

# **CS 520: Introduction to Artificial Intelligence**

## **FINAL PROJECT: IMAGE CLASSIFICATION**



### **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.

$$P(y_i | x_1, x_2, \dots, x_n) = P(x_1, x_2, \dots, x_n | y_i) * P(y_i) / P(x_1, x_2, \dots, x_n)$$

### **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 threeed\_feature list.
- Secondly, we computed likelihoods of each input feature with respective to their class and stored them in a threeed\_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
calc_m_std()
Time taken for 10 % data is [419.76261138916016, 420.12572288513184, 395.62225341796875, 401.8564224243164, 390.37394523620605]
mean time for 10 % data is 405.54819107055664
accuracy_list for 10 % data is [66.10000000000001, 63.80000000000004, 67.4, 63.2, 65.60000000000001]
mean accuracies for 10 % data is 65.22
standard deviation of accuracies for 10 % data is 1.53414471286121

Time taken for 20 % data is [953.3977508544922, 887.845516204834, 923.9652156829834, 947.2634792327881, 912.9443168640137]
mean time for 20 % data is 925.0832557678223
accuracy_list for 20 % data is [71.89999999999999, 70.6, 70.89999999999999, 71.7, 69.6]
mean accuracies for 20 % data is 70.93999999999998
standard deviation of accuracies for 20 % data is 0.8260750571225363

Time taken for 30 % data is [1794.586420059204, 1749.976396560669, 1789.2215251922607, 1755.824327468872, 1778.6519527435303]
mean time for 30 % data is 1773.6521244049072
accuracy_list for 30 % data is [73.1, 73.1, 72.3, 72.8, 72.7]
mean accuracies for 30 % data is 72.8
standard deviation of accuracies for 30 % data is 0.296647939483825

Time taken for 40 % data is [2977.243185043335, 2889.6214962005615, 2887.7830505371094, 3011.772632598877, 2918.3685779571533]
mean time for 40 % data is 2936.957788467407
accuracy_list for 40 % data is [73.8, 74.0, 74.2, 74.0, 74.3]
mean accuracies for 40 % data is 74.06
standard deviation of accuracies for 40 % data is 0.17435595774162746

Time taken for 50 % data is [4483.0322265625, 4536.591053009033, 5741.463899612427, 4520.827770233154, 4419.162034988403]
mean time for 50 % data is 4740.2153968811035
accuracy_list for 50 % data is [74.6, 73.2, 75.7, 75.6, 74.5]
mean accuracies for 50 % data is 74.72
standard deviation of accuracies for 50 % data is 0.9064215354899715

```

Naive\_Bayes.ipynb

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

+ Code + Text

```

[ ] 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

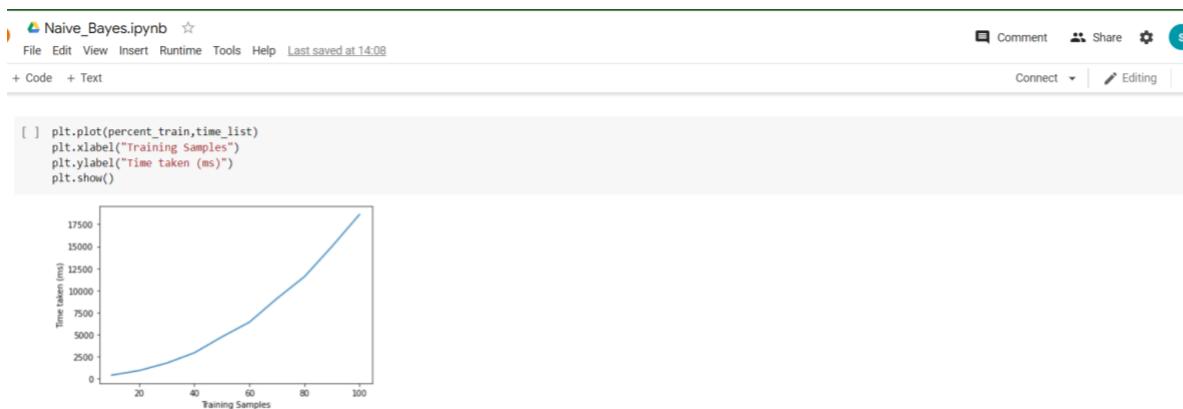
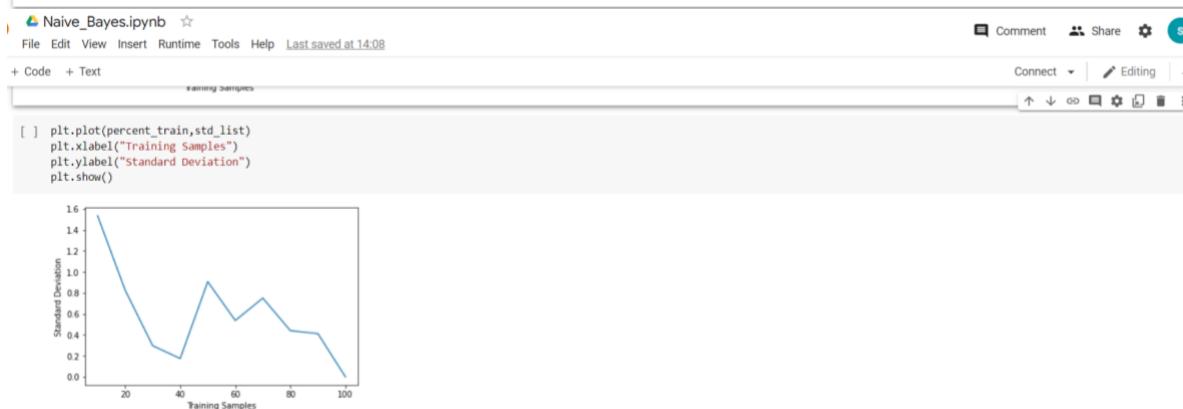
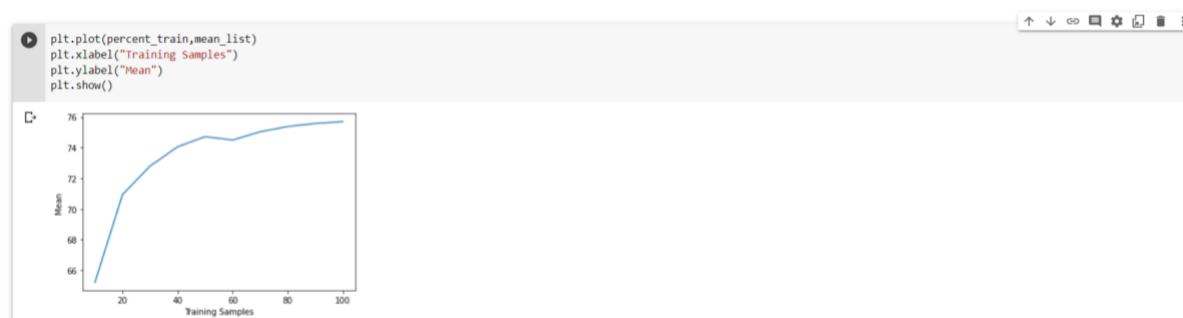
Time taken for 70 % data is [8840.784311294556, 8835.535049438477, 8831.073760986328, 10095.984697341919, 8850.327014923096]
mean time for 70 % data is 9090.740966796875
accuracy_list for 70 % data is [76.2, 74.7, 73.9, 75.2, 75.2]
mean accuracies for 70 % data is 75.03999999999999
standard deviation of accuracies for 70 % data is 0.749933303701061

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, 75.6, 74.9, 74.8]
mean accuracies for 80 % data is 75.38000000000001
standard deviation of accuracies for 80 % data is 0.4399999999999978

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, 18441.87378833618]
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

```



## **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 threeed\_feature list.
- Secondly, we computed likelihoods of each input feature with respective to their class and stored them in a threeed\_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:



The screenshot shows a Jupyter Notebook interface with the file NaiveBayesFace.ipynb open. The code cell [ ] contains the function definition for calc\_m\_std(). The output shows results for data sizes 10%, 20%, 30%, 40%, and 50%.

