Aviv Shisman 206558157 Itay Hassid 209127596

Write Up - Exercise 4

Intro

Relation: 'Live_In'

Explanation:

Our model is a rule - based one, we divide the rules to two groups:

1. The generalize rules (increase the recall) -> generare from the corpus the candidates to have the Live_in relation in their sentence.
2. The filtering rules (increase the precision) -> filter unrelated candidates

We used the spacy.nlp package to extract ner entities from the sentence , each entity have a number of useful fields that helps us to decide if to include him or not and later we also use these fields to filter some of them.

Note: We also used external file we created that help us to filter without over fit. (more info in the filter stage)

Note: after consulting with friends , we found that a rule based system will be the best course of action and will give us the best results in our opinion.

Program flow:

Get Input → train(optional - for filtering) →
Generation Stage (for each sentence in test(dev) extract relation using gen rules) →
Filtering Stage (for each sentence we now have candidates we will filter using rules) →
Eval Stage (getting F1,recall,precision -> comparing to the dev.annotations)

We will elaborate each Step in the following pages…

Cont to the next page

Generation Stage:

Goal: we want to check each sentence in the test(dev) corpus and decide if it might contain our relation, or not , for that we will use a set of rules - the generation rules (see generalize_rules function at the code for references)

this stage only get the candidates of our model not the final output, in this stage our aim is to find the most true candidates(ones that have our relation) we can → the highest recall we can get without hurting the precision too much (with unnecessary candidates).

recall our relation is 'Live_In' therefore we will look for a relations/clues the will imply that the current sentence can have the 'Live_In' relation.

The rules of this stage are divided to two groups also:

1. Main generalize rule (with small extensions)
2. More minor rules for little groups we did not count

Methodology: after calculating False negative we written to a file ('others\not found') all the false negative examples , we looked over each one of them and tried to find rules that will apply to a big portion of them and will not overfit.

Note: Before this stage we iterate over all the possible duos of entities in the sentence and send them to the generation rules to see if there is a relation to extract from these 2 entities, at the end all the relations from the sentence are sent to filtering.

Main rule

Input : 2 entities
Output: is there a relation in the sentence between those 2 entities

Rule: if ent1 is 'Person' and ent2 is 'GPE' → add (ent1,ent2) to relation candidates (function 'MainRules' for reference)

Note: Notice that we take care of cases which the entity has a title before like Ms. We also add '.' (dot) in case the entity appeared with a dot in the sentence.

Cont to the next page

Why we use this rule?

This is the most basic rule, every sentence with the 'Live_In' relation include a person(ent1) and the location he lives (ent2 , label=GPE).

We will see later that in some rare cases spacy extraction is not good (label person as organization or gpe as a person and so on… therefore we have the second set of rules)
Notice we don't care if the person is before or after the gpe because both are logical:
'Amy is from new york' , 'after just moving to new york, amy was ….'

→ in both cases amy lives in new york

It is also important to clarify that because we iterate over all the possible entities couples in the sentence, eventually we will get the right order => the person will be the first entity and the gpe will be the second entity → then the rule will classify them as a live_in relation. (in case they fulfil the rule)

Second rule set: (in case we did not find relations in the sentence using main rule)

Input : 2 entities
Output: is there a relation in the sentence between those 2 entities

First rule

We saw that a big portion of the recall mistakes contained sentences with the words 'in' or 'of' for example: ' aviv lives in tel -aviv , israel'
We set a hyper parameter = ' rad'

The rule : if ent1 is person and the sentence have 'in' or 'of' and the distance between them is lower than rad → add (ent1,ent2) to relation candidates.
(for reference look in function second rules)

Notice the higher the 'rad' → lower the recall loss but it if it will be too much high it can add a lot of unnecessary candidates that will lower the precision and f1 score.

Notice the order is important in this stage -> 'aviv of israel' and not 'israel of aviv'
→ The second example is a bad candidate.

Cont to the next page

Second rule

Meny times the first entity (person) was successfully extracted but the second (the gpe) Was counted as a ‘ORG’ (organization) or ‘Person’ (for example port arthur)

rule: if ent1 is a person and ent2 is a organization or a person (in title form) with a reasonable distance(hyper param, in absolute value) from the person → add it to relations.

