump

1946

New_York

NY

NYC

Queens

Trump,

Trump,

New_York_City

redirects

partOf

redirects

redirects

redirects
Candidates Finder

Idea: for each mention, select a small number of candidate vertices with **string similarity**

- We use a simple **index-based** string search
- Fuzzy matching with **2-grams and 3-grams**
- This provides a simple baseline (60-70% accuracy)
Results: exec. time of single steps

- Execution time is well divided between Candidate Finder and Disambiguation