CS 520: Introduction to Artificial Intelligence

FINAL PROJECT: IMAGE CLASSIFICATION

RUTGERS
UNIVERSITY

TEAM MEMBERS:

Sri Harsha Samayamanthula (ss4305)
Gummadi Rohith Madhu Chandra (rmg255)

NAÏVE BAYES ALGORITHM

Naive Bayes is a classification algorithm that predicts output class based on the data we present during the training.

$$P(yi \mid x1, x2, \ldots, xn) = P(x1, x2, \ldots, xn \mid yi) * P(yi) / P(x1, x2, \ldots, xn)$$

DIGIT CLASSIFICATION:

In the digit classification part, we will be given test data and find whether given image is 0 or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9. And have implemented five methods to get details.

1) Feature_extraction() method :

- Feature extraction is used to transform given image into a feature of features. For this purpose, we divide the whole image into grids of size (4x4).
- Now, we traverse in each grid and find either number of black pixels or white pixels. In our case, we have white pixels into account and calculated the percentage of white in the grid.
- Moreover, we rounded off each percentage into closest number by multiplying with 10. Now, the value we got is one of the features of the image. We calculate for all grids and append into a 2d list of each list size being 49.
- Our method at last returns image features and training labels.

2) Fit() method :

- Fit method trains our naïve bayes model by precomputing the prior probabilities and likelihoods of each feature of training image.
- Firstly, we create a 3d list containing all features of images with respective to their classes and named this as threed_feature list.
- Secondly, we computed likelihoods of each input feature with respective to their class and stored them in a threed_prob list. Wherever we get to know that likelihood is being zero for a feature, we included 0.00001 probability as it reduces inaccuracies of our algorithm.

3) testing() method :

- Testing method takes the input image features, training labels and predicts the output of test data and return a list of predictions for each image.
- At first, our testing method makes a call for calc_class_prior() which returns a dictionary containing each label class and its prior probability.
- Secondly, testing calls fit method to return precomputed likelihoods of input features which will be used in the upcoming part.

- Thirdly, we use validation data for rehearsal of the accuracy of our classifier. we traverse each validation data image feature and calculate the probability of that being 0,1,2,3,4,5,6,7,8,9. Now, we append the probabilities into a list and find the maximum of these. So, we append for each image and return prediction list. At last, traverse the prediction list and compare each of the values with the true validation labels and prints the accuracy. Our algorithm gives a good accuracy of 78.8 percent.

- Finally, we do the same testing for test data for our algorithm trained with 100 percent training data and gives a good accuracy of **75.7 percent**.

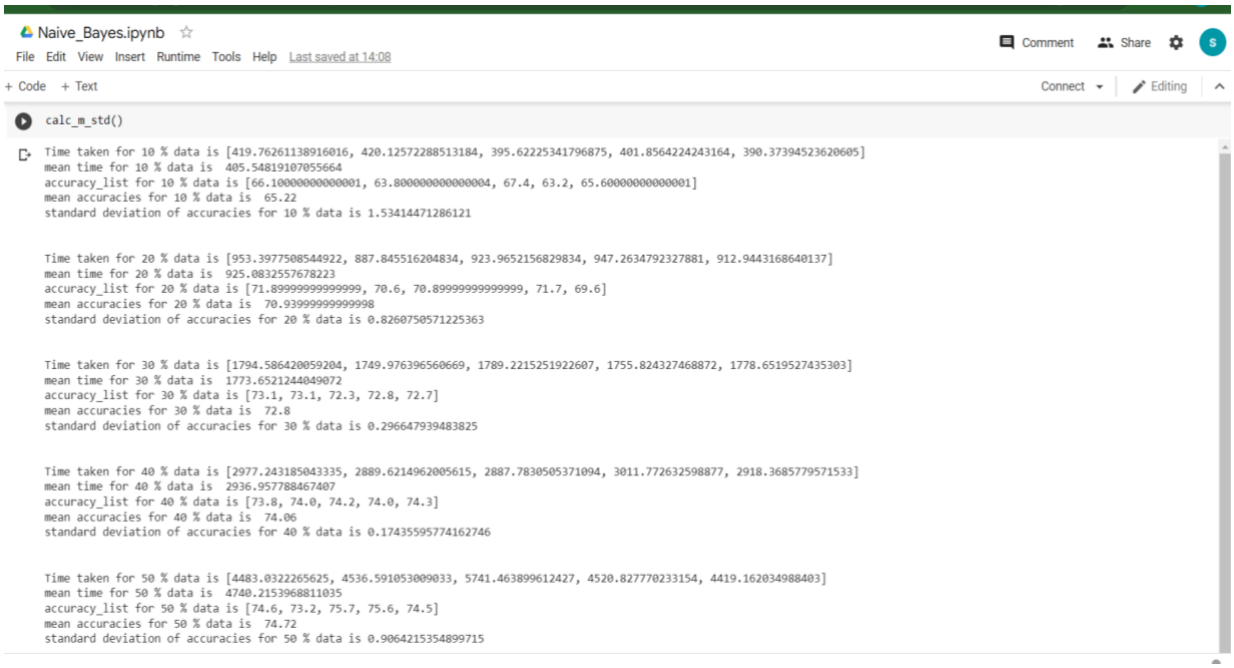
4) Training samples() method :

- Training_samples_method is used for predicting images based on the sample of training data given.
- Firstly, we take a list containing 10,20,30,40,50,60,70,80,90,100 which indicates amount of data needed for training. Now, we extract features from total_features based on the value of list.
- Secondly, this method calls testing method for every amount of data and prints the accuracy based on the predicted list.

5) calc_m_std() method :

- calc_m_std() method returns the average time taken, mean and standard deviation of accuracies of each amount of training data for particular number of iterations.
- At First, we take a list containing 10,20,30,40,50,60,70,80,90,100 which indicates amount of data needed for training.
- Secondly, for every iteration in range 5, we extract random features from total_features based on the value of list and append to the accuracy list. After completion of each amount of data extraction, we calculate mean and standard deviation of accuracies and print them.
- The following figure shows the average time taken, accuracies, mean and standard deviation for each training data.

Results:



Naive_Bayes.ipynb

File Edit View Insert Runtime Tools Help Last saved at 14:08

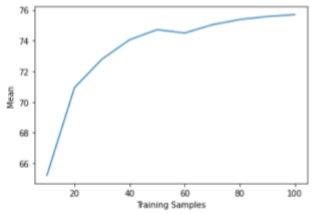
Comment Share S

+ Code + Text

Connect Editing

```
Time taken for 60 % data is [6433.502912521362, 6346.964597702026, 6429.115533828735, 6413.957834243774, 6447.494029998779]
mean time for 60 % data is  6414.206981658936
accuracy_list for 60 % data is [75.2, 74.6, 74.8, 73.6, 74.3]
mean accuracies for 60 % data is  74.50000000000001
standard deviation of accuracies for 60 % data is 0.5366563145999519


Time taken for 70 % data is [8840.784311294556, 8835.535049438477, 8831.073760986328, 10095.984697341919, 8850.327014923096]
mean time for 70 % data is  9090.740966796875
accuracy_list for 70 % data is [76.2, 74.7, 73.9, 75.2, 75.2]
mean accuracies for 70 % data is  75.03999999999999
standard deviation of accuracies for 70 % data is 0.7499333303701061


Time taken for 80 % data is [11604.221820831299, 11511.896848678589, 11530.518531799316, 11571.57015800476, 11632.598638534546]
mean time for 80 % data is  11570.161199569702
accuracy_list for 80 % data is [75.8, 75.8, 75.6, 74.9, 74.8]
mean accuracies for 80 % data is  75.38000000000001
standard deviation of accuracies for 80 % data is 0.4399999999999784


Time taken for 90 % data is [14823.44365119934, 14800.259113311768, 14762.859106063843, 14692.22640991211, 15919.605255126953]
mean time for 90 % data is  14999.678707122803
accuracy_list for 90 % data is [75.2, 76.0, 76.0, 75.7, 75.0]
mean accuracies for 90 % data is  75.58
standard deviation of accuracies for 90 % data is 0.41182520563947966


Time taken for 100 % data is [18347.965002059937, 18431.272268295288, 18322.606801986694, 19593.06263923645, 18441.873788833618]
mean time for 100 % data is  18627.356100082397
accuracy_list for 100 % data is [75.7, 75.7, 75.7, 75.7, 75.7]
mean accuracies for 100 % data is  75.7
standard deviation of accuracies for 100 % data is 0.0
```

```
plt.plot(percent_train,mean_list)
plt.xlabel("Training Samples")
plt.ylabel("Mean")
plt.show()
```



Naive_Bayes.ipynb

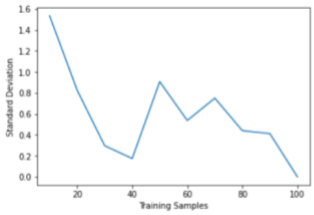
File Edit View Insert Runtime Tools Help Last saved at 14:08

Comment Share S

+ Code + Text

Connect Editing

```
[ ] plt.plot(percent_train,std_list)
    plt.xlabel("Training Samples")
    plt.ylabel("Standard Deviation")
    plt.show()
```



Naive_Bayes.ipynb

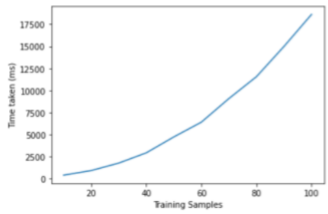
File Edit View Insert Runtime Tools Help Last saved at 14:08

Comment Share S

+ Code + Text

Connect Editing

```
[ ] plt.plot(percent_train,time_list)
    plt.xlabel("Training Samples")
    plt.ylabel("Time taken (ms)")
    plt.show()
```



FACE CLASSIFICATION:

In the face classification part, we will be given test data and find whether given image is Face or not. And have implemented five methods to get details.

1) Feature_extraction() method :

- Feature extraction is used to transform given image into a feature of features. For this purpose, we divide the whole image into grids of size (10x10).
- Now, we traverse in each grid and find either number of black pixels or white pixels. In our case, we have white pixels into account and calculated the percentage of white in the grid.
- Moreover, we rounded off each percentage into closest number by multiplying with 10. Now, the value we got is one of the features of the image. We calculate for all grids and append into a 2d list of each list size being 42.
- Our method at last returns image features and training labels.

2) Fit() method :

- Fit method trains our naïve bayes model by precomputing the prior probabilities and likelihoods of each feature of training image.
- Firstly, we create a 3d list containing all features of images with respective to their classes and named this as threed_feature list.
- Secondly, we computed likelihoods of each input feature with respective to their class and stored them in a threed_prob list. Wherever we get to know that likelihood is being zero for a feature, we included 0.00001 probability as it reduces inaccuracies of our algorithm.

3) testing() method :

- Testing method takes the input image features, training labels and predicts the output of test data and return a list of predictions for each image.
- At first, our testing method makes a call for calc_class_prior() which returns a dictionary containing each label class and its prior probability.
- Secondly, testing calls fit method to return precomputed likelihoods of input features which will be used in the upcoming part.
- Thirdly, we use validation data for rehearsal of the accuracy of our classifier. we traverse each validation data image feature and calculate the probability of that being 0,1. Now, we append the probabilities into a list and find the maximum of these. So, we append for each image and return prediction list. At last, traverse the prediction list and compare each of the values with the true validation labels and prints the accuracy. Our algorithm gives a good accuracy of 90.697 percent.
- Finally, we do the same testing for test data for our algorithm trained with 100 percent training data and gives a good accuracy of **90.66** percent.

4) Training samples() method :

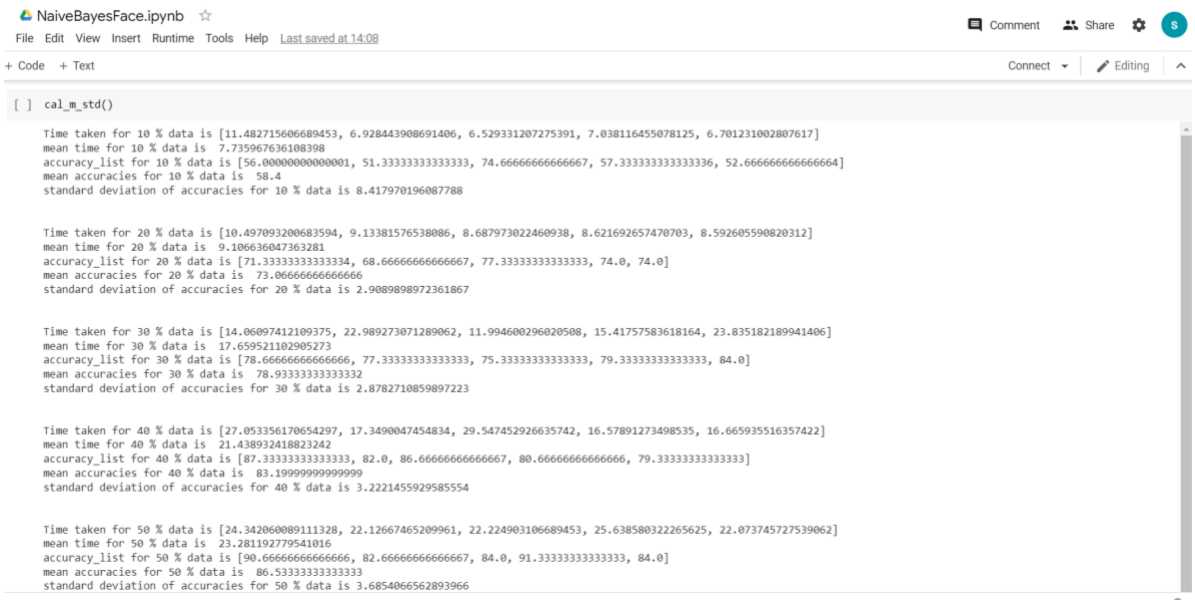
- Training_samples_method is used for predicting images based on the sample of training data given.