Notice we don’t care about the order because like the main rule the person can appear before or after the location.
(in the first rule of the second set the order is important because the person must appear before the ‘of’ or ‘in’ + place)

With this rule we can lower the recall loss (raise the ‘rad’) but notice that like before it comes with a cost → the higher the ‘rad’ is the lower the recall loss but it can add unnecessary candidates and lower the precision → we need to find the best rad that lower the recall loss in the smallest cost to the precision.

Third rule

Similar to the second rule, if the ent2 is a ‘NORP’ →
we saw in rare cases that ent2 was ‘NORP’ instead of gpe (e.g soviet was classified as ‘NORP’ by spacy) therefore we added the following rule, it also have a rad that affect the recall/precision like i described before, also here the order does not matter.(like previous rule)

rule: if ent1 is a person and ent2 is a ‘NORP’ (in title form) with a reasonable distance(hyper param, in absolute value) from the person → add it to relations.

Cont to the next page

Fourth rule

Like the second and third rule its goal is to get the candidates that are missing because the spacy extracted them as a wrong label.

Rule: if a ent1 is a 'ORG' (labeled it as org instead of a person) and ent2 is 'gpe' and the have a distance (in absolute value) of rad or lower between them → add (ent1,ent2) to relation.
Here the order does not matter and the rad has the same effect as the 2 rules before

Summarize:
@ Main rule → adds most of the candidates
@ First rule of second set → adds candidates by the logic we described
@ Second,Third,Fourth rules of the second set → add missing candidates due to spacy bad extraction.

Result :Our recall without filtering (and with filtering copies) got to **60-69%**(depends on the hyper parameters) **but** that was before filtering.

Recall loss: it is important to clarify that a large part of the recall loss is due to spacy unsuccessful extraction for name and names of countries, for example :

- Took ' the united states' instead of 'united states'
- Took ' Carlos' instead of 'Carlos Andres Perez'

Misc of this stage:

- We have a 'not found' file in 'others' directory, we used this file to raise the recall ,the file has all the not found examples.
- We have a 'why not found' and 'fix recall' → our unfiltered conclusion during the debugging of this stage.

Cont to the next page

Aviv Shisman 206558157 Itay Hassid 209127596

Filtering Stage

Goal: now we get from the generation step a list of possible candidates for our relation and we need to decide if it contains our relation or not.
(see filter function at the code for references)

Input: a sentence and its possible relations (can be none)

output: this stage outputs the final list of relations for each sentence in this stage,
In this stage our aim is the highest possible precision we can get without hurting the recall too much.

Methodology: after calculating False positive → our mistakes , using the annotations file , we found which example we need to filter (‘others/need to filter’) and found rules that will filter them while keeping the goal in mind(high precision without hurting the recall too much)

Note: in this stage we used a model file that contained information on the dependency and the pos tagging of correct relations from both the train and the dev, we will use it later to filter.

Filtering rules:

First rule

The simplest one → remove copies of relations it gives us a false high recall and precision.

Second rule

Also quite simple one , check if ent1(the person) has ‘at’ or ‘-’ before → implies it is not a person, if he does → remove the relation.

Why?
- At implies its a location
- the char ‘-’ implies it is part of other entity

Cont to the next page

Third rule

this rule checks if the sentence contains words that implies the sentence is not related to the 'Live_In' relation (labels like time,date,loc,ordinal etc)

If a sentence have more than threshold → we remove the all sentence and its relations

Why?

We found out that sentences that broke the threshold 99% of them were not live_in related so the removal was clean and did not reduce the recall , only the precision got up.

Note related to rule 6: if the sentence contain one of the unrelated labels we classify it as a suspect and if he is a suspect we use rule 6 on him later(rule 6 explained later)

Fourth rule

This rule is one of the major filter, and increases our precision by a significant portion. In this rule we use the pos,dep dictionaries → these are dictionaries we created in our "train" function → in the train function we went over the dev and test data and extracted from them dictionaries that represent the data, later we write them to a file one time ,and now every run we read them.

