Ryan Amaral
ryan.amaral@dal.ca

Entity Recognition and Entity Linking with Wikipedia

Introduction

In this project we explore the domain of Entity Recognition (ER) and Entity Linking (EL) backed by Wikipedia, which makes the EL task also known as wikification. We create a model which can do either task individually or both tasks together, using a collection of datasets, and backed by a Wikipedia graph structure [1].

ER is a classification task, where given a token in a piece of text, we must decide whether the token is an entity or not. A token being an entity means that the token refers to something.

EL is to take an entity and find what it actually refers to, based on some knowledge base, Wikipedia in our case. EL can be treated as a classification task, where given multiple candidates to link an entity to, we choose the one with the highest confidence (or even treat it as a regression task with highest score). The problem with this is that it is not leveraging the relativity between the candidates. To make use of this relativity, we use a Learning to Rank algorithm [6].

There are a few ways to do EL. One of the most effective but simple is **popularity**, which is to take the document that has the most incoming links. Another is **context**, which is to use words surrounding the entity currently trying to be linked, and use those words to search in the documents that are candidates to find the best match. And finally (of the ones we are using) is **coherence** [9], which takes the candidates of all entities in some chunk of text, and finds the combination of those which makes the most sense.

Data

For training the ER and EL models, we use two internally created datasets, which are generated with the link structure of Wikipedia [7,8].

For ER model training we use a dataset called *er-10000-cls-neg.txt*, and *er-10000-cls-pos.txt*, which are positive and negative examples of entities that we mix together when used. Table 1 describes the data.

Table 1: Description of each record of the er-10000-cls-*.txt dataset. Each record represents attributes of a token.

Column	**Description**
posBef	(Categorical) Part of speech tag before the token.
posOn	(Categorical) Part of speech tag of the toxen.
posAft	(Categorical) Part of speech tag after the token.

mentionProb	(Float) Probability that the text of the token is a link in Wikipedia.
cap	(Indicator) Whether the token is capitalized.
match	(Indicator) Whether there is an exact match of token text with a title in Wikipedia.
space	(Indicator) Whether the token contains a space.
ascii	(Indicator) Whether the token is ascii only.
isMention	(Indicator) Whether the token is a actually a mention/entity. The label.

For EL model training we used another internally created dataset, called *el-5000.txt.* Table 2 describes the data.

Table 2: Description of each record of the el-5000.txt dataset. Each record represents attributes of a candidate for the link of an entity.

Column	**Description**
trueEntity	(Indicator) Whether the candidate is the true link for the entity.
popScore	(Float) Popularity score.
ctx1Score	(Float) Context method 1 score.
ctx2Score	(Float) Context method 2 score.
w2vScore	(Float) Word2Vec [16] score.
cohScore	(Float) Coherence score.
mentionId	(Int) The ID of the mention/entity that this candidate is for. Group by this.

The datasets used for evaluating performance on the trained models to compare with other state of the art models are KORE [2], Aquaint [3], MSNBC [4], Wiki5000 [5], and Nopop, we refer to these datasets as the performance datasets. KORE uses examples that try to be tricky for the task of EL (explained further in Data section). Aquaint and MSNBC are news examples which are large, relative to the other datasets. Wiki5000 is comprised of the opening paragraphs of 5000 random Wikipedia pages. Nopop contains only records that no entity will have the most popular candidate as the correct one. The following is a sample from KORE.

{**"text"**: ["Müller", "scored", "a", "hattrick", "against", "England", "."],
"mentions": [[0, "Thomas_Müller_(footballer)"], [5, "England_national_football_team"]]}

We analyzed the level of similarity between the entity token text, and the text of the matching Wikipedia page for each of these datasets (except nopop) in figure 1. Part of KORE's difficulty could be attributed to the high amount of low similarities. The other datasets have relatively higher levels of similarity.