- Firstly, we take a list containing 10,20,30,40,50,60,70,80,90,100 which indicates amount of data needed for training. Now, we extract features from total_features based on the value of list.
- Secondly, this method calls testing method for every amount of data and prints the accuracy based on the predicted list.

5) calc_m_std() method :

- calc_m_std() method returns the average time taken, mean and standard deviation of accuracies of each amount of training data for particular number of iterations.
- At First, we take a list containing 10,20,30,40,50,60,70,80,90,100 which indicates amount of data needed for training.
- Secondly, for every iteration in range 5, we extract random features from total_features based on the value of list and append to the accuracy list. After completion of each amount of data extraction, we calculate mean and standard deviation of accuracies and print them.
- The following figure shows the average time taken, accuracies, mean and standard deviation for each training data.

Results:



NaiveBayesFace.ipynb

File Edit View Insert Runtime Tools Help Last saved at 14:08

Comment Share S

+ Code + Text

Connect Editing

```
Time taken for 60 % data is [37.15968132019043, 32.720088958740234, 35.60829162597656, 30.050992965698242, 31.792640686035156]
mean time for 60 % data is  33.466339111328125
accuracy_list for 60 % data is [82.0, 81.33333333333333, 88.66666666666667, 87.33333333333333, 80.0]
mean accuracies for 60 % data is  83.86666666666666
standard deviation of accuracies for 60 % data is 3.461534662865913


Time taken for 70 % data is [62.63399124145508, 43.98465156555176, 38.054704666137695, 41.36514663696289, 36.742210388183594]
mean time for 70 % data is  44.55614089965820
accuracy_list for 70 % data is [86.66666666666667, 88.66666666666667, 89.33333333333333, 90.0, 84.66666666666667]
mean accuracies for 70 % data is  87.86666666666667
standard deviation of accuracies for 70 % data is 1.9504985117770368


Time taken for 80 % data is [54.95858192443848, 50.02260208129883, 46.09537124633789, 66.83993339538574, 48.97308349609375]
mean time for 80 % data is  53.37791442871094
accuracy_list for 80 % data is [87.33333333333333, 88.0, 88.0, 89.33333333333333, 89.33333333333333]
mean accuracies for 80 % data is  88.39999999999999
standard deviation of accuracies for 80 % data is 0.7999999999999992


Time taken for 90 % data is [81.45356178283691, 56.55407905578613, 66.59936904907227, 72.51191139221191, 59.11660194396973]
mean time for 90 % data is  67.24710464477539
accuracy_list for 90 % data is [91.33333333333333, 89.33333333333333, 90.66666666666666, 90.66666666666666, 90.66666666666666]
mean accuracies for 90 % data is  90.53333333333333
standard deviation of accuracies for 90 % data is 0.6531972647421802


Time taken for 100 % data is [68.79234313964844, 67.840576171875, 76.41935348510742, 67.68584251403809, 74.6622085571289]
mean time for 100 % data is  71.08006477355957
accuracy_list for 100 % data is [90.66666666666666, 90.66666666666666, 90.66666666666666, 90.66666666666666, 90.66666666666666]
mean accuracies for 100 % data is  90.66666666666666
standard deviation of accuracies for 100 % data is 0.0
```

NaiveBayesFace.ipynb

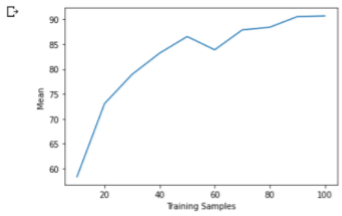
File Edit View Insert Runtime Tools Help Last saved at 14:08

Comment Share S

+ Code + Text

Connect Editing

```
plt.plot(percent_train,mean_list)
plt.xlabel("Training Samples")
plt.ylabel("Mean")
plt.show()
```



NaiveBayesFace.ipynb

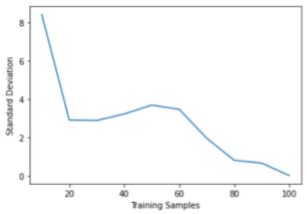
File Edit View Insert Runtime Tools Help Last saved at 14:08

Comment Share S

+ Code + Text

Connect Editing

```
[ ] plt.plot(percent_train,std_list)
    plt.xlabel("Training Samples")
    plt.ylabel("Standard Deviation")
    plt.show()
```



NaiveBayesFace.ipynb

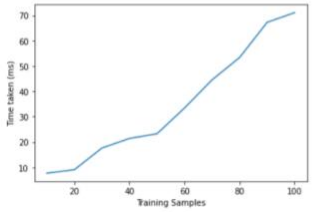
File Edit View Insert Runtime Tools Help Last saved at 14:08

Comment Share S

+ Code + Text

Connect Editing

```
[ ] plt.plot(percent_train,time_list)
    plt.xlabel("Training Samples")
    plt.ylabel("Time taken (ms)")
    plt.show()
```



Perceptron Classifier

- Perceptron Is basically a single layer Neural Network which uses a linear decision function to predict the output label value base on the calculated scores.

The linear function is given by the equation:

$F(xi, W) = W0 + W1*\phi1(xi) + w2*\phi2(xi) + \ldots\ldots wL*\Phi L(xi)$

Here "W" Is list of weights and "$\phi(x)$" is the list features in our feature list.

Given a new test point x, this predict its label y=true if
$F(xi, w) > 0$ and y=false if $F(xi, w) < 0$

Our Approach:

1. We Initialize the weights { Wj } for j=1..n, to be some random small values or they can be initialized as zeros. But in our approach we are initializing them to be zeros at the beginning. We initialized W0=0 at the beginning as our base weight.

2. We then perform feature extraction on our training set Xtrain and pass the initial weights and Xtrain features to the linear decision function.

3. Then the decision function calculates the value F(xi, W) for the each given image and based on the scores it chooses to perform the following actions.

Face Data Classification:

1. $F(xi, w) > 0$ and yi=True or $F(xi, w) < 0$ and yi=False then we don't perform any action.

2. If $F(xi, w) < 0$ and yi=true then $wj = wj + \phi j(xj)$
   we basically add the corresponding feature of the image to the our current weights and this continues till we reach the last image and update w0=w0+1.

3. If $F(xi, w)>0$ and y1=false then $wj = wj - \phi j(xj)$
   we basically subtract the corresponding feature of the image to the our current weights and this continues till we reach the last image and update w0=w0-1.

4. We run this complete iteration for few number of times and stop the process.

5. We then return the final weights {Wj for j=1..n} from the given training images and return the final base weight W0.

6. With this data we again calculate the scores for the test images and compare with the y label, if $F(xi, w)>0$ we decide that it is an image and vice versa.

Digit Data Classification:

1. The process is same for the Digit data as well, but here we have 10 different weights initialized to '0s', so the {W} is 2D list with sub list containing zeros. With that being said we have 10 different base weights initialized to 0's as well.

2. We calculate the F(xi,w) for each image and if the max index in the list of scores for a particular image is equal to the y label of the corresponding, then we don't do anything.

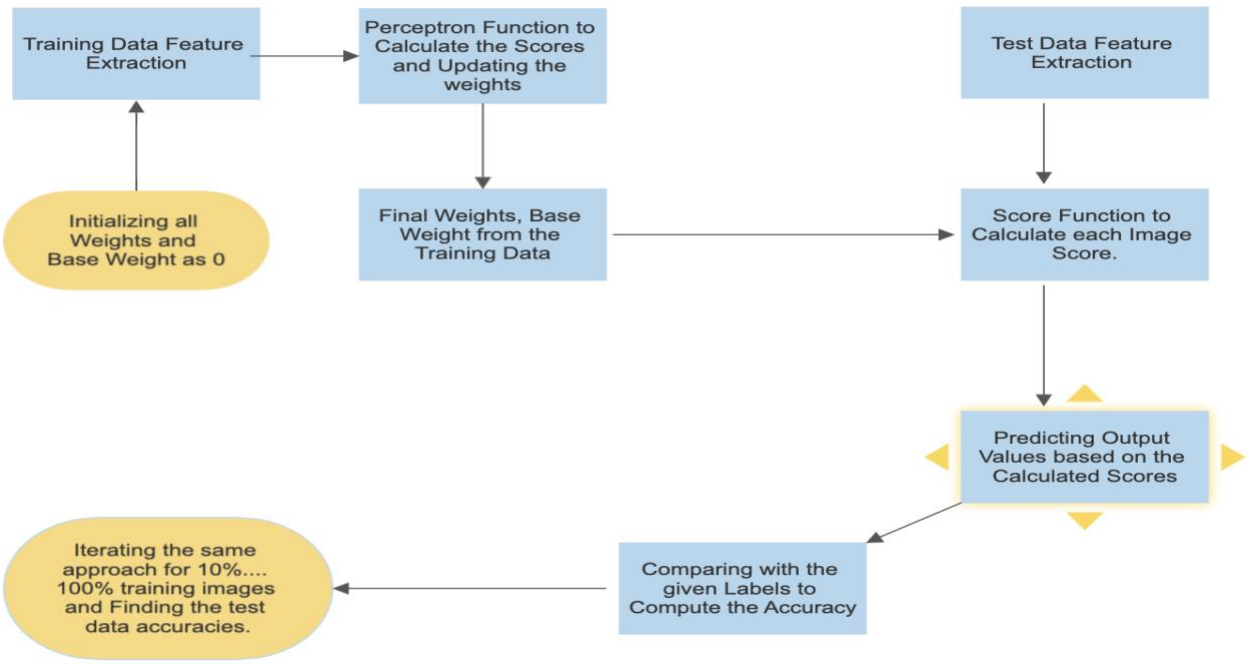
3. Otherwise, we subtract the corresponding feature of the image from the weights list i.e., subtract feature from list in the weights list whose position equal to the value of y label.

4. If the above condition arises, we update the values in the in the base weight list i.e., Wj to Wj-1 except for the corresponding label value position in the base weights list, where we add Wi=Wi+1 to it.

5. We continue to this process till we reach the final image and outputs the final trained weights list and base weights list, which we then use it on the test data to predict the output values.

6. In predicting the output, we take the max index of the particular images scores list and outputs it as the predicted value, and we calculate the accuracy by taking if there is a match or not with the y-labels and divide the total matches by total images.

Diagrammatic Representation of the approach

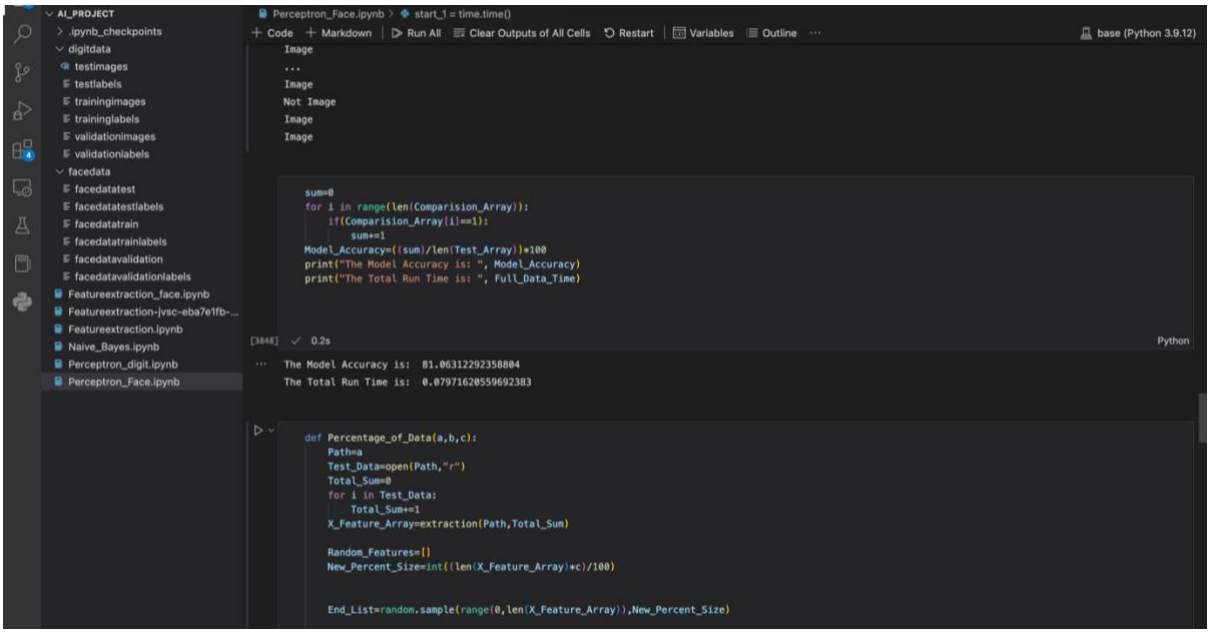


Results and Accuracies:

