

Assignment 3

Natural Language Processing

Shilpa Gupta (110948405)

Idea :

The Main task of this assignment was to get our hands dirty with different type of feature extraction task for the given training and test data in order to predict whether the given sentence contains “educated from” relation or not. This task was broadly divided into 4 parts in the first part we tried to just predict the relation directly by matching few generic regular expressions, there were different set of combinations to try including using only words and the POS tags of the words. The second part was to apply support vector machine classifier on set of features generated by bag of words approach. This is the basic way of generating the feature vectors corresponding to text.

Since in the bag of word, approach we tried in the second part, gives the feature vector to be very big, I slight improvement in this approach can be done by using brown cluster approach by clustering the words together and assigning the same ids to the words belonging to same cluster. This was our third part. In the fourth part we tried to include different features from the dependency parse tree of the sentence. Below are the all parts and their results explained in more detail.

Part 1 : (Regex Approach) :

After eyeballing the given data we can see that there are certain words phrases that indicates the “educated from” relation directly, for example phrases like “attended the”, “educated at”, “received a degree from”, “matriculated from” etc. the idea is to exploit these patterns and see how many of these given data points are directly classified rightly using these phrases.

Word Regular expression tried	Results
1. R'^.* graduated .*'	Precision : 70.61%
2. R'^.* studied .*'	Recall : 70.61%
3. R'^.* attended .*'	F1 : 70.61%
4. R'^.* received .*'	
5. R'^.* educated .*'	
6. R'^.* degree .*'	

Another idea in the same direction is to use POS tags along with the normal words in the regular expression it can capture more information regarding the phrase we are trying to capture for example we can say if we see “Entity 1 <verb> <Preposition> Entity 2” this seems more plausible and constrain our criteria for the sentence. Below are the results for this approach.

Word+POS Regular expression tried	Results
1. Above verbs followed by preposition 2. Above verbs followed by preposition followed by place noun 3. Pronoun followed by above verbs	Precision : 70.8% Recall : 70.8% F1 : 70.8%

Errors in this approach :

One of the example of the failure of this approach is received can be about anything not necessary about a degree below is the example : “received “Lifetime Achievement Award” from” regex classifier detected it as positive example by seeing received but here we are not talking about any degree of graduation.

Part 2 : (Bag of word Approach) :

This is the most simple way of representing the text into numbers, we first create a vocabulary of words which are part of our sentences. By sentences here I mean the intermediate text between entity 1 and entity 2, in this case the intermediate text between person and the institution mentioned. Let say the size of the vocabulary is N, we represent each sentence by a N length vector where each word has a unique index, we set this index to 1 or more depending upon how many times that word is mentioned in the given text.

In this approach, I tried 2 scenarios, First I took the sentences as it is. That means no modification/cleaning done on the sentences just converted all of them into lower case to maintain uniformity. Below are the results.

BOW without modifying sentences	Results
	Precision : 75.1% Recall : 83% F1 : 78.9%

Secondly, I removed the special characters and converted the sentence to all lower case and check on the accuracy again

```
intermediate_text = intermediate_text.lower()
intermediate_text = intermediate_text.replace("/", " ")
intermediate_text = intermediate_text.replace("-", " ")
intermediate_text = intermediate_text.replace("(", " ")
intermediate_text = intermediate_text.replace(")", " ")
intermediate_text = intermediate_text.replace(".", " ")
intermediate_text = intermediate_text.replace("?", " ")
intermediate_text = intermediate_text.replace(",", " ")
intermediate_text = intermediate_text.replace(";", " ")
intermediate_text.strip()
```

Below are the results :

BOW after modifying sentences	Results
	Precision : 72.7% Recall : 83.8% F1 : 77.8%

Another change modification to above approach is to use nltk tokenization in place of simple split to identify words in the sentence. I again trained and tested the model after tokenizing the sentences using nltk tokenizer. Below are the results

BOW +nltk tokenize	Results
	Precision : 73.4% Recall : 83.0% F1 : 77.9%

Errors in this approach :

In bag of word approach we generally try to predict the result based on what words are present in the given text. It does not accommodate the relation or the order of the word into the account. For Example if the text is “ her mother was graduated from ” in this case it has the right words but they are not talking about the Entity1.

Part 3 : (Brown Cluster Approach) :

In the above mentioned approach we create a word vector equal of the length of the vocab size. We can intuitively say that every word might not be of equal importance to lead to the decision. Considering each word uniquely is also increasing our feature vector size which is computationally expensive. The idea of Brown clustering of text is to cluster the words by them having embed into similar context. This way we will represent each sentence as a feature vector of length equal to the number of clusters. Each cluster has a unique id. And we assign each index a number based on how many words in that sentence belong to that particular cluster.

Also we can experiment with the length of the ID that we want to consider while clustering which will also eventually be the decision of fine or coarse grain clustering.