What they contain? → they contain all the pos and dependency tagging duos and how much we saw each one in both the train and the dev.

```
filter_dep = {dict} {('nsubj', 'poss'): 9, ('pobj', 'amod'): 6, ('nsubj', 'appos'): 8, ('appos', 'appos'): 3, ('pobj', 'compound'): 14, ('pobj', 'poss'): 3, ('pobj', 'pobj'): 12,
filter_pos = {dict} {('PROPN', 'ADJ'): 15, ('PROPN', 'PROPN'): 246}
```

In this screenshot you can see example of what the contain, we filter relations that can't be found in one of the dictionaries , because those dictionaries represent the date quite good and if a relation pos or dependency tags is not in there → high chance they are unrelated.

The rule: we go over all the relations candidates and get their dep and pos taggings → (ent1.pos,ent2.pos) ,(ent1.dep,ent2.dep) → if one of these duos is not in pos.keys or dep.keys we remove the relation.

(see the 4th filter in filtering_rules function for more info about the rule)

Cont to the next page

Fifth rule

These rule is also one of the major filtering rules, we check for each relation in the relation candidates if the distance between ent1 and ent2 is greater than rad (hyper param) in case it does we remove the relation.

Why?
We saw that in a lot of cases we suffered precision loss from sentences with big distance between the entities , furthermore if the distance is big we can also deduct that ent1 and ent2 are probably unrelated and therefore no part of the live_in relation.

The rule: for all the relations check if the distance is bigger the 'rad' if it does remove the relation.

Note: the order of the entities does not matter we calculate the distance in absolute value

Sixth rule
We check this rule on a sentence only if it's considered suspicious.
In this rule we assumed that in a relation of 'Live_In', ent1 is an ancestor of ent2, in other words, ent1 is above ent2 in the in the dependency tree.
Note: it's doesn't matter the actual order of the entities in the sentence.
Using spacy we can access the dependency tree of any name entity, so we look at ent1's successors 2 levels deep down and check if ent2 showing up in one of this levels.

We saw this technique make precision very high because it's find many wrong relations. But it also removes correct examples, probably because the 2 words too far from each other in the sentence and therefore they are more than 2 levels away, another possible explanation is that spacy analyzed incorrectly the sentence.
In order to try to fix this problem, I assumed that relations that don't have dependency relation may be close to each other in the sentence, so we set a minimal distance between 2 words so even if the words don't have dependency relation, but closer in <radius - hyper parameter> words away, they'll still be considered a relation.

Why?
Until now we didn't look at grammatical relation between 2 entities (which is not statistic like rule 4), and looking at the relation in the dependency tree gives us a lot of information to determine whether it's relation or not.

Cont to the next page

Summarize:

@ rules 1,2,3→ are simple filtering rules , not aggressive ones
@ rule 4 → major filtering rule that use the “model” we extracted and the information of the the dev and train files.
@ rule 5 → major filtering rule that filter by the distance of the entities
@ rule 6 → minor filtering rule that remove relation that don’t have connection in the dependency tree.

Results :

On train:

```
/usr/bin/python3.5 /home/magshimim/Desktop/NLP/ex4/extract.py model.file Corpus.TRAIN TRAIN.annotations
Hello! Welcome to our relation extractor
please wait a few seconds...

results!

recall= 0.4230769230769231 , precision=  0.3716216216216216 , f1=  0.3956834532374101
```

On dev:

```
/usr/bin/python3.5 /home/magshimim/Desktop/NLP/ex4/extract.py model.file Corpus.DEV DEV.annotations
Hello! Welcome to our relation extractor
please wait a few seconds...

results!

recall= 0.44274809160305345 , precision=  0.3972602739726027 , f1=  0.4187725631768953
```

Precision loss: i believe a big part of the precision loss is due to the recall being decreased , it is hard to filter without filtering good candidates , therefore as we saw the precision was higher on this stage but it came with a great cost to the recall.

It important to mention that we tried a lot of combinations and different types of filtering and these are the best rules that brought the highest f1 score.

Misc of this stage:

- We have ‘need to filter’ file that helped us to reduce the precision loss

Cont to the next page

Evaluation stage

relation	Train recall	Train precision	Train F1 score	Dev(test) recall	Dev Precision	Dev F1 Score
‘Live_In’	0.423	0.3716	0.3956	0.4427	0.3972	0.4187

Was calculated as:

Recall = TP/TP+FN = what we tagged right/ what we tagged right + what we thought was not related or accidently filter.

Precision = TP/TP+FP = what we tagged right/ what we tagger right + what we thought was right and need to filter.

F1 Score = (precision*recall)/ (precision +recall)

Cont to the next page