Name: Prachi Sadarangani
Date: 10/06/2023

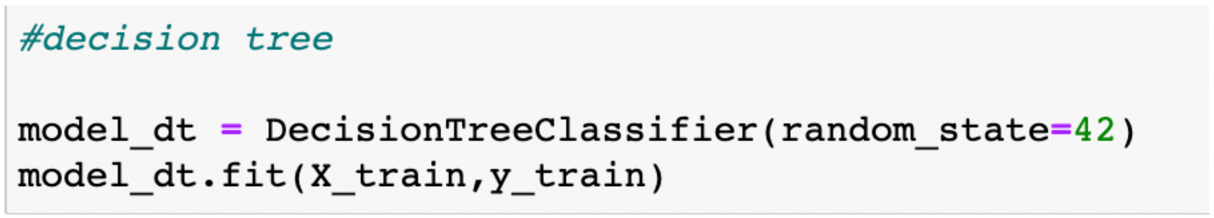
<u>Model Comparison for Handwritten Digits:</u>

The aim of this report is to compare two classification algorithms - the Decision Tree Classifier and Naïve Bayes - to find which model is better at predicting images of handwritten numbers from 0 to 9.

The dataset does not contain any null value and is divided into testing and training sets already. After loading the data, the training set is split into X and y where X contains the features (each of the pixel columns in this case) and y contains the target variable ('labels'). The same is done for the testing dataset. Once, the training and testing datasets are prepared, we begin with data modeling.

<u>Decision Tree:</u>

The first algorithm used is the Decision Tree Classifier. First, the training dataset is fit to a simple Decision Tree Classifier model without providing any type of tuning as shown below.



```
#decision tree

model_dt = DecisionTreeClassifier(random_state=42)
model_dt.fit(X_train,y_train)
```

Now, this model is tested using the testing dataset. The accuracy of the model is 78.32% which means 78.32% of the predictions in the test data are correct. The precision, recall, and f1 score of each of the labels and the weighted average are shown below.

```
              precision    recall  f1-score   support

           0       0.87      0.88      0.87       414
           1       0.90      0.94      0.92       478
           2       0.74      0.76      0.75       420
           3       0.72      0.72      0.72       446
           4       0.79      0.79      0.79       404
           5       0.70      0.71      0.71       388
           6       0.79      0.78      0.78       404
           7       0.85      0.84      0.85       465
           8       0.67      0.65      0.66       393
           9       0.75      0.73      0.74       386

    accuracy                           0.78      4198
   macro avg       0.78      0.78      0.78      4198
weighted avg       0.78      0.78      0.78      4198
```

The weighted average precision of the model is 78% which means that on average, when our model predicts a specific digit, it's correct about 78% of the time. However, in the above image, we can see that the precision is lowest for the digit ‘8’ which means every time the model predicted the number 8, only 67% of the time it predicted correctly. This could be because numbers like 3,6,9 are similarly shaped like 8 so they could have been incorrectly identified as 8. Likewise, we can see that the precision is the highest for the digit 1 which means that 90% of the time the model predicted a label as 1, it was actually 1. The weighted average recall for our model is 0.78 as well which means on average our model can find a specific digit correctly 78% of the time. The lowest recall is 65% which is for 8 which means out of every time 8 was present, the model predicted it correctly only 65% of the time. Additionally, for 1 it has the highest recall which is 94% which means out of every time 1 was in the image, it predicted it correctly 94% of the time. F1 score is the average of precision and recall, so in this case, it is also 78%.

Moreover, we have performed 3-fold cross-validation to see if the model is overfitting. The below image shows the results. As per the results, the model accuracy is about 78% and cross-validation scores are in the range of 73% to 75%. Since the accuracy is not significantly higher than the cross-validation scores, it means the model is only slightly overfitting.

```
cross-validation scores: [0.72857143 0.73766976 0.74839171]
```

