**Providing Matches for Context Disambiguation**

**Introduction**

The problem this system solves is as follows: Given an entity from a ReVerb tuple, quickly come up with k Freebase entities that are likely candidates purely based on string matching. For example, for the tuple “(Clinton, visited, New York)”, top candidate entities such as Bill Clinton and Hillary Clinton should be returned. This system employs a variety of matching techniques such as exact string matching, exact cleaned string matching, full word substring matching, Wikipedia alias matching, acronym matching, and Lucene spell correction matching. On a test set of 93 unique terms from real news articles, our best system is able to return the correct match in the top 5 matches 83% of the time at a speed of 150 ReVerb entities per second. This is a 41% improvement over the naïve approach of using only exact string matches and ordering on prominence.

**Naïve Approaches**

A naïve approach to solving this problem would be using only exact string matches and ordering on matches; however, this misses many matches such as: “US” matching “United States”, “Obama” matching “Barack Obama”, and “Gadafy” matching “Muammar al-Gaddafi”. When we evaluated this approach, we found that it achieved 42% accuracy.

Another simple approach would be to just take substring matches, ordering on prominence. When we evaluated this approach coupled with exact string matching and prominence ordering, we found that it achieved 58% accuracy. Used alone, it provided only 20% accuracy. The reason it is so low is because pure substring matching provides many low quality matches that push the desired matches down past the 5 match threshold. Also this method still misses matches such as “US” matching “United States” and “Gadafy” matching “Muammar al-Gaddafi”.

These simple approaches are able to correctly match some simple types of matches, but they miss more complex types of matches. Table 1 enumerates a list of the different types of matching that a system might be required to handle in order to increase accuracy to a reasonable level.

Table 1: Outlines cases handled by our system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | **Example Case** | | |
|  | **Match Type** | **ReVerb Entity** | **Freebase Entity** | **Handled?** |
| 1 | Exact match between arg1 and Freebase entity | Bill Clinton | Bill Clinton | Yes |
| 2 | Exact match between cleaned arg1 and cleaned Freebase entity | Titanic | Titanic (1997 film) | Yes |
| 3 | Arg1 is a substring of the Freebase match | Christ | Christian | No |
| 4 | Freebase match is a substring of arg1 | Christian | Christ | No |
| 5 | String overlap between arg1 and Freebase match | President Clinton | Bill Clinton | Yes |
| 6 | Arg1 is an abbreviation of a Freebase match | G.E. | General Electric | Yes |
| 7 | Misspelling match between arg1 and Freebase match | Gadafy | Muammar al-Gaddafi | Yes |
| 8 | Prominent Alias between arg1 and Freebase match | America | United States | Yes |
| 9 | Obscure Alias between arg1 and Freebase match | The Red Planet | Mars | No |

**Key Components in System**

In addition to the components listed below, we also incorporated a scoring algorithm for ordering matches. The naïve approaches ordered only on prominence, but this approach doesn’t take into account the fact that some matching techniques yield higher quality matches than others. For example, an exact string match is higher quality than a single word-overlap match. All of the matching techniques listed below use case insensitive matching.

