Project report

Natural language processing
Entity linking system

Adrian Spurr, Dejan Mircic, Lennart Van der Goten
{spurra, mircicd, lennartv}@ethz.ch

ABSTRACT
In this project report, we will describe our approach to solve the problem of entity linking. We give details on what has been attempted, the issues we have faced and the results obtained. We start with stating the problem statement, and continue on explaining our approaches to solve this. The next section describes the results obtained and in the final section we conclude our project.

Problem statement

There are two related problems we aim to solve. The first is the C2W task, where given a query q, the goal is to find the most probably entities referred to in q. The second, more challenging task is the A2W problem, where given a query $q = t_1t_2 \dots t_n$, as a sequence of n terms, the task is to detect non-overlapping mentions $m_i = t_{l_i}t_{l_i+1} \dots t_{k_i}$ and *annotate* them with the entity e they most likely refer to. In other words, our aim is to construct an algorithm that finds all most probable mention-entity pairs (m_i, e) in a given query. In our case, the entities are given as a Wikipedia page that describes unambiguously the underlying concept of said entity.
The main difficulty occurring is that queries can be intrinsically ambiguous, so that even humans may have issues in understand what is meant. To alleviate this, the data in the training set is annotated by multiple different people, so that the meaning of a query is defined by whatever the majority decides on.

Approaches

We decided to approach the problem by testing two approaches: A non-learning approach, inspired by baseline-1 performance and a learning approach, based on the algorithm as described in Cornolti et al. [1]. As will be described in the results section, we find that the SVM annotator outperforms the non-learning annotator (NLA)

Non-learning annotator

The baseline-1 algorithm is a simple greedy entity-linker to measure our actual algorithm against. It starts at the beginning of the query, greedily selects the longest mention and annotates it by assigning it the most probable entity. We were surprised at the performance we got and decided to implement a similar non-learning approach. First we produce the necessary candidate entities in a similar fashion as in [1]. We draw candidates from multiple sources for a given query q. The first source $\mathcal{E}_1$ is the set of wikipedia pages occurring in a bing search for q, the second $\mathcal{E}_2$ is the set appearing given the concatenation of q and the string $wikipedia$, whereas the third $\mathcal{E}_3$ consists of entities given by annotating the top 25 snippets using TagMe. We introduce a fourth set, $\mathcal{E}_4$, which consist of the wikipedia pages found by searching for each of the word n-gram of q, where n is a hyperparameter. With the additional set, we aim to increase the recall value of our approach. For the rest of the report, let $S_1 = \{\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3\}$ and $S_2 = \{\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3, \mathcal{E}_4\}$. All candidate entities are collected in a set E (i.e $E = S_1$ or $E = S_2$). The actual annotator works in the following way:

1. Build a set M with all possible mentions for a query q.
2. Form all possible pairs (m, e) with all the entities of E and all the mentions of M
3. Score each pair using a suitable scoring measure, measuring how adequate m refers to e and add to a list P.
4. Sort P.
5. Iteratively go through P, and add the highest scoring pairs which do not overlap with others picked. Add all to a set $\hat{P}$.
6. Return $\hat{P}$

The non-learning annotator greedily picks the best mention-entity pair according to a score value. We used the value of *anchorsAvgED* defined in [1], as the weighted average Levenshtein distance between a mention m and all anchor links that refer to e in Wikipedia. In the paper, they emphasise the importance of the feature. An alternative to *anchorsAvgED* is using a $word2vec$ model for computing the cosine similarity between m and e, using it as score. The performance for both are presented in the "Results" section.

SVM annotator

During initial experimentation of the previous approach, we tend to reach a higher recall and lower precision. Therefore we decided to add an SVM to shorten the number of entities, so that we achieve a higher precision while maintaining the higher recall. This approach is partly based on [1]. Given a query q and an entity e, the SVM classifies how applicable e is to q, which is the C2W problem. To this end, we extract the same features as in [1] which are applicable to e and q, with the exception of the rank feature, as the Bing queries returned unreliable results for it. With the help of the SVM, we get a probability estimate for each e, given q. We threshold the set of entities by a value t_1 and only keep those that are above a certain value in a set E. Then, we perform the actual annotation in the same way as in the NLA. During initial testing we realised that we had a high number of false positives. We therefore introduced a second threshold and modified point 5 to the following:

5. Iteratively go through P, and add the highest scoring pairs which do not overlap with others picked **and whose score are above a threshold** t_2. Add all to a set $\hat{P}$.

Intuitively, the threshold t_2 controls the trade-off between having good recall or precision, and needs to be optimised with respect to the F1 measure.

Implementation details To implement the SVM, we used libsvm [2]. We use an RBF kernel and adjusted the library to use the F1 score. We perform the training on both S_1 and S_2 to compare if our addition leads to any benefit. To prevent performing unnecessary work multiple times, we calculate the features of the training data once, normalise and cache them. The training data is labelled by checking if an annotation occurs in the gold standard or not. This leads to a very unbalanced training set, where the number of positives are outnumbered by the number of negatives. Therefore, we weigh the positives more heavier than the negatives. To find the optimal values of C, the weight of the slack variables and γ of the RBF kernel, we perform a 10-fold cross validation, doing a grid search over both parameters. To obtain appropriate values for t_1 and t_2, we undertook an additional grid search.

Results

For both annotators, we used the training data contained in the GERDAQ portions train-A, train-B and initially tested on the devel portion to generate candidate entities as mentioned above, for both the non-learning and SVM annotator, and using it to train the SVM. To perform the scoring of the entity-mention pairs with $word2vec$, we use a pretrained model built from the google news [3]. The precision, recall and macro-F1 score of both approaches, including our baseline-1 implementation can be seen in table . We see that for the NLA, we obtain slightly better F1 scores using *anchorsAvgED* scoring for the A2W task, whereas the $word2vec$ cosine distance is roughly similar on the C2W task. Using word 5-grams, we slightly increase the F1 score. For the SVM annotator, the $word2vec$ cosine distance is clearly superior to *anchorsAvgED* scoring. Additionally, the SVM annotator outperforms the NLA. Out of this reason, we decided to continue testing using only the SVM annotator, discarding the NLA. We trained on the GERDAQ portions train-A, train-B and devel, and tested on the test portion. The resulting macro and micro scores can be seen in table **??**.

Conclusion

We slightly outperform baseline-1 with the SVM annotator, and clearly outperform the NLA. Naturally, attempting to learn from data will tend to outperform a heuristic and/or greedy algorithm. We believe that we can increase the performance of our SVM annotator by including features that take the mention into account so that the final annotation procedure can benefit from learning as well. During our experiments, we noted that tuning many parameters, such as both thresholds of the SVM lead to a tradeoff between precision and recall, and we decided to find the optimal with respect to the F1-score.

Model	**Strict (A2W)**			**Relaxed (C2W)**		
	P	**R**	**F**	**P**	**R**	**F**
Baseline	0.46	0.514	0.418	0.502	0.565	**0.464**
NLA + anchorsAvgED + S_1	0.444	0.419	0.328	0.463	0.444	0.347
NLA + anchorsAvgED + S_2	0.349	0.451	0.33	0.368	0.484	0.353
NLA + *w2v* + S_1	0.341	0.441	0.322	0.361	0.477	0.348
SVM annotator + *w2v* + S_1	0.608	0.428	**0.424**	0.644	0.467	0.459
SVM annotator + *w2v* + S_2	0.599	0.383	0.391	0.626	0.416	0.419
SVM annotator + anchorsAvgED + S_1	0.766	0.343	0.349	0.773	0.349	0.355
SVM annotator + anchorsAvgED + S_2	0.666	0.305	0.314	0.667	0.306	0.315

Table 1: macro F1 scores on the development set

	Macro						**Micro**					
Model	**Strict (A2W)**			**Relaxed (C2W)**			**Strict (A2W)**			**Relaxed (C2W)**		
	P	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
SVM annotator + *w2v* + $t_2 = 0$	0.479	0.474	**0.431**	0.516	0.522	**0.471**	0.355	0.462	0.402	0.400	0.518	0.452
SVM annotator + *w2v* + $t_2 = 0.2$	0.597	0.413	0.417	0.635	0.449	0.452	0.447	0.374	**0.407**	0.500	0.418	**0.455**

Table 2: Macro and Micro F1 scores on the test portion of the GERDAQ set

REFERENCES

1. Marco Cornolti, Paolo Ferragina, Massimiliano Ciaramita, Stefan Rd, Hinrich Schtze, *A Piggyback System for Joint Entity Mention Detection and Linking in Web Queries*

2. Chih-Chung Chang and Chih-Jen Lin, *LIBSVM : a library for support vector machines*. ACM Transactions on Intelligent Systems and Technology, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm

3. https://s3.amazonaws.com/dl4j-distribution/GoogleNews-vectors-negative300.bin.gz