Final Project

Rutgers University : CS 520 (Image Classification)

Team Members :
Anamika Lochab (NetId: al1522 RUID: 219004636)
Ryan Feng (NetId: rf445 RUID: 175007719)
Raunak Negi (NetId: rn444 RUID: 218002701)

Introduction

In this Image classifcation project, we have implemented 3 classification algorithms namely, Naive Bayes Classifier, Perceptron Classifier and Neural Networks classifier algorithms. We are performing these classification actions on 2 different datasets of scanned digit, face data.
For the digit dataset we need to predict the label of digit(0-9) and for the image dataset, we need identify whether it is a face or not (0-1).

Data Pre-Processing

We did not use the already pre-processed data from the Berkeley files. The image files are transformed into matrix [i,j,k] where i is the number of sample images in file, j is the rows and k is the columns. The image file has black and gray pixels, but we consider them as same and thus replace "#" and "+" with 1 and empty spaces with 0.

Classification Algorithms

1. Naive Bayes

It is a probabilistic classification model which is based on Bayes theorem.

According to Bayes rule ,

$$P(Label/Data\_features) = \frac{P(Label)P(Data_features/Label)}{P(Data_features)}$$

$$P(Label = l/Data\_features = f) = \text{Probability of Label l given Data\_features f}$$

$$P(Label = l) = \frac{\text{Instances with label l}}{\text{Total instances}}$$

$$P(Data\_features/Label) = \frac{\text{Number of times feature took value f in instances with label l}}{\text{Number of label l instances}}$$

It determines the probabilty of a label given a set of features, we divide the image into a set of features, we find the probability of each feature, and the probability of predicted label is the max probability from all the feature probabilities.

2. Perceptron

Perceptron is a supervised machine learning algorithm for binary classification. It takes a row of data as input and predicts a label. The binary classifier is a function that determines whether a given input belongs to some particular class.

Weights w are initialized using random values and bias as 1, the activation function takes input w,b, and input data.

Activation f = w * I + b

$$\text{Output } \hat{y} = \begin{cases} 1 & \text{if } f > 0 \\ -1 & \text{if } f \leq 0 \end{cases} \tag{1}$$

If the value of $\hat{y}$ is different from the true label of the data, the weights and bias value are updated. If a label is misclassified then :

$$w = w + y_i x_i \text{ and bias } b = b + y_i$$

These values are updated until all predicted and true labels are the same. Then the obtained w and b are used for prediction.

Perceptron assumes that :

- Problem is binary classification
- It is linearly separable.

For face recognition, since we only need to determine if the given data represnts a face or not, perceptron is fine. But in digit, we have more than 2 values to classify, so the perecptron is redesigned and we use max function to resolve the non-linearity.

3. Neural Networks

Neural Networks is another supervised machine learning algorithm that can be used for classification. The data is entered into an input layer of neurons, which then outputs data to hidden layers of neurons, which continue outputting to other layers until they reach the final layer, which outputs a final result. Each neuron has different weights that determine the output of the neuron based on the input that is given. The weights that determine the outputs of the neurons are then updated based on whether the correct final result was produced.

Our implementation has three layers :

- Input Layer : We get the input data, it is transformed for the next layer as $f = w * x + b$ where w is the weight and b is bias.
- Hidden Layer : We are using sigmoid function for the hidden layer. This activation function gives and s- shaped curve. $Sigmoid(f) = \frac{1}{1+e^{-z}}$
- Output layer : The value we get from the hidden layer determines y. In case of Face data if $Sigmoid(f) > 0.5$ it is set to 1 else to 0. In case of digit data, the labels are transformed into one-hot format, and we use argmax to determine the label, the $Output = argmax(Sigmoid(f))$

The optimization is done for N number of iterations and the final w, b are then used for prediction.