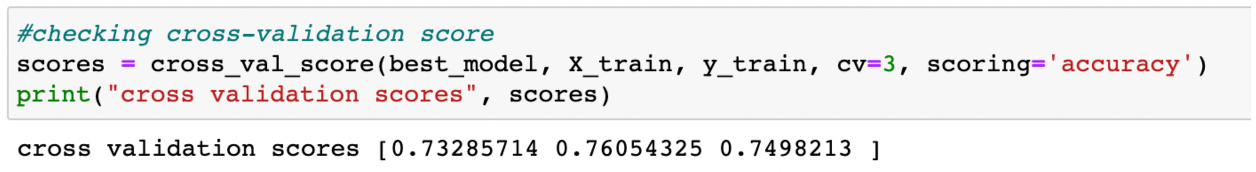
We have a decent model but now we will fine-tune it to see if it can get any better. To fine-tune our model, we will use the Randomized Search. The hyperparameter range that is used to fine-tune the model is shown below. We are also using the 3-fold cross-validation in our random search.

```
#fine-tuning
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV

param_dist = {
    'criterion': ['gini', 'entropy'],
    'max_depth': np.arange(10, 200, step=2),
    'min_samples_split': np.arange(2, 26),
    'min_samples_leaf': np.arange(1, 26)
}

random_search = RandomizedSearchCV(model_dt, param_distributions=param_dist,
                                   n_iter=100, cv=3, n_jobs=-1, verbose=2, random_state=42)
```

After running the above code with the above hyperparameter range, the randomized search finds the best hyperparameters to increase the model's accuracy. The below image shows the best hyperparameters

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
Best Hyperparameters:  {'min_samples_split': 5, 'min_samples_leaf': 4, 'max_depth': 118, 'criterion': 'entropy'}
```

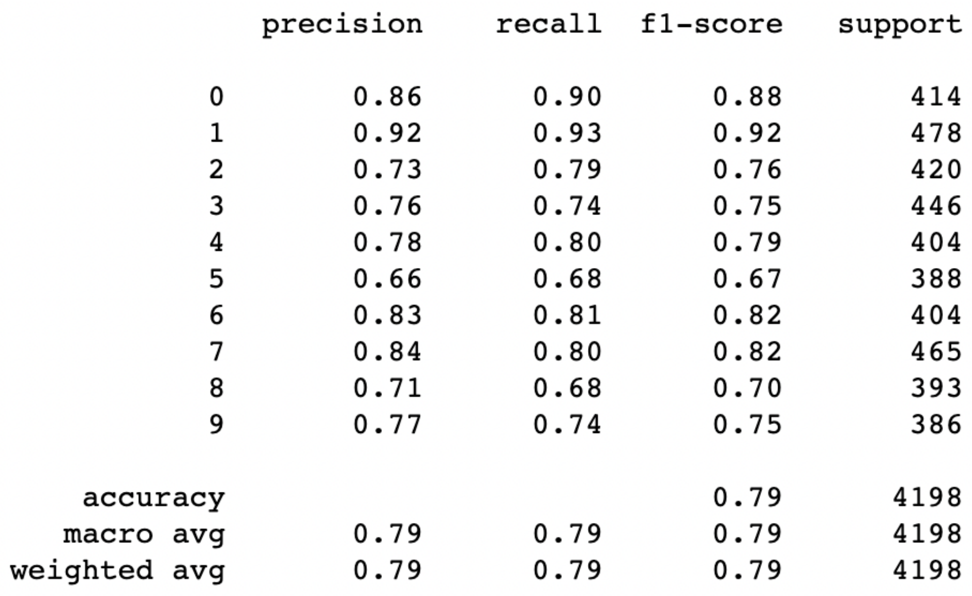
After fitting the above hyperparameters in the model, the accuracy of the model is 79.09%. The accuracy of the model has increased after fine-tuning it. The 3-fold cross-validation accuracy values are shown below. The average cross-validation score has also increased and since the values are now between 73% and 76%, this model is slightly overfitting.



