Yehoshua Stern 314963927

Eyal Cohen 207947086

Ass 4 NLP report

System description:

After many failed attempts with rule-based system, we finally decided to take the machine learning approach.

We decided to use linear regression from the sklearn library, we saw in our experiments that the simpler the model the better our results, which is reasonable considering the data was small

The model is a binary classifier, 1 for Live_In relation and 0 for all the rest.

Preparing the data:

We got the sentences from the .txt file and we pulled the data with spacy library, then after we did so we processed the data, for each two entities in a sentence we created a train example, if the entities appeared in the .annotations file and the relation between them was Live_In we considered this example with y = 1, and if it didn't we considered it as a bad example with y = 0, so basically the data is a binary data, 1 for live-in and 0 to everything else.

The TRAIN.txt file size was 345 lines long, with this method we succeeded to create 7200 training examples.

Features:

When creating the features we wanted to keep it as simple as we can because the lack of training data, we used the DictVectorizer class from sklearn to transform the features to a vector to be fed to the model.

The code for the features creating is in the create_features.py file.

Features list:

1. Entity 1 type
2. Entity 2 type
3. Concatenation of the two entity types (ent1_ent2)
4. Word before the first entity
5. Word after the second entity
6. Number of entities between the given two entities in the sentence
7. Length of the path between entities in the spacy dependency tree
8. Pos triplets' combinations between the entities in the sentence path
9. Pos triplets' combinations between the entities in the dependency tree path

Explanation of the pos combinations:

If we have the sentence – sent 10: Israel television rejected a skit by comedian Tuvia Tzafir…, let's say the first entity is Israel and the second entity is Tuvia Tzafir, the pos tags between the entities is: "television rejected a skit by comedian" → "NOUN VERB DET NOUN ADP NOUN".

Now we took every combination with size of 3 (we tried also 2 and 5 but 3 was the best size in our tests) from the pos list above and added it as a feature, the idea was that if later at the test we get a sentence with almost the same structure between the entities we would still get a lot of the same features even though it's not exactly the same list of pos tags between the entities.

For the dependency tree combinations, it's the same as the sentence only that it's the pos tags list on the route in the spacy dependency tree between the entities.

Other features we tried to add but did not improve the accuracy on the dev set:

1. Dependency edges (combinations of) on the dependency tree between the entities (led to overfitting in our opinion)

<u>Training:</u>

We took the training examples we created and fed it to linear regression model, we changed the call weights to be 0.8 to y = 1 and 0.2 to y = 0 in our tests it got a better f1 score from 50:50 split and from the opposite split (20:80).

<u>Prediction:</u>

We tried to add some rules when predicting but we decided not to use it because the accuracy on the train did not improve and the accuracy on the dev set reduced.

The rules we tried:

0. Rule 0 - if the first entity not in the person entities we got from the training set or the second entity is not in the list of places entities we got from the train set then change the tag to 0.
1. Rule 1 - if we have not entities between the two entities and the first one is place and the second one is person and the word " 's " appear between them then change the tag to 1
2. Rule 2 – if the words " live in " appear between the entities and we have no other entities between the entities change the tag to 1

These rules seen to work when looking at the training set because it was learned from there but when we tried it on the dev set we got less accuracy, so we decided to leave the model without these rules.

<u>Error analysis:</u>

We noticed that our classifier had good recall score but low precision, we think its because the use of combinations of pos tags as features, because we only have 110 Live_In examples we believe the classifier didn't have enough examples to learn the exact patterns, so some of the patterns it learned classified bad examples as good ones, and that's why the recall is higher.

Common mistakes:

Sent1695 – " Wang Shaohua , an official with China 's consulate in San Francisco , said those who fled ` ` were instigated by people with motives to bring damage to China. ' ' " :

Our classifier outputted 3 times about the same entity "Wang Shaohua" that he lives in different places, we think its because of the way we modeled the problem with the combinations of pos tags, when two place entities are in the same sentence and are closer to each other the classifier might give false positive because only one of them is correct and have most of its features like a good example but because the other entity is close to it, it will have almost the same features and get classified as true example.

Sent4590 – " In 1969 , James Earl Ray pleaded guilty in Memphis , Tenn. , to the assassination of civil rights leader Martin Luther King Junior ":

James Earl Ray Live_In Memphis – it was told in the sentence that he pleaded guilty there (that’s in the train sentences), we assume that if we had more data the model was powerful enough to distinguish between sentences of the format (someone who did something at a place).

Sent3073 - " U.S. (USTR) Mickey Kantor , who came to Japan amid the political confusion within the Hosokawa administration caused by the economic stimulation plan , stated before he left for home that " If we see no actual results , we might have to consider other options . " ":

Mickey Kantor Live_In Japan: in this example the true output was Mickey Kantor Live_In U.S and the model output both, we thought of adding a heuristic that if we got one output about an entity so the next time we wont output that he lives in a different place, but when we looked at the data we saw examples that had more then one place for the same entity, so we decided to live it like it is.

<u>Results for Live-In relation:</u>

Dataset:	Train	Dev
Recall	0.75	0.48
Precision	0.73	0.46
F1	0.74	0.47