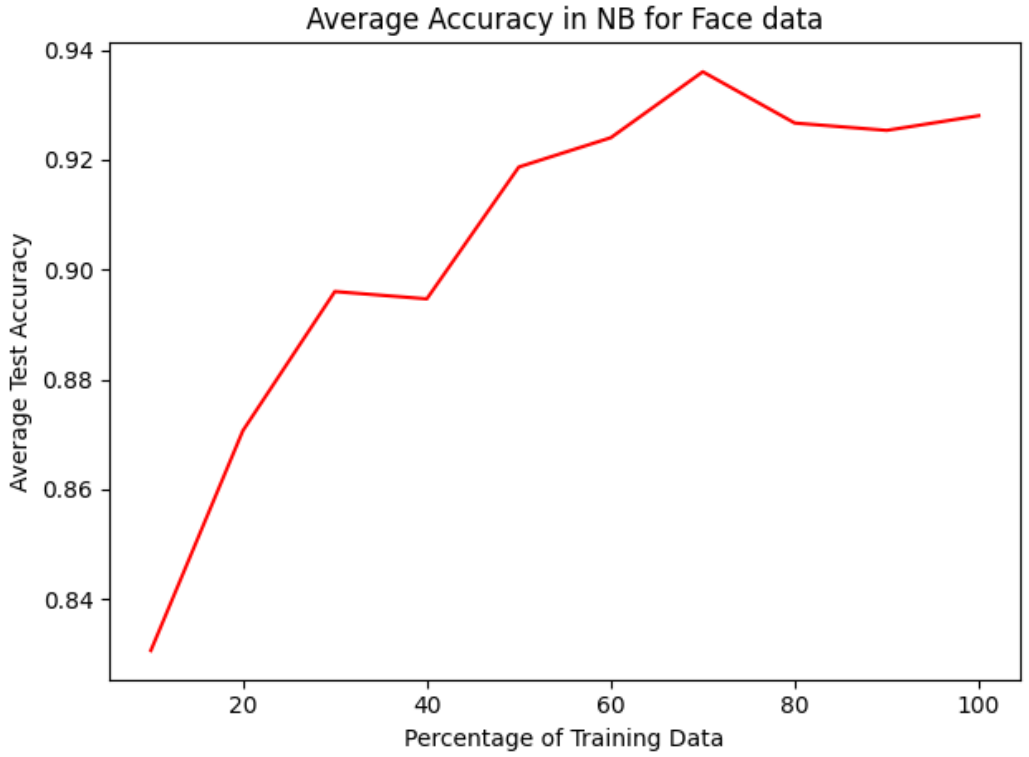
Result

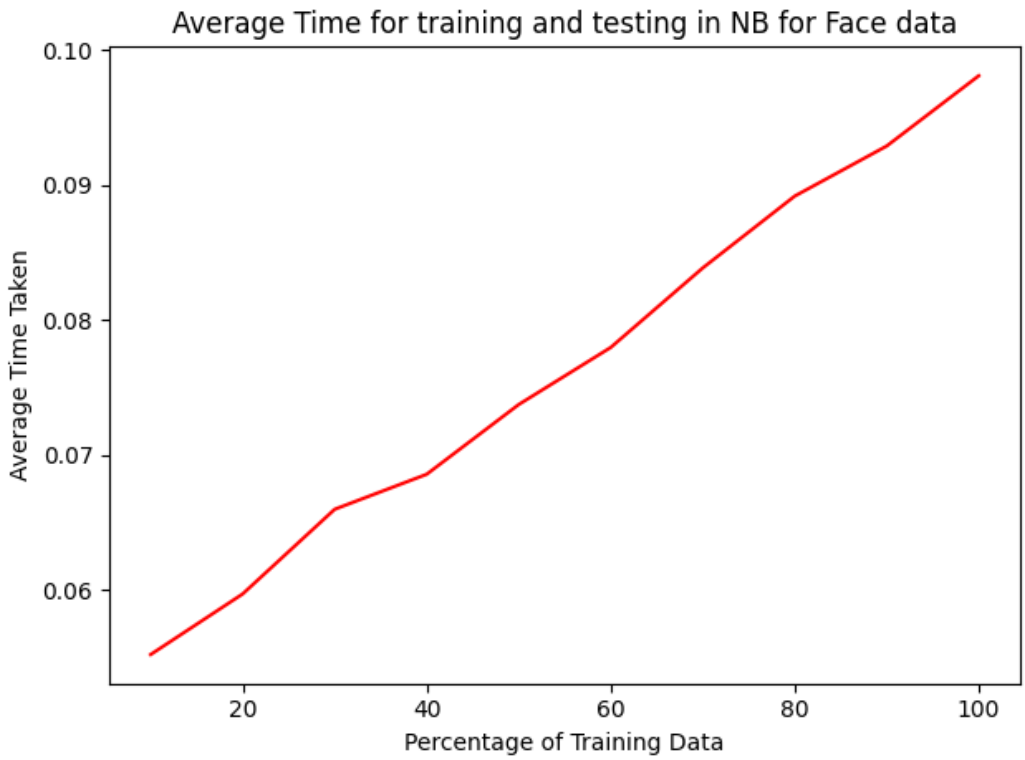
We have implemented the above algorithm with different percentages of training data from $10\% to 100\%$. We see in the results below how the accuracy changes as we increase the training data and the time taken by the model.

