Google Knows Who is Famous Today
Building an Ontology From Search Engine Knowledge and DBpedia

Christopher Ochs, Tian Tian, James Geller
New Jersey Institute of Technology (NJIT)
Newark, NJ, USA

Soon Ae Chun
City University of New York (CUNY)
Staten Island, NY, USA
Overview

• Background

• Building an Ontology of Famous People
  • Who are the famous people?
  • Extracting Useful Relationships

• Dynamically Expanding our Ontology

• Conclusions and Future Work
Suggested Completions

Search engines, such as Google, provide suggested query completions to users.
Ambiguous Query Suggestions

- Query suggestions are often **ambiguous**.
- What if a user searches for:
  
  Penn Station, Paul Simon, George Bush

- **Homonyms** present many interesting problems for query suggestions.

- Ambiguous query suggestions will usually lead to results for the **most common or popular** homonym.
Ontology-Supported Web Search (OSWS)

• To address this problem we have been developing the **Ontology-Supported Web Search (OSWS)** System.

• OSWS provides **disambiguated query completions** generated using an **ontology**.
  
  • [Tian, Geller, Chun. Vienna 2010]
  • [Tian, Geller, Chun. London 2011]
Ontology-Supported Web Search

michael jordan
michael jordan soccer player
michael jordan soccer player belongs to club boreham wood f.c.
michael jordan soccer player plays for team arsenal f.c.
michael jordan soccer player plays for team eastbourne borough f.c.
michael jordan soccer player plays for team stevenage f.c.
michael jordan soccer player plays for team chesterfield f.c.
michael jordan soccer player plays for team england national under-17 football team
michael jordan basketball player
michael jordan basketball player plays position small forward
michael jordan basketball player plays position shooting guard
michael jordan basketball player plays for team washington wizards
michael jordan basketball player plays for team chicago bulls
michael jordan basketball player attended college north carolina tar heels men s basketball
michael jordan basketball player born in brooklyn
Improving the OSWS Ontology

• The first iteration of the OSWS ontology was severely limited in domain.

• It lacked information we felt users would find useful; such as:
  Album names, movie titles, teams athletes play on

• To further validate our design, we needed to build a new ontology that covered a much larger domain.
Overview

• Background

• Building an Ontology of Famous People
  • Who are the famous people?
  • Extracting Useful Relationships

• Dynamically Expanding our Ontology

• Conclusions and Future Work
Building an Ontology of Famous People

• We are building a special purpose ontology focused on famous people.

• How do we know somebody is famous? How do we know someone is famous right now?

• Google knows who is famous today! We define someone as famous if his or her name appears as a Google query suggestion.

• If a name is a suggestion it implies many people have searched for it.
Building the A-List

- Used **U.S. Census Data** to find most common first names.
- First we built the **A-List**, the “most famous” people.
- Created by querying Google’s query suggestion API with **just first names**.
- *John* → John Jay, Johnny Depp, John Deere, etc.
- Using this method we found **5,286** candidates.
The B-List and C-List

• Next we queried in the form “n1 l1,” where l1 is a letter from the alphabet. E.g. Gary B, Will F, Albert E

• This is the B-List. It contains 132,896 candidates.

• Finally, we queried “n1 l1 l2” where l2 is another letter. E.g. Gary Bu, Will Fe, Albert Ei

• This set was named the C-List. There were nearly 1,000,000 candidates.
Matching Names to People

• The **A-List** was the starting point of our ontology.

• **How** do we determine if a candidate name can refer to person?

  Sam Adams, Sterling Silver, Tian Tian

• **How** will we populate our ontology with relevant information?
DBpedia

- **DBpedia** is one of the world’s largest multi-domain ontologies.
- Contains information on over 3.5 million entities, including over 360,000 people.
  - [Bizer et. al 2009]
- DBpedia is built from Wikipedia pages and other ontologies.
Building an Ontology of Famous People

• DBpedia is a fantastic resource, but could not be used “out of the box” for OSWS.

• Not structured in a way as to provide good query suggestions.

• Take what DBpedia has and tune it to our own needs.

• We extracted a subset of DBpedia and restructured it for our own use.
Finding Famous People

• Our system queried DBpedia with the 5,286 names in the A-List.

• Class and Wikipedia category information was then analyzed.

• If we found an entity belonging to class “person,” or Wikipedia categories “year_births” or “year_deaths,” we considered them a person.