```

NaiveBayesFace.ipynb ☆
File Edit View Insert Runtime Tools Help Last saved at 14:08
Comment Share ⚙ $ Connect ▾ | ✎ Editing ▾
+ Code + Text

[ ] calc_m_std()
Time taken for 10 % data is [11.482715606689453, 6.928443908691406, 6.529331207275391, 7.038116455078125, 6.701231002807617]
mean time for 10 % data is 7.735967636108398
accuracy_list for 10 % data is [56.00000000000001, 51.3333333333333, 74.666666666666667, 57.3333333333333, 52.666666666666664]
mean accuracies for 10 % data is .58.4
standard deviation of accuracies for 10 % data is 8.417970196087788

Time taken for 20 % data is [10.497093200683594, 9.13381576538086, 8.687973022460938, 8.621692657470703, 8.592605590820312]
mean time for 20 % data is 9.106636047363281
accuracy_list for 20 % data is [71.3333333333334, 68.666666666666667, 77.3333333333333, 74.0, 74.0]
mean accuracies for 20 % data is .73.066666666666666
standard deviation of accuracies for 20 % data is 2.90898972361867

Time taken for 30 % data is [14.06007412109375, 22.989273071289062, 11.994600296020508, 15.41757583618164, 23.835182189941406]
mean time for 30 % data is 17.659521102905273
accuracy_list for 30 % data is [78.666666666666666, 77.3333333333333, 75.3333333333333, 79.3333333333333, 84.0]
mean accuracies for 30 % data is .78.9333333333332
standard deviation of accuracies for 30 % data is 2.8728710859897223

Time taken for 40 % data is [27.053356170654297, 17.3490047454834, 29.547452926635742, 16.57891273498535, 16.665935516357422]
mean time for 40 % data is 21.43893418823242
accuracy_list for 40 % data is [87.3333333333333, 82.0, 86.666666666666667, 80.666666666666666, 79.3333333333333]
mean accuracies for 40 % data is .83.19999999999999
standard deviation of accuracies for 40 % data is 3.2221455929585554

Time taken for 50 % data is [24.342060089111328, 22.12667465209961, 22.224903106689453, 25.638580322265625, 22.073745727539062]
mean time for 50 % data is 23.281192779541016
accuracy_list for 50 % data is [90.666666666666666, 82.666666666666667, 84.0, 91.3333333333333, 84.0]
mean accuracies for 50 % data is .86.5333333333333
standard deviation of accuracies for 50 % data is 3.6854066562893966

```

NaiveBayesFace.ipynb

```

File Edit View Insert Runtime Tools Help Last saved at 14:08
+ Code + Text
[ ] 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, 88.666666666666667, 87.33333333333333, 80.0]
mean accuracies for 60 % data is 83.866666666666666
standard deviation of accuracies for 60 % data is 3.461534662865913

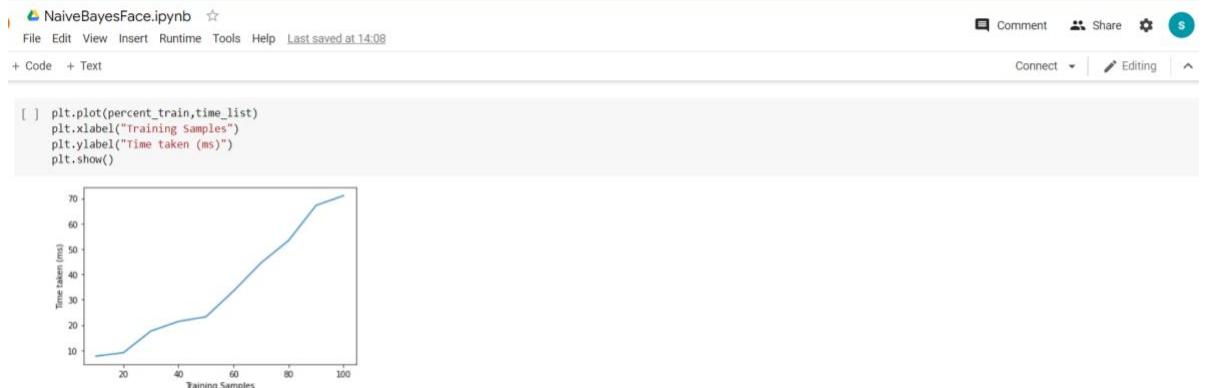
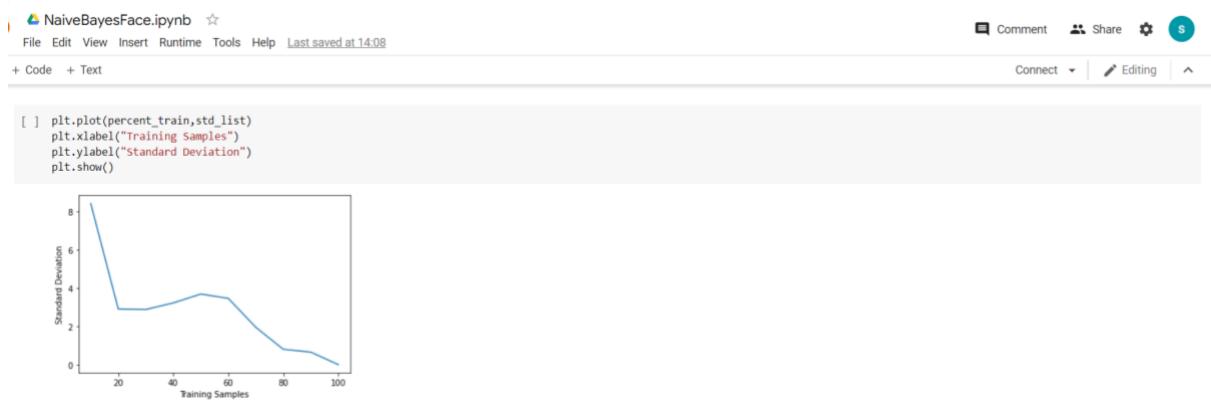
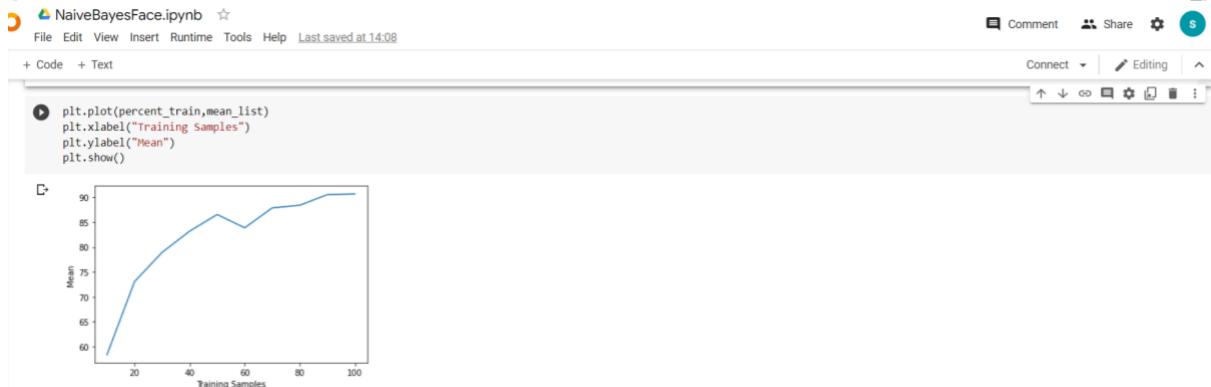
Time taken for 70 % data is [62.63399124145508, 43.98465156555176, 38.054704666137695, 41.36514663696289, 36.742210388183594]
mean time for 70 % data is 44.5561408996582
accuracy_list for 70 % data is [86.666666666666667, 88.666666666666667, 89.33333333333333, 90.0, 84.666666666666667]
mean accuracies for 70 % data is 87.866666666666667
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 0.7999999999999999

