AGR9012 - Image and Text Processing for Data Science

Sebnem Keles
27435249

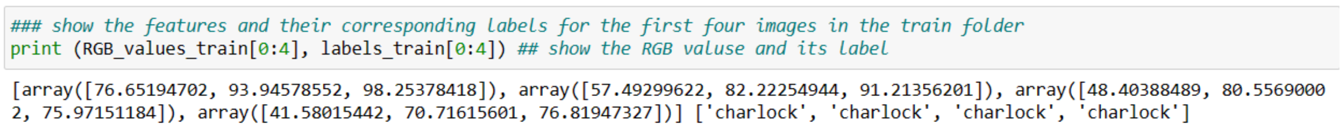
UNIVERSITY OF
LINCOLN

Task 1: Image Processing

Preprocessing:

Initially, I attempted to preprocess the images using grayscale to make my image data simpler, histogram equalization to adjusts the contrast, image denoising to eliminate undesirable noise from images and Canny Detector to highlight the edges. All of these procedures are meant to enhance image quality and facilitate more effective analysis. After that, in order to minimize data and expedite the machine learning process, I carried out feature extraction using mean value. I applied each of these processes on my test, validation and train datasets.

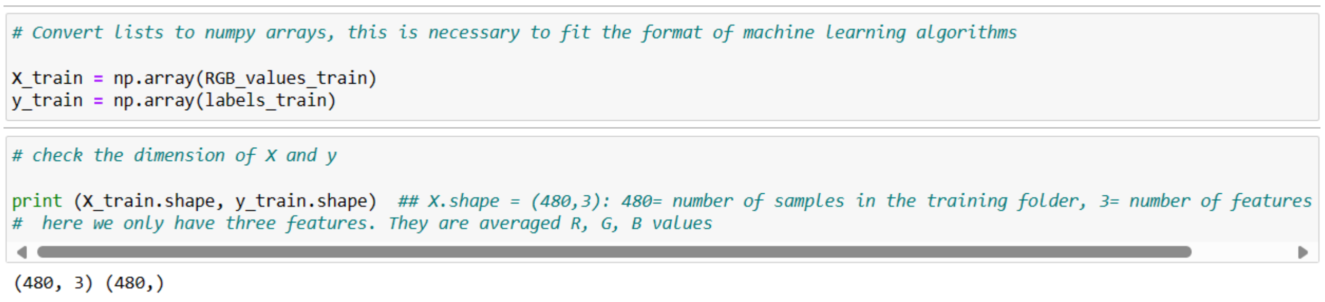
Firstly, using RGB_values_train, the average RGB values of each image are first observed as a sequence. '[0:4]' is then used to ensure that only the first four images display these values, and label_train is used to allow the first four images' label to be displayed.



```
### show the features and their corresponding labels for the first four images in the train folder
print (RGB_values_train[0:4], labels_train[0:4]) ## show the RGB valuse and its label
```

```
[array([76.65194702, 93.94578552, 98.25378418]), array([57.49299622, 82.22254944, 91.21356201]), array([48.40388489, 80.5569000
2, 75.97151184]), array([41.58015442, 70.71615601, 76.81947327])] ['charlock', 'charlock', 'charlock', 'charlock']
```

Then, arranged the data for training with separated into X and Y and monitor the trainings' dimensions. As mentioned below, 480 indicates the number of samples, while 3 indicates the number of features.

