Project 1 Report

Language Identification with sklearn

This project is about language identification using sklearn.
We have build a NLP pipline to achieve the task using linear classification (Part1), and Multilayer Perceptron (Part2).
In a first step, the most important distributional properties of the dataset are described.
The dataset contains a total of 65954 samples split into 52675 training samples and 13279 test samples.
Every sample comprises a “tweet” column with the textual content and a “label” column with the associated language.
The histogram shows that there are many labels with very low frequency (strong class imbalance).
For example 50 labels occur less than 20 times in the whole dataset.
However, English occurs more than 20,000 times.
We are dealing with this class imbalance in a later step.

Part 1

We merged the given training and test data into one dataset, so we had to write less code for the pre-processing.
At first we cleaned the data by removing emojis, twitter @ mentions, numeric patterns etc..
Since the tweet itself is the only featuretext corpus that we are given, we thought about possible features that we could engineer
and that would help us with classifying the language a tweet is written in.

- **Alphabet**: the detected alphabet in use

- **Character Count**: the number of characters in each text observation
- **Word Count**: the number of words in each text observation
- **Character Density**: the average characters per word. Divides the word_count by the character_count and creates an average relative relation between the latter two variables. This feature standardizes the length of a word based on how many words and characters there are in a text input.

- **Punc Count**: counts the number of spacing in a text input. Some languages may use more punctuations.

- **Num Consec Vowels**: counts the number of consecutive vowels in a text.

- **Encode Ascii**: checks whether a text can be encoded by utf-8.

We tried these features however, they did not increase our test accuracy in the end and therefore we removed them again.

We found that the imbalanced class problem is badly influencing the accuracy result. In order to compensate for class imbalance, we used back translation of underrepresented languages and upsamling to augment those data.
The dataset is split into train and test dataset using a 90:10 split.
TfidfVectorizer was used in order to tokenize the Tweets. The reason why we chose the tf-idf vectorizer instead of the Count Vectorizer is that tf-idf vectorizer produces a more normalized representation of word. Compared to Count Vectorizer, the tf-itf way gives more weights to the rare word or the unique prefix or suffix. GridSearchCV (5-fold cross validation) is used in order to find the best combination of hyperparameters for the tokenizer and the model.
The following parameters were passed to GridSearchCV.
We did not pass word as a possible analyzer to the TfidfTokenizer since we have languages in the dataset which do not use spaces as word separation.
For the SGDClassifier the following parameter were passed to GridSearchCV.

- **TfidfTokenizer > ngram_range** : (1,3), (1,4)
- **TfidfTokenizer > analyzer**: char
- **SGDClassifier > alpha**: 1e-4, 1e-5, 1e-6
- **SGDClassifier > loss**: perceptron, log, hinge
- **SGDClassifier > penalty**: elasticnet
- **SGDClassifier > class_weight**: balanced
- **SGDClassifier > early_stopping**: True, False

GridSearchCV yields us with the following best hyperparameter combination

- **TfidfTokenizer > ngram_range**: (1, 4)
- **TfidfTokenizer > analyzer**: char
- **SGDClassifier > alpha**: 1e-06
- **SGDClassifier > loss**: log
- **SGDClassifier > penalty**: elasticnet
- **SGDClassifier > class_weight**: balanced
- **SGDClassifier > early_stopping**: False

In the end we received a test accuracy of 93%
with a micro/macro f1 of both 93%.

For the MultinomialNB classifier the following parameter were passed to GridSearchCV.

- **TfidfTokenizer > ngram_range** : (1,2), (1,3), (1,4)
- **TfidfTokenizer > analyzer**: char
- **MultinomialNB > alpha**: 1, 0.8,0.6, 0.4, 0.2, 0

GridSearchCV yielded us with the following best hyperparameter combination

- **TfidfTokenizer > ngram_range** : (1,4)
- **TfidfTokenizer > analyzer**: char
- **MultinomialNB > alpha**: 0

In the end we received a test accuracy of 86%
with a micro f1 84% and macro f1 of 86%.

Part 2

In the second part we were supposed to build a Multilayer Perceptron model.
We did not do the data cleaning and preprocessing again.
We just load the augmented dataset that we saved (dataset.csv) during the first part of the exercise.
We then tried to train a MLPClassifier using GridSearchCV, however after running for 10 hours our runntime disconnected
and we could not retrieve any results from that. Therefore, we decided to run 12 different hyperparamter combinations individually and compare the classification reports
with each other. The following hyperparamters combinations were tried out

- **MLPClassifier > hidden_layer_sizes**: 100, 200, 500
- **MLPClassifier > solver**: adam, sgd
- **MLPClassifier > activation**: tanh, relu

The weighted average accuracies under different configurations are respectively as follows: - **hidden_layer_sizes > 100, solver > adam, activatio > tanh**: 92% - **hidden_layer_sizes > 100, solver > adam, activatio > relu**: 92% - **hidden_layer_sizes > 100, solver > sgd, activatio > tanh**: 72% - **hidden_layer_sizes > 100, solver > sgd, activatio > relu**: timeout - **hidden_layer_sizes > 200, solver > adam, activatio > tanh**: 92% - **hidden_layer_sizes > 200, solver > adam, activatio > relu**: 91% - **hidden_layer_sizes > 200, solver > sgd, activatio > tanh**: timeout - **hidden_layer_sizes > 200, solver > sgd, activatio > relu**: timeout - **hidden_layer_sizes > 500,..**: timeout

In the end, running these 12 hyperparameter combinations also took over about 30 hours. Although we didn't manage to get the accuracy from the configurations with hidden_layer_size=500, because of the colab crash problems after long hours of trainning, we found that with 100 and 200 hidden_layer_sizes, we can already achieve quite well trainning outcome. While the model performed quite well when the Adam optimizer was used, it performed much worse when it was trained by SGD.

The best model that got trained with Adam received an accuracy of 92% and micro f1 of 92% and macro f1 of 89%. In contrast, The best model that got trained with SGD received an accuracy of 77% and micro f1 of 72% and macro f1 of 8%. This is much worse in comparison. Also, between 100 and 200 hidden_layer_sizes, the results do not differ too much, both achieving similar level of 92% in accuracy.