1. Face Data Classification:

Model Accuracy for 100% of Training Data is: **"81.063%"**

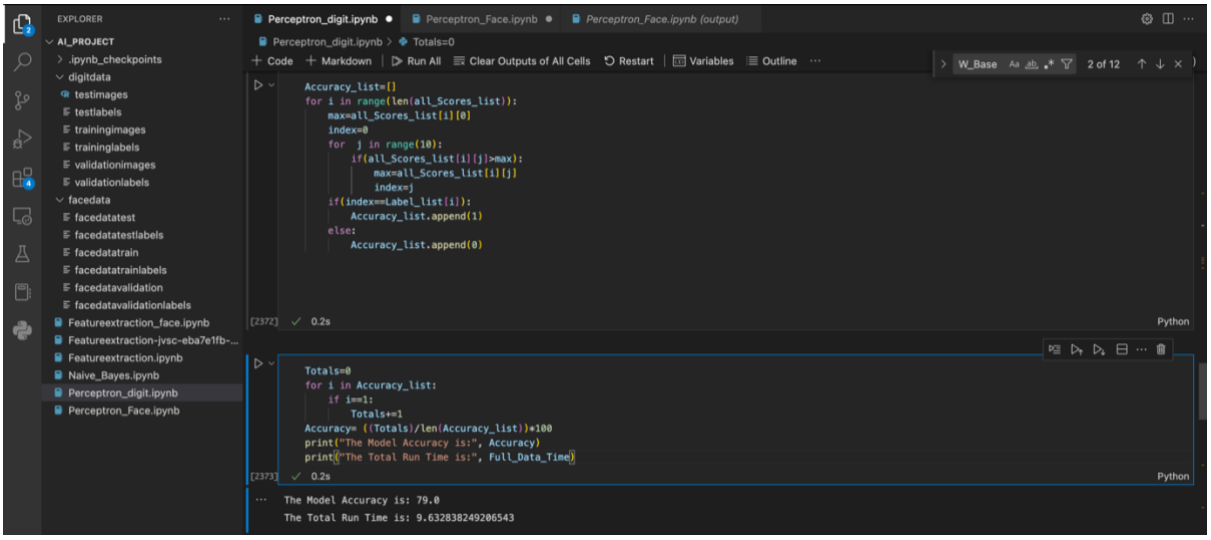
100% Training Data Run Time: **"0.0797" Sec**



2. Digit Data Classification:

Model Accuracy for 100% of Training Data is: **"79%"**

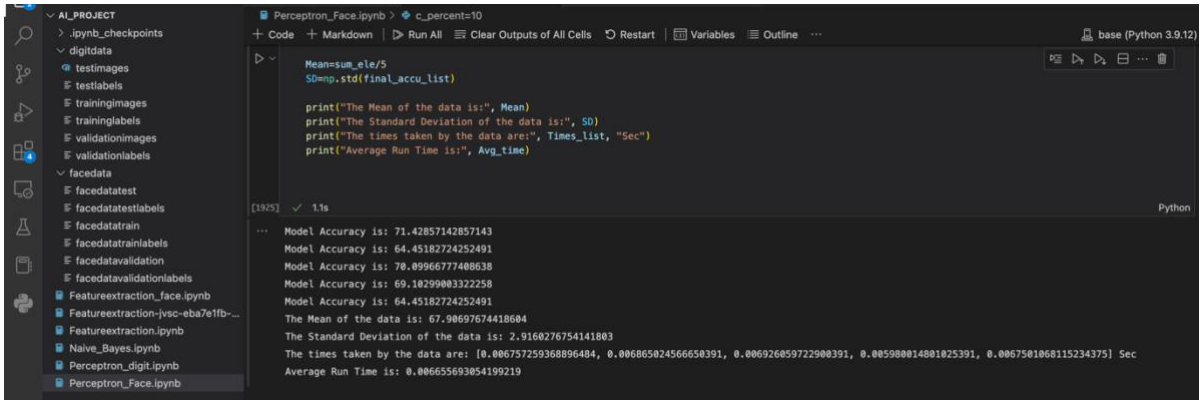
100% Training Data Run Time: **"9.632" Sec**



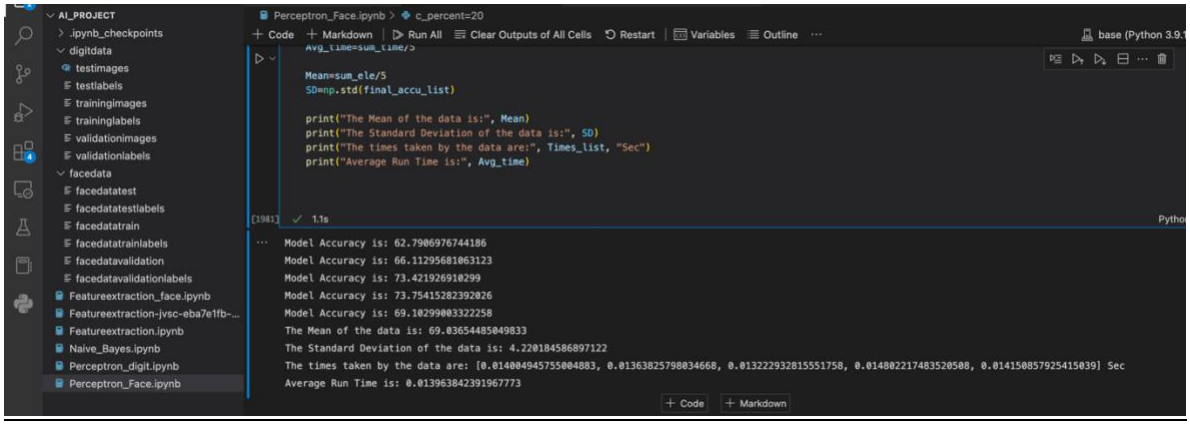
ACCURACIES AND RUNTIMES FOR VARIOUS PERCENTAGE OF TRAINING DATA

1. Face Data Classification:

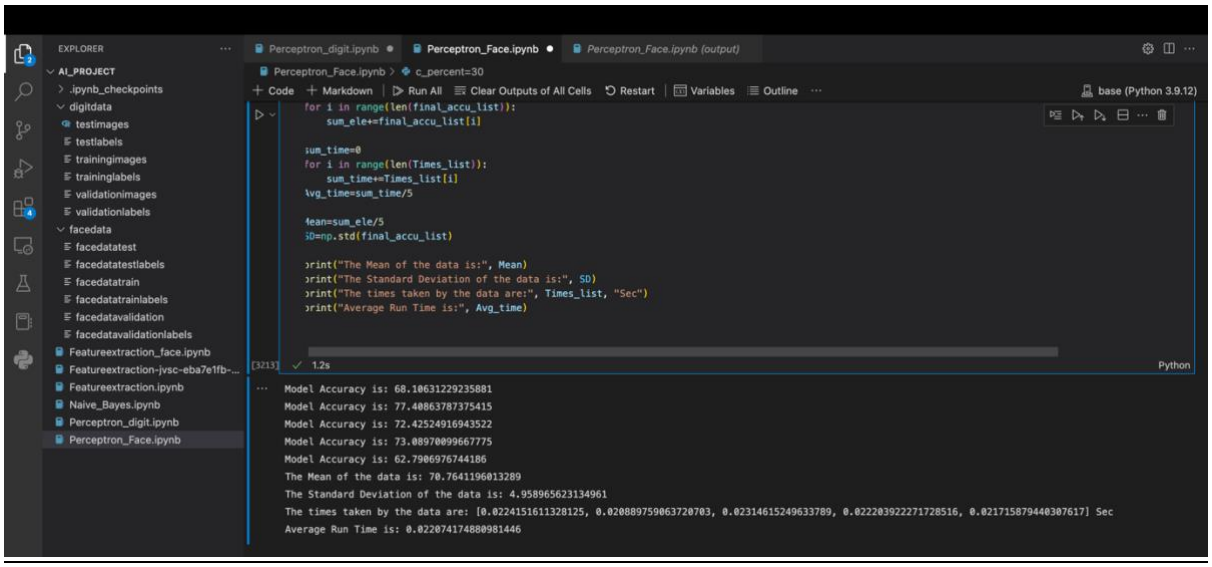
10% (5 Iterations)



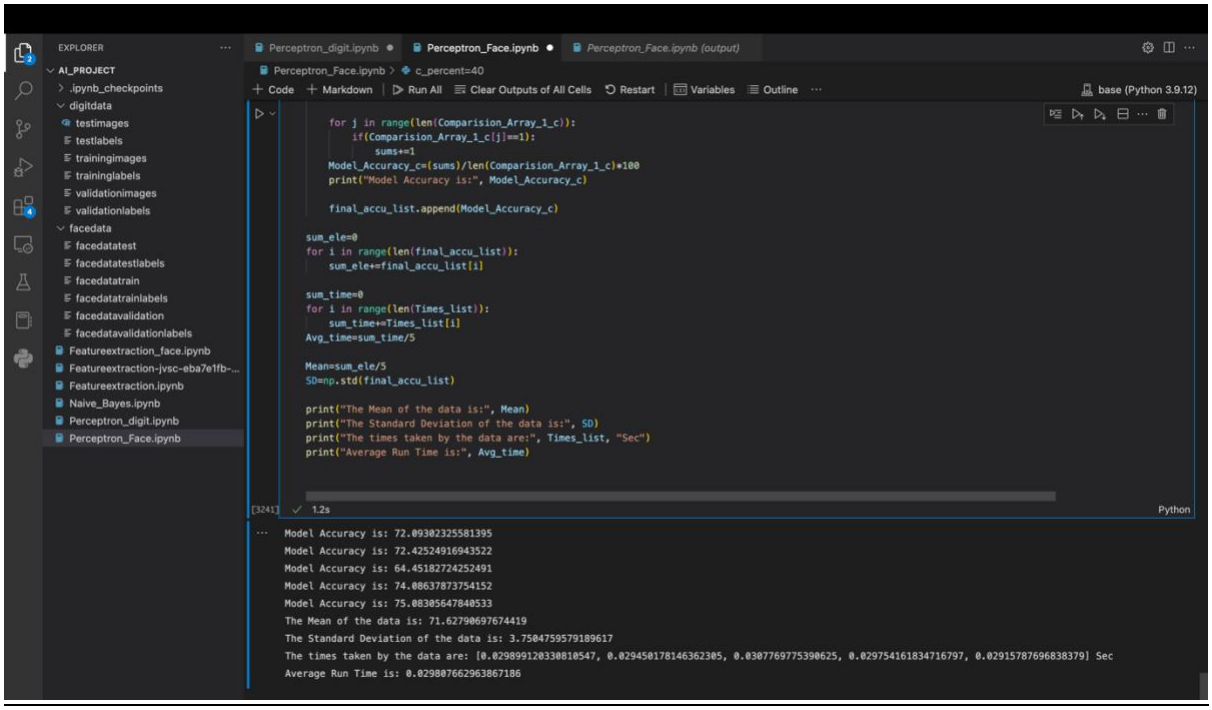
20% (5 Iterations)



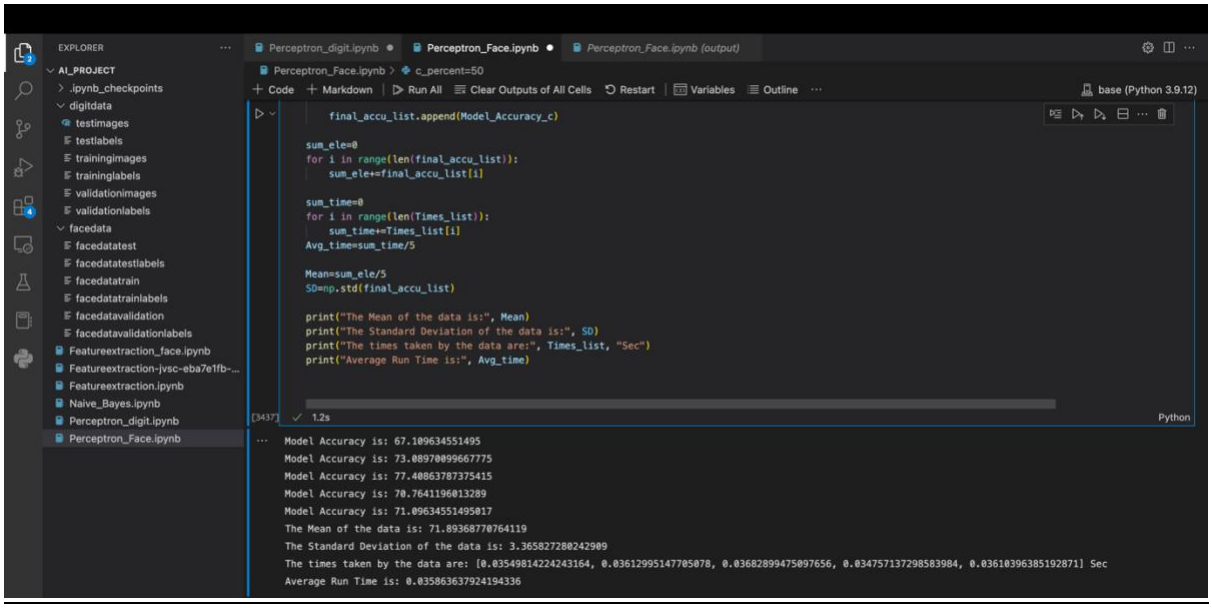
30% (5 Iterations)



