Natural Language Processing, Homework 1 Report

Dorjan Hitaj, Matricola 1740478

April 1, 2017

1 System Overview

This assignment required to train our own morphological segmentation model for English as described in the paper provided. The paper treats morphological segmentation as a structured classification problem by assigning each character of a word, one of the 4 classes which are (B-beginning M-middle, E-end of the word and S-single character). Moreover they use also 2 markers to represent the start and end of the word. This task was handled by using CRF with Averaged Perceptron. For implementing, I choose **Python** due to the flexibility and wast resources that this language provides. The crucial library that I used is the **sklearn_crfsuite** library which provides the CRF with Averaged Perceptron. Other used libraries are **numpy** for array processing, **pickle** for model saving.

1.1 Preprocessing

The first step was reading and translating all the words contained in the provided files into the corresponding labelling. For this I wrote a function named **allfilewords2label** which takes as the input a file and returns a python dictionary in which the **key** is the word and the **value** is the word translated to the labelling system. To explain how I made the word labelling I will demonstrate how the function treats each line of the file by considering the first line of the **training.eng.txt** which is **ablatives ablative:ablative_A s:+PL**. Firstly check if there are two or more annotation and consider the first one. After I split the line by spaces. Then at the index 0 of the split is located the actual word that will serve as a key for the dictionary. To this word I concatenate in the beginning a ***** to denote the beginning and in the end a ˆ to denote the end of it. After that I loop in the remaining part of the array, I split each element by the colon **(:)** and consider the first part. The first part is treated like this: If the length of that part is 1 this means that it is a single character and I concatenate **S** to the labelling string, else I concatenate **B** and then (morpheme_length -2) **M**'s and in the end an E. (morpheme_length-2) M because 2 are the begin(B) and end(E) labels. After that to match the start and end labels that I put to the actual word I concatenate also an ***** at begining and a ˆ in the end. So according to my function the line above is translated into a dictionary element like this: key= ***ablativesˆ** and the value= ***BMMMMMMMESˆ**. To parse also the combined data set in which the crowd annotations are in a different shape I modified the parsing function so that when the line that is read is of the type of crowd annotations I get the morphemes by splitting all the line by spaces and the rest is treated the same as explained above. To distinguish between two types of annotations I searched for the colon **(:)**, which is present in the provided annotations and is not present in the crowd annotations.

1.2 Feature Encoding

Next I encoded the features. For each word, for each character of that word there is going to be dictionary that will contain the feature set of that character. This will result in a list of dictionaries for a single word, and a **list of lists of dictionaries**(required input format of CRF) for the all words. To do the feature encoding I wrote 2 functions. One function named **character2feature** which takes a word, the character index and delta and returns the feature set for that character. The feature set contains right and left substrings of the word, of a length up to delta of the character for which we are building the feature set. The other function is named **word2features** and returns the feature set of all characters of a word as a list of dictionaries. And after that to create the input for the CRF I call **word2features** for all words and the result is stored as a list.

1.3 Training Phase

Then I pass the inputs to the CRF for training. In the parameters I have placed the **max_iterations**=100 which is the max number of iterations that is supported for the 'AP'. Else I have placed also 2 other parameters which contributed in increasing the performance. They are: **all_possible_transitions**, which set to true CRFsuite