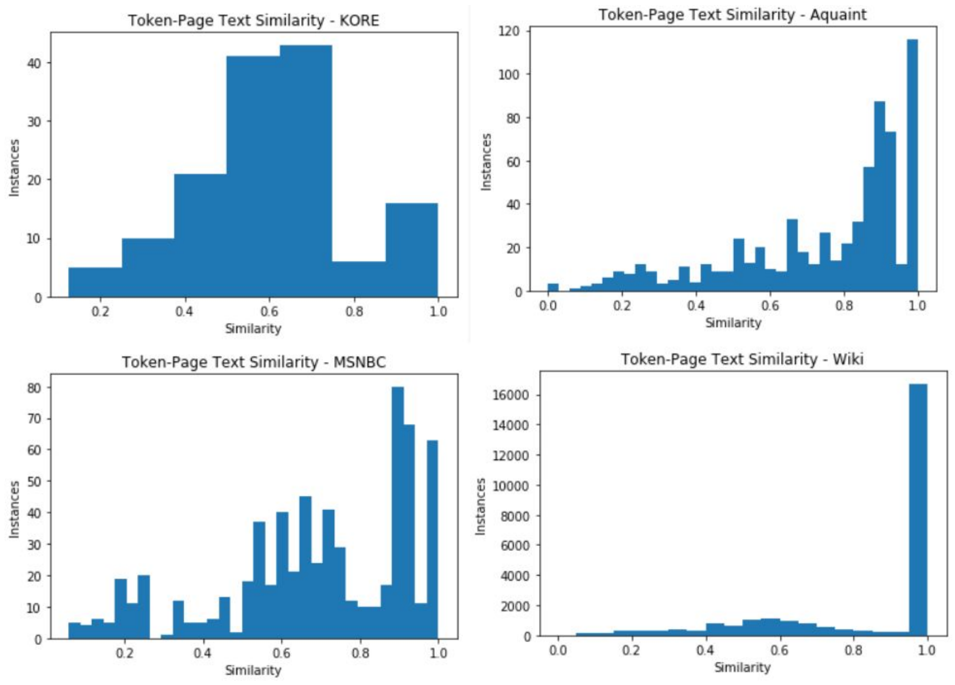


Figure 1: Text similarity between entity tokens, and the titles of the Wikipedia pages they link to.

Hypothesis/Purpose

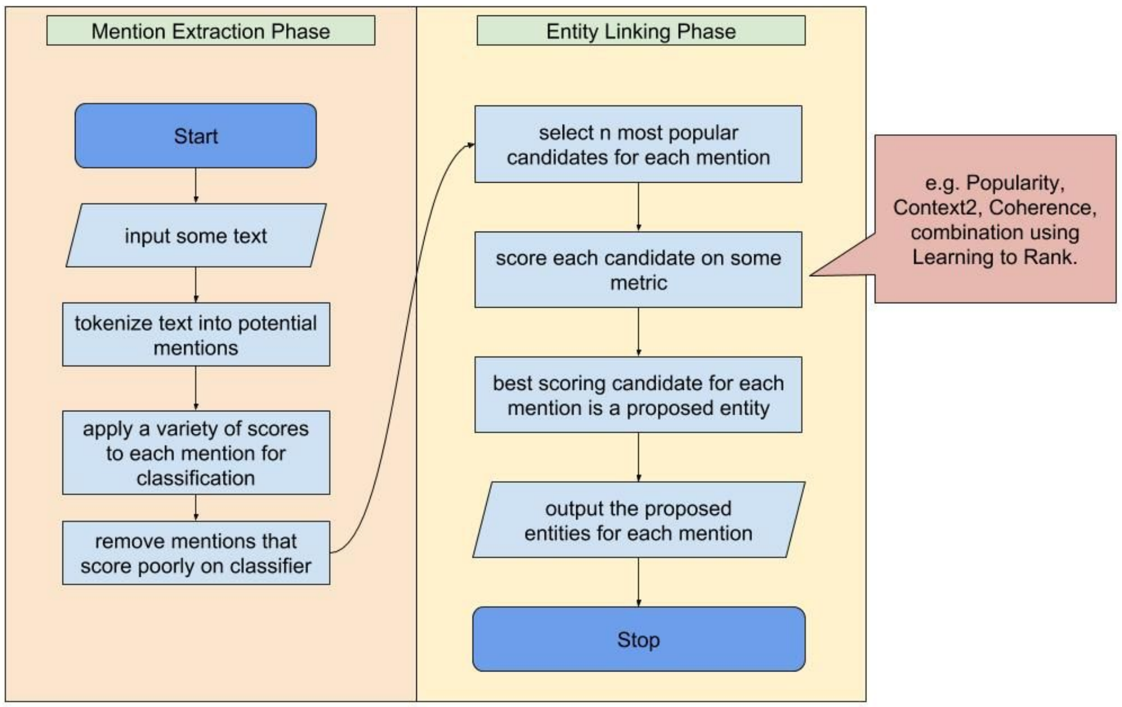
Our main hypotheses / goals are less theoretical and more competitive, because we are doing tasks that are known to work, and already have benchmarks. Our goals were mainly to be competitive with state of the art approaches in ER and EL, and to do better than popularity score on EL. One more theoretical hypothesis is that the part of speech tags of the token before and after the token in question for ER, are important for classification.