```
#checking cross-validation score
scores = cross_val_score(best_model, X_train, y_train, cv=3, scoring='accuracy')
print("cross validation scores", scores)
```

```
cross validation scores [0.73285714 0.76054325 0.7498213 ]
```

The precision, recall, and F1 score are shown below.



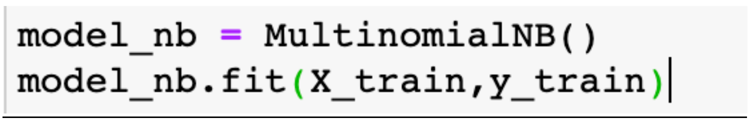
```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88       414
           1       0.92      0.93      0.92       478
           2       0.73      0.79      0.76       420
           3       0.76      0.74      0.75       446
           4       0.78      0.80      0.79       404
           5       0.66      0.68      0.67       388
           6       0.83      0.81      0.82       404
           7       0.84      0.80      0.82       465
           8       0.71      0.68      0.70       393
           9       0.77      0.74      0.75       386

    accuracy                           0.79      4198
   macro avg       0.79      0.79      0.79      4198
weighted avg       0.79      0.79      0.79      4198
```

From the above image, we can see that the precision, recall, and F1 score have increased as well. We can also see that the precision and recall for the digit 8 have also increased. However, this model does not predict the digit 5 as well as the previous model.

Since, this fine-tuned model has a higher accuracy, precision, recall, and F1 score, we will choose this model.

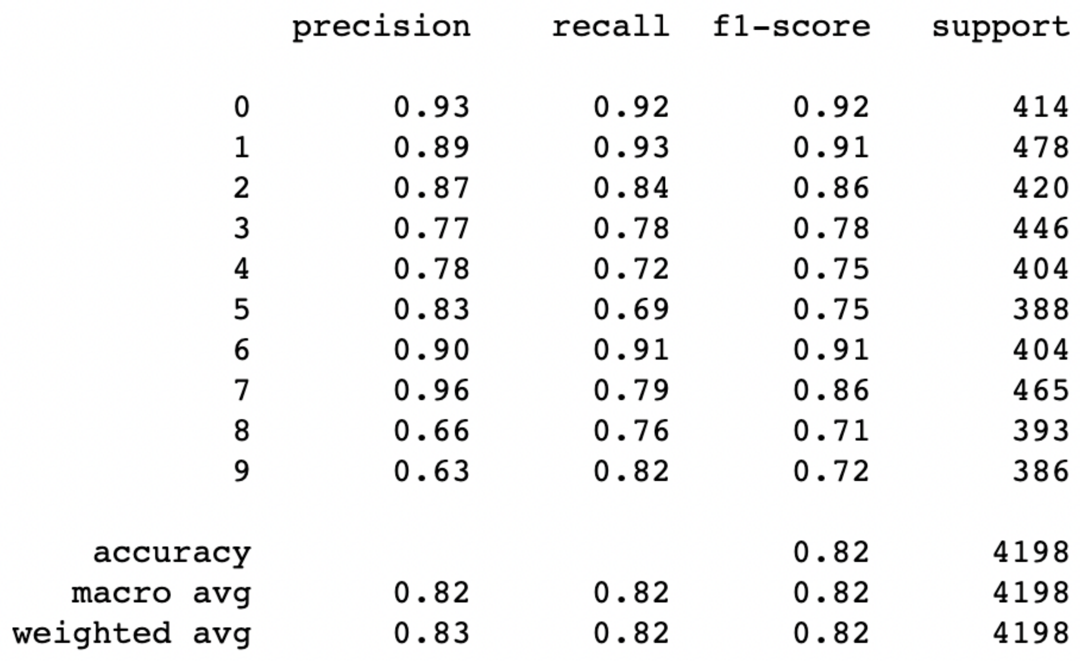
Naïve Bayes Model:

To build the Naïve Bayes Model, we are using the Multinomial Naïve Bayes as it is used for discrete data. The Gaussian Naïve Bayes is not being used because the data is not normalized. First, we use the simple Gaussian Naïve Bayes model as shown below.