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 0.6531972647421802

```



## 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:**

$$F(x_i, W) = W_0 + W_1 \phi_1(x_i) + w_2 \phi_2(x_i) + \dots + w_L \phi_L(x_i)$$

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  
 $F(x_i, w) > 0$  and  $y=false$  if  $F(x_i, w) < 0$

### **Our Approach:**

1. We Initialize the weights {  $W_j$  } 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  $W_0=0$  at the beginning as our base weight.
2. We then perform feature extraction on our training set  $X_{train}$  and pass the initial weights and  $X_{train}$  features to the linear decision function.
3. Then the decision function calculates the value  $F(x_i, W)$  for the each given image and based on the scores it chooses to perform the following actions.

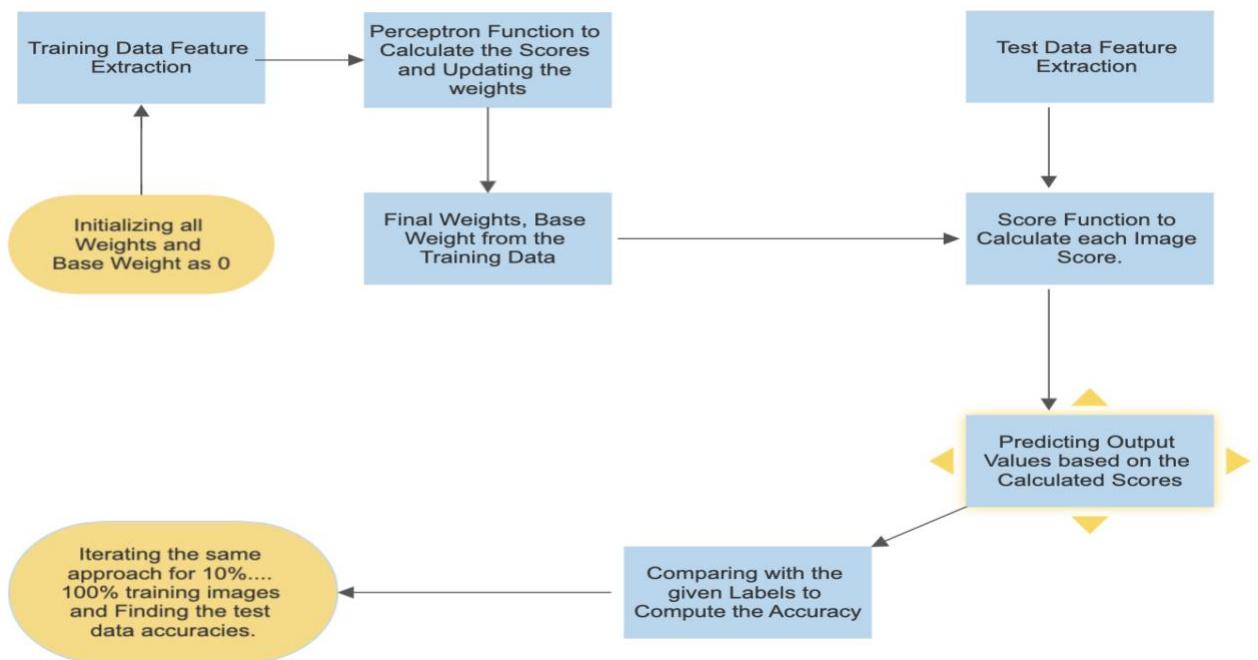
### **Face Data Classification:**

1.  $F(x_i, w) > 0$  and  $y_i=True$  or  $F(x_i, w) < 0$  and  $y_i=False$  then we don't perform any action.
2. If  $F(x_i, w) < 0$  and  $y_i=true$  then  $w_j = w_j + \phi_j(x_j)$   
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  $w_0=w_0+1$ .
3. If  $F(x_i, w)>0$  and  $y_i=false$  then  $w_j = w_j - \phi_j(x_j)$   
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  $w_0=w_0-1$ .
4. We run this complete iteration for few number of times and stop the process.
5. We then return the final weights {  $W_j$  for  $j=1..n$  } from the given training images and return the final base weight  $W_0$ .
6. With this data we again calculate the scores for the test images and compare with the  $y$  label, if  $F(x_i, w)>0$  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  $F(x_i, w)$  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.,  $W_j$  to  $W_{j-1}$  except for the corresponding label value position in the base weights list, where we add  $W_i = W_{i+1}$  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**

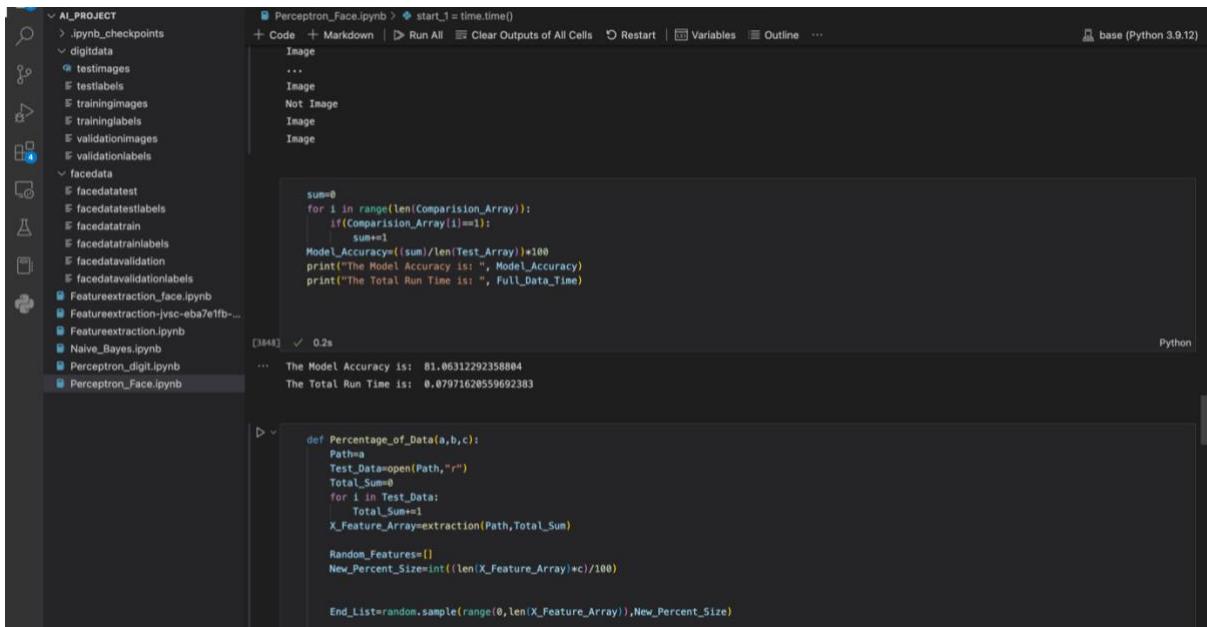


## Results and Accuracies:

### 1. Face Data Classification:

Model Accuracy for 100% of Training Data is: “81.063%”

100% Training Data Run Time: “0.0797” Sec



The screenshot shows a Jupyter Notebook interface with a sidebar containing project files like AI\_PROJECT, digitdata, and faceadata. The main area displays a Python script named Perceptron\_Face.ipynb. The code calculates accuracy and run time, with output cells showing the results.

```
AI_PROJECT
  .ipynb_checkpoints
  digitdata
    testimages
    testlabels
    trainingimages
    traininglabels
    validationimages
    validationlabels
  faceadata
    facetedatest
    facetedatatestlabels
    facetedatatrain
    facetedatatrainlabels
    facetedatavalidation
    facetedatavalidationlabels
  Featureextraction_face.ipynb
  Featureextraction-jvsc-e8a7e1fb...
  Featureextraction.ipynb
  Naive_Bayes.ipynb
  Perceptron_digit.ipynb
  Perceptron_Face.ipynb

Perceptron_Face.ipynb
  start_1 = time.time()
  Image
  ...
  Image
  Not Image
  Image
  Image

sum=0
for i in range(len(Comparision_Array)):
    if(Comparision_Array[i]==1):
        sum+=1
Model_Accuracy=(sum/len(Test_Array))*100
print("The Model Accuracy is: ", Model_Accuracy)
print("The Total Run Time is: ", Full_Data_Time)

[3848] ✓ 0.2s
... The Model Accuracy is: 81.06312292358804
The Total Run Time is: 0.07971620559692383
```

```
def Percentage_of_Data(a,b,c):
    Path=a
    Test_Data=open(Path,"r")
    Total_Sum=0
    for i in Test_Data:
        Total_Sum+=1
    X_Feature_Array=extracion(Path,Total_Sum)

    Random_Features=[]
    New_Percent_Size=int((len(X_Feature_Array)*c)/100)

    End_List=random.sample(range(0,len(X_Feature_Array)),New_Percent_Size)
```

### 2. Digit Data Classification:

Model Accuracy for 100% of Training Data is: “79%”

100% Training Data Run Time: “9.632” Sec

```

    Accuracy_list=[]
    for i in range(len(all_Scores_list)):
        max=all_Scores_list[i][0]
        index=0
        for j in range(10):
            if(all_Scores_list[i][j]>max):
                max=all_Scores_list[i][j]
                index=j
        if(index==Label_list[i]):
            Accuracy_list.append(1)
        else:
            Accuracy_list.append(0)

Totals=0
for i in Accuracy_list:
    if i==1:
        Totals+=1
Accuracy= (Totals)/len(Accuracy_list)*100
print("The Model Accuracy is:", Accuracy)
print("The Total Run Time is:", Full_Data_Time)

```

The Model Accuracy is: 79.0  
The Total Run Time is: 9.63283249206543

## ACCURACIES AND RUNTIMES FOR VARIOUS PERCENTAGE OF TRAINING DATA

### 1. Face Data Classification:

10% (5 Iterations)

```

Mean=np.sum(ele)/5
SD=np.std(final_accu_List)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

```

[1925] ✓ 1.1s

Model Accuracy is: 71.42857142857142  
Model Accuracy is: 64.45182724252491  
Model Accuracy is: 70.89966777408638  
Model Accuracy is: 69.1029903322258  
Model Accuracy is: 64.45182724252491  
The Mean of the data is: 67.90697674418604  
The Standard Deviation of the data is: 2.9166276754141803  
The times taken by the data are: [0.006757259368896484, 0.006865024566650391, 0.006926059722900391, 0.005988014801825391, 0.0067501068115234375] Sec  
Average Run Time is: 0.006655693054199219

20% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with a sidebar labeled "EXPLORER" containing files like "AI\_PROJECT.ipynb", "digitdata.ipynb", "facedata.ipynb", and several feature extraction and perceptron-related files. The main area displays code in "Perceptron\_Face.ipynb" and its output. The code calculates mean, standard deviation, and average run time based on a list of accuracy values. The output shows results for 5 iterations.