Early in our experimenting we found that popularity score is very effective for such a naive method, on some more straight forward datasets. So we want to make sure that we build for robustness by using but not strongly relying on popularity score.

ER and EL are very useful for at-least a few things. One of them is for document clustering that can go beyond pure text analysis, to get a little bit of understanding of the documents [10]. Another use is for searching, where you can find results based on the meaning of the query, more so than just the exact words used. And more simplistically, you can use these techniques to automatically generate links on documents for human readers, just like we can see on Wikipedia.

Our Model

Our model, called Wikisim [1] (sometimes referred to as multi because it makes use of combining multiple methods) does the full workflow, from raw text to a collection of linked entities. The full process is described in figure 2.



Mention Extraction Phase
Entity Linking Phase
Start
input some text
tokenize text into potential mentions
apply a variety of scores to each mention for classification
remove mentions that score poorly on classifier
select n most popular candidates for each mention
score each candidate on some metric
e.g. Popularity, Context2, Coherence, combination using Learning to Rank.
best scoring candidate for each mention is a proposed entity
output the proposed entities for each mention
Stop

Figure 2: Flowchart that shows the full flow of data through our model.

The first phase from the raw text is tokenization. We handle this by using Solr Text Tagger [11], and have it return all overlaps. An example of this is if we are given the text "*She is the Queen of England.*", the tokenizer will return to us "*She*", "*is*", "*the Queen*", "*the Queen of England*", "*Queen*", "*Queen of England*", "*England*", "*.*". Where each quoted section is a token. We keep all of these overlaps in place until the ER phase.

For ER, for each token, we get all of the attributes as described in figure 2. Then feed them into our trained ER model.

For finding our ER model, we tried AdaBoostClassifier, BaggingClassifier, GradientBoostingClassifier, and RandomForestClassifier all from scikit-learn [12]. We trained each of these models with grid search on the parameter *n_estimators* at 100, 200, and 300, with 70% of our dataset. We then performed cross validation with the remaining 30% of the dataset on five folds. The results are summarized in figure 3, where we can see that gradient boosting performed the best. It is suspicious however, how well all of the models did. From our results, we can also conclude our hypothesis of the importance of part of speech tags before and after the token.