```
model_nb = MultinomialNB()
model_nb.fit(X_train,y_train)
```

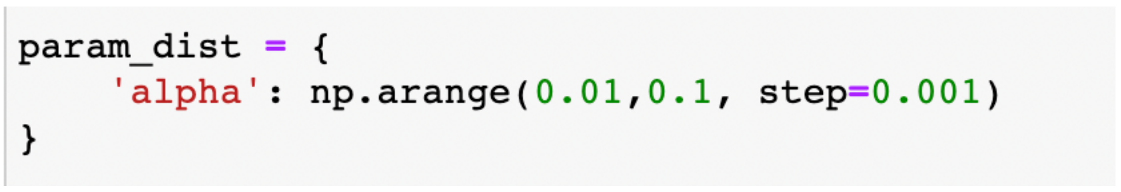
The accuracy of this model is 81.75%. The below image shows the precision, recall and F1 score for the model.



```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92       414
           1       0.89      0.93      0.91       478
           2       0.87      0.84      0.86       420
           3       0.77      0.78      0.78       446
           4       0.78      0.72      0.75       404
           5       0.83      0.69      0.75       388
           6       0.90      0.91      0.91       404
           7       0.96      0.79      0.86       465
           8       0.66      0.76      0.71       393
           9       0.63      0.82      0.72       386

    accuracy                           0.82      4198
   macro avg       0.82      0.82      0.82      4198
weighted avg       0.83      0.82      0.82      4198
```

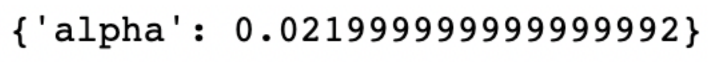
The precision, recall, and F1 score of the model is 82%. This model has a low precision but a high recall for the digit 9 which means every time the model has predicted a digit as 9, only 63% of the time it has correctly predicted it. Moreover, every time the digit was actually 9, it predicted the digit correctly 82% of the time. Likewise, for the digit 8, the precision is low, but the recall is high. Moreover, in this model the F1 score for each of the digits is greater than 70% which was not the case for the Decision Tree model.

Now, to see if this model can get any better, we will fine-tune it. To do so, we will use a randomized search and define the parameters as shown below. In Randomized Search, we have defined the cross-validation as 3.



```
param_dist = {
    'alpha': np.arange(0.01,0.1, step=0.001)
}
```

The below image shows the best hyperparameter which means the alpha value should be 0.022 to get the best accuracy.



```
{'alpha': 0.021999999999999992}
```

After applying this hyperparameter in the model, the accuracy is now 81.78%. There is not a significant improvement in the model. The precision, recall and F1 score are shown below.

```
              precision    recall  f1-score   support

           0       0.93      0.92      0.92       414
           1       0.89      0.93      0.91       478
           2       0.87      0.84      0.86       420
           3       0.78      0.78      0.78       446
           4       0.78      0.72      0.75       404
           5       0.82      0.69      0.75       388
           6       0.91      0.91      0.91       404
           7       0.95      0.79      0.86       465
           8       0.66      0.76      0.71       393
           9       0.63      0.82      0.71       386

    accuracy                           0.82      4198
   macro avg       0.82      0.82      0.82      4198
weighted avg       0.83      0.82      0.82      4198
```

The precision, recall, and F1 score seem to be the same as before. Hence, the fine-tuning didn't help much. Since the accuracy is a bit higher, we will use the fine-tuned model.

To see if this model overfits or not, we have performed 3-fold cross-validation on it. We get the below results. Since the range of the scores is from 81.62% to 84.42% and the accuracy of this model is 81.77%, it is in the range which means that the model is not overfitting.

```
cross validation scores [0.82285714 0.81629736 0.84417441]
```

<u>Model Comparison:</u>

Out of the two fine-tuned algorithms, the Naïve Bayes model has a higher accuracy. The precision, recall, and F1 score for the Naïve Bayes model is also better. Moreover, the Decision Tree model is slightly overfitting but the Naïve Bayes model is not. Additionally, Naïve Bayes model also runs faster. The decision tree model has 118 as its max_depth which means it has 118 levels which could add to the computational time which is not the case for Naïve Bayes. Additionally, we are using only one hyperparameter to fine-tune the Naïve Bayes whereas, for the Decision tree, we are using 3. Hence, it is more time-consuming to fine-tune a Decision Tree. After taking the above points into consideration, we can conclude that Naïve Bayes model is better than the Decision Tree model in this scenario.