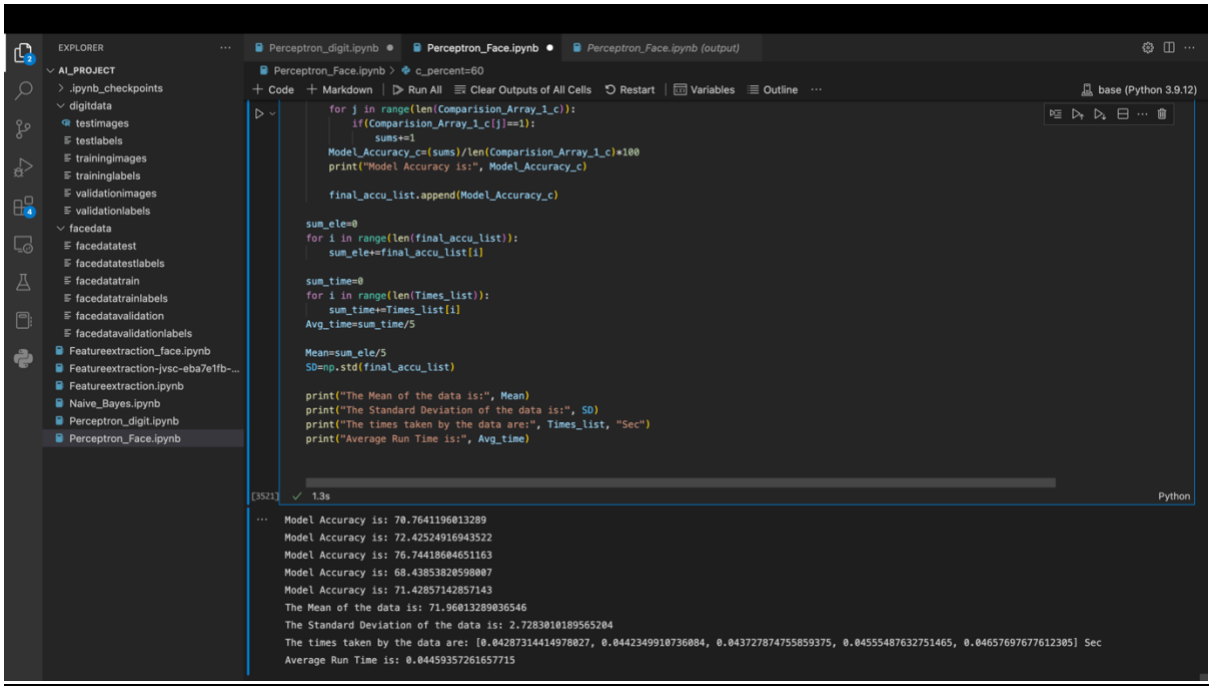
40% (5 Iterations)



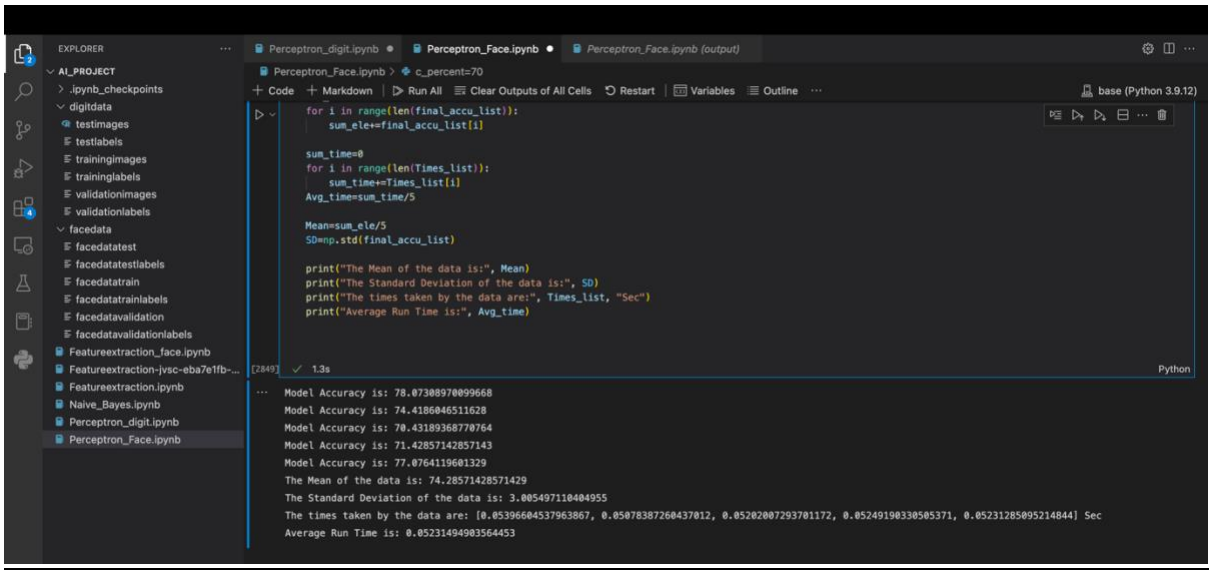
50% (5 Iterations)



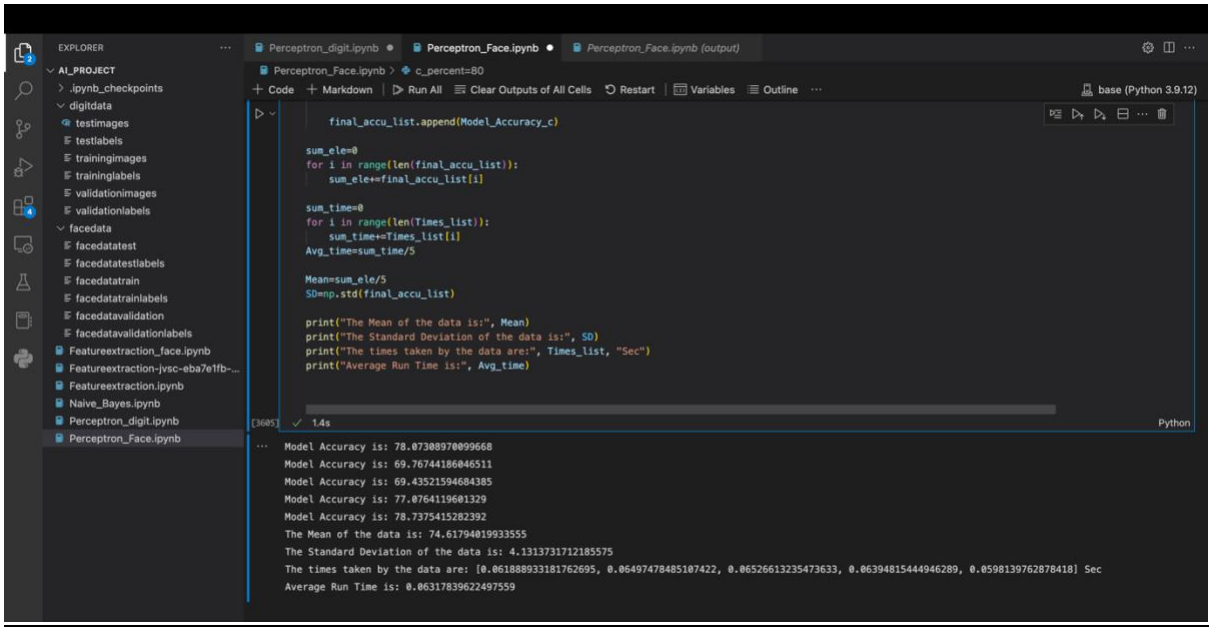
60% (5 Iterations)



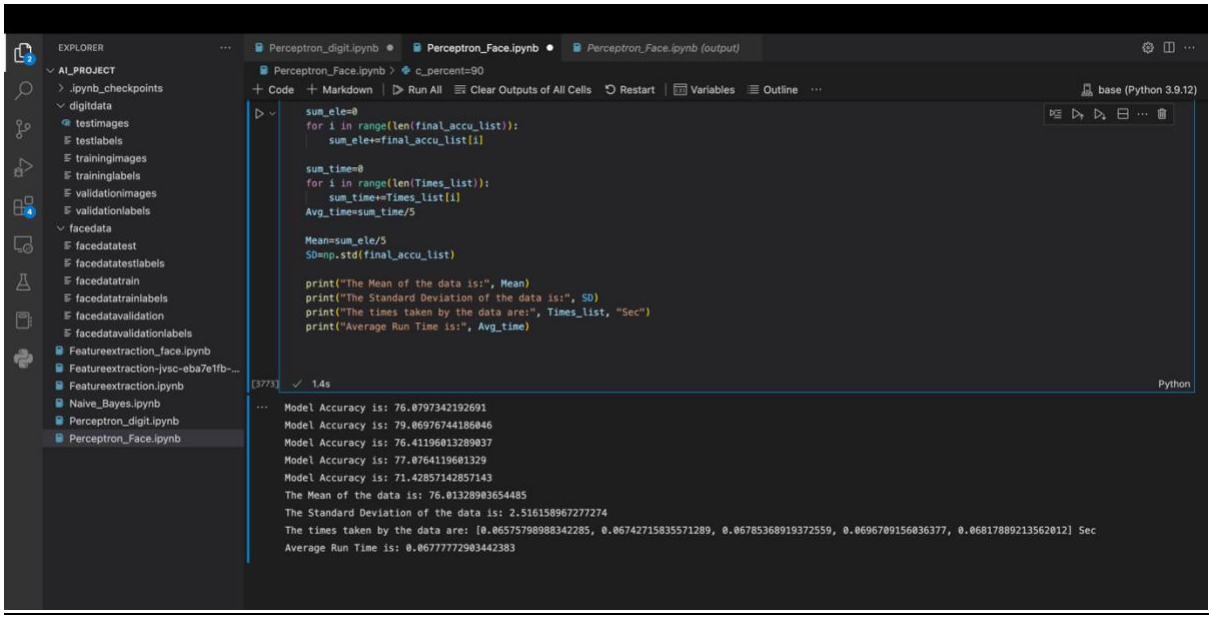
70% (5 Iterations)



80% (5 Iterations)

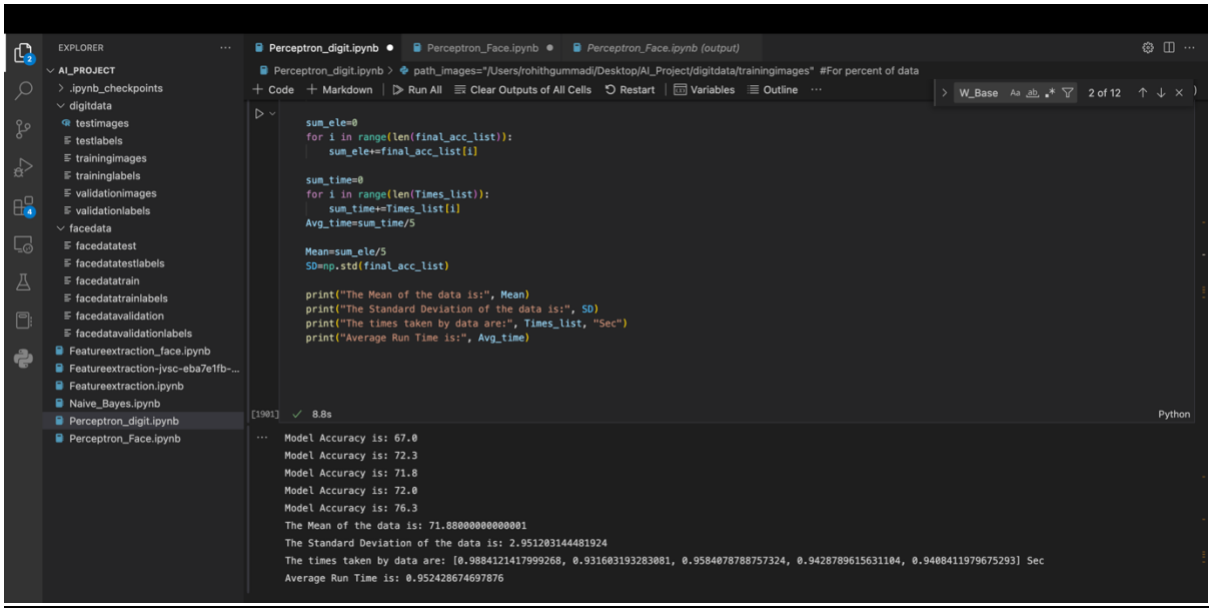


90% (5 Iterations)