• Using this method, we found that there were at least 3,241 people in the A-List.
Overview

• Background

• Building an Ontology of Famous People
  • **Who are the famous people?**
  • Extracting Useful Relationships

• Dynamically Expanding our Ontology

• Conclusions and Future Work
Classifying Famous People

• For each person in the A-List our system extracted information from DBpedia.

• We used the DBpedia Person Hierarchy. Many entities could be directly mapped into our ontology.

• Many names were not found within DBpedia’s ontology, or their classification was not specific. E.g. “person”
Yet Another Great Ontology (YAGO)

• YAGO (Yet Another Great Ontology) is based off of Wikipedia and WordNet.

• Instances are assigned to classes based on their Wikipedia categories.
  • [Suchanek, Kasneci, Weikum. 2007]

• YAGO categories are often far too specific ("AmericanPeopleOfCanadianDescent"), so we go “one level up” the YAGO hierarchy to the more general WordNet synsets and map from there.
Mapping From YAGO

- Analyzed entities in both DBpedia’s ontology and YAGO to create a mapping between the ontologies.
- We mapped the top 85 YAGO classes in the A-List.

<table>
<thead>
<tr>
<th>Broader YAGO Class Name</th>
<th>DBpedia Ontology Mapping</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actor</td>
<td>Actor</td>
</tr>
<tr>
<td>Anthropologist</td>
<td>Scientist</td>
</tr>
<tr>
<td>Blogger</td>
<td>Writer</td>
</tr>
<tr>
<td>Drummer</td>
<td>MusicalArtist</td>
</tr>
<tr>
<td>Marine</td>
<td>MilitaryPerson</td>
</tr>
</tbody>
</table>
Mapping From YAGO (cont.)

• We map all YAGO classes an instance belongs to, and then choose the most common mapping.

• For example, Ronald Reagan would be mapped as:

<table>
<thead>
<tr>
<th>YAGO Class</th>
<th>WordNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>AmericanFilmActors</td>
<td>Actor</td>
</tr>
<tr>
<td>UnitedStatesArmyOfficer</td>
<td>MilitaryOfficer</td>
</tr>
<tr>
<td>PresidentsOfTheUS</td>
<td>President</td>
</tr>
<tr>
<td>AmericanPoliticiansOfIrish..</td>
<td>Politician</td>
</tr>
<tr>
<td>DelegatesToTheRNC</td>
<td>Delegate</td>
</tr>
<tr>
<td>GovernorsOfCalifornia</td>
<td>Governor</td>
</tr>
</tbody>
</table>

• This method gave us 67% correlation and more specific classification for 16% of instances.
Other Mapping Methods Used

• Wikipedia adds tags to page names, e.g. John_Adams_(American_football)

• Wikipedia pages have an abstract, the first paragraph, usually in the form $x$ [is|was] a $y$, where $x$ is a name and $y$ is a class.
  • Thomas "Tom" Hanks is an American actor, producer, writer, and director.

• Projects such as Unipedia (ICSC 2010) use both mapping methods as well.
  • [Kalender, Jiangbo Dang, Uskudarli. 2010]
Choosing The Best Class

• When multiple sources existed we performed all of the mapping methods.

• We chose the classification that is lowest in the hierarchy, the most specific, to be the class an instance is assigned to.

• For example, if DBpedia gives a classification of “Musical Artist” and our YAGO mapping gives a classification of “Artist,” we use “Musical Artist.”
Overview

• Background

• Building an Ontology of Famous People
  • Classifying Famous People
  • Extracting Useful Relationships

• Dynamically Expanding our Ontology

• Conclusions and Future Work
Extracting Useful Relationships

• Previous OSWS ontology was severely lacking in class-specific relationships and attributes. [Tian, Geller, Chun. 2010].

• DBpedia has many useful attributes & relationships, but had to be organized for search queries.

• OSWS required relationships that were most useful for suggested query completions.

• Next we identified and extracted useful relationships and attributes from DBpedia.
Choosing The Best Relationships

• For each class we counted how often a type of relationship occurred.

• If a type of relationship existed for fewer than 50% of instances of a class, we removed it.

• Removed the most commonly occurring relationships.  
  E.g. Abstract, image, category, etc.

• Performed a manual review of removals and remaining relationships.
Improving Relationships

• Redundant relationships were fairly common.

