CS 6320 Natural Language Processing

Fall 2020 – Mithun Balakrishna

Final project

Extracting Relation between two named entities

Group name: NLP Programmers

Members: 1) Bala Santosh Balija(bxb180030)

2) Krishnan Vijayaraghavan(kxv190006)

PROJECT DESCRIPTION:

The goal of this project is to extract relations between two named entities in a sentence. We use the TAC KBP dataset that contains 41 different relation types. The input is a sentence with two annotated entities e1 and e2 (contains the span of the two named entities). First, we read the data file and process the information in the files in such a way that suits our model. Next, a series of tasks are performed to extract various NLP based features from the sentences. Then, a machine-learning, statistical or heuristic based approach is implemented to determine the relation and its direction for the given pair of arguments(entities). The final output is the one of the 41 relations in the train set and its direction((e1,e2) or (e2,e1)). The direction does not exist if there is no relation between the annotated entities.

Two examples are provided below:

1. On Tuesday the parent company <e1> Kaupthing Bank </e1> announced that it had received a 500-million-euro loan from <e2> Iceland </e2> 's central bank to facilitate operations while the government announced that it had nationalised the country's second - largest bank Landsbanki .

In the above sentence, the two annotated entities are Kaupthing Bank and Iceland.

The output for the sentence is org:country_of_headquarters(e1,e2).

Here, org:country_of_headquarters is the relation between the entities and the direction is (e1,e2).

2. It was brutally occupied by <e1> Benito Mussolini </e1> 's Fascist Italy from 1936 to 1941 ending with its liberation by British <e2> Empire </e2> and Ethiopian Patriot forces .

In the above sentence, there is no relation between the entities. So, there is no direction as well.

So, the output is labelled as no_relation.

PROPOSED SOLUTION:

FIRST TASK:

The first task is to read the train data file and process the sentences.

First, we extract the different lines from the text corpus (train data file). Then, the e1 and e2 entity words are extracted and stored in a list. The e1 and e2 start and end markers are removed from the train sentences along with the indexes. The words in between the named entities are also extracted. Since the train sentences are extracted separately, the relations of the train sentences are extracted separately and stored in a list. Finally, we have list of e1, e2, train sentences, words in between the entities and the relation of the train sentences.

SECOND TASK:

The second task is to extract the NLP features from the processed sentences. There various subtasks within this task are

1) **Tokenization:** We tokenize the words of every sentence. nltk package is used - nltk.word_tokenize(sentence)

2) **Lemmatization:** We lemmatize the tokenized words of every sentence. nltk package is used - wnl = WordNetLemmatizer() ---> wnl.lemmatize(token)

3) **Part-of-speech tag:** POS Tagging is done on the lemmatized words. Nltk package is used - nltk.pos_tag(lemmatized_word)

4) **Dependency parsing:** The dependency relations for each lemmatized word in a sentence are determined using spacy library –

```
doc= nlp(sentence)

for item in doc:

sentence_dependency.append([item.text, item.dep_, item.head.text, item.head.pos_])
```

5) **Wordnet:** We extract hypernym features. First, the synset of every lemmatized word paired with its pos tag is determined. Next, the hypernym of each synset of every word in the sentence is found.

6) **NER tags:** The named entity tags of every noun in the sentence is extracted using spacy library.

```
doc= nlp(sentence)

for entity in doc.ents:

    ner_sentence[entity.text] = entity.label_
```

THIRD TASK:

The third task is to pick the features and implement a machine learning model that would help to predict the relation and direction. We store very feature in a list. So, the features included in the model are:

i) Annotated e1:

We extract the entity 1 of every sentence in the train and store it in a list. However, this is done in the initial text processing phase itself.

ii) **Annotated e2:**

Similar to e1, the entity 2 is extracted.

iii) **Length of words in between the entities:**

In the text processing phase, the words in between the entities is obtained. Using this list, we calculate the length of words in between the entities.

iv) **Words between the entities:**

The words between the entites obtained in the preprocessing step are fed into the model using the one Hot encoding technique.

v) **Part-of-speech tags of words in between the entities**:

Similar to the previous feature, the pos tags of words in between the entities are obtained using the words in between. We first calculate the start and end indexes of the words in between the entities. Using the indexes, we extract only the pos tags of the words in between the start and end indexes and create a part of speech string for each of the train sentence by extracting the first two characters of the part of speech of the words in between the entites separated by space.