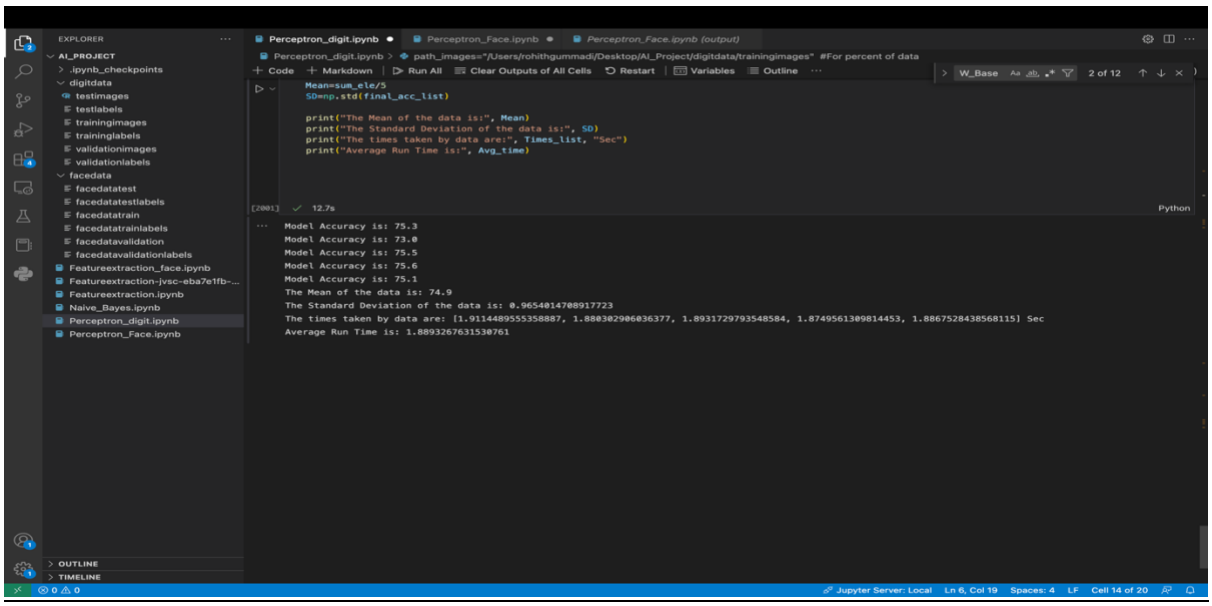


2. Digit Data Classification:

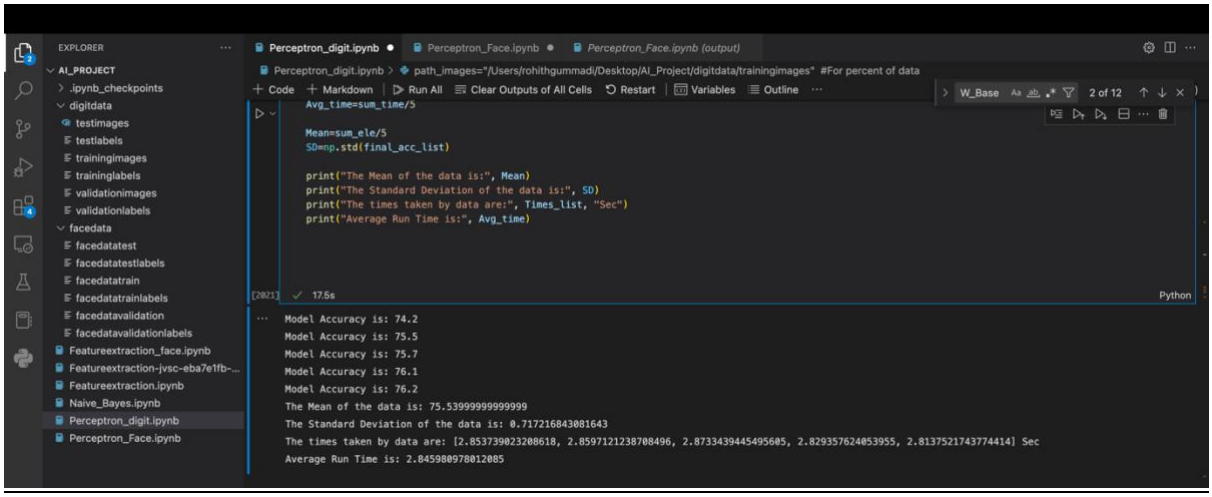
10% (5 Iterations)



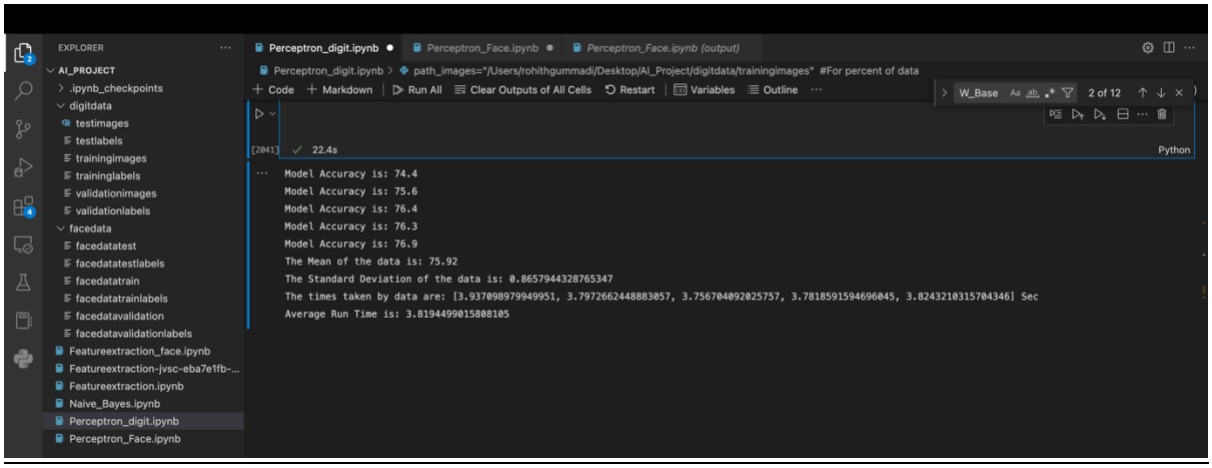
20% (5 Iterations)



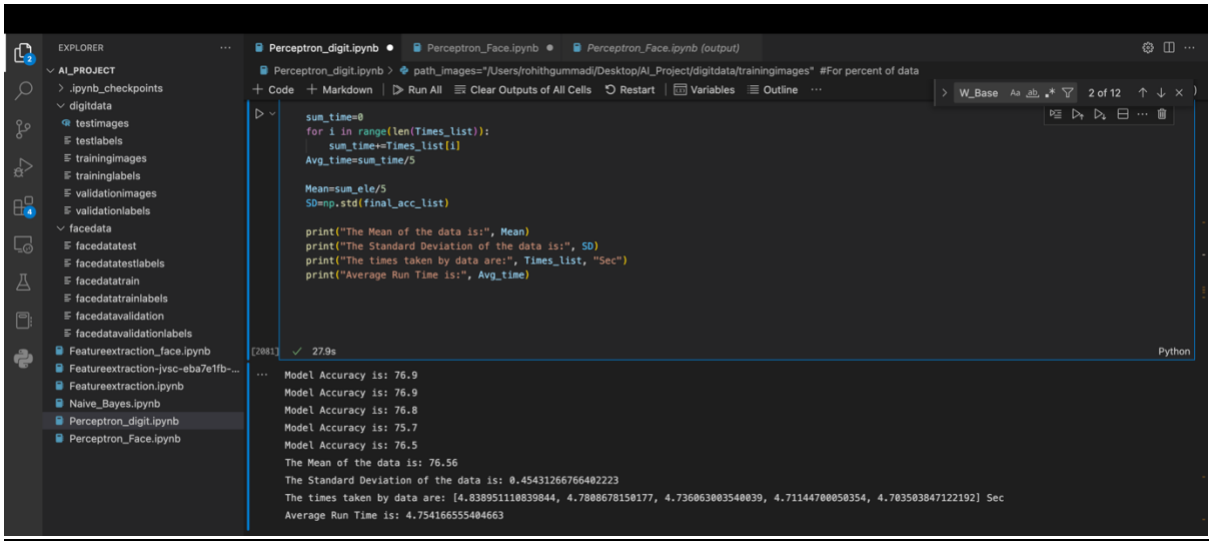
30% (5 Iterations)



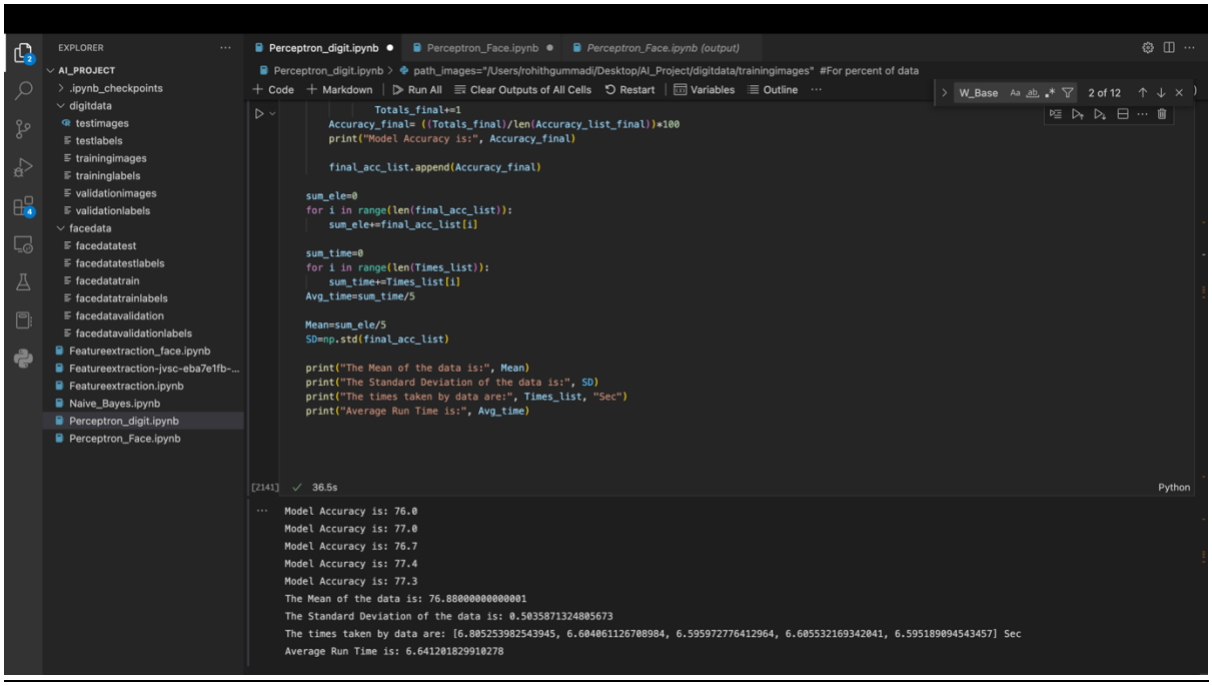
40% (5 Iterations)



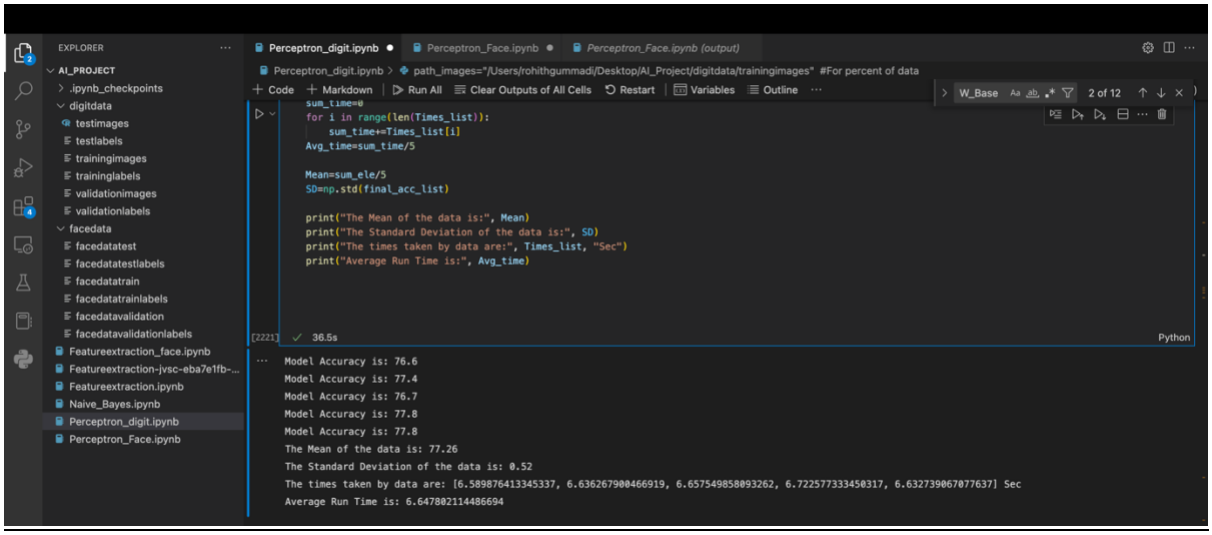
50% (5 Iterations)