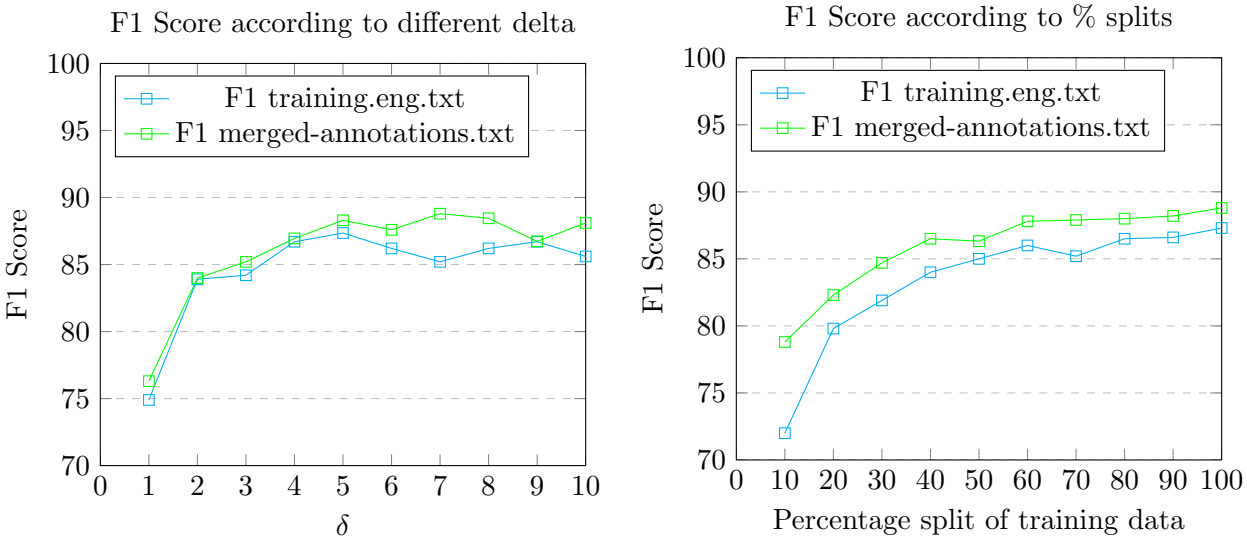
generates transition features that do not even occur in the training data like negative ones, **all_possible_states** which if set to true, CRFsuite generates state features that associate all of possible combinations between attributes and labels. These extra transitions and state features helped the algorithm to learn more and thus perform better on test set.

1.4 Performance Evaluation

After that I take the predictions and use them to calculate the precision recall and f1_score. To do that I created a function named **compute_precision_recall_fscore** which takes to input parameters: the gold standard labelling and the predicted labelling from the CRF. By iterating on the lists I treat every (gold standard, predicted labelling) pair in this way: Use **numpy** to convert them into arrays. Then I get the indexes where E and S are positioned in gold standard and predicted labeling by means of numpy.where() function. I calculate the correctly placed boundaries by counting the number of times a correct boundary is present in the predicted output and increment **H-number of correctly placed boundaries**. This is done for each (gold standard, predicted output pair). Moreover, I calculate the number of incorrect boundaries that are present in the predicted labeling (**I-wrong boundaries in the output**) and also the **D-missing boundaries that are present in gold standard but are missing in the predicted output**. After having calculated H, I, and D compute the precision, recall and f1_score by applying their corresponding formulas.

2 Results and Analysis

The best performance obtained from my model after being trained with the full training set and also jointly using the development set during learning was: **f1_score = 0.8735**, recall = 0.8423, precision = 0.91 with a δ = 5. With the combined data set my models best performance was: **f1_score = 0.888**, recall = 88.2, precision = 0.8935 with a δ = 7.



In the graph to the left is the F1-score of the model with δ from 1 to 10 trained with each of the training files. The delta value is crucial into the performance due to its effect on the size of the feature sets of a word. In both models, from 1 to 5 the performance is always bigger than previous round. Above 5 there is a fluctuation of performances in both models. The model trained with all data peaks performance at delta equal to 7, while the model trained with the initial training data peaks performance at a delta of 5.

As noted in the graph the model performed better by learning from the combined dataset but the increase in performance was not high. I think that this came due to the low complexity and low variety of the majority of morphemes that came from a part of the crowd annotations. This small variety maybe was not enough to make the model perform on a higher scale with other parameters equal as before. Another problem that might have occurred is over fitting. The extra data may have mislead the model into learning less than expected.

On the right graph is displayed the performance of the models trained with random samples of the training file. As expected, since the training.eng.txt has less data, the performance curve is very steep at the beginning due to the small sample size. It gets smother after 50-60%. Above that the fluctuation on performance becomes smaller. The performance curve obtained from the combined dataset fluctuates less because of the bigger samples, as expected. Even though I have performed many runs for each percentage split the combined dataset is outperforming the base training set on each split.