```
Mean=sum_ele/5
SD=np.std(final_accu_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[1981]: 1.1s
...
Model Accuracy is: 62.7906976744186
Model Accuracy is: 66.11285681065123
Model Accuracy is: 73.421926910299
Model Accuracy is: 73.75415262392026
Model Accuracy is: 69.18299803322258
The Mean of the data is: 69.03654485049833
The Standard Deviation of the data is: [0.014004945755004883, 0.0136325798034668, 0.013222932815551758, 0.014802217483520588, 0.014150857925415039] Sec
Average Run Time is: 0.013963842391967773
```

30% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with a sidebar labeled "EXPLORER" containing files like "AI\_PROJECT.ipynb", "digitdata.ipynb", "facedata.ipynb", and several feature extraction and perceptron-related files. The main area displays code in "Perceptron\_Face.ipynb" and its output. The code calculates mean, standard deviation, and average run time based on a list of accuracy values. The output shows results for 5 iterations.

```
sum_time=0
for i in range(len(final_accu_list)):
    sum_time+=final_accu_list[i]

avg_time=sum_time/5

mean=sum_ele/5
SD=np.std(final_accu_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[3213]: 1.2s
...
Model Accuracy is: 68.18631229235881
Model Accuracy is: 77.40863787375415
Model Accuracy is: 72.42524916943522
Model Accuracy is: 73.08970099667755
Model Accuracy is: 62.7906976744186
The Mean of the data is: 70.7641196013280
The Standard Deviation of the data is: 4.958965623134961
The times taken by the data are: [0.0224151611328125, 0.020889759063720703, 0.02314615249633789, 0.022203922271728516, 0.021715879448387617] Sec
Average Run Time is: 0.022074174880981446
```

40% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER:** Shows the project structure under AI\_PROJECT, including digitdata, facedata, and various .ipynb files.
- CELLS:** The current cell contains Python code for calculating accuracy and run times across 5 iterations. The output shows the following results:

```
for j in range(len(Comparision_Array_1_c)):
    if(Comparision_Array_1_c[j]==1):
        sums+=1
Model_Accuracy_c=(sums)/len(Comparision_Array_1_c)*100
print("Model Accuracy is:", Model_Accuracy_c)

final_accu_list.append(Model_Accuracy_c)

sum_ele=0
for i in range(len(final_accu_list)):
    sum_ele+=final_accu_list[i]

sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_accu_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

... Model Accuracy is: 72.09382325581395
Model Accuracy is: 72.42524916943522
Model Accuracy is: 64.45182724252491
Model Accuracy is: 74.08637873754152
Model Accuracy is: 75.08385647848653
The Mean of the data is: 71.62798697674419
The Standard Deviation of the data is: 3.7504759579189617
The times taken by the data are: [0.02989120330810547, 0.029450178146362305, 0.0307769775390625, 0.029754161834716797, 0.02915787696838379] Sec
Average Run Time is: 0.029887662963867106
```

50% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER:** Shows the project structure under AI\_PROJECT, including digitdata, facedata, and various .ipynb files.
- CELLS:** The current cell contains Python code for calculating accuracy and run times across 5 iterations. The output shows the following results:

```
final_accu_list.append(Model_Accuracy_c)

sum_ele=0
for i in range(len(final_accu_list)):
    sum_ele+=final_accu_list[i]

sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_accu_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

... Model Accuracy is: 67.109634551495
Model Accuracy is: 73.0897089966775
Model Accuracy is: 77.40863787375415
Model Accuracy is: 78.7641196013289
Model Accuracy is: 71.09634551495017
The Mean of the data is: 71.89368778764119
The Standard Deviation of the data is: 3.365827288242989
The times taken by the data are: [0.03549814224243164, 0.03612995147705078, 0.03682899475097656, 0.034757137298583984, 0.03610396385192871] Sec
Average Run Time is: 0.035863637924194336
```

60% (5 Iterations)

```

for j in range(len(Comparision_Array_1_c)):
    if(Comparision_Array_1_c[j]==1):
        sum+=1
Model_Accuracy_c=(sum)/len(Comparision_Array_1_c)*100
print("Model Accuracy is:", Model_Accuracy_c)

final_accu_list.append(Model_Accuracy_c)

sum_ele=0
for i in range(len(final_accu_list)):
    sum_ele+=final_accu_list[i]

sum_time=0
for i in range(len(Times_list)):
    sum_time+=times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_accu_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[3521] ✓ 1.3s
...
Model Accuracy is: 70.7641196013289
Model Accuracy is: 72.4254916943522
Model Accuracy is: 76.74418604651163
Model Accuracy is: 68.43853820598007
Model Accuracy is: 71.42857142857143
The Mean of the data is: 71.96013280036546
The Standard Deviation of the data is: 2.72830189565204
The times taken by the data are: [0.84287314414978027, 0.8442349910736084, 0.843727874755859375, 0.84555487632751465, 0.8465769776712305] Sec
Average Run Time is: 0.84459357261657715

```

70%(5 Iterations)

```

for i in range(len(final_accu_list)):
    sum_ele+=final_accu_list[i]

sum_time=0
for i in range(len(Times_list)):
    sum_time+=times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_accu_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[2849] ✓ 1.3s
...
Model Accuracy is: 78.07388970099668
Model Accuracy is: 74.4186046511628
Model Accuracy is: 78.43189358778764
Model Accuracy is: 71.42857142857143
Model Accuracy is: 77.0764119601329
The Mean of the data is: 74.28571428571429
The Standard Deviation of the data is: 3.005497110404955
The times taken by the data are: [0.8539664537963867, 0.85878387268437012, 0.85282007293701172, 0.85249198330505371, 0.85231285095214844] Sec
Average Run Time is: 0.85231494983564453

```

80%(5 Iterations)

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER:** Shows the project structure under AI\_PROJECT, including subfolders like digitdata, facedata, and various .ipynb files.
- PERCELLS:** The current cell contains Python code for calculating accuracy and run times across 5 iterations.
- OUTPUT:** The output pane displays the results of the executed code, including Mean, Standard Deviation, and Average Run Time.

```

final_accu_list.append(Model_Accuracy_c)

sum_ele=0
for i in range(len(final_accu_list)):
    sum_ele+=final_accu_list[i]

sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_accu_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[3605] ✓ 1.4s
Model Accuracy is: 78.07308970099668
Model Accuracy is: 69.767441866046511
Model Accuracy is: 69.43521594684385
Model Accuracy is: 77.0764119601329
Model Accuracy is: 78.7375415282392
The Mean of the data is: 74.61794019933555
The Standard Deviation of the data is: 4.1313731712185575
The times taken by the data are: [0.061888933181762695, 0.06497478485107422, 0.06526613235473633, 0.06394815444946289, 0.0598139762878418] Sec
Average Run Time is: 0.0631783962249759

```

90% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER:** Shows the project structure under AI\_PROJECT, including subfolders like digitdata, facedata, and various .ipynb files.
- PERCELLS:** The current cell contains Python code for calculating accuracy and run times across 5 iterations.
- OUTPUT:** The output pane displays the results of the executed code, including Mean, Standard Deviation, and Average Run Time.

```

final_accu_list.append(Model_Accuracy_c)

sum_ele=0
for i in range(len(final_accu_list)):
    sum_ele+=final_accu_list[i]

sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_accu_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by the data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[3775] ✓ 1.4s
Model Accuracy is: 76.0797342192691
Model Accuracy is: 79.06976744186846
Model Accuracy is: 76.41196813289037
Model Accuracy is: 77.0764119601329
Model Accuracy is: 71.42857142857143
The Mean of the data is: 76.01328903654485
The Standard Deviation of the data is: 2.516158967277274
The times taken by the data are: [0.06575798988342285, 0.0674271583571289, 0.06785368919372559, 0.0696709156036377, 0.06817889213562012] Sec
Average Run Time is: 0.06777772983442383

```

## 2. Digit Data Classification:

10% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER** pane on the left showing files in the AI\_PROJECT directory, including Perceptron\_digital.ipynb, Perceptron\_Face.ipynb, and Perceptron\_Face.ipynb (output).
- Code Editor** pane containing Python code to calculate mean, standard deviation, and average run time from lists of accuracy and times.
- Output pane** showing the results of the code execution:

```
sum_ele=0
for i in range(len(final_acc_list)):
    sum_ele+=final_acc_list[i]

sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_acc_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[190]  ✓  8.8s
...
Model Accuracy is: 67.0
Model Accuracy is: 72.3
Model Accuracy is: 71.8
Model Accuracy is: 72.0
Model Accuracy is: 76.3
The Mean of the data is: 71.88000000000001
The Standard Deviation of the data is: 2.951203144481924
The times taken by data are: [0.9884121417999268, 0.931603193283081, 0.9584078788757324, 0.9428789615631104, 0.94084111979675293] Sec
Average Run Time is: 0.952428674697876
```

20% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER** pane on the left showing files in the AI\_PROJECT directory, including Perceptron\_digital.ipynb, Perceptron\_Face.ipynb, and Perceptron\_Face.ipynb (output).
- Code Editor** pane containing Python code to calculate mean, standard deviation, and average run time from lists of accuracy and times.
- Output pane** showing the results of the code execution:

```
Mean=mean_ele/S
SD=np.std(final_acc_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[280]  ✓  12.7s
...
Model Accuracy is: 75.3
Model Accuracy is: 73.0
Model Accuracy is: 75.5
Model Accuracy is: 75.6
Model Accuracy is: 75.1
The Mean of the data is: 74.9
The Standard Deviation of the data is: 0.9654014780917723
The times taken by data are: [1.911448955538887, 1.888302906636377, 1.8931729793548584, 1.8749561309814453, 1.886752843856815] Sec
Average Run Time is: 1.8893267631530761
```

30% (5 Iterations)

The screenshot shows the Jupyter Notebook interface with the code cell containing the following Python script:

```
Perceptron_digit.ipynb • Perceptron_Face.ipynb • Perceptron_Face.ipynb (output)
Perceptron_digit.ipynb > path_images="/Users/rohitgummadi/Desktop/AI_Project/digitdata/trainingimages" #For percent of data
+ Code + Markdown | Run All | Clear Outputs of All Cells | Restart | Variables | Outline ... | W_Base Aa ab * 2 of 12 ↑ ↓ ×
Avg_time=0
sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/len(Times_list)

Mean=sum_ele/5
SD=np.std(final_acc_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[2821] ✓ 17.5s
```

The output of the code cell is:

```
Model Accuracy is: 74.2
Model Accuracy is: 75.5
Model Accuracy is: 75.7
Model Accuracy is: 76.1
Model Accuracy is: 76.2
The Mean of the data is: 75.53999999999999
The Standard Deviation of the data is: 0.717216843081643
The times taken by data are: [2.853739023208618, 2.859712138788496, 2.8733439445495605, 2.829357624053955, 2.8137521743774414] Sec
Average Run Time is: 2.845980978812085
```

40% (5 Iterations)

The screenshot shows the Jupyter Notebook interface with the code cell containing the following Python script:

```
Perceptron_digit.ipynb • Perceptron_Face.ipynb • Perceptron_Face.ipynb (output)
Perceptron_digit.ipynb > path_images="/Users/rohitgummadi/Desktop/AI_Project/digitdata/trainingimages" #For percent of data
+ Code + Markdown | Run All | Clear Outputs of All Cells | Restart | Variables | Outline ... | W_Base Aa ab * 2 of 12 ↑ ↓ ×
[2841] ✓ 22.4s
sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/len(Times_list)

Mean=sum_ele/5
SD=np.std(final_acc_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[2841] ✓ 22.4s
```

The output of the code cell is:

```
Model Accuracy is: 74.4
Model Accuracy is: 75.6
Model Accuracy is: 76.4
Model Accuracy is: 76.3
Model Accuracy is: 76.9
The Mean of the data is: 75.92
The Standard Deviation of the data is: 0.8657944328765347
The times taken by data are: [3.937098979949951, 3.797266244883057, 3.756704092025757, 3.7810591594696045, 3.8243210315704346] Sec
Average Run Time is: 3.8194499015880185
```

50% (5 Iterations)

The screenshot shows the Jupyter Notebook interface with the code cell containing the following Python script:

```
Perceptron_digit.ipynb • Perceptron_Face.ipynb • Perceptron_Face.ipynb (output)
Perceptron_digit.ipynb > path_images="/Users/rohitgummadi/Desktop/AI_Project/digitdata/trainingimages" #For percent of data
+ Code + Markdown | Run All | Clear Outputs of All Cells | Restart | Variables | Outline ... | W_Base Aa ab * 2 of 12 ↑ ↓ ×
sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/len(Times_list)

Mean=sum_ele/5
SD=np.std(final_acc_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[2881] ✓ 27.9s
```

The output of the code cell is:

```
Model Accuracy is: 76.9
Model Accuracy is: 76.9
Model Accuracy is: 76.8
Model Accuracy is: 75.7
Model Accuracy is: 76.5
The Mean of the data is: 76.56
The Standard Deviation of the data is: 0.45431266766402223
The times taken by data are: [4.838051110839844, 4.7880678150177, 4.736063003540039, 4.71144700050354, 4.703503847122192] Sec
Average Run Time is: 4.754166555404663
```

60% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER:** Shows the project structure under "AI\_PROJECT" with files like "Perceptron\_digital.ipynb", "Perceptron\_Face.ipynb", and "Perceptron\_Face.ipynb (output)".
- Code Cell:** Contains Python code to calculate accuracy and run times.
- Output:** Displays the results of the code execution, including Model Accuracy and Average Run Time.

```

Totals_final=1
Accuracy_final= (Totals_final)/len(accuracy_list_final)*100
print("Model Accuracy is:", Accuracy_final)

final_acc_list.append(Accuracy_final)

sum_ele=0
for i in range(len(final_acc_list)):
    sum_ele+=final_acc_list[i]

sum_time=0
for i in range(len(times_list)):
    sum_time+=times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_acc_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[21M] ✓ 36.5s
...
Model Accuracy is: 76.0
Model Accuracy is: 77.0
Model Accuracy is: 76.7
Model Accuracy is: 77.4
Model Accuracy is: 77.3
The Mean of the data is: 76.88000000000001
The Standard Deviation of the data is: 0.5035871324805673
The times taken by data are: [6.805253982543945, 6.604861126708984, 6.595972776412964, 6.605532169342041, 6.595189094543457] Sec
Average Run Time is: 6.641201829910278
  
```

70% (5 Iterations)

The screenshot shows a Jupyter Notebook interface with the following details:

- EXPLORER:** Shows the project structure under "AI\_PROJECT" with files like "Perceptron\_digital.ipynb", "Perceptron\_Face.ipynb", and "Perceptron\_Face.ipynb (output)".
- Code Cell:** Contains Python code to calculate accuracy and run times.
- Output:** Displays the results of the code execution, including Model Accuracy and Average Run Time.

```

sum_time=0
for i in range(len(Times_list)):
    sum_time+=Times_list[i]
Avg_time=sum_time/5

Mean=sum_ele/5
SD=np.std(final_acc_list)

print("The Mean of the data is:", Mean)
print("The Standard Deviation of the data is:", SD)
print("The times taken by data are:", Times_list, "Sec")
print("Average Run Time is:", Avg_time)

[22M] ✓ 36.5s
...
Model Accuracy is: 76.6
Model Accuracy is: 77.4
Model Accuracy is: 76.7
Model Accuracy is: 77.8
Model Accuracy is: 77.8
The Mean of the data is: 77.26
The Standard Deviation of the data is: 0.52
The times taken by data are: [6.589876413345337, 6.636267900466919, 6.657549858093262, 6.722577333458317, 6.632739067077637] Sec
Average Run Time is: 6.647882114486694
  
```

80% (5 Iterations)

```

    sum_ele=0
    for i in range(len(times_list)):
        sum_time+=times_list[i]
    Avg_time=sum_time/5

    Mean=sum_ele/5
    SD=np.std(final_acc_list)

    print("The Mean of the data is:", Mean)
    print("The Standard Deviation of the data is:", SD)
    print("The times taken by data are:", Times_list, "Sec")
    print("Average Run Time is:", Avg_time)

[230] 46.4s
...
Model Accuracy is: 77.4
Model Accuracy is: 77.60000000000001
Model Accuracy is: 77.60000000000001
Model Accuracy is: 77.4
Model Accuracy is: 77.8
The Mean of the data is: 77.56
The Standard Deviation of the data is: 0.1496662954709522
The times taken by data are: [8.611618995666504, 8.477760791778564, 8.6039056396484, 8.465496063232422, 8.551790714263916] Sec
Average Run Time is: 8.542121124267577

```

90% (5 Iterations)

```

    Mean=sum_ele/5
    SD=np.std(final_acc_list)