The part of speech string obtained in the previous step for each training sentence is converted to a vector using the one Hot encoding technique, by considering the unique number of bigrams of part of speech in the whole training sentence.

vi) **NER tags of e1 and e2:**

Initially, we create empty list for every sentence. The NER tags list created in Task 2 is iterated and the NER tags of only the entities e1 and e2 is extracted.

vii) **Previous word of e1 and the next word of e2:**

The words before entity_1 and after entity_2 are extracted and stored in a list. A unique string is constructed with the combination of these words, which are converted to a vector using one Hot encoding technique.

viii) **Hypernyms of e1 and e2:**

We already calculated the hypernym features for all the sentences. There may be zero, one or more hypernyms for a lemmatized word. The top one hypernym is extracted for every word and stored in a list as this would minimize the unnecessary features. Next, the hypernyms of only e1 and e2 are extracted from the list. Firstly, for e1, the hypernyms of every word in e1 is extracted. Then, only the hypernym of the last word in e1 is extracted as the last word in an entity is the most significant compared to other words. The same process is done for e2.

ix) **Root verb:**

We extracted the root verb for each training sentence, using the dependency parser of spacy calculated in the Task2. Root verbs for each sentence is obtained by iterating the dependency parse list and checking the condition (for a word if its head is same as itself then it is the root verb of that sentence). The obtained root verbs are converted to vectors using one Hot encoding.

x) **Root verb position:**

This is the only feature where the one hot encoding was done at the same time. Initially, the index of the root verbs is stored by iterating and indexing the dependency parse list that was done in task2. Next, by iterating through the train sentences the index positions of e1 and e2 words in the train sentences are determined. There are few rare cases where the entity 1 or 2 or both are empty. Those scenarios are handled by assigning the indexes of e1 and e2 to some default values. For example, in the case where both the entities are empty, the e1 was assigned the first index and the e2 was assigned the last index of that sentence. Lastly, the indexes of e1, e2 and root words are compared and assigned appropriate one hot encodings.

xi) **Levin Class:**

The Levin class of the root verb of each training sentence is given to the model as a feature. The Levin class vector is obtained from shortest dependency path between the entities. If the root verb is present in the shortest dependency path, then the corresponding Levin class of the root verb is obtained and fed to the model using the one Hot encoding technique.

While performing one Hot encoding on the features we have obtained all the unique values in the train data and assigned each of the unique value a specific index and stored in a dictionary. We are allocating an array of length equal to the length of the unique values plus one. This extra plus one is used to store if any unseen values appear in the test data.

Then, the features of the train data are combined into a single list. These features are then flattened so that it can be sent into the machine learning model. The features are fed into a SVM model.

We did two variations of training the model.

In the first approach, we considered two separate classifiers to predict the relation and direction. All the mentioned features are considered to train the SVM classifier for relation. The dependency features namely Levin class, root verb and root verb position are considered to train the SVM classifier for direction.

In the second approach, we trained a single SVM classifier for predicting both the relation and direction. If two relations have the same relation and different directions, they are considered as different classes unlike the first approach.

FOURTH TASK:

We run the model on the test set.

As we did two different approaches that is, we considered two different classifiers in one approach and only a single classifier in the other approach, the accuracy, precision, recall, F1 score and time elapsed are calculated for three different cases. We also experiment with different hyperparameters of the SVM classifier.

First approach:

Predicting only relation using different kernels with gamma value 0.1 and C value 1

	Accuracy	Precision	Recall	F score	Time elapsed
linear	0.286	0.226	0.468	0.232	72.123
poly	0.172	0.138	0.371	0.129	73.215
rbf	0.191	0.075	0.203	0.056	73.452

Predicting only direction using different kernels with gamma value 0.1 and C value 1

	Accuracy	Precision	Recall	F score	Time elapsed
linear	0.503	0.365	0.420	0.344	13.323
poly	0.514	0.337	0.322	0.245	13.558
rbf	0.520	0.347	0.336	0.278	12.213

Now, we see the combined accuracy of the instances where both the relation and direction are true (still they are separate classifiers). We experiment with combinations of few kernels.

- Combined accuracy if linear kernels are used for both the relation and direction is 0.166.
- Combined accuracy if relation(linear) and direction(poly) kernels is 0.178.
- Combined accuracy if relation(linear) and direction(rbf) kernels is 0.180.

Second Approach:

Prediction both relation and direction using single classifier with linear and poly kernels.

	Accuracy	Precision	Recall	F-score	Time elapsed
linear	0.225	0.166	0.370	0.163	79.357
poly	0.143	0.109	0.179	0.091	74.360

PROGRAMMING TOOLS:

PROGRAMMING LANGUAGE:

- Python 3.6

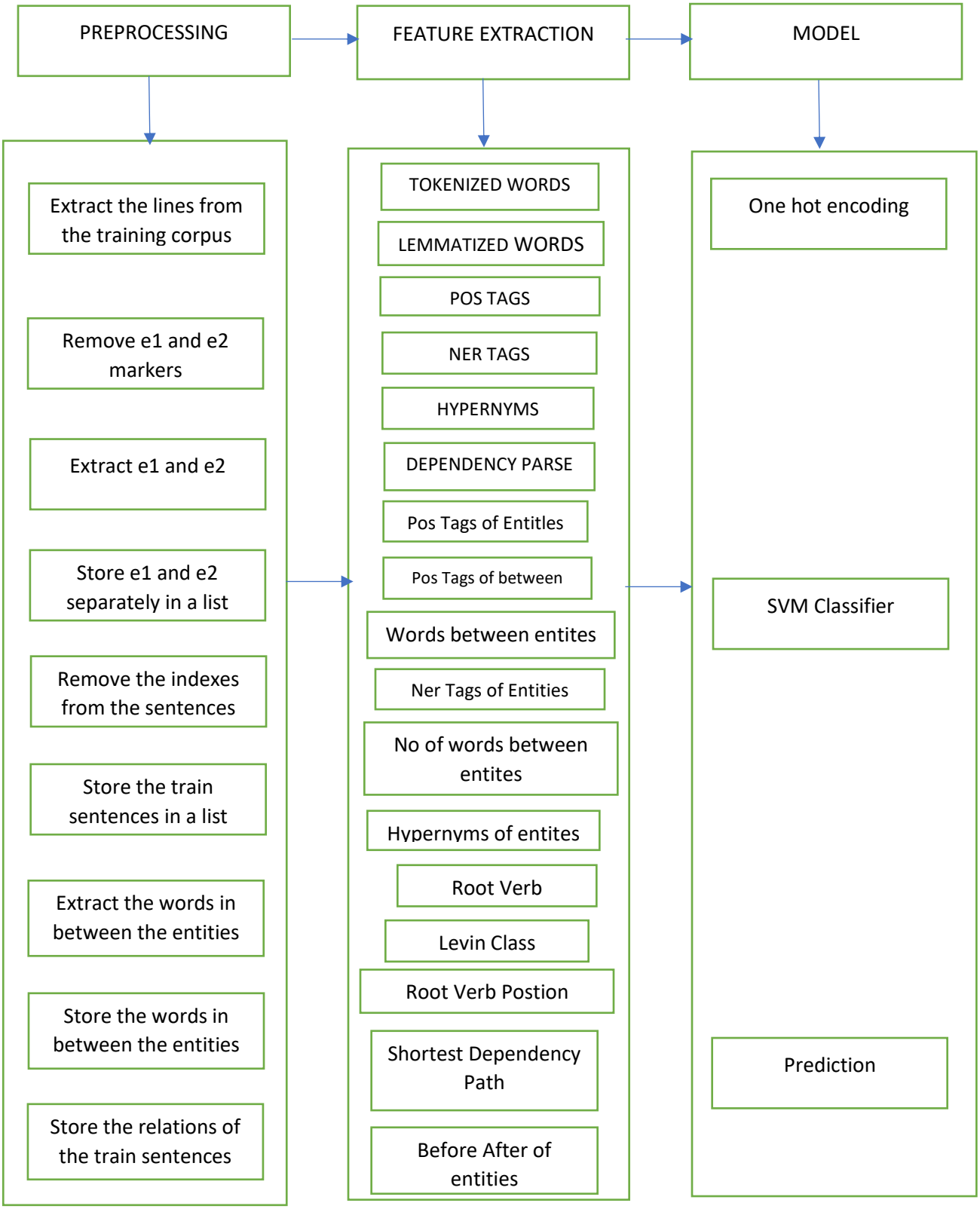
TOOLS USED:

- **Jupyter notebook**

LIBRARIES USED:

- Numpy
- NLTK
    - Tokenization
    - Lemmatization
    - Part of speech tagging
    - Hypernyms
- Spacy
    - Dependency parse
    - Named Entity Recognition Tags
- Scikit learn

ARCHITECTURE DIAGRAM



RESULTS AND ERROR ANALYSIS:

- Maximum accuracy of 28% was achieved without considering the direction while maximum accuracy of 22% was achieved when the direction was considered.

PROBLEMS ENCOUNTERED AND SOLUTION:

1) While extracting the hypernyms of the words in the train sentences, there may be many hypernyms for a word. This caused lot of problems while performing one Hot encoding as there were many features to train. So, we only picked the top hypernym of every word instead of keeping all the hypernyms. Many entities in given training set have many words for an entity. Such entities are handled by picking only the hypernym of the last word as the last word is the most significant one in many sentences.

2) As there are many features during training, it became almost impossible to train the model. The matrix was too sparse. So, we used csr matrix to create compressed a compressed sparse matrix. It improved the speed of training the model.

3) While computing the shortest dependency path, the graph does not accept more than one word. So, in sentences where the entities have multiple words, only the first word of the entity is fed to the graph.

PENDING ISSUES:

- First, we trained two separate classifiers to train the relation and direction of the entities separately. In that case since we included only three features (dependency based) for determining the direction. So, the accuracy was low where both the relation and the direction was classified correctly.

POTENTIAL IMPROVEMENTS:

- To solve the above issue and train the two separate classifiers separately, more features may be included to train the direction classifier. The accuracy may also improve by selecting better machine learning model and its hyperparameters. Else, a single classifier may be used for determining both the relation and direction that was eventually used by us.