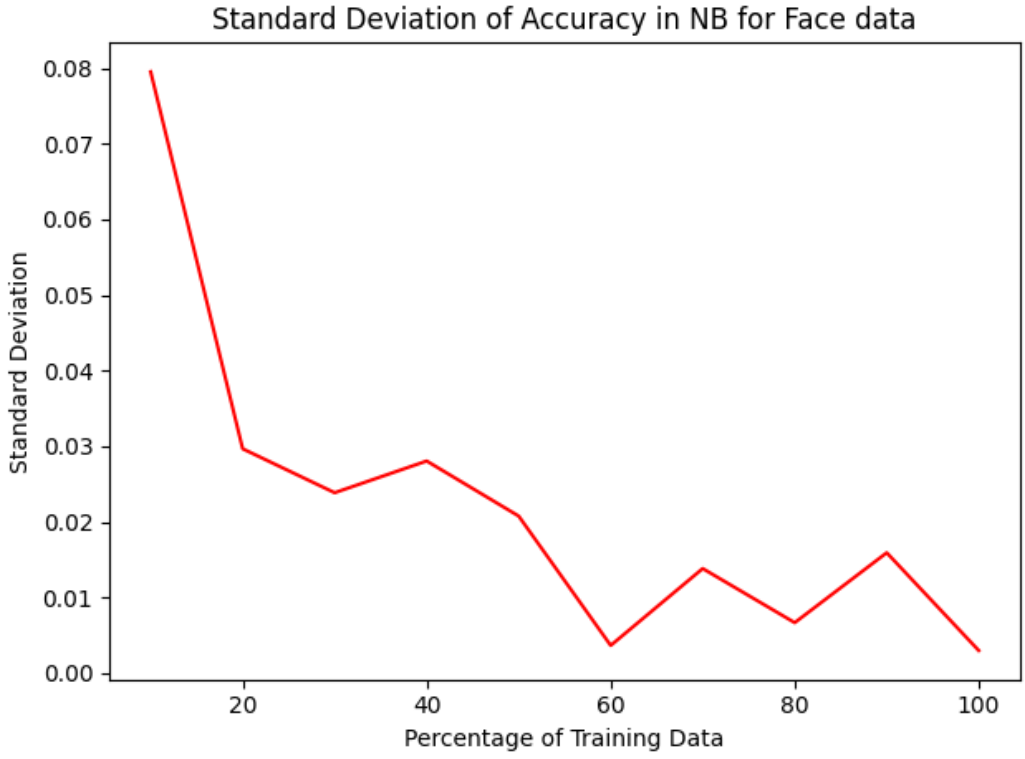
1. Naive Bayes

We look at the accuracy of the Naive Bayes classifier as the training data increases, and also the time taken by the model to train and predict. We can see that accuracy generally improves as data increases, and is thus a function of training data. We also see that the time for training also increases as data increases. The standard deviations between accuracy also generally decreases as data increases.