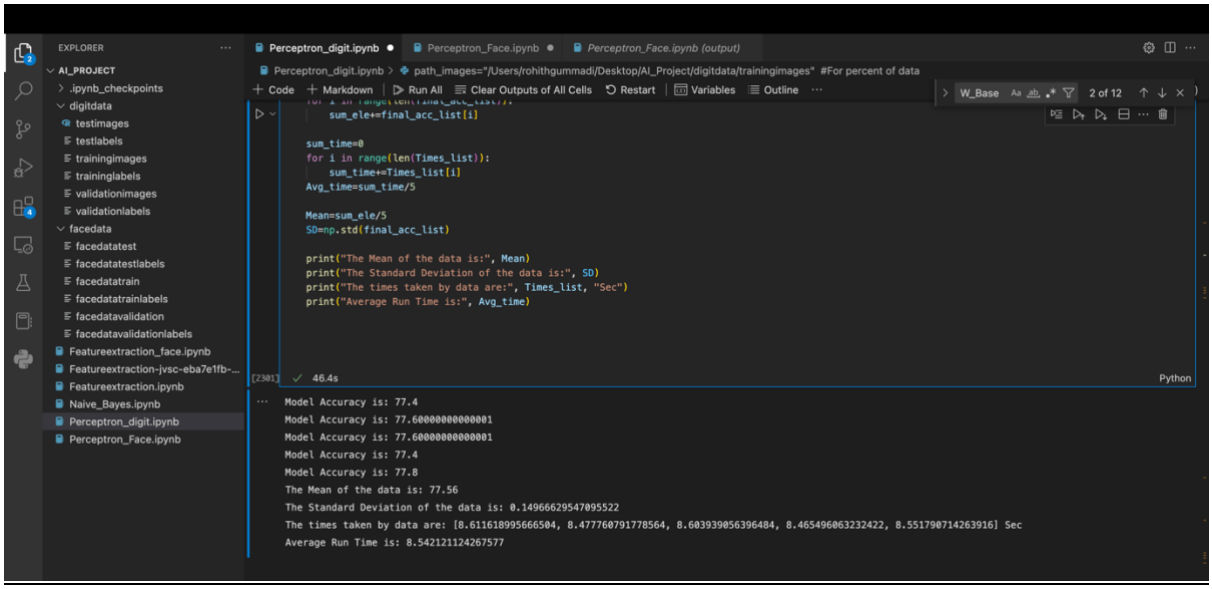
60% (5 Iterations)



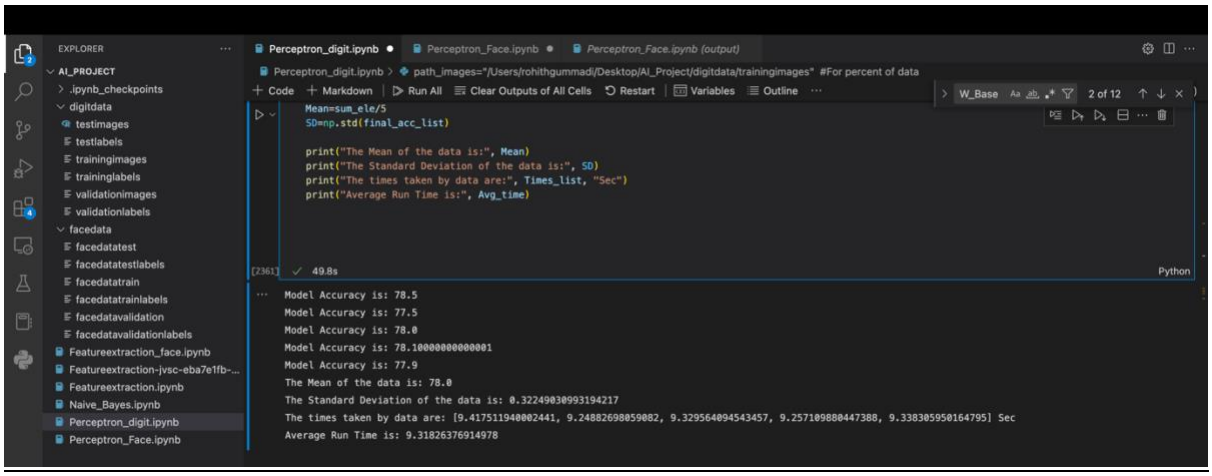
70% (5 Iterations)



80% (5 Iterations)

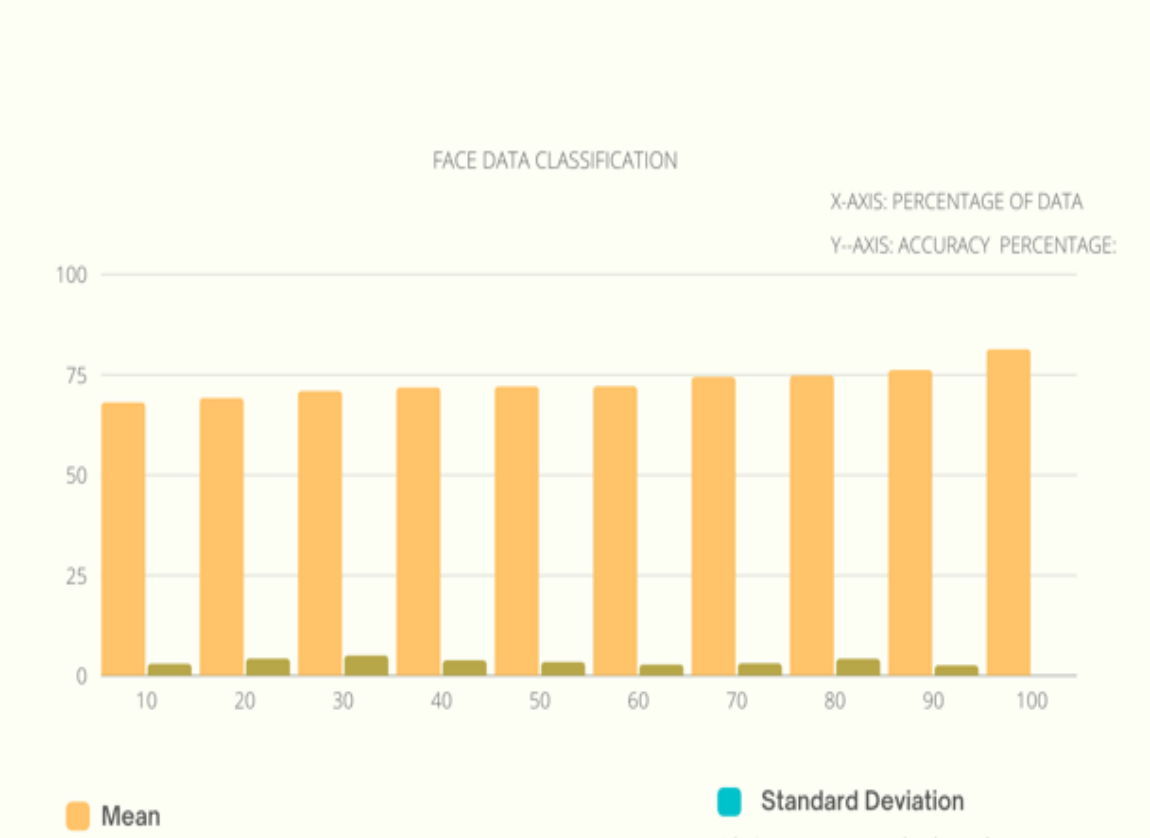


90% (5 Iterations)

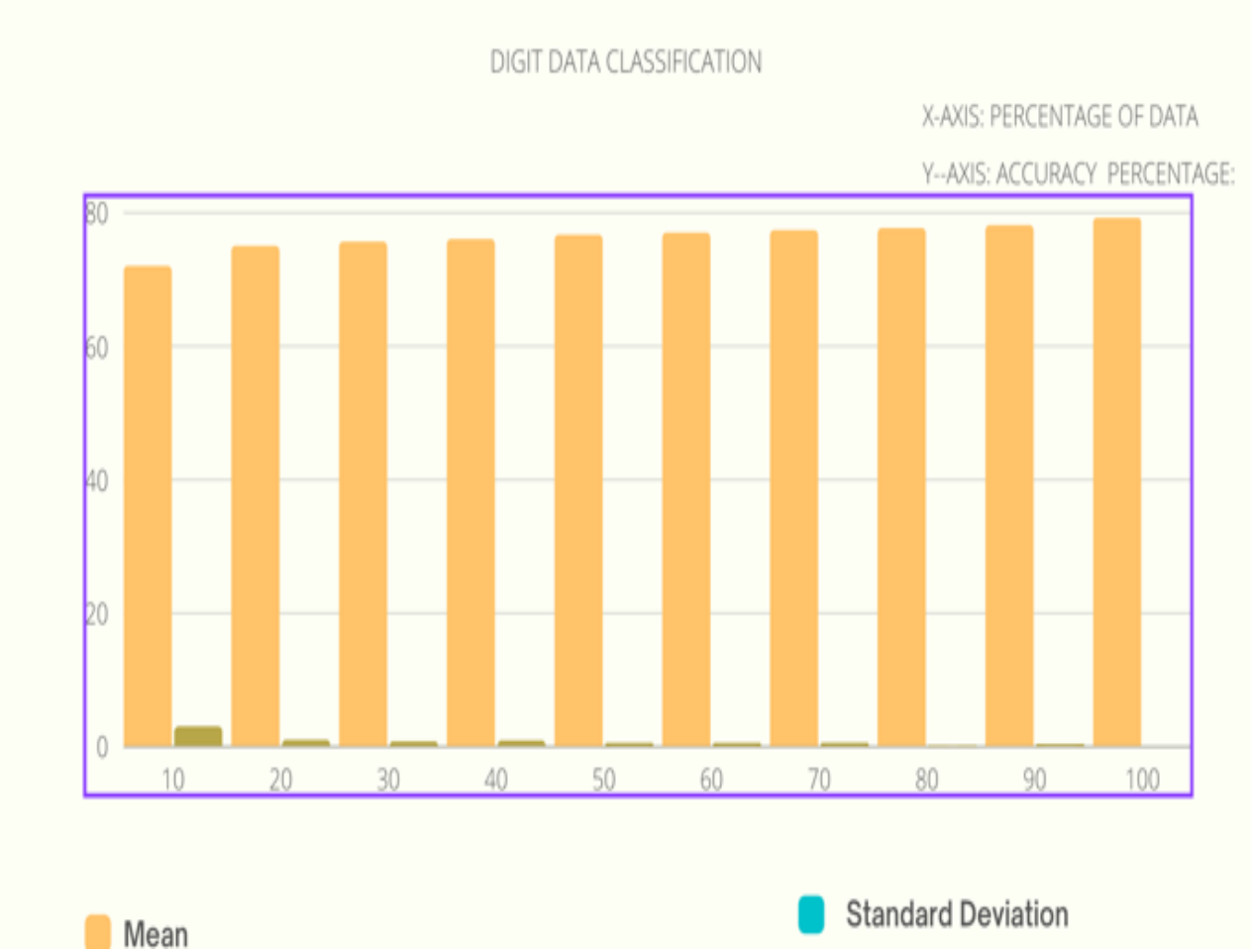


Graphs for Mean of Accuracies and Standard Deviation:

FACE DATA:

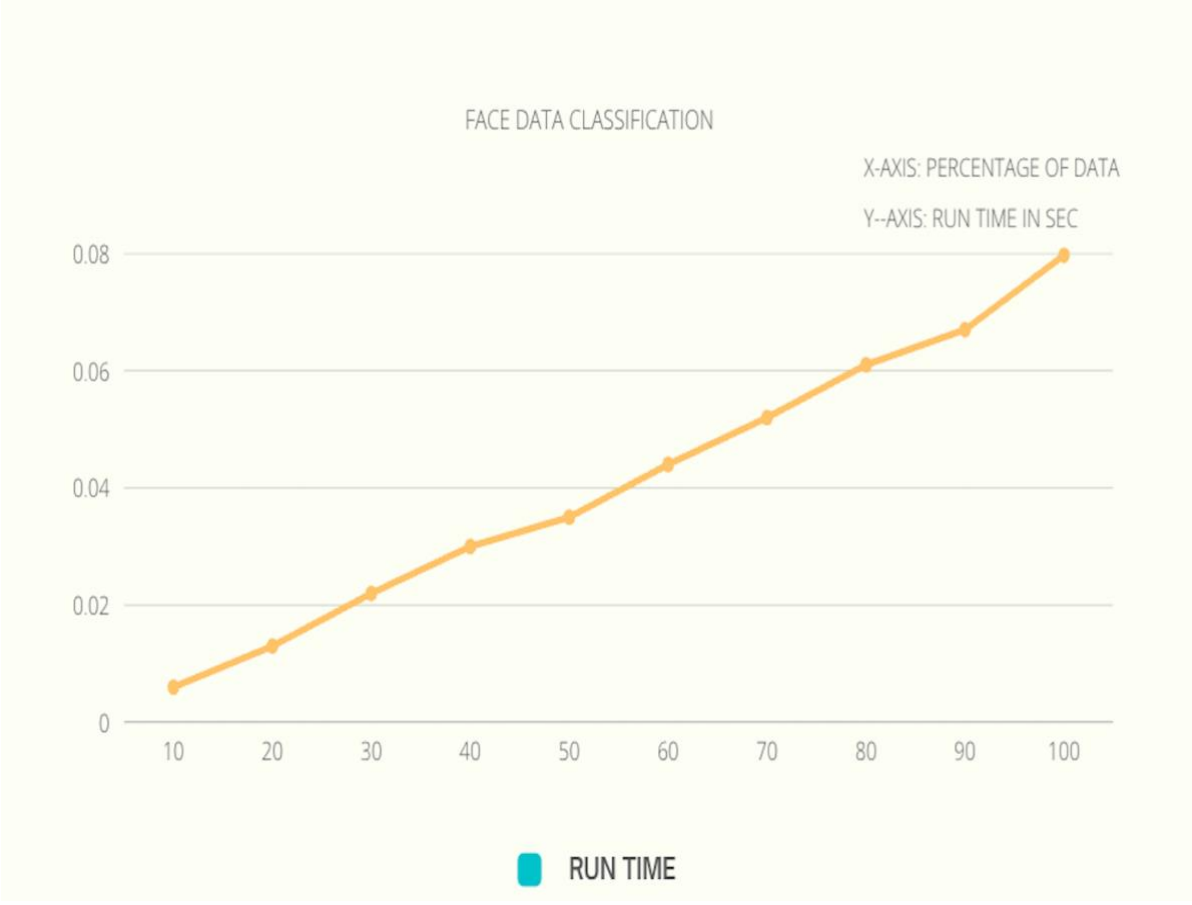


DIGIT DATA:

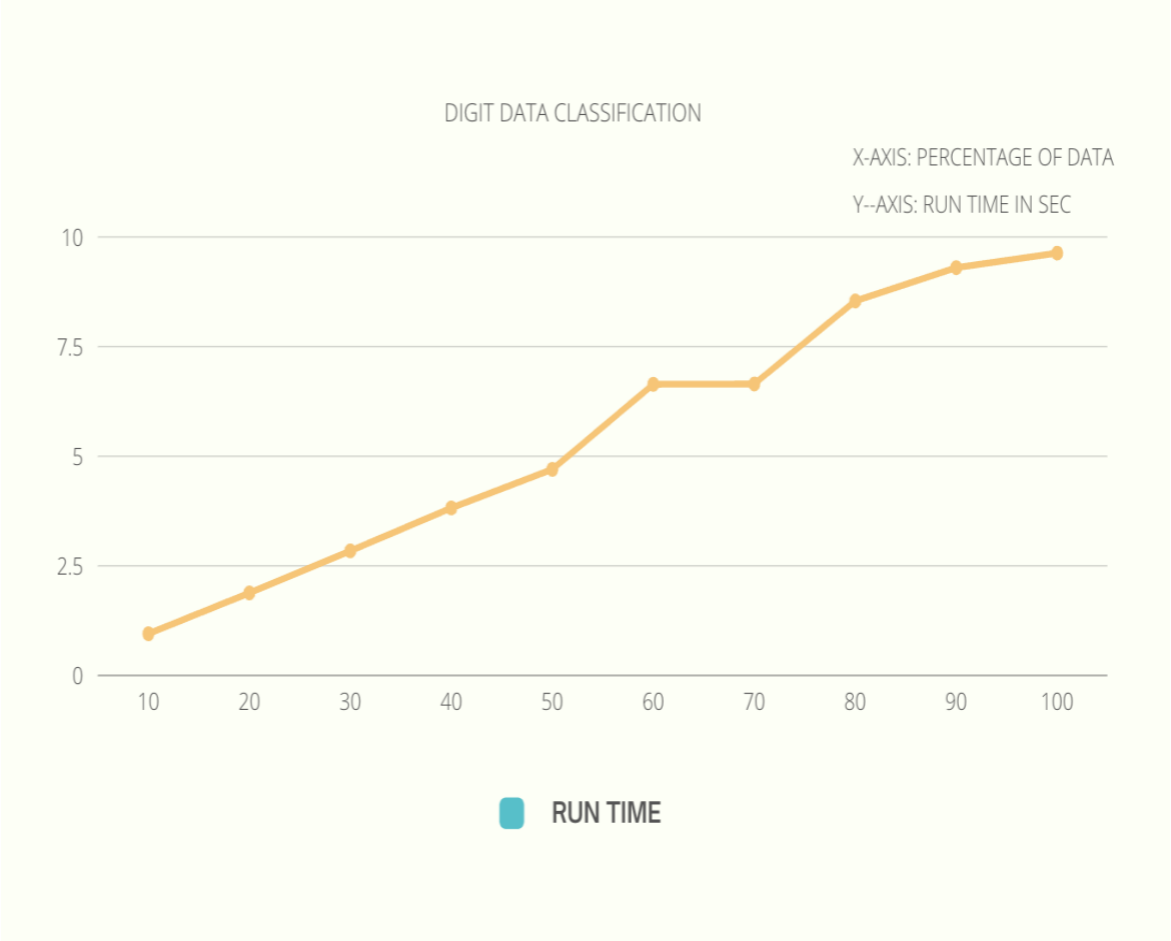


Graph for Mean Run Times:

Face Data:


FACE DATA CLASSIFICATION
X-AXIS: PERCENTAGE OF DATA
Y--AXIS: RUN TIME IN SEC
0.08
0.06
0.04
0.02
0
10
20
30
40
50
60
70
80
90
100
RUN TIME

DIGIT DATA:



K NEAREST NEIGHBOURS ALGORITHM:

KNN is a supervised machine learning algorithm that predicts output based on the highest number of votes of a class from the K nearest neighbours.

DIGIT CLASSIFICATION:

In the digit classification part, we will be given test data and find whether given image is 0 or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9. And have implemented three methods to get details.

1) Feature_extraction() method :

- Feature extraction is used to transform given image into a feature of features. For this purpose, we divide the whole image into grids of size (4x4).
- Now, we traverse in each grid and find either number of black pixels or white pixels. In our case, we have white pixels into account and calculated the percentage of white in the grid.

- Moreover, we rounded off each percentage into closest number by multiplying with 10. Now, the value we got is one of the features of the image. We calculate for all grids and append into a 2d list of each list size being 49.
- Our method at last returns image features and training labels.

2) Validation and Testing:

- After feature extraction, we fit Knn classifier with validation data to check how good our algorithm works. For validation, accuracy was 90.5 which is very good to proceed.
- Finally, for test data, our Knn algorithm gives an accuracy of 87.7

3) Training samples() method :

- Training_samples_method is used for predicting images based on the sample of training data given.
- Firstly, we take a list containing 10,20,30,40,50,60,70,80,90,100 which indicates amount of data needed for training. Now, we extract features from total_features based on the value of list.
- Secondly, this method calls testing method for every amount of data and prints the accuracy based on the predicted list

4) calc_m_std() method :

- calc_m_std() method returns the average time taken, mean and standard deviation of accuracies of each amount of training data for particular number of iterations.
- At First, we take a list containing 10,20,30,40,50,60,70,80,90,100 which indicates amount of data needed for training.
- Secondly, for every iteration in range 5, we extract random features from total_features based on the value of list and append to the accuracy list. After completion of each amount of data extraction, we calculate time ken, mean and standard deviation of accuracies and print them.
- The following figure shows the time taken, accuracies, mean and standard deviation for each training data.

Results:

KnnDigit.ipynb

File Edit View Insert Runtime Tools Help Last saved at 14:09

Comment Share S

+ Code + Text

Connect Editing

```
calc_m_std()
```

```
Time taken for 10 % data is [71.41304016113281, 84.73658561706543, 62.927961349487305, 80.50942420959473, 71.79021835327148]
mean time for 10 % data is  74.27544593811035
accuracy_list for 10 % data is [77.60000000000001, 79.4, 75.9, 79.9, 80.5]
mean accuracies for 10 % data is  78.66
standard deviation of accuracies for 10 % data is 1.6859418732566063


Time taken for 20 % data is [95.65877914428711, 82.80253410339355, 98.62112998962402, 85.65402030944824, 97.7623462677002]
mean time for 20 % data is  92.09976196289062
accuracy_list for 20 % data is [84.5, 82.8, 82.39999999999999, 83.2, 81.3]
mean accuracies for 20 % data is  82.84
standard deviation of accuracies for 20 % data is 1.0442221985765308


Time taken for 30 % data is [121.10757827758789, 124.22323226928711, 121.28233909606934, 102.15473175048828, 100.76022148132324]
mean time for 30 % data is  113.90562057495117
accuracy_list for 30 % data is [82.89999999999999, 82.8, 84.5, 85.7, 85.39999999999999]
mean accuracies for 30 % data is  84.25999999999999
standard deviation of accuracies for 30 % data is 1.2175385004179555


Time taken for 40 % data is [155.37071228027344, 114.3643856048584, 135.6825828552246, 118.00527572631836, 124.64475631713867]
mean time for 40 % data is  129.6135425567627
accuracy_list for 40 % data is [85.1, 85.0, 84.7, 84.39999999999999, 85.6]
mean accuracies for 40 % data is  84.96
standard deviation of accuracies for 40 % data is 0.40298883359219745


Time taken for 50 % data is [156.3551425933838, 142.12298393249512, 133.12458992004395, 153.63812446594238, 135.29682159423828]
mean time for 50 % data is  144.1075325012207
accuracy_list for 50 % data is [86.2, 85.9, 85.7, 85.6, 85.6]
mean accuracies for 50 % data is  85.8
standard deviation of accuracies for 50 % data is 0.2280350850198308 3
```

KnnDigit.ipynb

File Edit View Insert Runtime Tools Help Last saved at 14:09

Comment Share S

+ Code + Text

Connect Editing

```
Time taken for 60 % data is [167.89555549621582, 169.15535926818848, 153.50985527038574, 146.63243293762207, 141.60799980163574]
mean time for 60 % data is  155.76024055480957
accuracy_list for 60 % data is [86.4, 85.9, 85.3, 86.1, 86.0]
mean accuracies for 60 % data is  85.94000000000001
standard deviation of accuracies for 60 % data is 0.3611094017053576


Time taken for 70 % data is [166.9931411743164, 169.07119750976562, 170.7615852355957, 169.69585418701172, 155.17663955688477]
mean time for 70 % data is  166.33968353271484
accuracy_list for 70 % data is [87.0, 86.1, 85.6, 86.2, 85.2]
mean accuracies for 70 % data is  86.02
standard deviation of accuracies for 70 % data is 0.6079473661428266


Time taken for 80 % data is [182.26051330566406, 170.63093185424805, 174.85880851745605, 172.8353500366211, 179.05282974243164]
mean time for 80 % data is  175.92768669128418
accuracy_list for 80 % data is [86.9, 86.6, 87.2, 87.0, 86.8]
mean accuracies for 80 % data is  86.9
standard deviation of accuracies for 80 % data is 0.20000000000000284


Time taken for 90 % data is [186.90967559814453, 199.32007789611816, 185.86015701293945, 185.8358383178711, 197.39484786987305]
mean time for 90 % data is  191.06411933898926
accuracy_list for 90 % data is [87.3, 87.9, 87.6, 87.1, 87.9]
mean accuracies for 90 % data is  87.55999999999999
standard deviation of accuracies for 90 % data is 0.3200000000000044


Time taken for 100 % data is [209.4416618347168, 212.8455638885498, 192.81268119812012, 199.01275634765625, 195.9991455078125]
mean time for 100 % data is  202.0223617553711
accuracy_list for 100 % data is [87.7, 87.6, 87.7, 87.7, 87.7]
mean accuracies for 100 % data is  87.67999999999999
standard deviation of accuracies for 100 % data is 0.04000000000000 3415
```

KnnDigit.ipynb

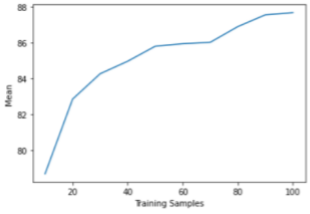
File Edit View Insert Runtime Tools Help Last saved at 14:09

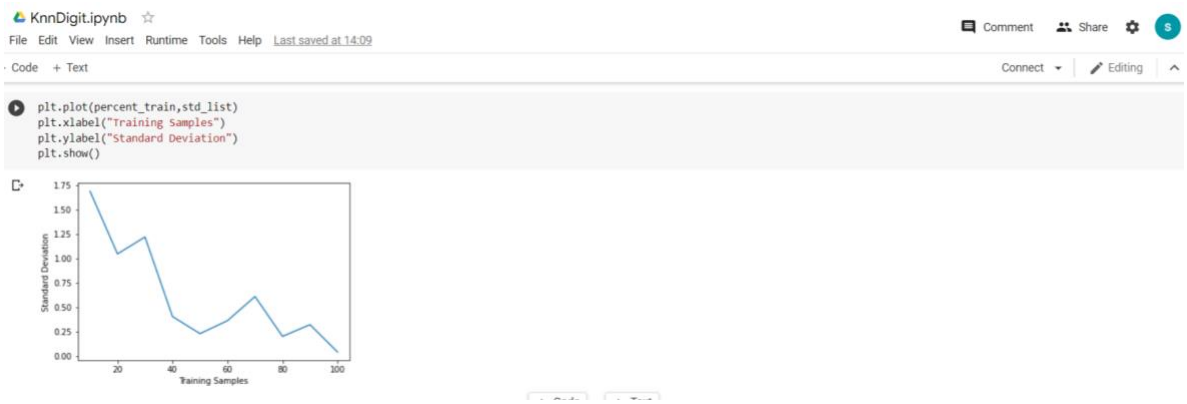
Comment Share S

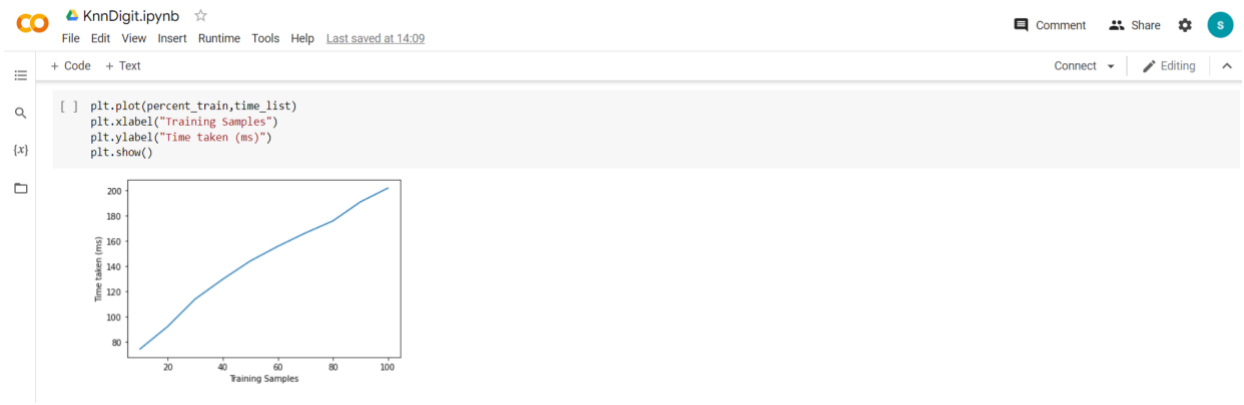
Code + Text

Connect Editing

```
[ ] plt.plot(percent_train,mean_list)
    plt.xlabel("Training Samples")
    plt.ylabel("Mean")
    plt.show()
```







FACE CLASSIFICATION:

In the face classification part, we will be given test data and find whether given image is Face or not. And have implemented similar methods to get details.

