NLP Assignment 1: CRF tagging in Movie Queries

Question 1 : Split the training data into 80% training, 20% development set
Split the training data (training_data) into two lists: one split of the first 80% of the instances of training_data, which you will use for training your CRF, and the remaining 20% for testing. Once you've done this re-run the above code such that the tagger is trained on the 80% split and tested on the 20% split, and you obtain the classification report output and confusion heatmap output for the results of testing. Do not use the test data as it is above for testing/viewing results for now. Record the results by saving the classification report output as a string somewhere in the notebook for future reference as you go through.

Answer 1
To split the given training set into 80/20 train and test, the method train_test_split is imported from the class sklearn.model_selection. The parameter test_size with value 0.2 specifies that the 20% of the data be split as test set and the rest is used for training purpose.
Post this operation, both train and test sets have 20% lesser instances compared to the given train and test sets. Although the accuracy remained at 82%, other metrics such as precision, recall and f1-score are reduced by 5%, 3% and 3% respectively. Hence, it can be concluded that the given model performed better as it had more data to train on compared to the second model.

Question 2 : Error analysis 1: False positives
Performing error analyses is a key part of improving your NLP applications. For the 5 classes which have the lowest precision, according to the results table from your 20% development data, print out all the sentences where there is a false positive for that class (i.e. the label is predicted in the predicted label for a given word by the tagger, but this is not present in the corresponding ground truth label for that word). HINT: This may most easily be achieved by editing the code above beginning print("testing tagger...") and ending print("done"). Have a look at these errors closely, and think about which features could be added to reduce the number of these errors.

Answer 2
False positives were highest for I-Opinion among the five classes examined. They mostly came from I-Origin and I-Award apart from the unknowns on the words such as ‘best’, ‘films’, ‘writing’ – which is understandable since they were trained on both I-Opinion and I-Award, the classifier finds them relevant to both classes. The results for the other classes are also on the similar lines. To improve this, Bigram or an appropriate n-gram feature could be used to help differentiate the semantic of the word.

Question 3 : Error analysis 2: False negatives
For the 5 classes which have the lowest recall, according to the results table from your 20% development data,, print out all the sentences where there is a false negative for that label (i.e. the label is present in the ground truth label for a given word, but that label is not predicted for that word by the tagger). HINT: This may most easily be achieved by editing the code above beginning print("testing tagger...") and ending print("done"). Have a look at these errors closely, and think about which features could be added to reduce the number of these errors.

Answer 3
False negatives were highest for B-Character_Name among the 5 classes examined. They were mostly names of people that could have easily belonged to I-Plot or B-plot. The semantic of the word was not clearly understood by the classifier due to the unigram feature. Class B-Plot was dominated by articles such as ‘a’, ‘the’. Most articles could easily belong to any of the classes. Also, the sentences in I-Plot and B-Plot are very similar in context. This could have misled the classifier. More

strict distinction between the context of I-Plot and B-Plot could reduce the error count. Including n-grams with an appropriate n will help overcome limits of unigram approach for most classes.

Question 4 : Incorporating POS tags as features (15 marks)
Use the CRF part-of-speech (POS) tagger as shown below to add POS tags to the words in the training data. Do this by altering the preProcess function above. Note the CRF tagger only takes strings as input so you will have to concatenate the word and POS tag together (with a special symbol, e.g. @), and you will also have to then split on this special symbol in the feature extraction function get_features to get the word and POS tag - modify that function so it uses the POS tag in addition to the word (currently using the word only is achieved by feature_list.append("WORD_" + token). Re-run the training and testing code on your 80%/20% training/dev split from question 1 and record the results from the classification report as text in this file for comparison of the accuracy metrics against not using POS tags- try to see any improvemements across the classes.

Answer 4
Using the given 'posttagger', each word in a sentence can be tagged with an appropriate Parts of Speech (POS). '.join' function will concatenate the word and its POS tag so it can be used with a CRF tagger. Without using the POS tag as a feature, the accuracy of the model was at 82%, averages of precision, recall and f1-score at 61%, 51% and 55% respectively.
By adding the POS tag as one of the features (done by splitting the concatenation and appending '*"WTAG_" + tag'* to the feature list), the precision, recall and f1-score improved by 1%, 2% and 1% respectively.
By indicating the parts of speech for each word, the model is able to associate each word with its POS, thereby making better predictions than before.
POS tagging has increased the count of true positives and reduced false positives overall. False negatives have also reduced for most classes with the exception of class I-Plot - it has higher false negatives than the before, when POS tags were not used. Upon examination, most words belonging to I-plot have ambiguity of whether they must be a verb or a noun. POS tagging technique has limitations in terms of deciding the semantic of a word. For example, the word shot in sentence "That is a good shot" have different meanings – it could be a verb or a noun. Such words create ambiguity for a POS tagger leading to poor classification. Again, a n-gram model with an appropriate n will be able to provide some clarity on the context and reduce ambiguity.

Question 5 : Feature experimentation and other optimization for optimal macro average
Experiment with different features by further adjusting the get_features function, and modifying it to get the best results in terms of macro average f-score (i.e. average f-score across all classes) on your 20% development data. Iteratively try different functions, briefly describe the method and record the results in the classification report format and make sure you describe this in your report.
Feature tips: You could try more suffixes/prefixes of the current word than those currently extracted, you could use windows of the next and previous tokens (of different sizes, e.g. the previous/next N words/tags). As you try different feature functions, use the techniques you used in Q2 and Q3 to see the kind of errors you are getting for lower performing classes, in addition to the confusion matrix over classes.
Model tips: After your feature engineering, for even better performance you could try optimizing the hyperparameters by adjusting the training_opt dictionary values, experimenting with changing the minimum document frequency, c1 and c2 regularization parameters for https://www.nltk.org/api/nltk.tag.crf.html#nltk.tag.crf.CRFTagger.__init__ e.g. changing the initialization to make the minimum feature frequency 2 (rather than the default 1) would be:
ct = CRFTagger(feature_func=get_features, training_opt={"feature.minfreq":2})

Leave the get_features functions in the state you used to get the highest macro average f-score on your 20% development set, then re-train the model on ALL the training data and print the classification report for the original test data (i.e. from the test file trivia10k13train.bio.txt) as your final piece of code.

Answer 5

Adding POS tags as a feature provided a boost to the performance. Further experimentation with the below features already given in the assignment:

1. Capitalization
2. Punctuation
3. Number
4. Suffix up to length 3
5. The word
6. POS tag of the word

revealed that features such as suffix and number were more important compared to punctuation and capitalization. The accuracy of the model without suffix as a feature dropped to 40% while it remains at 80% when punctuation and capitalization were removed. This is mostly because the given training set has most words in lowercase and contains very less punctuation. This behaviour will change when the dataset changes.

To boost the performance of the model, a few more features were considered

1. Start of the sentence
2. End of the sentence
3. Prefix of lengths up to 2, 3 and 4
4. Suffix of lengths up to 3 and 4
5. Previous word
6. POS tag of previous word
7. Next word
8. POS tag of next word

Multiple combinations of these features were used in experimenting with the model to arrive at the best performance. Values in the range of 0 to 0.3 for regularization parameters were also considered. All of these experiments gave accuracies in the range of 80-85%. Some of the results stored in text files are attached with this report and notebook.

With just 3 features - suffix up to length 3, previous and next word, model's accuracy was at 84% and its f1-score was at 62%. This clearly indicates that these 3 features are among the most significant for the model. It also makes sense that these features put together provide strong context of a word thereby reducing ambiguity. Examining the confusion matrix – overall is a good improvement. Class I-plot however, had very high false positives and false negatives; highest confusion with B-Plot. High similarity in the context of the sentences in I-Plot and B-Plot could be the cause for this confusion.

Next, adding all 14 of the above-mentioned features (prefix and suffix up to lengths 4) and setting regularization parameters to 0.2 and 0.01 the model took extremely long time to train. This could have been due to the high dimensionality offered by the large set of features. The regularization parameters needed to be reset to reduce the dimensionality. To optimise the train time, punctuation feature (that proved to be less significant earlier) was removed and regularization parameters were set to 0.25 and 0.3. The model accuracy went up to 85% and the average f1-score was 65%. The POS tags of previous word and next word along with the prefixes may have helped this bump.

After a few combinations of various prefix and suffix lengths, the below feature set

1. The word

2. POS tag of the word
3. Start of the sentence
4. End of the sentence
5. Capitalization
6. Number
7. Prefix of lengths up to 3
8. Suffix of lengths up to 3
9. Previous word
10. POS tag of previous word
11. Next word
12. POS tag of next word

and regularization parameters c1=0.2 and c2=0.1 provided the highest average f1-score of 66%. With accuracy at 84%, to avoid overfitting, no further modifications to these settings were made.
Examining the confusion matrix, the true positives have increased compared to the previous iteration. However, the confusion between the classes I-Plot and B-Plot remains.

Finally, the model was trained with above 12 feature set with c1=0.2 and c2=0.1 and the entire training set as given. It was accurate to 84%, with an f1-score average 64%. The behaviour with the classes I-Plot and B-Blot seem to be similar on this set as well. Maybe creating a new class for the words that are misclassified most often could help eliminate the confusion. Alternatively, the construct of what constitutes I-Plot and B-Plot could be examined, and rules can be incorporated to make this distinction clearer.