ECS763P/U NLP Assignment 1: Fake News Detection

Introduction:

In this assignment, we have created a model which can make predictions on the fake_news dataset to detect fake news. In steps 1-3, we implemented a LinearSVC model with basic pre-processing and tested the performance of the model using K-fold testing. In steps, 5 and 6 we implement various techniques to improve the performance of our model.

1. **Simple data input and pre-processing:**

a. The ***parse_data_line*** function takes the data line list which is a list of all the column values in a single row. The desired column values– labels (class) and statements (feature) are selected from the line. The label, which is a degree of fakeness, is converted to a binary decision – REAL or FAKE. The converted label and text are returned as a tuple.

b. The ***pre_process***ing simply splits a text into a list of tokens using .split().

2. **Simple feature extraction:**

a. The ***to_feature_vector*** function takes a pre-processed text (list of tokens/words in this case) and creates a dictionary of words(keys) and their weight(values). The weight is 1 for each word occurring in the list. A global dictionary is created to keep track of all the words in the statement corpus.

3. **Cross-validation on training data:**

a. The ***cross_validate*** function divides the 80% training set obtained from the split function and further splits it into 10-fold – 1 Validation data, 9 train data. The classifier is then trained on train data and predictions are made on validation data of each fold.

b. A classification report is generated for each fold which includes precision, recall, f-score, and accuracy. The average of all these values is calculated and stored in list results_avg.

c. The average result of all the folds was found to be:
Avg Precision: 0.57, Avg Recall: 0.57,
Avg F-score: 0.57, Avg Accuracy: 0.57

4. **Error Analysis:**

a. In this section, we carry out analysis on all the falsely predicted labels by generating a heatmap which gives the count of all TP, FP, FN, and TN and printing all the False Positives(Pred-FAKE, Truth-REAL) and False Negatives(Pred-REAL, Truth-FAKE) for the FAKE labels in files FP.txt and FN.txt respectively.

b. **Observations:**
1. We can see that there are a lot of words in their various forms e.g., ‘lives’, ‘live’, ‘living’. This can lead to a large feature set and can cause the model to overfit. This can be tackled with the help of various pre-processing techniques.
2. The tokens used have no context present to them. E.g., ‘lies on the floor’ and ‘lies to people’ have 2 completely different meanings but both ‘lies’ if taken in unigram will be interpreted equally. This can be improved by using a higher sequence of tokens that can give each feature contextual meaning.
3. It can be observed in the dataset that the total number of REAL labels is much more than the total number of FAKE labels. This adds more weight to the REAL label. Which can be observed in the heatmap.

5. **Optimising pre-processing and feature extraction.**

Experiment 1: Improving pre-processing
One of the initial important steps of NLP is pre-processing. This is to make sure we have good-quality data. There are different methods of pre-processing a feature text before using it for training a model or making predictions. The below points discuss various pre-processing techniques used in the ***pre_process*** function that has shown to have a positive or negative effect on the model performance:

I. **Remove Punctuation**:
The punctuation removal process helps in identifying each word equally. For e.g., ‘wonder’ and ‘wonder!’ are identified as the same after the removal of punctuation (!). Although in this scenario, this process affected the overall accuracy negatively. Hence not included as punctuations are

important for giving meaning to a sequence.

II. **Conversion to lowercase**:
In this technique, we convert all the words in the token list 'tokens' to lowercase form. This also is used to treat each word equally without being affected by capitalisation.

III. **Stop word removal**:
Stop word removal is a process of eliminating all prepositions/very frequent functional words. By removing these words, we remove the low-level information from our text to give more focus to the important information.

IV. **Lemmatising:**
Lemmatization considers the context and converts the word to its meaningful base form. E.g., started, starting → start.

All the above techniques and some more when performed individually had a negative to no significant effect on the performance but together decreased the average performance of the classifier by 0.01 but looking at the confusion matrix heatmap the count of FAKE prediction increased slightly as compared to just splitting the text into tokens.

Experiment 2: N-gram tokens
In this technique, we take an N sequence of words and count how often these sequences occur in the texts. This way we can add contextual information to a word which in turn would increase the quality of the information received by the classifier.
In Q2, we calculated the occurrence of a unigram (single word). To further increase the accuracy, we tried using bi-gram and trigram sequences. The following was observed-
Bi-gram (2 words) sequence increased the average performance by 0.01.
Tri-gram sequence had a positive effect on average Recall and average accuracy.

Experiment 3: Stylistic features - Number of words per sentence.
To observe patterns in our texts we can use a stylistic feature like the number of words in each statement and add the result in the feature dictionary as an additional feature.
After performing the above experiments, we observed that the count of 'REAL' prediction is much higher than the overall count of 'FAKE' prediction i.e., the number of FNs is much higher than FPs. For e.g., in the 1st fold, the count of FN is 217 whereas for FP it is 118.

Experiment 4: SVM parameters: **class-weight = {0:0.8,1:0.2}**
As observed in Q4, we have more REAL records in the database than FAKE. When it comes to an unbalanced dataset, we can use the SVC parameter 'class_weight'. Since class 'REAL' has more weightage than class 'FAKE', we need to add more weightage to class 'FAKE'.
This parameter did increase the 'FAKE' pred count and had a positive effect on the average precision and the F-score.

Average performance in Q5:
Avg Precision: 0.58, Avg Recall: 0.58, Avg F-score: 0.58, Avg Accuracy: 0.58

6. **Using other metadata in the file:**
So far, we have just included 1 column ('Statement') from the dataset as a feature. There are other useful columns in the dataset which can help improve classifier performance as it provides more information in the feature set about different aspects.
Below are some of the other columns used along with the statement feature-
Observations:
After analysing and trying multiple combinations, the columns 'speaker', 'speaker_job_title' and 'state_info' along with 'Statement' gave the overall best result of 61%

Experiments	**Avg. Precision**	**Avg. Recall**	**Avg. F-score**	**Avg. Accuracy**
Preprocessing	0.56	0.56	0.56	0.56
2-gram tokens	0.57	0.57	0.57	0.57
3-gram tokens	0.57	0.58	0.56	0.58
Stylistic Feature	0.57	0.58	0.56	0.58
SVM	0.58	0.58	0.58	0.58
+ Speaker	0.6	0.61	0.6	0.61
+col[5]+col[6]+col[7]	0.61	0.61	0.61	0.61

Conclusion: After performing all of the techniques mentioned above the final best performance of the model against unseen test data was around 60%.