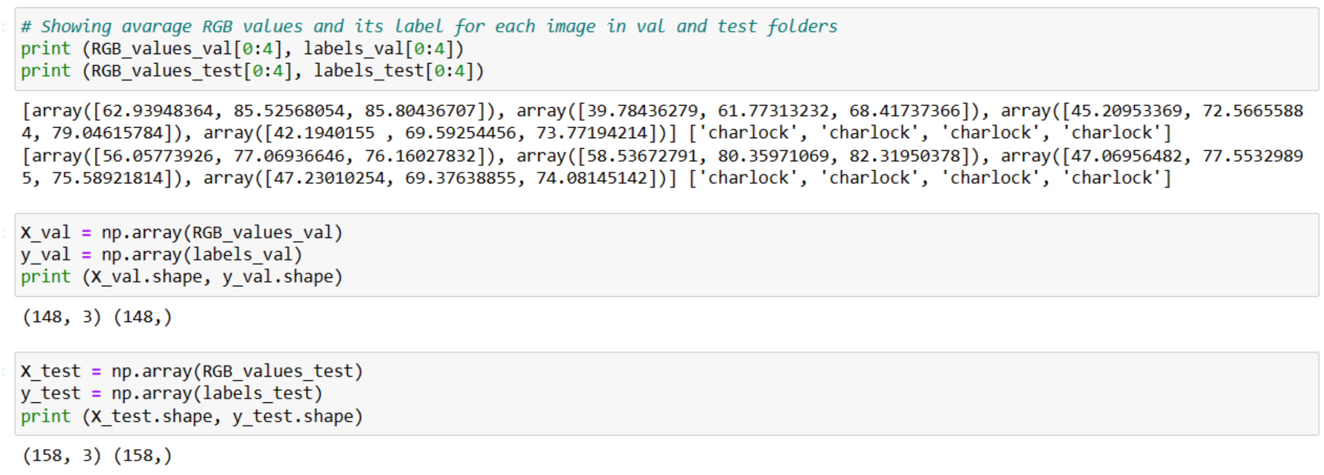


```
# Convert lists to numpy arrays, this is necessary to fit the format of machine learning algorithms

X_train = np.array(RGB_values_train)
y_train = np.array(labels_train)
```

```
# check the dimension of X and y

print (X_train.shape, y_train.shape)  ## X.shape = (480,3): 480= number of samples in the training folder, 3= number of features
#  here we only have three features. They are averaged R, G, B values
```

```
(480, 3) (480,)
```

I have finished the preprocessing step after following the same procedure for our validation and test datasets as well and now ready to build a machine learning model.



```
# Showing avarage RGB values and its label for each image in val and test folders
print (RGB_values_val[0:4], labels_val[0:4])
print (RGB_values_test[0:4], labels_test[0:4])
```

```
[array([62.93948364, 85.52568054, 85.80436707]), array([39.78436279, 61.77313232, 68.41737366]), array([45.20953369, 72.5665588
4, 79.04615784]), array([42.1940155 , 69.59254456, 73.77194214])] ['charlock', 'charlock', 'charlock', 'charlock']
[array([56.05773926, 77.06936646, 76.16027832]), array([58.53672791, 80.35971069, 82.31950378]), array([47.06956482, 77.5532989
5, 75.58921814]), array([47.23010254, 69.37638855, 74.08145142])] ['charlock', 'charlock', 'charlock', 'charlock']
```

```
X_val = np.array(RGB_values_val)
y_val = np.array(labels_val)
print (X_val.shape, y_val.shape)
```

```
(148, 3) (148,)
```

```
X_test = np.array(RGB_values_test)
y_test = np.array(labels_test)
print (X_test.shape, y_test.shape)
```

```
(158, 3) (158,)
```

Modelling:

In addition to Logistic Regression, I also examined the results of Gaussian Naive Bayes and K-Nearest Neighbours models for comparison. As seen below, the GNB training achieved a score of 60%, whereas the KNeighbors model attained an 85% score.

```
# Gaussian Naive Bayes and the score of the model [2]
from sklearn.naive_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X_train, y_train)

gnb.score(X_train, y_train)

print('Training Score:', gnb.score(X_train, y_train))
```

```
Training Score: 0.6
```

```
# KNeigbours Classifiers and the score of the model [2]
from sklearn.neighbors import KNeighborsClassifier

kn = KNeighborsClassifier()

kn.fit(X_train, y_train)

kn.score(X_train, y_train)

print('Training Score:', kn.score(X_train, y_train))
```

```
Training Score: 0.85625
```