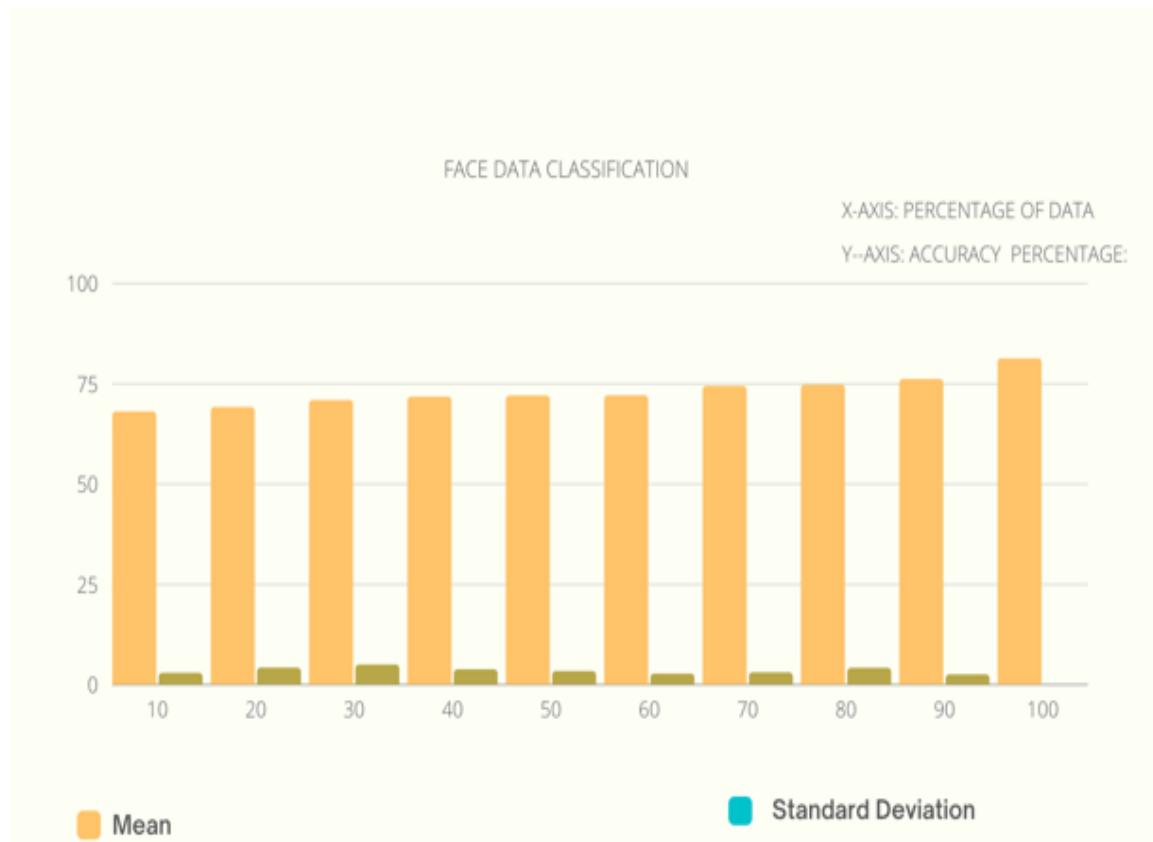
    print("The Mean of the data is:", Mean)
    print("The Standard Deviation of the data is:", SD)
    print("The times taken by data are:", Times_list, "Sec")
    print("Average Run Time is:", Avg_time)

[230] 49.8s
...
Model Accuracy is: 78.5
Model Accuracy is: 77.5
Model Accuracy is: 78.0
Model Accuracy is: 78.10000000000001
Model Accuracy is: 77.9
The Mean of the data is: 78.0
The Standard Deviation of the data is: 0.32249030993194217
The times taken by data are: [9.417511940002441, 9.24882698859882, 9.329564094543457, 9.257109888447388, 9.338305950164795] Sec
Average Run Time is: 9.31826376914978

```

## Graphs for Mean of Accuracies and Standard Deviation:

## **FACE DATA:**

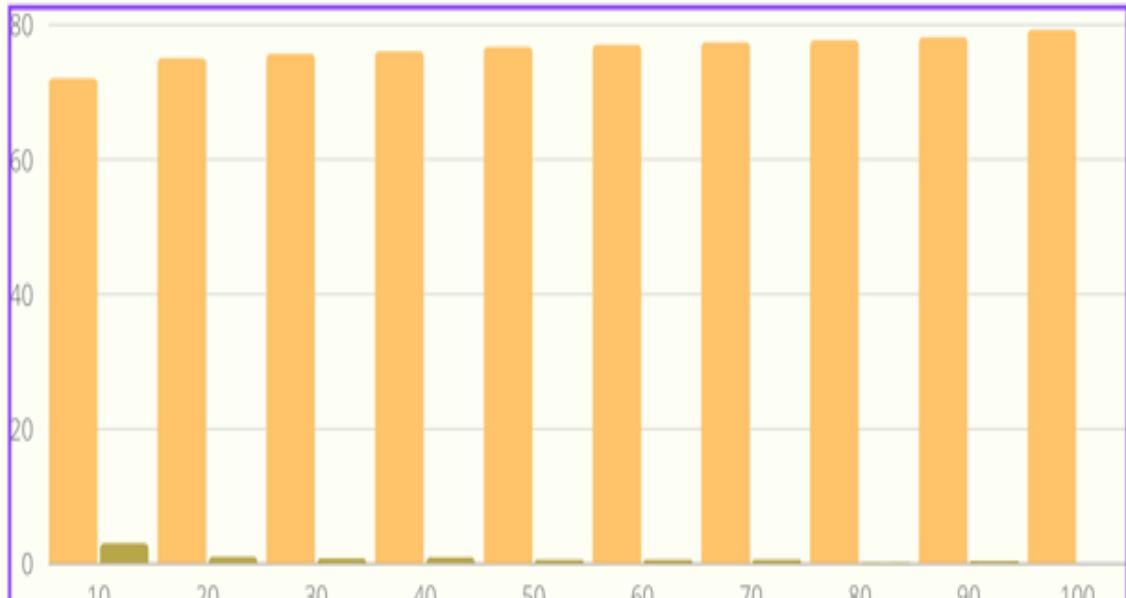


## **DIGIT DATA:**

### DIGIT DATA CLASSIFICATION

X-AXIS: PERCENTAGE OF DATA

Y-AXIS: ACCURACY PERCENTAGE:

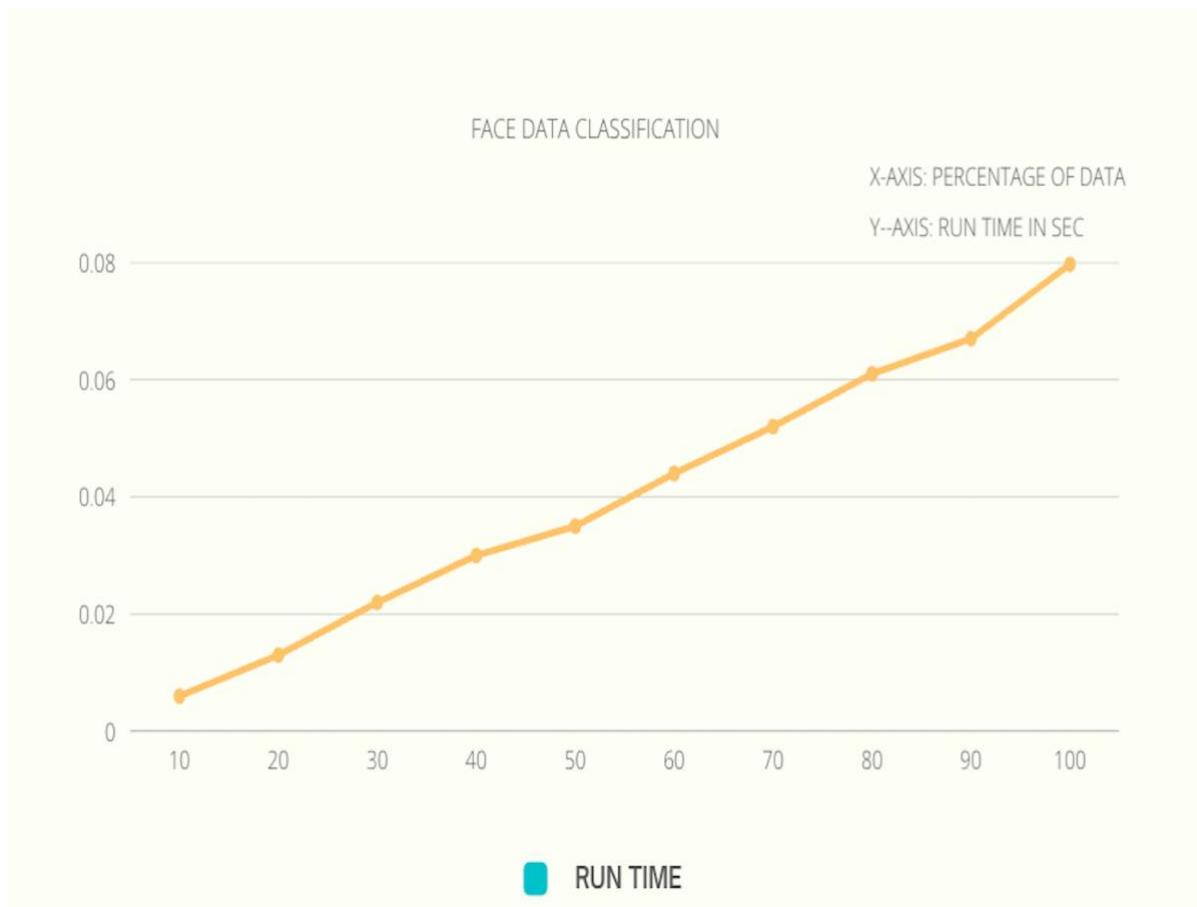


Mean

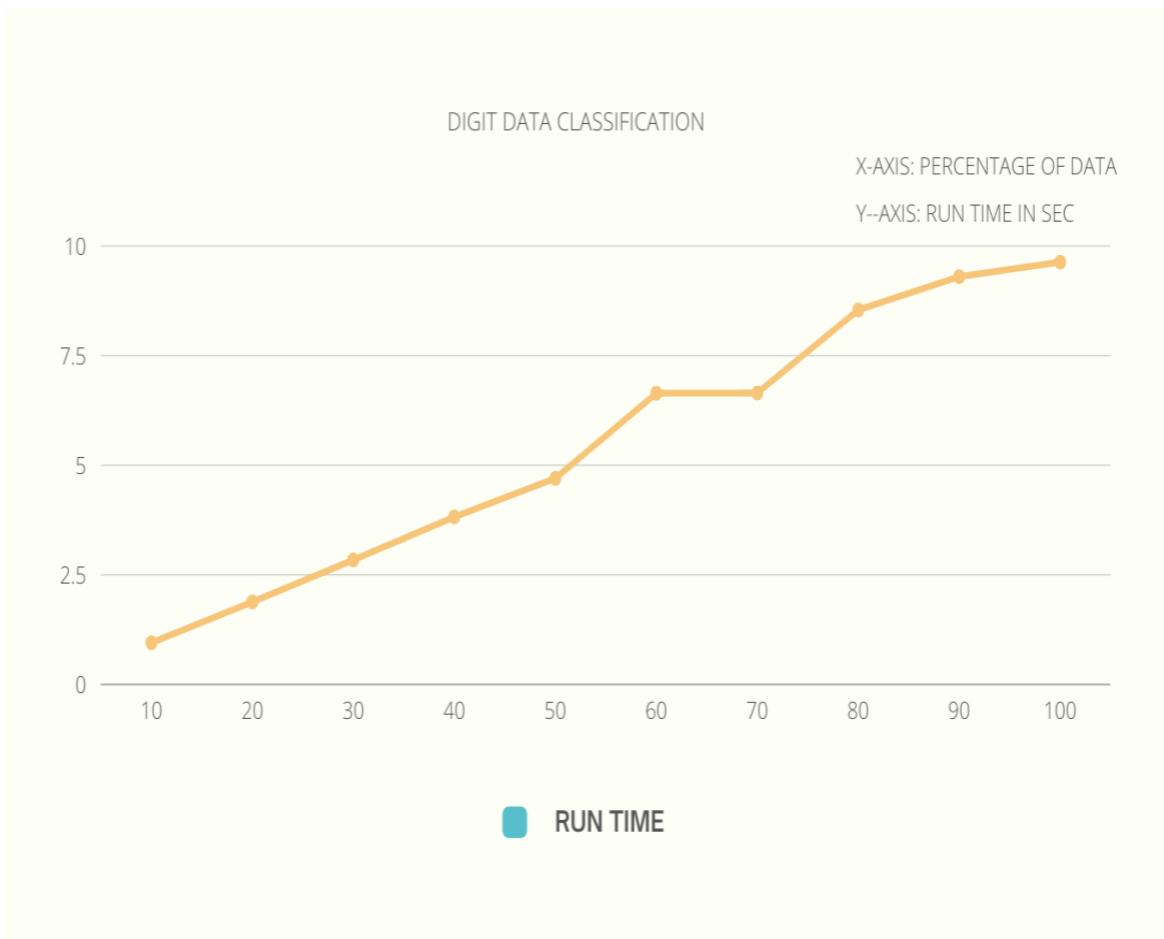
Standard Deviation

### Graph for Mean Run Times:

Face Data:



**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

+ Code + Text

```
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 74.2754459381035
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 0.22803508501983083
```

KnnDigit.ipynb ☆

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

+ Code + Text

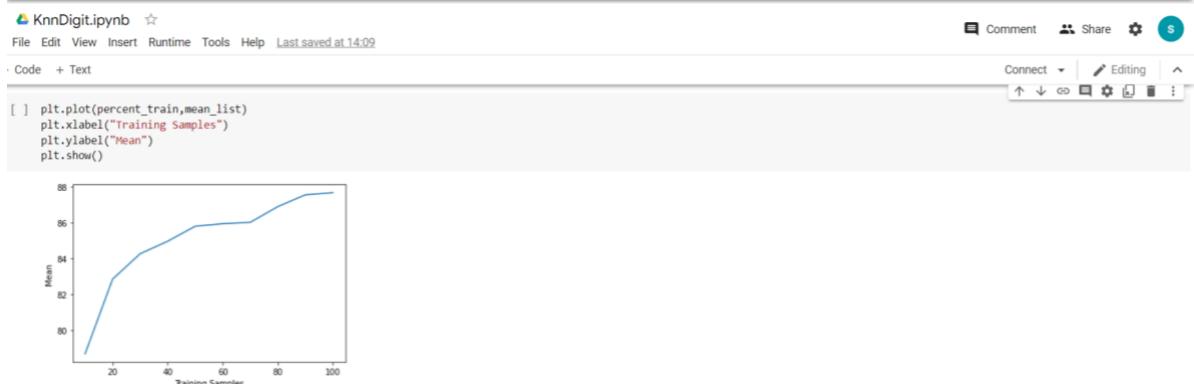
```
Time taken for 60 % data is [167.89555549621582, 169.1553926818848, 153.50985527038574, 146.63243293762207, 141.60799980163574]
mean time for 60 % data is 155.76024055408957
accuracy_list for 60 % data is [86.4, 85.9, 85.3, 86.1, 86.0]
mean accuracies for 60 % data is 85.940000000000001
standard deviation of accuracies for 60 % data is 0.3611094017053576

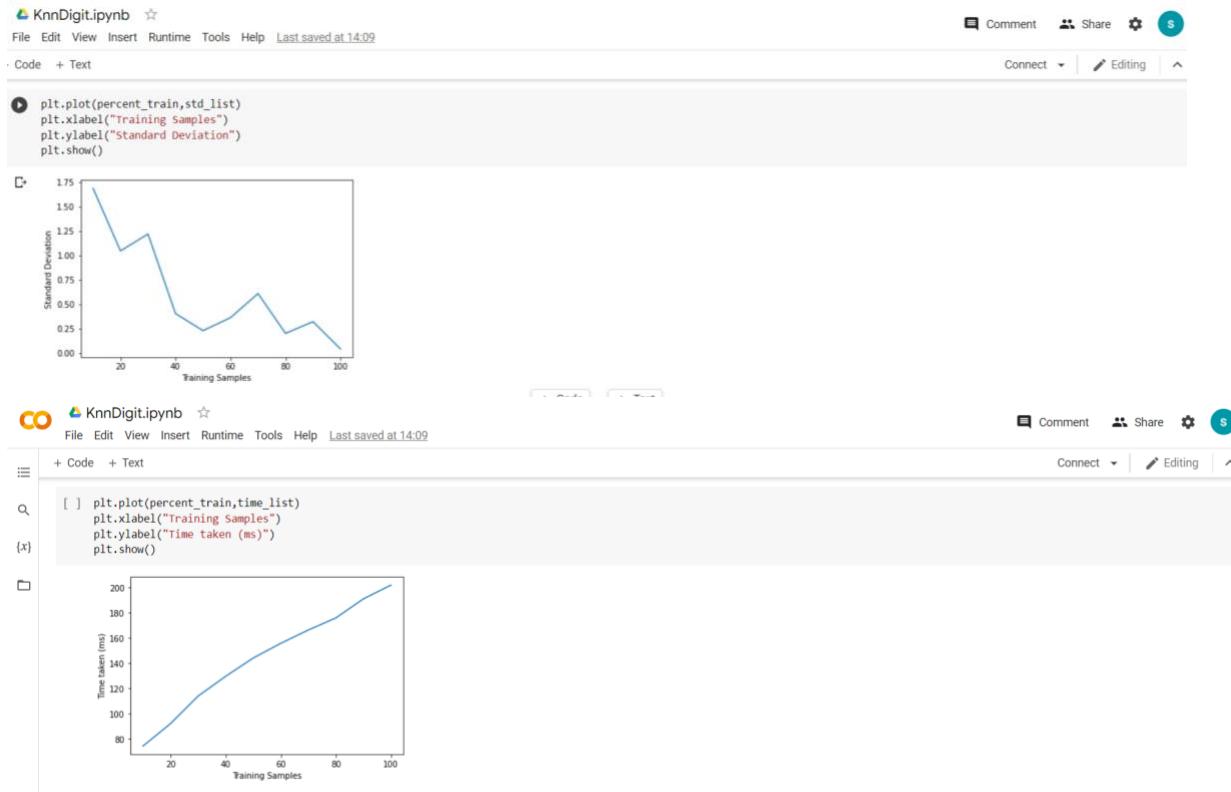
Time taken for 70 % data is [166.9931411743164, 169.07119750976562, 170.7615852355957, 169.69585418701172, 155.17663955688477]
mean time for 70 % data is 166.33968353271484
accuracy_list for 70 % data is [87.0, 86.1, 85.6, 86.2, 85.2]
mean accuracies for 70 % data is 86.02
standard deviation of accuracies 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 175.9276669128418
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 0.040000000000003415
```





## 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

+ Code + Text

```
▶ 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 21.6782569852530
█ accuracy_list for 10 % data is [68.666666666666667, 66.0, 54.666666666666664, 60.666666666666667, 70.666666666666667]
█ mean accuracies for 10 % data is 64.13333333333334
█ standard deviation of accuracies for 10 % data is 5.80268137253087

Time taken for 20 % data is [40.013790130615234, 20.699024200439453, 18.30887794494629, 22.2930908203125, 21.846771240234375]
mean time for 20 % data is 24.63231086738957
accuracy_list for 20 % data is [75.33333333333333, 76.0, 73.33333333333333, 67.33333333333333, 62.666666666666667]
mean accuracies for 20 % data is 70.9333333333332
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 27.921532584594727
accuracy_list for 30 % data is [68.0, 60.666666666666667, 68.666666666666667, 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, 26.52645110839844, 33.01763534545894, 21.383285522460938, 20.572662353515625]
mean time for 40 % data is 27.2295951814326172
accuracy_list for 40 % data is [70.0, 72.666666666666667, 71.33333333333334, 69.33333333333334, 68.666666666666667]
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, 37.7898216247586, 37.077903747558594, 26.851415634155273, 31.006574630737305]
mean time for 50 % data is 31.563806533813477
accuracy_list for 50 % data is [66.0, 76.0, 74.666666666666667, 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

+ Code + Text

```
▶ 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, 68.666666666666667, 72.666666666666667, 73.3333333333333, 70.666666666666667]
█ mean accuracies for 60 % data is 71.866666666666667
█ 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 30.37843704232328
accuracy_list for 70 % data is [72.666666666666667, 72.666666666666667, 72.0, 74.0, 70.0]
mean accuracies for 70 % data is 72.266666666666668
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, 70.666666666666667, 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 71.866666666666667
standard deviation of accuracies for 90 % data is 1.654623152798776

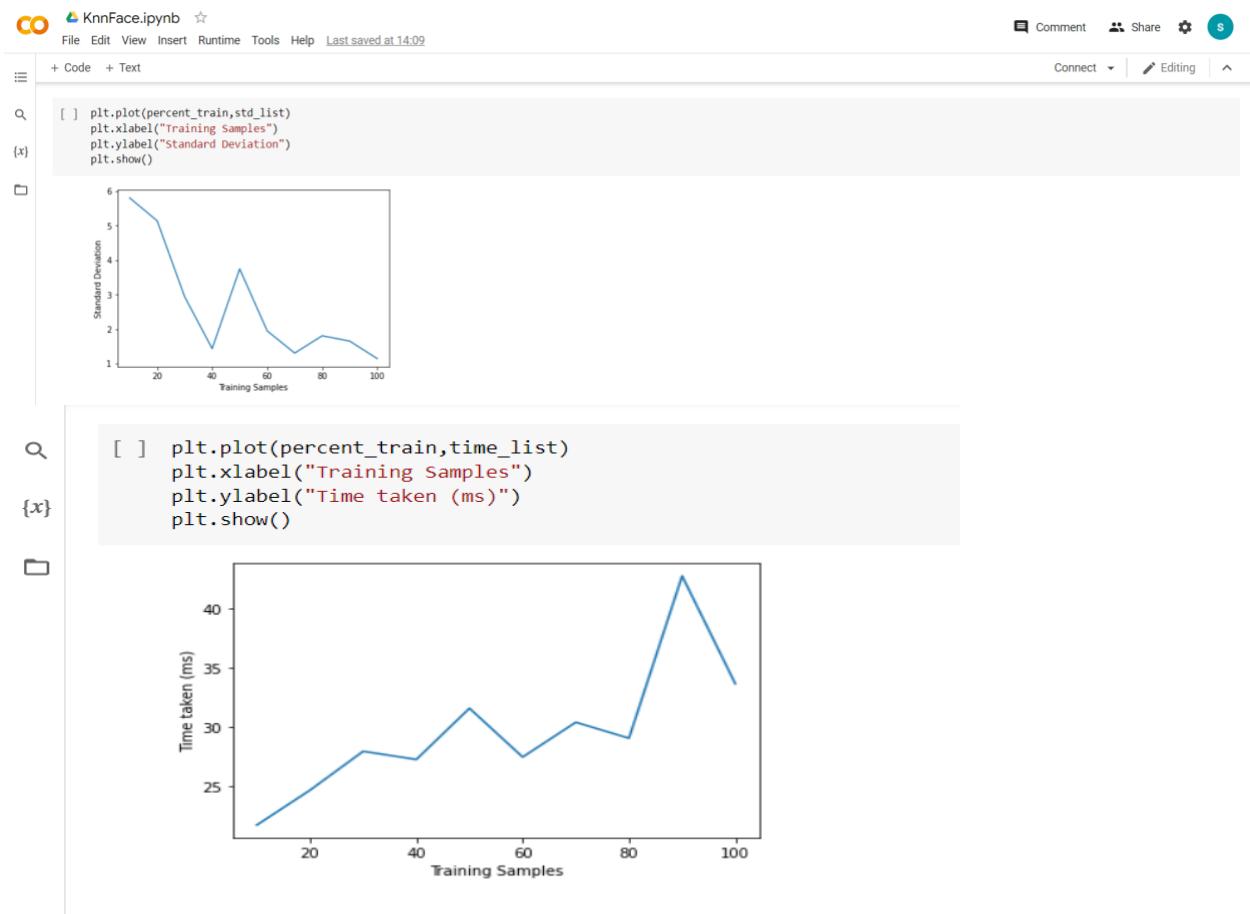
Time taken for 100 % data is [42.617002595825195, 33.33020210266113, 31.08811378479004, 31.360149383544922, 29.801368713378906]
mean time for 100 % data is 33.63938331604004
accuracy_list for 100 % data is [73.33333333333333, 72.0, 72.666666666666667, 70.0, 72.666666666666667]
mean accuracies for 100 % 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

+ Code + Text

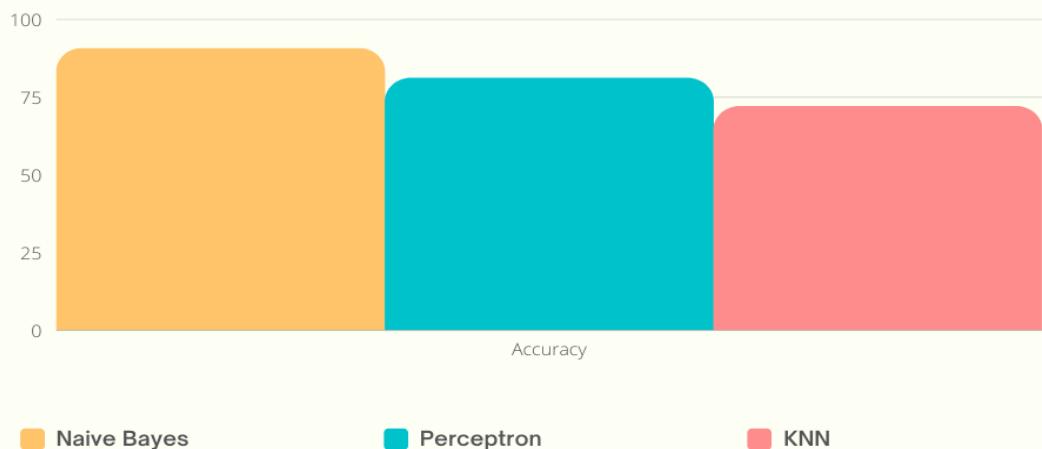
```
[ ] plt.plot(percent_train,mean_list)
plt.xlabel("Training Samples")
plt.ylabel("Mean")
plt.show()
```



## COMPARISION OF ACCURACIES OF OUR ALGORITHMS:

### **1. FACE DATA CLASSIFICATION:**

## Face Data Classification



## 2. DIGIT DATA CLASSIFICATION:

### Digit Data Classification