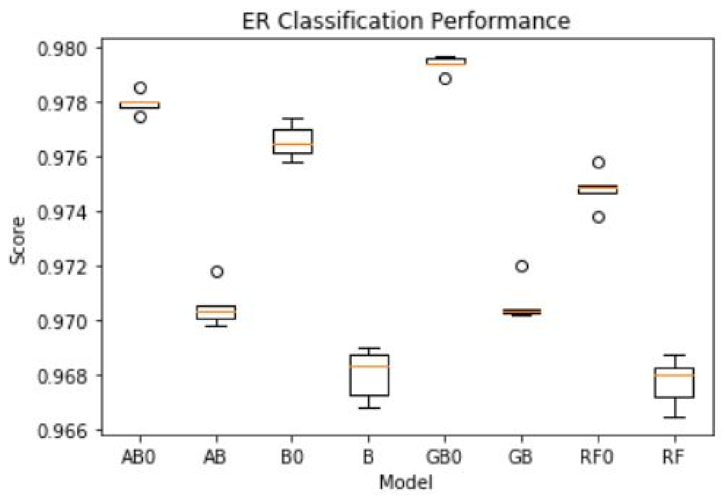


Figure 3: Performance results from cross validation of our tried ER models. AB = AdaBoost, B = Bagging, GB = Gradient Boosting, RF = Random Forest. Only those suffixed with 0, are trained with the features posBef and posAft included.

Once the ER model determines all of the tokens, we then remove the overlaps in the tokens, based on what are the entities and probabilities that the token is a link on Wikipedia.

Now we generate candidates for each of the entities. Our primary method of generating candidates is to use popularity score to generate the top 20 candidates. We also tried generating half of the candidates this way, and half by context score. We called this method hybrid, which gives us better theoretical results, as shown in figure 4. However in the final model this did not give us better results, possibly due to giving extra bias to our context methods.

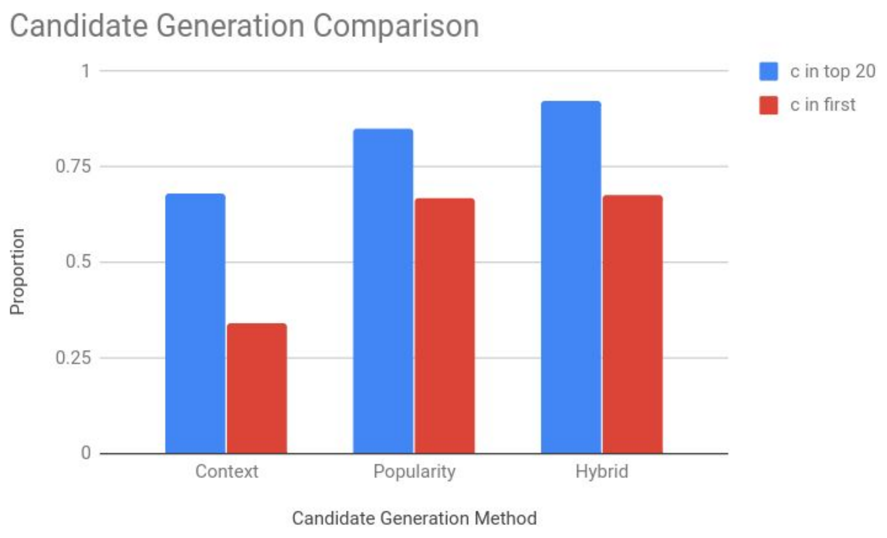


Figure 4: Comparison of candidate generation methods

Despite the hybrid method not giving better actual results, using a similar technique actually did give us better results. Which was to try to generate all 20 candidates by popularity, and only if all 20 weren't filled, we would fill the remaining by context. A major reason this helps us because if there is a bad misspelling in an entity, the popularity method may not pick up any candidates, but context can.

We now score each of the candidates with our EL model with the features described in table 2. We will now explain the two different context scores. The first is based on how well the context of the entity is found specifically in the candidates page. The second method is to look for all instances, of any page, of the a string similar to the context, and finds the page that is linked to most in this context.

The goal of the EL model is to choose the best candidate for each of entities.

Our final EL model is a Learning to Rank algorithm called LambdaMART [15], implemented in a python module called pyltr [14]. We measured performance on a metric called Normalized Discounted Cumulative Gain (NDCG), which measures how well rankings are made. Our model performed better without including the Word2Vec score as shown in table 3.

Table 3: Learning to Rank model performance with and without Word2Vec. Evaluated on the EL model test set.

Model	**NDCG Score**
With Word2Vec	0.94355
Without Word2Vec	0.96004

Results

We test different parts of our model on the performance datasets. One test is just ER, another is just EL, and finally, ER and EL together. The metrics used are precision, recall, and f1, because that is what the benchmarks use.

For ER we tested our model against two other freely available models, CoreNLP [17] and TAGME[18]. The results are shown in figure 5. It can be seen that we are quite close to the other models overall.

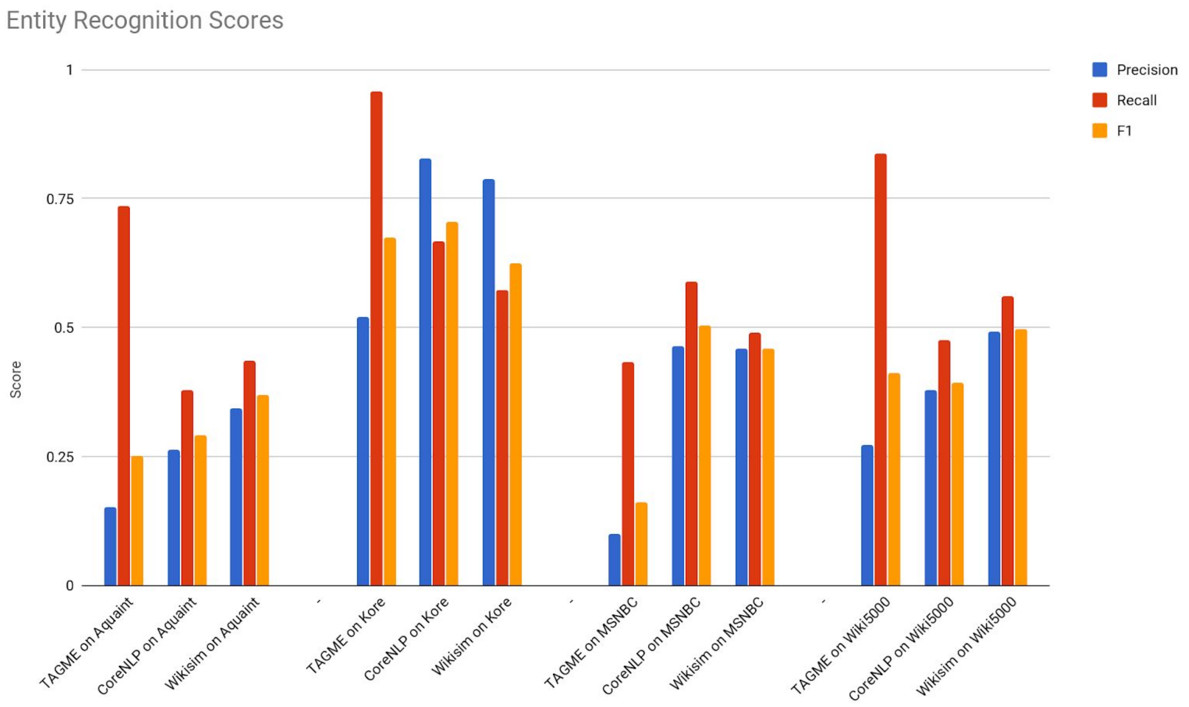


Figure 5: Comparison of TAGME, CoreNLP, and Wikisim (us) on ER.

For ER and EL together we compare ourselves to just TAGME (CoreNLP is ER only). We also show the results of each of our individual basic methods. The results are seen in figure 6. Once again, like in ER we are competitive with the other model. And it is good news that we are beating out the popularity score.