After evaluating the performance of various machine learning methods on the train dataset, I examined the accuracy rates of these models to determine which one yielded superior prediction results for test dataset. The accuracy rates for Logistic Regression, Gaussian Naive Bayes, and K-Nearest Neighbors are approximately 82%, 56%, and 76%, respectively.

```
# Checking the accuracy for each model(Logistic Regression, Decision Tree and Random Forest respectively)
from sklearn.metrics import accuracy_score

y_pred_clr = clf.predict(X_test)
y_pred_gnb = gnb.predict(X_test)
y_pred_kn = kn.predict(X_test)
accuracy_clr = accuracy_score(y_test, y_pred_clr)
accuracy_gnb = accuracy_score(y_test, y_pred_gnb)
accuracy_kn = accuracy_score(y_test, y_pred_kn)
print('Accuracy for Logistic Regression :', accuracy_clr)
print('Accuracy for Gaussian Naive Bayes:', accuracy_gnb)
print('Accuracy for KNeighbors Classifier:', accuracy_kn)
```

```
Accuracy for Logistic Regression : 0.8227848101265823
Accuracy for Gaussian Naive Bayes: 0.5632911392405063
Accuracy for KNeighbors Classifier: 0.7658227848101266
```

Finally, afterward diligently fine-tuning the hyperparameters on the validation dataset using Support Vector Machine and GridSearchCV, I achieved best accuracy score of approximately 72% and a test accuracy of around 82%, alongside identifying the best hyperparameters. As Wong et al. (2019) mentioned 'Tuning the hyperparameters of individual algorithms in a super learner may help optimize performance.' So, we expect to have better result for Machine Learning models after tuning hyperparameters according to the best params which is '4.0' in this case.

```python
# Tune hyperparameters of the model [3]
from sklearn.svm import LinearSVC
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(LinearSVC(), {'C': [1.0, 2.0, 4.0, 8.0]})
grid.fit(X_val, y_val)


print("Best Hyperparameters: ", grid.best_params_)
print("Best Accuracy Score: {:.2f}%".format(grid.best_score_ * 100))

best_svm = grid.best_estimator_
test_accuracy = best_svm.score(X_val, y_val)
print("Test Accuracy: {:.2f}%".format(test_accuracy * 100))
```

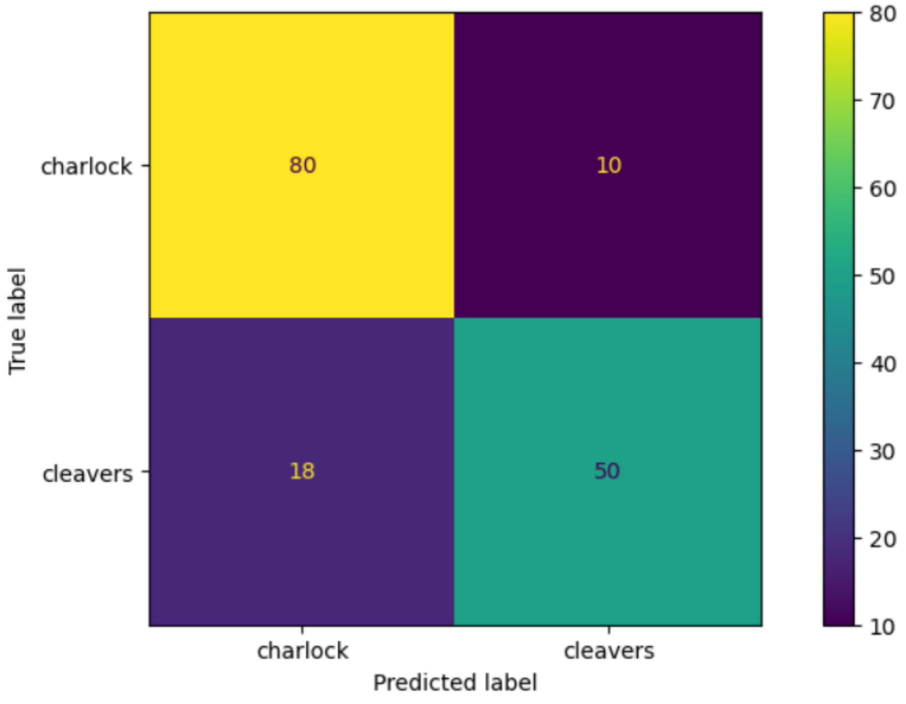
```
Best Hyperparameters:  {'C': 4.0}
Best Accuracy Score: 71.54%
Test Accuracy: 81.76%
```

After completing all prediction processes, I utilized the Confusion Matrix to examine the counts of true and predicted labels for both charlock and cleavers, thereby assessing the efficacy of the outcome. According to the matrix the number of True Positive instances produced by the model on the test data is 80 for charlock, while the number is 50 for cleavers. Moreover, upon observing the True Label and Predicted Label outcomes, it's evident that Charlock attained a noteworthy prediction accuracy of 89% (80/90 = 0.88...), whereas Cleaver's prediction accuracy stands at approximately 74% (50/68 = 0.735...) and overall score for Confusion Matrix is about %81.