Below are my experiments and results:

No. of clusters	ID length	Results
100	11 (Max)	Precision : 66.1% Recall :98.5% F1 :79.1%
100	10	Precision : 65.6% Recall :98.9% F1 :78.9%
100	9	Precision : 65.8% Recall : 98.9% F1 : 79%
100	8	Precision : 65.4% Recall :99.7% F1 :79%
100	7	Precision : 65.4% Recall :100% F1 :79.1%
100	6	Precision : 65.4% Recall :100% F1 :79.1%
100	5	Precision : 65.4% Recall :100% F1 :79.1%

100	4	Precision : 65.4% Recall :100% F1 :79.1%
100	3	Precision : 65.4% Recall :100% F1 :79.1%

I Finalized with the cluster ID 11 which is the max, because it is giving the better F1 score than others.

Errors in this approach :

Although this approach is computationally less expensive due to small feature vector size, but it still consist of the same problem of not taking into consideration the order or relation of the words. If the word is used in the sentence but not in right order, this model going to return true.

Part 4 : (Dependency tree Approach) :

The idea behind this approach is to extract the relevant features from the dependency parse of the given intermediate text, and use these features to predict the relation. I have used Stanford Dependency parser for this purpose. There are different features possible in this approach which can be exploited in order to be useful for predicting a relation.

For this purpose I first made the sentences by appending person and the institution at the front and back of the intermediate text. Also since person and institution can be made of multiple words I made them one word by removing spaces in order to consider it 1 word entity in dependency tree.

Eg.

RufusWheelerPeckham and isabella adeline; his mother died when he was only nine. following his graduation from **TheAlbanyAcademy**

Code for extracting below features can be found in gen_dep_features.py

After parsing above sentence using the stanford dependency parser, I extracted below features.

1. Length of the left path :

This is the length of the path from E1 (person) to the root. Below is the sample code to extract this

```

unvisit_tree(tree)
person_node = tree[0]
left_path_len = 0
start = person_node
while start['head'] != '0' and start['visited'] == False:
    start['visited'] = True
    left_path_len += 1

```

```
start = tree[int(start['head'])-1]
```

Since there were cycles of 2 nodes in the tree at some places I had to mark the nodes visited in order to avoid cycles, If there is cycles we just break from the loop and consider the length of path before cycle to be the left path length. I believe that this feature will relate to the closeness of the person from the root verb, which will be relevant to identify whether we are talking about the person's education or someone else's.

2. Length of the right path :

Same as above, this length is the length of the path from the E2 (institution) to the root. This feature as well will indicate if the institution is related to the root verb.

3. Length of left path + length of right path :

This feature is nothing but the sum of above 2 features, I think this feature will play role in identifying how closely the person and the institution are related.

4. Bag Of Word representation of left path dependency labels :

For this features I represented each dependency label as a one ID in the feature vector and for every tree added one to the corresponding index position if that particular label is present in the left path. This feature vector will help us to detect if there is any particular label sequence of pattern is present in the left path of the dependency tree (path from the E1 (person) to the root)

5. Bag Of Word representation of Right path dependency labels :

Same as above this is the Bag of word representation of the dependency labels of the right path which is the path from E2 (institution) to the root.

6. Bag Of Word representation of left path words POS tags :

This is the Bag of word representation of the POS tags of the words occurring in left path. This will help us in identifying or detecting if there is a pattern of POS tags in the left path.

7. Bag Of Word representation of right path words POS tags :

Same as above, this is the Bag of word representation of the POS tags of the words occurring in right path. The code for extracting these features can be found in `gen_dep_features.py`

8. Are root of E1 and E2 is same? :

This feature I included to check if person and the institution are part of same sentence. For this purpose I extracted the root of the left path by travelling through the heads sequentially starting from the person node, and similarly extracting the root of the right path by travelling the heads sequentially starting from the institution node. And at the end check if both the roots are same, if yes value of this feature is 1, 0 otherwise. I included

this feature to avoid the cases where person and institution are not in directly relation and not part of the same sentence.

I run the SVM classifier by taking only above mentioned features, below are the results

Dep Feats	Results
	Precision : 72.7% Recall : 85.9% F1 : 78.8%

Errors in this approach :

1. First issue in this approach was, while finding the path from Entity 1 to root or Entity 2 to root there were cases when there was cycles in the graph. Because of which our estimation of the length of left and right path in the tree is not right in some cases.
2. In few of the above features our assumption is that root verb in the dependency tree will be related to education, but it might not be the case. For example, the sentence can say “marry was born in the town near the boston university” here our dependency tree might start by the verb born, and might not be related to education relation at all but our classifier consider it the positive case.

Kitchen Sink & Final/Best Results :

Total features that we are considering


Feature vector size = {0/1 class detected by regex classifier} (1) + {BOW representation by Number of clusters after tokenization} (100) + {length of left path} (1) + {length of right path} (1) + {length of left+right path}(1) + {BOW of left path labels} (184) + {BOW of right path labels} (184) + {BOW of left path POS tags} (17) + {BOW of right path POS tags} (17) + {Are root of E1 and E2 is same} (1)

= 507

Results :

Kitchen Sink	Results
	Precision : 72.9% Recall : 86.2% F1 : 79%

The final results goes as follows :



Feature Group	Precision	Recall	F1
Manual Regex	70.8%	70.8%	70.8%
BOW	75.1%	83%	78.9%
BOW w/ NLTK Tokenizer	73.4%	83.0%	77.9%
Brown Clusters	66.1%:	98.5%	79.1%
Dependency	72.7%	85.9%	78.8%
Kitchen Sink	72.9%	86.2%	79%