1. *Scoring Algorithm*: The scoring algorithm assigns a weight to each type of matching, and the score is computed by adding all of the weights together. The weights are on a scale of 0 to 100 For example, if a Freebase entity has an exact string match, 100 points would be added to its overall score. Every match’s score is initialized to be the natural log of its inlink count. Normalizing the prominence is necessary to prevent the prominence from effecting the score too much.
2. *Exact String Matching*: Uses hash table stored in memory for fast lookup.
   1. Generally provides a few very high quality matches such as “Bill Clinton” matching “Bill Clinton”
   2. *Scoring*: If a Freebase entity is an exact string match, 100 points are added to its score. An exact match is the best type of match, which is why it is assigned the highest weight possible.
3. *Exact Cleaned String Matching*: Freebase entities are cleaned, then loaded into a hash table for fast lookup of a cleaned ReVerb entity. Cleaning an entity involves removing commas, periods, quotes, html special characters, and removing parts of string in parentheses (“Titanic (1997 film)” would become “Titanic”)
   1. Handles cases like “Great Britain&rsquo” matching “Great Britain”.
   2. *Scoring*: If a Freebase entity is an exact cleaned string match, 20 points are added to its score. This was chosen as the same as a single word-overlap match because cleaned strings are generally about the same quality as word-overlap matches and when we increased the weight of cleaned matches it decreased accuracy by pushing word-overlap matches below the top 5. A match is only considered to be a cleaned string match if it is not a word-overlap match, abbreviation match or exact string match.
4. *Wikipedia Alias Matching*: Uses a list generated from Wikipedia of different articles that refer to the same entity in Wikipedia. For example, the article with the title “America” refers to the article with the title “United States”. Aliases are cleaned as in (2) and loaded in a hash table for fast lookup of a cleaned ReVerb entity.
   1. Handles cases like “America” matching “United States” or “Feb” matching “February”
   2. *Scoring*: An exact alias match generally provides a very high quality match, but still not quite as good as an exact string match, which is why these matches get a weight of 90.
5. *Word-Overlap Matching*: Determines how many words overlap between two strings. This method is better than pure substring matching for two reasons. One is that substring matching is much slower than this technique. For this method, a hash table can be loaded with each word for fast lookup of a cleaned ReVerb entity, while in substring matching; many string comparisons need to be made for each match. Another reason is that this method returns higher quality matches. Full word matches are generally better than partial word matches. Using only word-overlap matching achieves 57% accuracy, which is a 37% improvement over substring matching.
   1. Handles cases like “William Jefferson Clinton” matching “William Clinton”.
   2. *Scoring*: For each word that overlaps between the ReVerb entity and the Freebase entity, 20 points is added to the score. Most of these match types only have one word overlapping, which generally gives just a 20 point boost. The lower score makes sure that word-overlap that has high prominence will still most likely be lower than the other types of matching, which generally give higher quality results. A match is only considered to be a word-overlap match if it is not an exact string match or abbreviation match.
6. *Abbreviation Matching*: If a ReVerb entity looks like an acronym, then it is matched to the acronyms of all Freebase entities with at least 20 inlinks. A ReVerb entity looks like an acronym if it is composed of all capital letters. It could also have periods in between the letters, but is not required to. The acronyms for Freebase entities with more than 20 inlinks are computed prior to matching and loaded into a hash table for fast lookup. Freebase entities with fewer than 20 inlinks generally do not have prominent acronyms associated with them. On our dataset, all of the acronyms’ correct matches were found in the top 50,000 Freebase entities. The 20 inlink threshold includes around 300,000 of the top entities in Freebase.
   1. Handles cases like “U.S.” matching “United States” or “GE” matching “General Electric”
   2. *Scoring*: An abbreviation match has a weight of 80 points. This is because even though filtering out the less prominent Freebase entities for acronym matching removes many low quality matches, exact string matching and wiki alias matching still yields higher quality matches. A match is only considered to be an abbreviation match if it is not already an exact string match.
7. *Non-Plural Matching*: If a ReVerb entity ends with an “s”, it is stripped off and the stub is used to query against both the exact string hash table and the substring hash table. Plural entities are not handled by the Word-Overlap method, so a special case had to be made for them.
   1. Handles cases like “Egyptians” matching “Egyptian”
   2. *Scoring*: These matches receive the same score as they would if they were normal exact string or substring matches, since most words with a trailing “s” stripped off are non-plural forms or are nonsense words.
8. *Lucene Matching*: This type of matching gets spelling suggestions from a Lucene index based off of all of the words used in Freebase entities. Lucene performance is much slower than other types of matching, which is why it is only used if there are no high scoring matches returned from the other types of matching. If there is not at least one entity with a score over 100 points, Lucene will get spelling suggestions and then all of the matching techniques listed above are performed on the top 5 matches. Any duplicate matches are ignored.
   1. Handles cases like “Gadafy” matching “Muammar al-Gaddafi” and “Democrats” matching “Democratic Party (United States)”
   2. *Scoring*: Lucene matches’ weighted match scores (excluding the prominence score) are multiplied by two factors. The first is the similarity score that Lucene returns with each spelling suggestion, which is between 0 and 1. The second is a constant Lucene scoring factor of 0.2. Lucene provides many low quality matches, and these two factors, used in combination, pushes down less prominent entities below any word-overlap matches, which are generally better matches even than exact Lucene matches. Prominent entities still get pushed above other normal matches because the normalized prominence score is not affected by the Lucene weighting factor.

**Evaluation**

We used a test set of actual new articles created from queries to Yahoo Boss’s news interface. We queried "Watson" and "Jeopardy", "Wael" and "Egypt", and "the". Each of the 3 queries returned several hundred news articles. We then ran ReVerb on these news articles, and extracted out the top 50 most frequent arg1 entities from each set. This is a small dataset, and good for experimenting with which ReVerb arguments might not be in Freebase, and what kind of matching algorithms are needed to correctly match with entities from news articles. This dataset provided 150 ReVerb arg1s, of which 122 were unique entities. Duplicate entities were filtered out by hand.