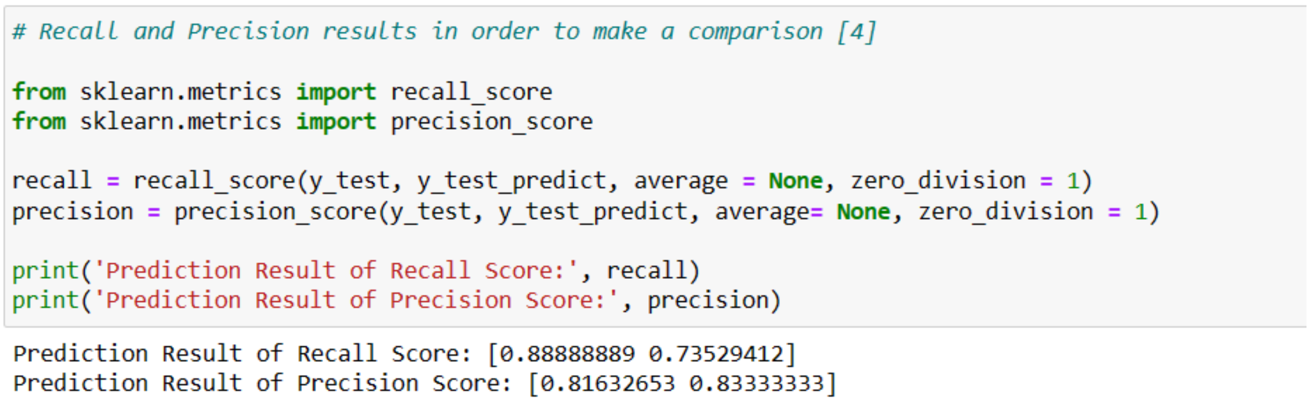
Evaluation:

```python
import matplotlib.pyplot as plt
from sklearn.metrics import ConfusionMatrixDisplay

fig, ax = plt.subplots(figsize=(10, 5))
ConfusionMatrixDisplay.from_predictions(y_test, y_test_predict, ax=ax)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x2ab8399c4f0>
```



In the following code, I wanted to check the recall and precision matrix scores to see if they differed from the confusion matrix. In the Recall Matrix, Charlock's correct prediction score was 0.888..., while Cleavers' correct prediction score was 0.735..., resulting in an overall score of approximately 81% in this matrix as well.



```
# Recall and Precision results in order to make a comparison [4]

from sklearn.metrics import recall_score
from sklearn.metrics import precision_score

recall = recall_score(y_test, y_test_predict, average = None, zero_division = 1)
precision = precision_score(y_test, y_test_predict, average= None, zero_division = 1)

print('Prediction Result of Recall Score:', recall)
print('Prediction Result of Precision Score:', precision)
```

```
Prediction Result of Recall Score: [0.88888889 0.73529412]
Prediction Result of Precision Score: [0.81632653 0.83333333]
```

Conclusion:

In conclusion, following the completion of my pre-processing procedures to prepare the images for machine learning methods, I evaluated them using three distinct machine learning models. The logistic regression model yielded the highest accuracy, achieving approximately 82%. The results from applying three different metrics showed no significant variance, and we can say that our prediction result is usable with an estimation rate of approximately 81%. However, employing alternative methods, models or preprocessing could potentially elevate this percentage further.

References:

Wong, Jenna, et al. "Can Hyperparameter Tuning Improve the Performance of a Super Learner?" *Epidemiology*, vol. 30, no. 4, July 2019, pp. 521–531, https://doi.org/10.1097/ede.0000000000001027. Accessed 13 July 2020.