1) **Feature_extraction method():**

   Validation and Testing:
   After feature extraction, we fit Knn classifier with validation data to check how good our algorithm works. For validation, accuracy was 72.75 which is good to proceed. Finally, for test data, our Knn algorithm gives an **accuracy of 72%**

2) **Training samples() method :**

3) **calc_m_std() method :**

Results:

KnnFace.ipynb

File Edit View Insert Runtime Tools Help Last saved at 14:09

Comment Share S

+ Code + Text

Connect Editing

```
cal_m_std()
```

```
Time taken for 10 % data is [20.212888717651367, 17.122745513916016, 24.9178409576416, 26.60512924194336, 19.53268051147461]
mean time for 10 % data is  21.67825698852539
accuracy_list for 10 % data is [68.66666666666667, 66.0, 54.666666666666664, 60.66666666666667, 70.66666666666667]
mean accuracies for 10 % data is  64.13333333333334
standard deviation of accuracies for 10 % data is 5.802681372530387


Time taken for 20 % data is [40.013790130615234, 20.699024200439453, 18.30887794494629, 22.2930908203125, 21.846771240234375]
mean time for 20 % data is  24.63231086730957
accuracy_list for 20 % data is [75.33333333333333, 76.0, 73.33333333333333, 67.33333333333333, 62.66666666666667]
mean accuracies for 20 % data is  70.93333333333332
standard deviation of accuracies for 20 % data is 5.139822737972366


Time taken for 30 % data is [32.57250785827637, 30.101776123046875, 30.0137996673584, 23.354530334472656, 23.565053939819336]
mean time for 30 % data is  27.921533584594727
accuracy_list for 30 % data is [68.0, 60.66666666666667, 68.66666666666667, 66.66666666666666, 68.0]
mean accuracies for 30 % data is  66.4
standard deviation of accuracies for 30 % data is 2.939387691339812


Time taken for 40 % data is [34.64794158935547, 26.526451110839844, 33.017635345458984, 21.383285522460938, 20.572662353515625]
mean time for 40 % data is  27.229595184326172
accuracy_list for 40 % data is [70.0, 72.66666666666667, 71.33333333333334, 69.33333333333334, 68.66666666666667]
mean accuracies for 40 % data is  70.4
standard deviation of accuracies for 40 % data is 1.4360439485692011


Time taken for 50 % data is [25.09331703186035, 37.78982162475586, 37.077903747558594, 26.851415634155273, 31.006574630737305]
mean time for 50 % data is  31.563806533813477
accuracy_list for 50 % data is [66.0, 76.0, 74.66666666666667, 74.0, 69.33333333333334]
mean accuracies for 50 % data is  72.0
standard deviation of accuracies for 50 % data is 3.747591819348052
```

KnnFace.ipynb

File Edit View Insert Runtime Tools Help Last saved at 14:09

Comment Share S

+ Code + Text

Connect Editing

```
Time taken for 60 % data is [32.82594680786133, 27.04143524169922, 22.689342498779297, 25.30503273010254, 29.302358627319336]
mean time for 60 % data is  27.432823181152344
accuracy_list for 60 % data is [74.0, 68.66666666666667, 72.66666666666667, 73.33333333333333, 70.66666666666667]
mean accuracies for 60 % data is  71.86666666666667
standard deviation of accuracies for 60 % data is 1.9504985117770368


Time taken for 70 % data is [32.51075744628906, 27.33325958251953, 25.098323822021484, 36.739349365234375, 30.210494995117188]
mean time for 70 % data is  30.378437042236328
accuracy_list for 70 % data is [72.66666666666667, 72.66666666666667, 72.0, 74.0, 70.0]
mean accuracies for 70 % data is  72.26666666666668
standard deviation of accuracies for 70 % data is 1.3063945294843622


Time taken for 80 % data is [35.62641143798828, 29.195547103881836, 25.171995162963867, 25.866031646728516, 29.27851676940918]
mean time for 80 % data is  29.027700424194336
accuracy_list for 80 % data is [73.33333333333333, 70.66666666666667, 68.0, 69.33333333333334, 71.33333333333334]
mean accuracies for 80 % data is  70.53333333333335
standard deviation of accuracies for 80 % data is 1.808621328833403


Time taken for 90 % data is [59.52620506286621, 50.069332122802734, 35.89582443237305, 35.300493240356445, 33.04767608642578]
mean time for 90 % data is  42.767906188964844
accuracy_list for 90 % data is [73.33333333333333, 71.33333333333334, 74.0, 69.33333333333334, 71.33333333333334]
mean accuracies for 90 % data is  71.86666666666667
standard deviation of accuracies for 90 % data is 1.654623152798776


Time taken for 100 % data is [42.617082595825195, 33.33020210266113, 31.08811378479004, 31.360149383544922, 29.801368713378906]
mean time for 100 % data is  33.63938331604004
accuracy_list for 100 % data is [73.33333333333333, 72.0, 72.66666666666667, 70.0, 72.66666666666667]
mean accuracies for 100 % data is  72.13333333333334
standard deviation of accuracies for 100 % data is 1.14697670227235
```

KnnFace.ipynb

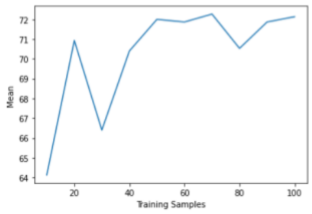
File Edit View Insert Runtime Tools Help Last saved at 14:09

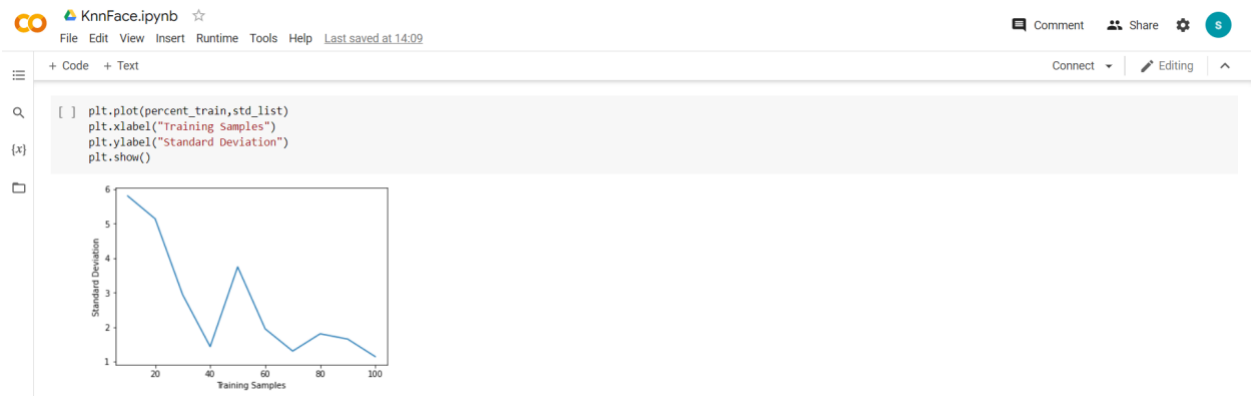
Comment Share S

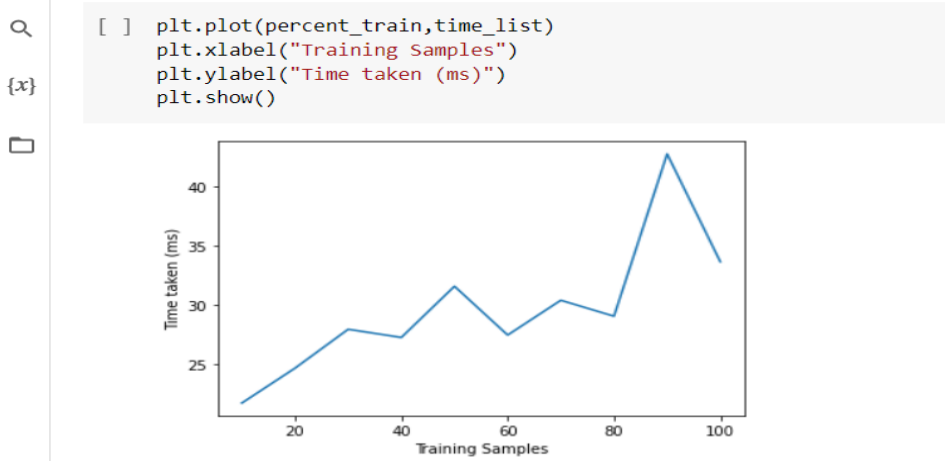
+ Code + Text

Connect Editing

```
[ ] plt.plot(percent_train,mean_list)
    plt.xlabel("Training Samples")
    plt.ylabel("Mean")
    plt.show()
```

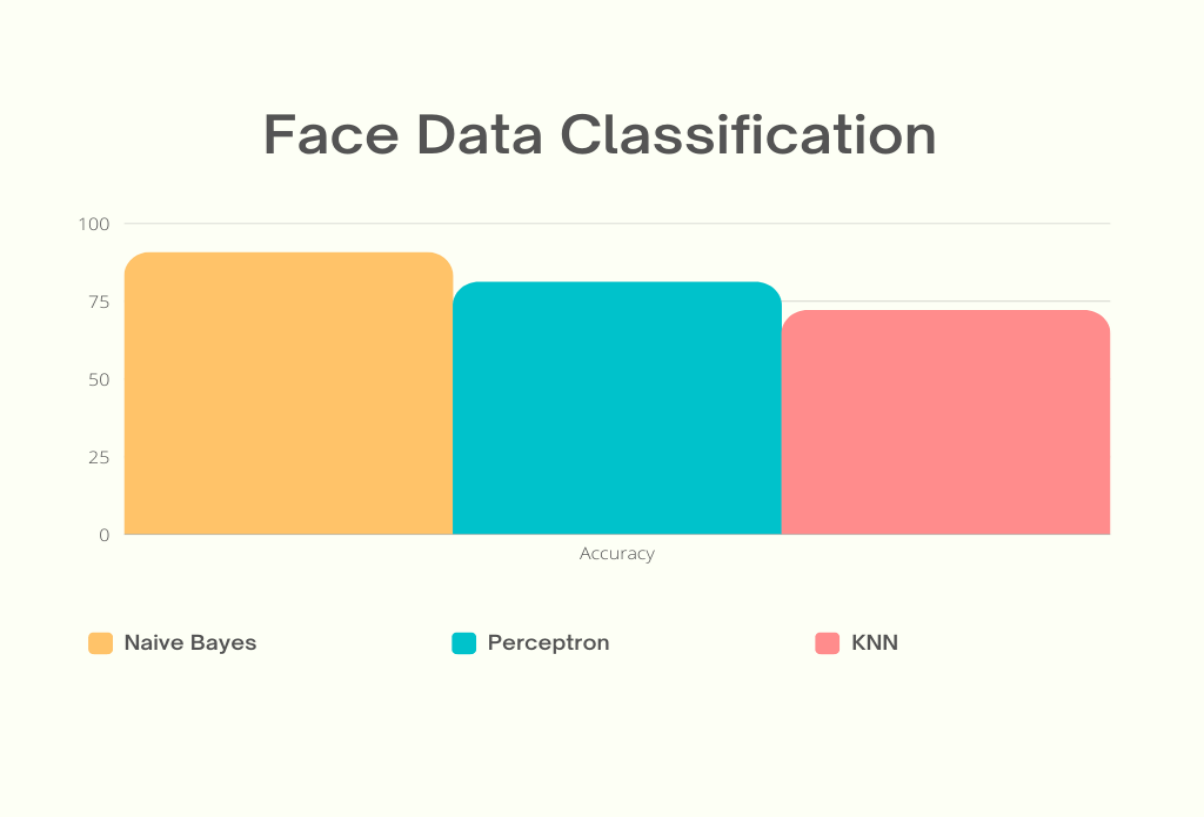






COMPARISION OF ACCURACIES OF OUR ALGORITHMS:

1. **FACE DATA CLASSIFICATION:**



2. DIGIT DATA CLASSIFICATION: