READme

Folder descriptions

1. data:
   1. Contain given data(test and train files)
   2. idx2Label.pickle, idx2Word.pickle, label2Idx.pickle and word2Idx.pickle are stored dictionaries that I used for converting words and data labels to indices in my deep learning approach.
   3. sentences.npy and tags.npy are NumPy arrays storing the sentences and BIO tags for each word in every sentence as obtained from the file train.txt
   4. processed\_sents.npy are cleaned and pre-processed obtained from sentences.npy
   5. pos.npy and processed\_tags.npy are the parts of speech (POS) and BIO tags for processed\_sents.npy
   6. processed\_sents\_dl.npy and processed\_tags\_dl.npy are the pre-processed and cleaned sentences and tags I used for the deep learning approach
2. embeddings: Folder storing the Glove embeddings I used in the deep learning approach
3. models: Various models I trained for the task
4. results: Final predictions on the test data
5. scripts:
   1. EDA.ipynb contains all the EDA code
   2. pre-processing.ipynb contains code for pre-processing and cleaning of data
   3. ensemble-learning.ipynb contains code for the use of ensemble learning for the task.
   4. knn-randomforest.ipynb contains code for the use of KNN and random forests for the task.
   5. traditional-ml.ipynb contains code for the use of Perceptron, Passive Aggressive classifier, SGD classifier and Multinomial Naive Bayes for the task.
   6. lstm-approach.ipynb contains code for the use of Bi-LSTM for the task.
   7. lstm-testdata-predictions.ipynb contains code for the use of Bi-LSTM for making predictions on the test data

My approach:

1. I first performed EDA to analyze the data. (Refer to file EDA.ipynb).
2. I plotted the number of samples that belonged to each class and noticed that I was dealing with class imbalance. The class ‘O’ heavily dominated each class.
3. I plotted the word cloud for each class. This gave me a good idea of the most frequent words in every class. Due to this I got the idea of using KNN as one of the models. Why so is explained later.
4. I also noticed in the training data file; each sentence was separated by a blank line. I used this information to spilt the data into sentences and save them in the NumPy array sentences.npy. The NumPy array tags.npy contains the BIO tags per sentence.
5. Then I decided to pre-process and clean the data. (Refer to file pre-processing.ipynb).
6. Load in the NumPy arrays sentences.npy and tags.npy. Then clean the sentences by eliminating URLs, Twitter @s and non-alphabetic characters.
7. I also extract the Parts of speech (POS) features for every word in every sentence here. Since for machine learning models, the number of POS for a sentence should be equal to the number of words in that sentence, skip the sentences where this condition is not satisfied. Save these processed arrays as processed\_sents.npy, pos.npy and processed\_tags.npy
8. For my deep learning approach, I did not use POS, so for that I can use all sentences. These are stored as processed\_sents\_dl.npy and processed\_tags\_dl.npy.
9. After that I tried some traditional ML approaches like Perceptron, Passive Aggressive classifier, SGD classifier and Multinomial Naive Bayes (Refer to traditional-ml.ipynb) (Results discussed later). Since these approaches require rows of data to be in the format (POS, word, BIO tag), I flatten the POS, sentence, and tags NumPy arrays obtained from pre-processing and then feed them to these models
10. After that I tried using KNN and random forests (Refer to knn-randomforest.ipynb). From the word cloud I had the intuition that there are certain words and words like them that belong to a class. So, if I could represent the words as embeddings and the use those embeddings for KNN or random forests I might get a good result. I used Glove embeddings to represent the words. I used the NumPy array processed\_sents\_dl.npy, processed each sentence word by word and represented it by its embedding. Then I fed these words and their labels to my models. (Results discussed later)
11. I also tried ensemble learning for this task (Refer to ensemble-learning.ipynb). As above I used the word embeddings. I used KNN, random forest, Perceptron and Passive aggressive classifier as my ensemble models. I tried ensemble learning to mitigate the class imbalance problem.
12. Finally, I tried a Bi-LSTM deep learning approach (Refer to lstm-approach.ipynb for the steos I took).
13. Since the Bi-LSTM gave the best results I use it to make predictions in lstm-testdata-predictions.ipynb

Order of execution:

1. EDA.ipynb
2. pre-processing.ipynb
3. traditional-ml.ipynb
4. knn-randomforest.ipynb
5. ensemble-learning.ipynb
6. lstm-approach.ipynb
7. lstm-testdata-predictions.ipynb

Metric:

For my metric using accuracy is not a good idea since we have class imbalance problem. Using Precision, Recall and F1 score is a good idea. However, with respect to F1 score too, I calculated the F1 score for all classes except ‘O’. This helped me find me best model accurately and is what I used as my metric for evaluation of each model. If I include ‘O’ in calculating the F1 score, it skews the results, and every model looks good due to the large number of samples of ‘O’.

Results:

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Precision | Recall | F1 score |
| SGD classifier | 0.69 | 0.07 | 0.11 |
| Perceptron | 0.55 | 0.27 | 0.35 |
| Multinomial NB | 0.45 | 0.31 | 0.3 |
| Passive Aggressive Classifier | 0.49 | 0.27 | 0.34 |
| KNN | 0.48 | 0.19 | 0.27 |
| Random forest | 0.54 | 0.25 | 0.34 |
| Ensemble | 0.57 | 0.15 | 0.22 |
| Bi-LSTM | **0.52** | **0.47** | **0.49** |

**That is why I think using a Bi-LSTM with word embeddings is the best approach for this problem**

Future work:

If I had more time and resources here are the things I would do:

1. Collect more data on the minority classes.
2. Heavy penalty the model for misclassification of minority classes. Implement this in the loss function.