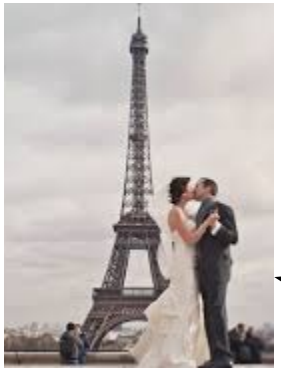
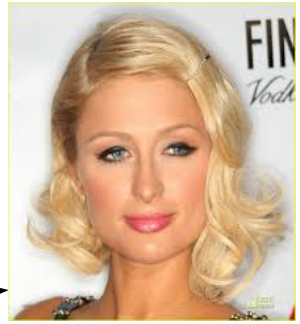
LODIE 2

Genevieve Gorrell
Johann Petrak
Kalina Bontcheva

What do we want to accomplish?

- Identify mentions of concepts in the Knowledge Base (KB) in arbitrary text, including Tweets
- Link mentions to the concept in the KB that best matches the meaning in the given context.
- Determine when a mentioned concept is not in the KB (OOKB / NIL)
- Do this efficiently for a KB with millions of concepts and with dozens or hundreds of concept candidates per mention.





... went to Paris on Thursday ...

Identify Mentions: What?

- Identify KB concept labels: Gazetteer of known possible labels for all concepts
  - DBpedia instance labels
  - DBpedia name/nickname properties (from WP templates)
  - YAGO labels of mapped YAGO instances
  - Labels from redirected WP pages (spelling variations)
  - Labels from WP Disambiguation pages
  - Anchor text from intra-WP links to that concept
- Additional mentions from NER which may be variations of known multi-word labels

Identify Mentions: How?

- For matching, normalize labels and text:
  - case, white-space, punctuation
  - encoding / representation of accented characters
- Millions of labels, even after normalization
- Often many candidate concepts/URIs per label
- Will need a lot of information for each candidate concept:
  - URI
  - Original case
  - concept type (Org, Pers, …)
  - Frequency statistics

  => Cannot directly use a Gazetteer for all of this

Identify Mentions: How?

- Use gazetteer to just identify mentions, no data
- Prepare a database that maps each mention text to all the information we need.
- Prepare as much in advance as possible so we do not need to spend time on it in the pipeline
  - Preparation only needed infrequently, may require a lot of computing resources
- Desired output: all the information for each mention text

Candidate Preparation

- Sources (numbers just for EN):
  - DBpedia labels: ~10M triples
  - YAGO labels: ~27M triples (including non-mappable)
  - DBPedia properties: ~26M triples
  - WP page links: ~172M links
  - DBpedia/Airpedia types: 1.6M/10M triples
- Make sure URIs are normalized:
  - proper %-encoding style (varies between DBP version)
  - use IRIs not URIs everywhere
- Make sure labels are normalized

Candidate Preparation: Normalization

Normalize and filter labels:

- exclude obvious cases (hundreds of chars, “List of …”, numbers only, …)
- normalize for case-insensitive matching
  => but remember original case!
- canonical representation of accents etc.
- multiple white-space, punctuation
- Extract parentheses info, e.g
  “Jean Lemaire (painter)”

Candidate Preparation: Merge

- Gather information per URI: class, frequency of related WP page link in WP articles, DBpedia properties …
- Gather information per label: original spelling, source (redirected page, disambiguation, canonical page, yago), frequencies, ….
- Gather information per label/URI pair: relative frequency of label used with this URI (WP page link), relative frequency of URI used with this label (“commonness”)
- Merge information and generate a de-normalized representation: key/value where the key is the label and the value is an array of rich URI-information, one element for each URI.

Candidate Preparation: Example

"yorkshire" →

- original_label="Yorkshire", all_labels=["Yorkshire","The Yorkshire Mafia"], uri="dbp:The_Yorkshire_Mafia", uriByLabel=0.0, sources=["dbp_name"],....
- original_label="Yorkshire", all_labels=["Yorkshire","Yorkshire, Ohio", "Yorkshire, OH"], uri="dbp:Yorkshire,_Ohio", sources=["dbp_labels"], uriByLabel=3.310E-3, airpClass=dbpo:PopulatedPlace, …
- original_label="Yorkshire",uri="dbp:Yorkshire", all_labels=["Yorkshire","Yorks","County of Yorkshire",...], uriByLabel=0.68,airpClass=dbpo:PopulatedPlace,parentheses=["UK","England"],...

Identify Mentions: How?

- Use ExtendedGazetteer PR to just match the labels (~9M cleaned labels, 158M on disk, ~900M memory)
- For each matched label, look up the de-normalized data from a key/value store (~21G on disk)
- Each candidate from the list is represented as an annotation, the fields as features in the FeatureMap
- In addition, store the list of annotation ids of each candidate annotation in the mention annotation.
- Subsequently, a lot of processing happens on entire candidate lists.

Reducing Candidates – Overall Strategy

- Initially over-annotate
- Remove obvious rubbish early on
- Deal with obvious cases early on
- Deal with overlaps early on
- Calculate scores for remaining candidates
- Pick best candidate or decide it must be OOKB

=> when to filter or when to just calculate a score?
=> how to decide based on scores and other features?

Reducing Candidates

- Reduce based on POS tags: require at least one proper noun (plus some special cases like country demonyms)
  - Problem: wrong POS tag will do unrecoverable harm, but POS tags will often be wrong in Tweets, Title-case text..
  - Currently working on including a recaser
- Reduce based on known patterns, e.g.
  - [Mention1], [Mention2] with Mention1 containing location candidates, Mention2 containing location candidates and some pairs of candidates related, e.g. “Yorkshire, Ohio”
  - [Mention1] ([Mention2]) with Mention1 a multi word term and Mention2 something that looks like an acronym, e.g. “Inter-Services Intelligence (ISI)”
- Reduce based on known overlap patterns, e.g. a mention of a person name within a longer mention of a person name
- ….

Choosing a Candidate

- Now we have a good quality candidate list
- Useful information such as different kinds of commonness information and the original labels is available
- We need to make a final decision about which is the best candidate

Semantic Relatedness: Have/Want

- Have:
  - lists of candidates for each detected mention
  - context words
  - other context info (e.g. Tweet metadata)
- Want:
  - determine semantic relatedness between each candidate and the context
  - determine semantic relatedness between candidate pairs of different mentions

Context for Tweets

- Tweets are very short, barely any context at all
  - Try to retrieve additional context from what we have
- Resolve hash-tags:
  - split multiword hash-tags
  - add hash-tag expansion candidates as text
- Resolve user screen names:
  - retrieve user profile information from Twitter and add as annotated text (location, description, language …)
- Resolve linked web pages:
  - retrieve content of web page and add as text
  - filter web page content to remove boilerplate/navigation

Disambiguation in LODIE

- **Scoring and feature creation**—applying metrics indicating the quality of the candidate
  - Contextual similarity
  - Structural similarity (relationships between candidates on nearby mentions)
  - Case match, class match ...
- **Selection**—choosing the best candidate based on scores and other information
  - Rule-based, intuitive approaches
  - Machine learning

The Challenge—Beat Commonness!

Measure	F1
Random choice of candidate (baseline)	0.39
Relative label frequency by URI in Wikipedia	0.65
Relative URI frequency by label in Wikipedia	**0.78**
URI frequency in Wikipedia	**0.77**
Label URI combination frequency in Wikipedia	**0.78**
Label frequency in Wikipedia	0.36

- It's hard to beat just picking the most common entity!
- However, if we can supplement that information with other valuable sources, we might just be able to nudge the score up a bit, even if the individual metric doesn't beat commonness

(All results presented are on a corpus of 200 manually annotated gold standard tweets.)

Contextual Similarity

- Each candidate for a mention is a DBpedia page describing that entity
- DBpedia page contains different kinds of information about the entity
- We can learn something about the likelihood of that candidate being correct by comparing the context in which the mention appears with the information on the DBpedia page
- How to do this? Vector space approaches

Contextual Similarity

- Word bag vectors are constructed from the context window of the mention and the content of the DBpedia page for each candidate
  - Stop words are removed
  - Words are lower-cased
- A semantic space is prepared in advance from an appropriate corpus—the semantic space records what words appear together in the same document
- Context and candidate vectors are mapped through the semantic space to expand them with related terms
- The resulting vectors are then compared for similarity

Contextual Similarity—Implementation

- We use the Airhead API to implement contextual similarity
- We have prepared a semantic space using Airhead, over a subset of abstracts in DBpedia
- We use TFIDF to improve the quality of the semantic space, by upweighting more meaningful words
- Airhead provides a variety of similarity metrics
  - Cosine, Euclidean, Jaccard index, etc. ..

Contextual Similarity

- We have implemented a PR for the project that takes the following parameters:
  - Context window length (chars)--this rounds down to the nearest whole word
  - Similarity function
  - Init-time parameter for which database to connect to and which table, to access the content of the DBpedia page
- We use three separate PRs to implement contextual similarity for different types of information about the candidate
  - Abstracts
  - All textual content for the entity
  - All textual content including the results of following links to related entities

Contextual Similarity

- Results
  - The three different information types (abstracts, all textual data and all textual data including indirect) seem to give similar results
    - Probably no need to include all three!
  - Overall result is much lower than commonness but significantly higher than choosing a random candidate
  - This information differs from commonness and so it may be combined with commonness to achieve a score higher than is possible using commonness alone

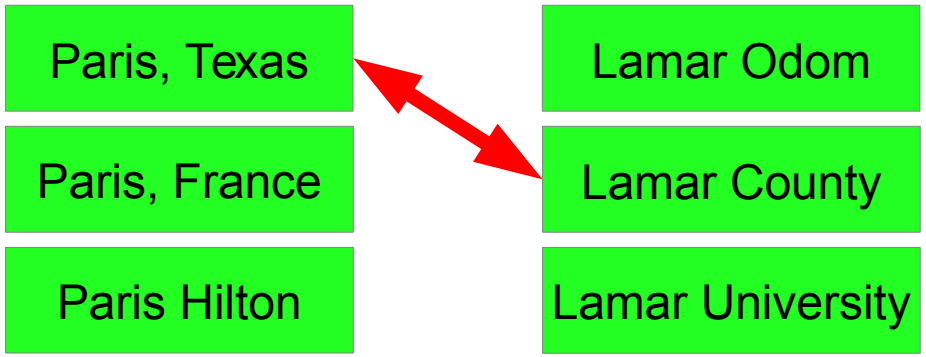
Contextual Similarity—Further Research

- Other ideas within contextual similarity include:
  - Process the semantic space with random indexing
    - This gave a worse result, though the semantic space was much smaller and the resulting PR was faster
  - Process the semantic space with LSA
    - Still to try—implementation issues to resolve!
  - Different similarity metrics
    - Cosine seems to give the best result of the ones tried, but systematic experimentation would be a good idea!
  - **Different representative texts**
    - Comparing context of the mention with the context in which the entity normally appears might work better than the abstract for that entity

Structural Similarity

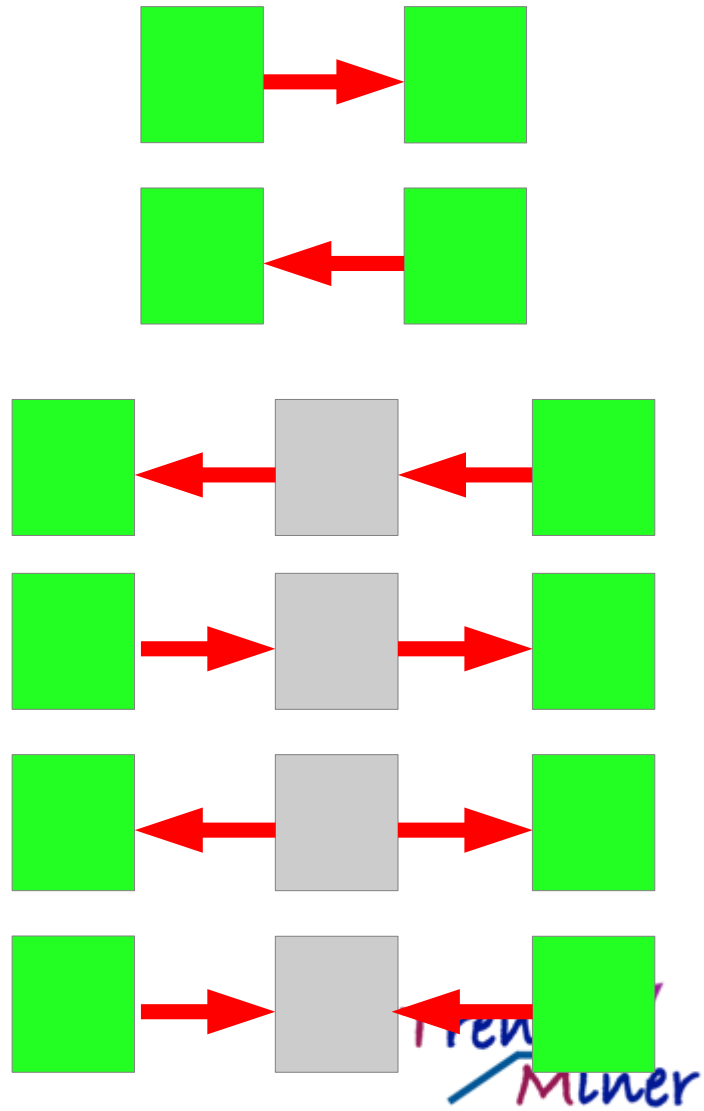
- Candidates on adjacent mentions are more likely to be correct if they are known to be related to each other

... Paris, and Lamar schools ...



Structural Similarity

- Relation types
  - Direct connection—inbound
  - Direct connection—outbound
  - Indirect:
    - unidirectional—both inbound
    - unidirectional—both outbound
    - shared parent
    - shared child



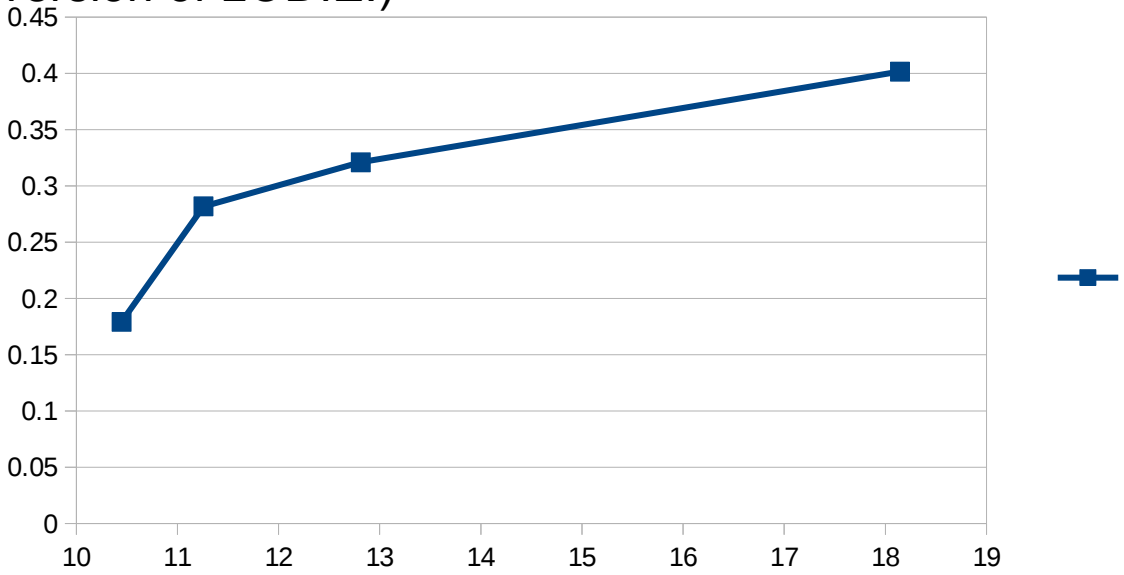
Structural Similarity—Some Results

Link Type	F1
Random	0.39
Unidirectional direct link	0.41
Bidirectional direct link	0.43
Shared parent	0.48
Shared child	0.42
Unidirectional indirect link	0.41
Bidirectional indirect link	0.42
Combined	0.49

Structural Similarity—Slow!

Relation Type	Time (millis)	Log Time	F1
Shared parent	75858358	18.1443784489	0.4015605406
Indirect	366706	12.8123157161	0.3209839508
Shared child	77442	11.2572845481	0.2817679558
Direct	34431	10.4467126003	0.1790426908

(Results shown are on TAC 2010 with nils excluded, no backoff to random candidate and on an earlier version of LODIE.)



Structural Similarity—Practicality

- Capping inbound links
  - Shared parent is slow because some popular entities have very many inbound links (the US has ~200,000!)
  - We can speed up the PR by not calculating a result for an entity with more than X inbound links—but there is a cost

Max inbound links (shared parent)	F1
1000,000	0.45
10,000	0.40
1000	0.32

(TAC 2010 data, key spans only, backoff to random candidate.)

Structural Similarity—Contribution

- Hopefully structural similarity provides a different type of information than commonness, and has the potential to improve overall performance in combination with it
- However, it is possible that structural similarity simply provides an alternative measure of commonness!
  - Perhaps shared parent is such a successful relation because very common concepts tend to score higher
- Two entities having a shared relation is intuitively a highly valuable source of information, but DBpedia is sparse in providing these relations
- Future work could involve supplementing the relations database
- We can test to see if shared parent contributes anything useful in addition to commonness—if not we can exclude this very slow calculation

Coreference in Scoring

- Coreference has been implemented in LODIE using the ANNIE orthographic coreferencer
- If two mentions are identified as being coreferences, then scoring metrics are improved by using the longest, most detailed mention only for scoring.
  - This mention will have the shortest, best quality candidate list.
  - Time is saved by not scoring poorer quality mentions.
- Coreference has been used in both PRs presented
- Coreference normally improves the result for that scoring metric by a few percent, and makes it faster too.

Further Features

- Class match
  - ANNIE is used to determine what class of entity we expect at that point in the text
  - We can then calculate, for each candidate, whether that candidate would be a class match to ANNIE
- String and POS features for this and adjacent tokens
- Commonness metrics
  - As discussed earlier, these are calculated in advance for all entities in DBpedia
- Case match
  - Context case is determined (sentence/title case, all lower, all upper) to decide whether the case of the mention is useful information
  - In the case that case appears to be well-formed and valid, candidates that aren't a case match are penalized
  - Future work will involve case-correcting the input text

Choosing a Candidate

- This can be presented as a machine learning classification problem
  - Each candidate becomes an instance
  - Scores etc. become features for the machine learning
  - Class is true or false for whether it is the right answer
  - ML task is to decide if a candidate is right or not based on features
  - This may mean that we have multiple trues, or none, in which case we can use the score assigned by the machine learner (e.g. probability) to select the best

Problems with ML for Disambiguation

- Too many negative candidates creates an extremely conservative learner
  - This problem was improved by reducing the candidate list to the five most common—we can do this with relatively minor loss of recall
- Probability may not be a probability—is this number really the best way to select the best candidate?
- It can take a long time to train some algorithms, and it isn't always practical to try everything we want to

More on Candidate Selection

- Would like to try more algorithms
  - Reliance on the probability for choosing between multiple “true” candidates means PAUM isn't suitable. However it would be good to find a solution to this because PAUM is fast
  - Decision trees show promise but take a long time to train. Perhaps it's worth it though ..
- Rule-based approaches
  - Simple additive combinations of metrics, including commonness or with commonness as a backoff only, has failed to improve on commonness alone

Acknowledgements

Development of the LODIE Entity Linking system is partially supported by the European Union under the Information and Communication Technologies (ICT) theme of the 7th Framework Programme for R&D (FP7), grant TrendMiner (287863)