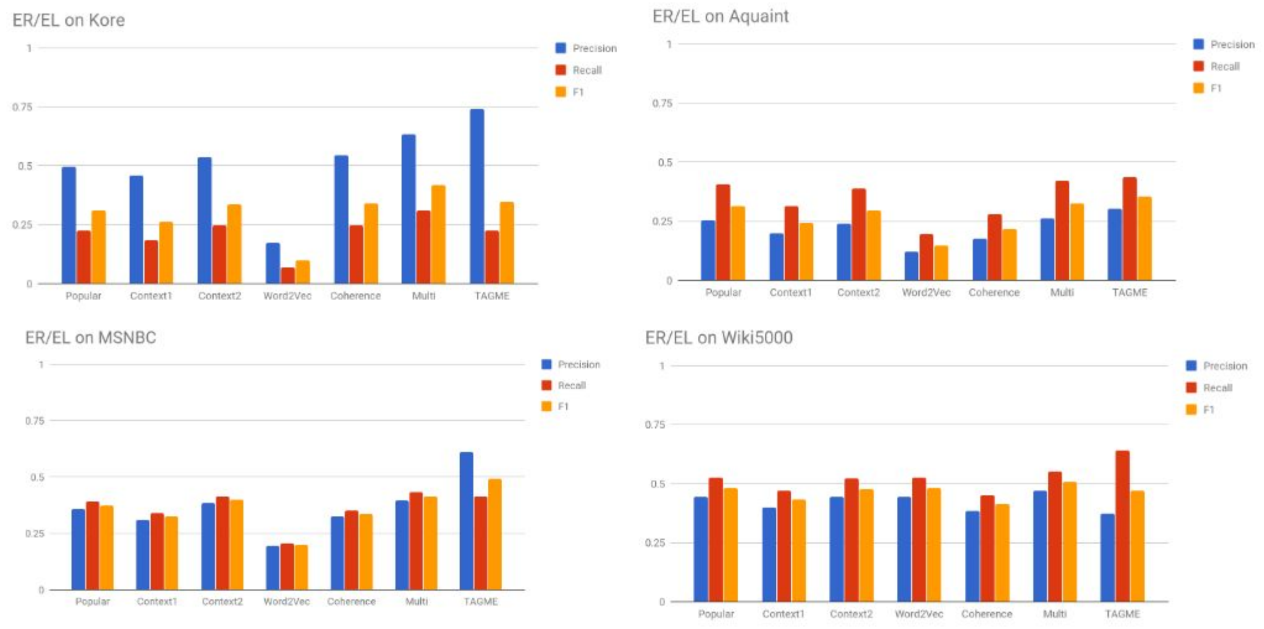


Figure 6: Performances of each individual method, our final model (Multi) and TAGME on the combined ER/EL task.

We also show the result of combined ER/EL on our Nopop dataset in figure 7. This one is interesting because it shows us that coherence, which does not make use of popularity does the best here, and the advanced models (Multi, and TAGME) are brought down a level due to their usage of popularity score. As a side note, popularity has a small amount of score because some of the entities may not have had any popularly picked candidates, so they have contextually generated candidates only, which can give us some result for popularity, also we used the truth set of tokens to calculate Nopop, so a token may be slightly different and more helpful.

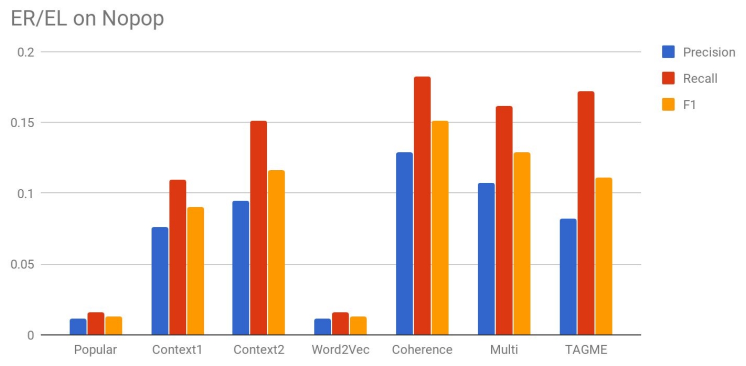


Figure 7: Performances of each individual method, our final model (Multi) and TAGME on the combined ER/EL task on the Nopop dataset.

We could not test just EL on TAGME because tagme does not support the input of entities. So we use an online benchmark framework called Gerbil [19] which tests models backed by DBpedia [20]. We were unable to test our model on this framework because we used Wikipedia instead of DBpedia, but we believe the scores should be fair to compare, since DBpedia should have the same pages as Wikipedia. The results are shown in figure 8. Our model is comparatively weak on KORE but performs roughly on point with top models on the other datasets.

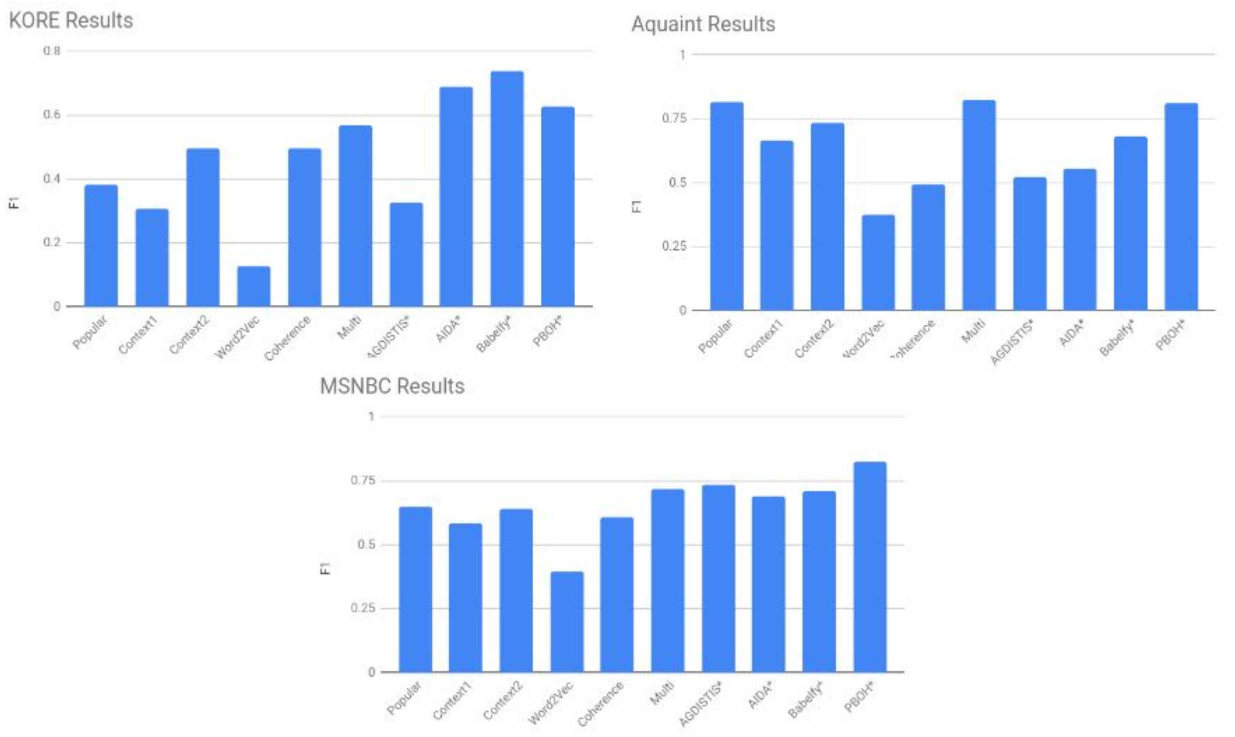


Figure 8: Comparing models on some common datasets. AGDISTIS, AIDA, Babelfy, and PBOH come from Gerbil. Multi is our model.

Our other two datasets, Wiki5000 and Nopop, aren't on Gerbil, so we test them only with our final model, and the individual metrics. Our model performs better than the individual methods all over, as shown in figure 9.

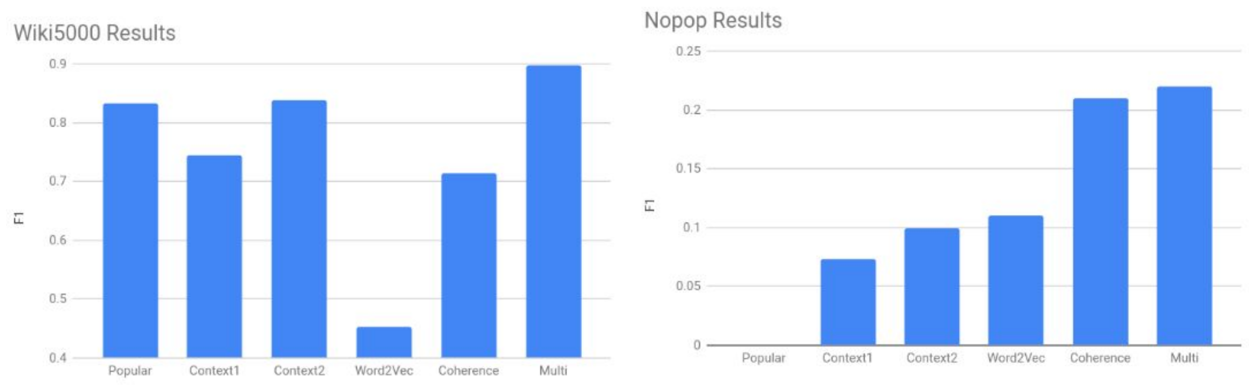


Figure 9: Comparing our individual measures and final model on two datasets (datasets not on Gerbil).

Conclusion

We created an ER/EL system which performs competitively with state of the art models, and which performs better than the naive metric of popularity. The ER model uses a gradient boosting classifier, and the EL model uses LambdaMART, a Learning to Rank algorithm.

This research was limited by the fact that I no longer had access to the server and necessary services on it to perform the full model workflow. I was only able to further analyze the datasets, and train/test new models for ER and EL on the training files, but not actually run them in a live Wikification scenario. But I still got some good new analysis of the ER and EL model training.

In the future, something we can do is jointly perform ER and EL, this can be referred to as JNERD [21], and has been shown to provide better results. An interesting problem we faced is that of stop-words, phrases like “is a” and “the” would show up as entities, because they have corresponding pages on Wikipedia, but the majority of the time, you don’t want this. And finally, it would be nice to change the underlying structure of this project to run on DBpedia instead of Wikipedia, so that we can get our model on Gerbil to be directly compared to other state of the art models.

References

[1] https://github.com/asajadi/wikisim
[2]
https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/aida/downloads/
[3] https://catalog.ldc.upenn.edu/LDC2002T31
[4] http://cogcomp.org/page/resource_view/4
[5] https://github.com/asajadi/wikisim/blob/master/datasets/ner/wiki-mentions.5000.json
[6] https://en.wikipedia.org/wiki/Learning_to_rank
[7] https://github.com/Ryan-Amaral/wikisim/tree/master/wikification/learning-data
[8] https://github.com/Ryan-Amaral/wikisim/blob/master/wikification/ml-model.ipynb
[9] https://link.springer.com/chapter/10.1007/978-3-319-59569-6_48
[10] https://arxiv.org/pdf/1807.07777.pdf
[11] https://github.com/OpenSextant/SolrTextTagger
[12] http://scikit-learn.org/stable/
[13] https://github.com/jma127/pyltr/blob/master/pyltr/metrics/dcg.py
[14] https://github.com/jma127/pyltr
[15]
https://www.microsoft.com/en-us/research/publication/from-ranknet-to-lambdarank-to-lambdamart-an-overview/?from=http%3A%2F%2Fresearch.microsoft.com%2Fpubs%2F132652%2Fmsr-tr-2010-82.pdf
[16] https://en.wikipedia.org/wiki/Word2vec
[17] https://stanfordnlp.github.io/CoreNLP/
[18] https://tagme.d4science.org/tagme/
[19] http://aksw.org/Projects/GERBIL.html
[20] https://wiki.dbpedia.org/
[21] http://aclweb.org/anthology/D15-1104