• For example: Tom Hanks has
  • dbppedia:starring Road to Perdition
  • dbpedia-owl:starring Road to Perdition

Along with many others.

• Redundant relationships were combined into a single relationship, such as starring.
Tom Hanks stars in the movie Saving Private Ryan.
Improving Relationships (cont.)

- Some relationships were ambiguous. E.g. “writer of” could mean: writer of a song, writer of a television show, writer of a book, etc.

- If a significant portion of relationship targets had different types, we split the relationship into many relationships of finer granularity.

- In the above example, we created “writer of song,” “writer of book,” “writer of movie,” etc.

- Produced, Starring, Performed, Writer, etc.
Our ontology is person-focused, DBpedia isn’t.

All relationships have sources that are people.

It is common to see relationships in the form [Movie] starring [Actor].

For many relationships our system reversed the subject and object.

For example, in DBpedia a relationship is [Forest Gump] starring [Tom Hanks].

While building an instance our system reversed the relationship to be [Tom Hanks] stars in film [Forest Gump].
Promoting Attributes to Relationships

- Certain **attributes** in DBpedia were promoted into full relationships.
- Attributes such as *instruments played* or *genres* were turned into relationships.
- In DBpedia a musician may have the attribute **“plays instrument”** with the value **“piano, keyboard.”**
- We split the list at the commas and create multiple relationships.
Relationship Targets

• The targets of non-interpersonal relationships were organized into a hierarchy similar to DBpedia.

• For inter-personal relationships we had to consider the issue of recursion.

• If a target person does not exist in our ontology then he or she is not considered famous. However, the target needs to exist to complete the new instance.

• We mark the target person as a “stub,” an incomplete instance.

• Stubs are not provided as query suggestions.
Query Google suggestion API with common first names

If a suggestion is a possible name, add to A-List

For all names in the A-List, query DBpedia with name

If name exists in DBpedia, save type and relationship data

Remove all names not correlating to a Person type

Create mappings for YAGO/disambiguation tags

Assign all names to a DBpedia ontology Person type

Analyze types of relationships that exist for each Person type

Keep most common relationships for each Person type

Keep most useful relationships for each Person type; analyze targets

Split relationships where necessary; build target hierarchy

For each person in the A-List, add them to the ontology and extract relationship data from DBpedia
Overview

• Background

• Building an Ontology of Famous People
  • Classifying Famous People
  • Extracting Useful Relationships

• Dynamically Expanding our Ontology

• Conclusions and Future Work
Dynamically Expanding the OSWS Ontology (D-OSWS)

• Keeping an ontology up to date is a labor-intensive and error-prone task.

• Required a way to **dynamically add new instances** while maintaining nearly the same quality as the A-List ontology.

• We created the **Dynamic Ontology-Supported Web Search (D-OSWS) System.**
Dynamic Ontology-Supported Web Search

• **D-OSWS** automatically adds new instances to the ontology during normal use. Uses systems developed for building A-List ontology.

• When a user enters a search term into OSWS, we follow the same process as when we built the A-List.

1. Query Google Suggestion API.
2. Query DBpedia with suggestions.
3. Add any instances NOT in our ontology.
4. Return the new suggestions to the user.
Return Suggestions To User

User Enters Query

Search Ontology For Suggestions

Query Google Suggest API

Query DBpedia for Missing Instances

Add New Instances

Add New Suggestions to D-OSWS

Add New Relationships & Targets
Overview

• Background

• Building an Ontology of Famous People
  • Classifying Famous People
  • Extracting Useful Relationships

• Dynamically Expanding our Ontology

• Conclusions and Future Work
Conclusions

• OSWS disambiguates query completions in a user-friendly manner by creating suggestions from an ontology.

• We built a new ontology by finding famous people via Google.

• The ontology consists of nearly 22,000 concepts linked by over 90,000 relationships; centered around 3,241 people in the A-List.

• The D-OSWS system dynamically extends the ontology during use.
Future Work

• Higher granularity in person hierarchy
• New data sources & DBpedia Live
• Search What I Mean (SWIM)
  • Do What I Mean (DWIM) [Teitelman]
• Quality assurance mechanisms
• Building the B-List and C-List ontologies
• Filtering search results with ontology
Try It Out!

- The Ontology-Supported Web Search is live right now.
- Feel free to try it out.
- D-OSWS will be live in the near future, including B-List ontology.

http://osws.njit.edu
Thank You!

Questions?