NLP Assignment 1 CRF tagging in Movie Queries **Yash Desai**

Q1) For the first question we split out data into 2 parts, the first 80% of the data as the training data and the remaining 20% as the test data. This has been achieved using the *train_test_split* function from sk-learn and adding the parameters *shuffle = False* and *stratify = True* which prevents the randomness of the split function and helps me to get the first 80% of the data as train and the rest 20% as the test. Thereafter the tagger is trained using the 80% split train data and tested on the 20% split test data using the saved model *crf_nlu.tagger_new.* The testing accuracy achieved was *83%* and the precise macro_average came out to be *0.55352 (rounded to 5 decimal places).* To get a better idea the confusion matrix was obtained for the predicted labels which helped to visualise the number of True Positives (the diagonal of the matrix), False Positives, False Negatives and the True Negatives.

Q2) The second question asks to print all the sentences which contain false positive words that belong to the 5 classes which have the lowest precision score (using the 20% Test Data). In order to achieve the same, a for loop was used to extract the sentences from our Test Data and another for loop inside it (a nested for loop) to work on the individual words.
Now, to calculate the false positives an *if statement was used* which checks if the *predicted labels* are not equal to the *true labels* and if the *predicted labels* are there in those 5 classes with the lowest precision. Now inside the same nested for loop, the following information was appended to a *data-frame:- predicted label, true label, falsely classified word, index of the word inside the sentence and the sentence itself.* The table contained 729 rows which meant there were a total 729 different words which were falsely classified. These observations may have a single sentence printed multiple times i.e for each false positive in it. Further the unique number of sentences were found out which would have at-least one false positive in it. This was achieved using the *drop_duplicates method* of pandas data-frame. The number of unique sentences which had at least one false positive were 619. To decrease the number of false positives in a sentence we need to improve the precision score which can be achieved by improving the overall f1 score (harmonic mean of precision and recall) by adding new and relevant features like 'POS' and next/previous words to our model.

Q3) The third question asks to print all the sentences which contain false negative words that belong to the 5 classes which have the lowest recall score (using the 20% Test Data).
It is exactly the same as the previous question where we calculated the false positives. The only change is that we check that the true labels are in the 5 classes with lowest recall instead of the predicted labels.The number of false negatives came out to be 459. Just like the previous question this 459 observations may have a sentence printed multiple times i.e for each false negative in it.
Further the unique number of sentences were found out which would have at-least one false negative in it. This was achieved using the *drop_duplicataes method* of pandas data-frame.
The number of unique sentences which had at least one false negative are 201.
To decrease the number of false negatives in a sentence we need to improve the recall score which can be achieved by improving the overall f1 score (harmonic mean of precision and recall) by adding new features. This has been achieved in the final question of this assignment.

Q4) A model is said to perform better if it has more number of features (not unnecessary which may lead to overfitting). In order to achieve this, an extra feature (Pos Tags) was added along with the word to our data set. As required, it was added in the form of a string as the CRF tagger would only take string inputs. The data now looked like ['Word+POS','CRF']. This was done for both the training and the testing data. Now at the time of extracting the features we split the word and pos again and add them as separate features. Adding this extra pos tag has increased the macro_average from *0.553* to *0.559* which is not significant but gives a hint on how to improve model performance.

Q5) As it is observed in Question 4 that the macro-avg (f1-score) increases by a small percentage when we add the POS tags as features, now more relevant and meaningful features have been added to the model. The model which was trained in the previous question had a macro-avg score of about 0.56. This has been improved by adding the following features in the feature set.
'is_first_capital' 'is_first_word' 'is_last_word' 'is_complete_capital' 'is_numeric' 'is_alphanumeric' 'prefix_3' 'prefix_4' 'prefix_5' 'prev_word': 'next_word' 'suffix_3' 'suffix_4' 'suffix_5' 'word_has_hyphen' 'word' 'pos' 'previous pos' 'next_pos' 'punctuation'
These features were added iteratively by checking the difference in performance due to each set. Four different sets of features were created and tested to get the optimal feature set and hyperparameters which is shown in the notebook. The macro-average increased gradually in each step due to the addition of new features. While testing for different features on the development dataset(80:20) it was observed that the 1st prefix, 2nd prefix, 1st suffix and the 2nd suffix were decreasing the macro avg score, so these 4 features were removed from our feature set.The new features seem to have impact on the model performance, for example considering the next and the previous words as features the model would have decreased probability of falsely classifying 'to' into a 'TO' class'. The word 'to' belongs to the class preposition ('IN') when used in a sentence but when the sentence is broken into words, the model will assign a higher probability for the class 'TO'. Hence it helps to have previous and next words as features in our feature set. Moreover, to better overcome the misclassifications, the previous and the next POS tags are added as features as well. The 'word_has_hyphen' feature helps the model to identify if the word is a connecting word. To further improve the performance of the model we have added 4 hyper parameters while training. We first set the max_iterations as '100' which shall prevent the model from excessive training and avoid overfitting. Moreover, we set the min frequency of features as '2', c1=0.25 (the coefficient for L1 regularisation) c2=0.3 (coefficient for L1 regularisation). The values of c1, c2 are usually in the range 0.2-0.3 hence with some testing, the best values obtained were 0.25 for c1 and 0.3 for c2. To get a detailed summary of model performance on individual classes, I have created this comparison where one can see the improvement of the macro-avg(f1 score) for different classes. A drastic change can be observed for the classes *I-Soundtrack* and *B-Soundtrack*. From a score of almost zero they increased to 0.32 and 0.4 respectively.

- *B-Actor 0.87 --> 0.93*
- *B-Director 0.78 --> 0.86*
- *B-Quote 0.38 --> 0.48*
- *B-Soundtrack 0.00 --> 0.4*
- *I-Director 0.81 --> 0.88*
- *B-Opinion 0.37 --> 0.44*
- *I-Opinion 0.10 --> 0.24*
- *B-Origin 0.47 --> 0.57*
- *B-Character_Name 0.37 --> 0.58*
- *I-Soundtrack 0.00 --> 0.32*
- *I-Quote 0.63 --> 0.71*

Result:- A model is said to make better and more accurate predictions if it has a higher macro-avg (f1 score). The f1 score is the harmonic mean of precision (takes into account false positives) and recall (takes into account false negatives) which means greater the f1 score lesser the number of false positives and false negatives. The macro-avg (f1-score) of the model trained on the original dataset without adding new features was 0.58 and the score after adding new features has increased to 0.66. Hence, I can conclude that the model performance improves if we include more meaningful and relevant features while training.