We then manually labeled the ReVerb entities by examining the articles they were pulled from and matching them to a Freebase entity. ReVerb entities that did not have matching Freebase entities were not used when computing overall accuracy, since they would always be missing. ReVerb entities with multiple possible matching Freebase entities were not used because they represent extraction errors by ReVerb. Our system still provides matches for those entities, but it is difficult to say whether or not it is providing the correct matches since there are multiple possible correct answers.

We then tested our matching system on this set of entities and were able to achieve 83% accuracy for the top 5 matches. In other words, 83% of the time, the correct match was in the top 5 matches. We were able to process an average of 240 ReVerb entities per second, not including pre-computation. The system takes a minute or two to set up the hash tables used for matching and load Freebase into memory. While running, the system takes around 3GB of memory, which is a rather large amount, but it is manageable by a cluster of servers.

Graph 1

As displayed by graph one, the more matches that are returned, the higher accuracy that the system achieves; however the context matching system that these matches are passed off to cannot handle a large number of matches for two reasons. One is that providing too many matches makes it more difficult to determine which one is the correct entity from the context. The other is that the context matching system is slow, so providing even a few more matches hurts performance significantly.

**Tradeoffs**

When deciding whether or not to add a feature to the system we considered two criteria: how much it slowed the system down, and how it affected the top 5 results.

String distance and substring matching, which are implemented by Lucene matching, were not included as a major part of the system (Lucene is only used when no other methods yield high quality matches) because they were much slower than the other methods and because they also bloated the top 5 matches with many irrelevant matches. For example “Khaled” matches to “The Persian Gulf” because “Khaled” has a similar string distance to “Khaleej” which is a Wikipedia alias for “The Persian Gulf”. Substring matching also provides many low quality matches, seeing as it only achieved 20% accuracy when used alone.

Exact string matching and word-overlap matching were included as necessities for many types of matches. When using hash tables, these methods are also very fast. One downside of these methods is that they take up a lot of memory, since both methods create several million entry hash tables. Cleaned exact string matching, Wikipedia alias matching, and abbreviation matching also created several million more hash table entries, but they were also very fast for lookup. They also provide many high quality matches, especially Wikipedia alias matching. Exact string matching, cleaned exact string matching, Wikipedia alias matching, and abbreviation matching account for 2.1% of the overall time spent matching per entity. Word-overlap matching accounts for 46% of the time spent matching. This is mainly because it provides 98% of the total potential matches. Many of those matches are not high quality matches, and have lower match scores than matches retrieved through other methods.

Lucene matching took the largest majority of the time spent matching at 52%, even though it was only used on 30% of the dataset. Recall that Lucene is only used when a ReVerb entity does not have a match with a score of 100 or greater. There weren’t many ReVerb entities that this method helped, so if it was necessary to speed up the system, the threshold that determines when Lucene is used could be lowered so that it would only be used in a case where there were only very low scoring matches.

**Conclusion**

Some surprising results that we learned during this project were that substring and string distance matching methods were not very effective when used alone, while Wikipedia alias matching, exact string matching, and word-overlap matching performed very well. The reasons that the stricter matching performed better than the weaker substring and string distance matching are because most of the entities that ReVerb extracts are not misspelled and are usually contain whole words that match to a Freebase entity. Many times when words seem to be misspelled or shortened, the Wikipedia aliases will still provide the correct matches for those when word-overlap and exact matching may have missed them. This is because when an alias/shortening appears online, especially in new articles, there is a good chance it also appears on Wikipedia, and will then appear in for our alias matching.

Some possible extensions to this system include the following:

* Speeding up word-overlap matching: Currently, word-overlap matching returns far more matches than is necessary, which is part of the reason why it takes so long. To speed it up, we could divide the hash table lookup into *k* buckets, ordered by prominence, and only search the next hash table if there haven’t been a sufficient number of matches found so far.
* Tweaking the effect Lucene matching has on the top 5 matches: Currently, Lucene returns as many matches as it finds for the top 5 spelling suggestions. This is part of the reason that Lucene matches have to be weighted so low, since they can push out all of the word-overlap matches from the top 5. To help this problem, we could only return the top 3 or so matches retrieved for all of the top 5 Lucene spelling suggestions. This way, good Lucene matches could still be taken account for, but so will word-overlap matches.
* Performance upgrades: Finding a way to filter out low quality matches before computing the scoring/sorting algorithm would help speed. Better allocation and management of the data structures used could help reduce memory usage and possibly speed up pre-computation speed.

By showing an improvement in the overall accuracy of over 40% from the naïve matching method of exact string matching, this system is a valuable addition of systems for mapping textual extractions to ontologies by providing the initial matches, which helps such systems improve overall performance.