NLP HW2 report

Jary Pomponi

I have chosen a knowledge based approach using graphs to accomplish this homework, the novelty approach is in the graph construction. The graph, like ones used in literature, $G = (V, E)$ is an undirected graph where V are synsets and E are the set of all the edges. An edge connects two vertexes whether there is a semantic connection between them. I have used NetworkX library to build and manage the graph. Usually in literature the graph is built from a train set in unsupervised way: for each ambiguous lemma all of the possible synsets are added to the graph. My approach is a supervised one: for each ambiguous lemma I have added only the correct synset and all the ones that are connected to that one. I have done this by means BabelNet, used to retrieve all the synsets associated to training lemmas and synsets connected to those lemmas. I will call this graph *Gsup*.

I have done experiments with two different kind of graph: i'll call the first one *Gsup* and the second one $Gsup_w$. In *Gsup* all the edges had same weights whilst in $Gsup_w$ I have used a co-occurrence matrix to assign edges' weight in order to encode strongly the connection between lemmas in the train set. To build the matrix I have used a context window on documents. After some preliminary experiments I have chosen a context size of ten words. We will see that both approaches overcome given baseline.

For each graph I have used the same prediction approaches, all based on calculating pagerank vector and using it to predict most the likely synset for each ambiguous lemma.[1] Before using any technique a subset of lemmas from the test set has to be added to the graph. To do that I retrieve from BabelNet all possible synsets concerning each lemma in the subset with the help of pos tag associated to that lemma. To avoid noisy data I get only synsets from WordNet. Then those synsets are added to the graph but semantic relationship that involves them will become edges only if the target vertex is alredy present in the graph. Choosing subset is the main difference between the prediction approaches and will be explained later. In order to predict the most likely synset the following approach is used in each algorithm: pagerank vector of the graph is computed and then between each possible synsets, of a given lemma, the most probable one is picked. Usually the initial probabilities are placed on the synset generated from lemmas in the subset; the only exception is the static algorithm where all nodes have the same initial probabilities.

The first prediction approach uses two different methods to calculate pagerank. In the first method, static prediction S_p, the entire test set is added to the graph and then pagerank vector is computed and used to disambiguate lemmas. In the second one, mass prediction M_p, the probability are placed on synsets extracted from test set, so the probability are zero for the other nodes. The second approach, document prediction D_p, is similar to mass prediction but instead of adding all the train set to the graph I add only current document and disambiguate all lemmas appearing in that document. In the third one, path prediction $SubPath_P$, I add all the document, exactly as in D_p, but then, for each synset associated to a lemma all the possible paths not exceeding a given length starting from the current synset are calculated using Dijkstra algorithm. Then all the synsets appearing in those paths are used to extract a subgraph from the current one. After that pagerank vector is computed on that subgraph followed by the prediction step. This approach is slow and I have implemented a cache method in order to speed up computation and avoid to calculate paths from a synset many times. Some experiments have shown that the best upper limit length is 2. This value highly depends on the graph's size, since using higher values with this dataset will results in a number of nodes in the subgraph close to the nodes in original graph, decreasing the score obtained.

Table 1 shows the results of those approaches applied on the two types of graph. We can notice that the method used to build *Gsup* leads to good results using even though it does not consider the contexts with a co-occurrence matrix. In fact even the worst prediction method on *Gsup* leads to results that overcome the baseline except for senseval3 dataset which still is very close to the baseline. I want also to underline that the worst prediction method used on $Gsup_w$ not only overcome the best one on *Gsup* but also the baseline on each dataset. This is mainly because in the graph it is encoded also how lemmas appear in the documents thanks to co-occurrence matrix. Talking about the prediction methods we can say that those based on documents has to be preferred because lead to better results. This is due the fact that adding only a document per time avoids to add useless or noisy informations to the graph. Path prediction is the best one because exploits to at maximum this concept considering only synsets that are very close to a given lemma. In the end I can say that this approach works well because of BabelNet, that contains a lot of useful information and correlation between synsets.

[1]All of them can be found in graph.py file. Some of them are not working or not good and can be found in not_used.py

Table 1: Results on eval dataset

Graph used	Prediction algorithm	senseval2	senseval3	semeval2007	semeval2013	semeval2015
$Gsup$	Static S_p	52.89	46.70	36.92	51.03	51.56
	Static Mass M_p	52.45	46.75	37.58	51.27	51.17
	Document D_p	53.59	47.62	39.12	54.44	53.71
	Path prediction $SubPath_P$	53.85	47.24	38.90	50.72	51.76
$Gsup_w$	Static S_p	58.98	56.32	45.05	59.24	58.31
	Static Mass M_p	58.72	56.05	43.29	59.30	58.41
	Document D_p	59.81	56.75	45.71	60.40	62.42
	Path prediction $SubPath_P$	**60.21**	**57.08**	**46.37**	**61.55**	**62.42**
MFS Baseline		52	47.7	35	49.0	51.2