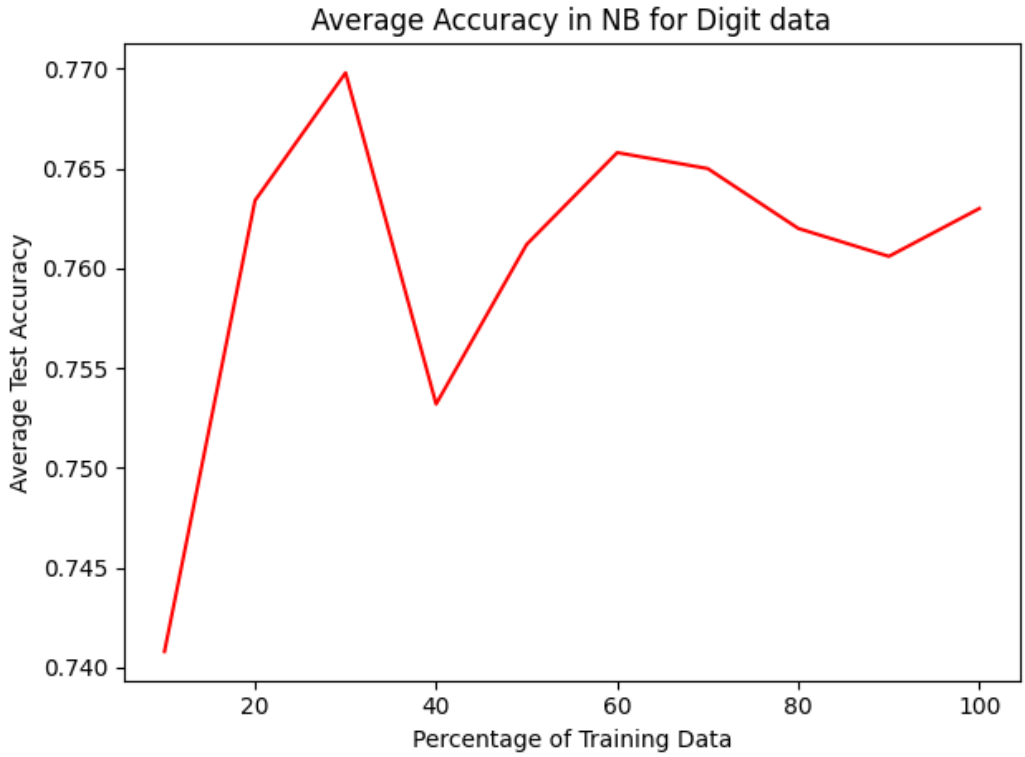
For face data:

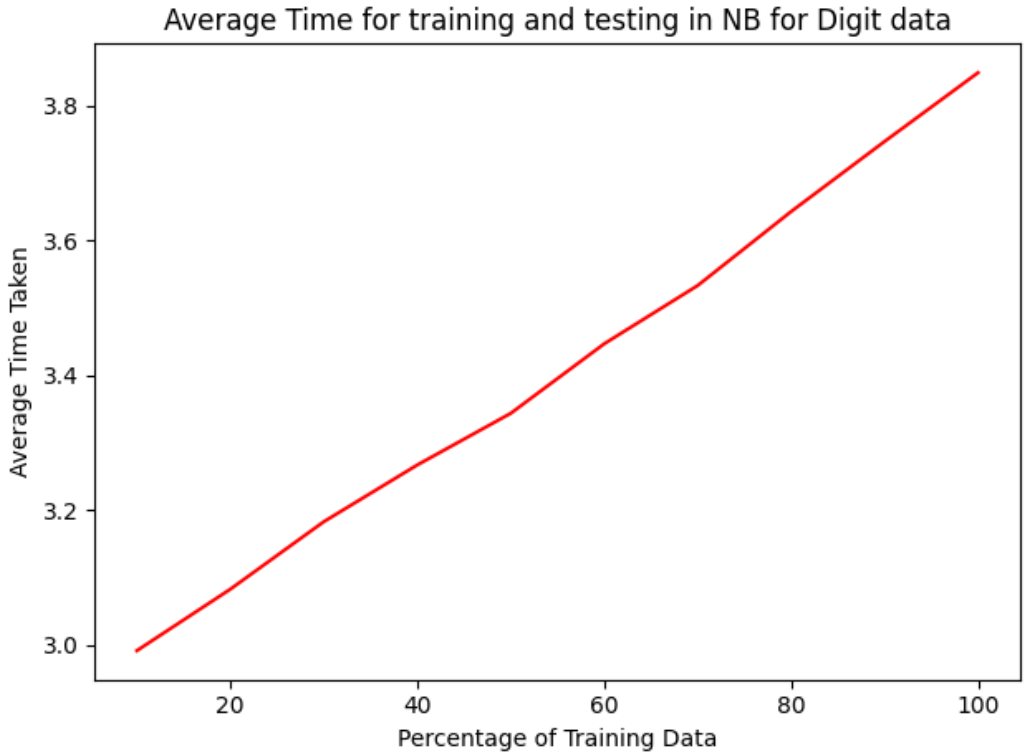


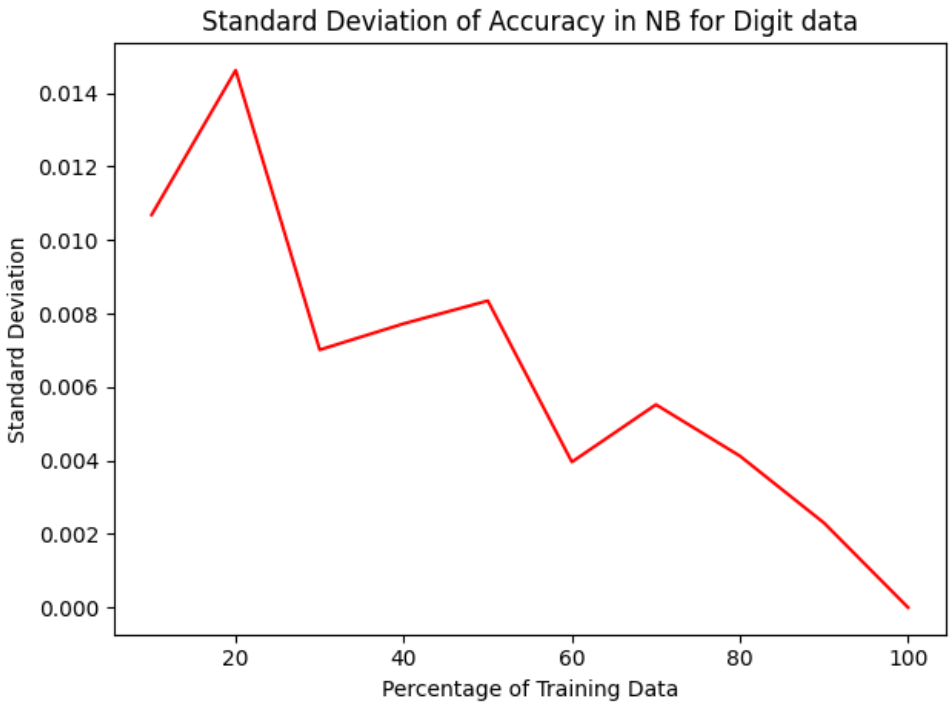

Average Time for training and testing in NB for Face data
Average Time Taken
0.10
0.09
0.08
0.07
0.06
20
40
60
80
100
Percentage of Training Data


Standard Deviation of Accuracy in NB for Face data
Standard Deviation
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
20
40
60
80
100
Percentage of Training Data

For digit data:



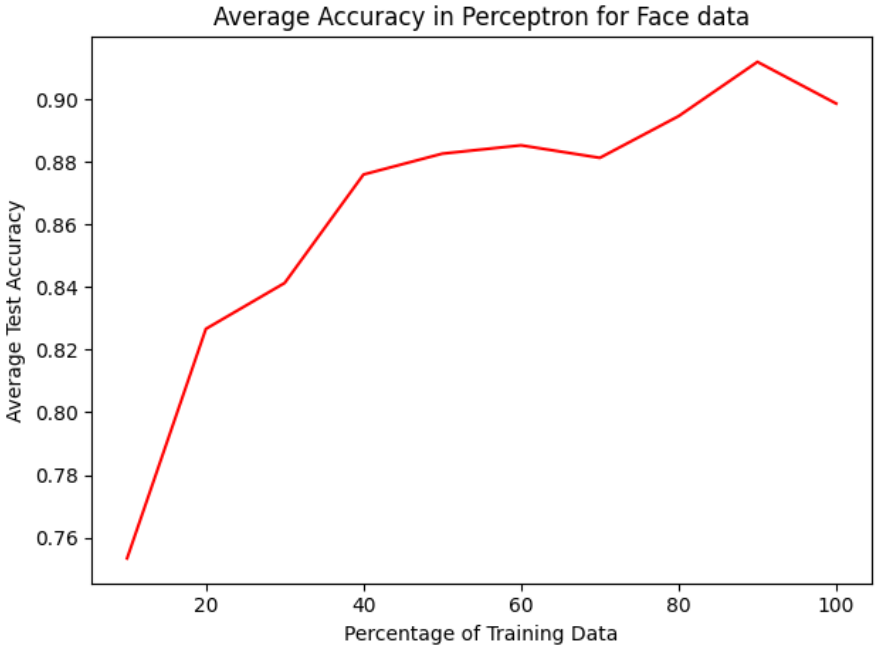


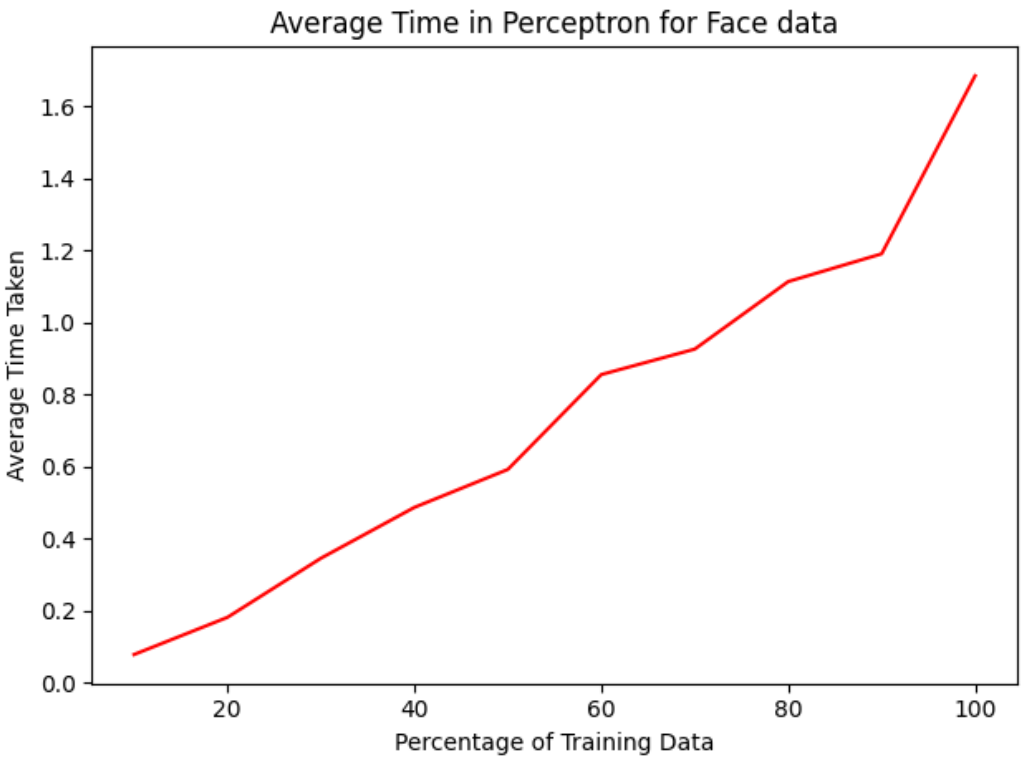


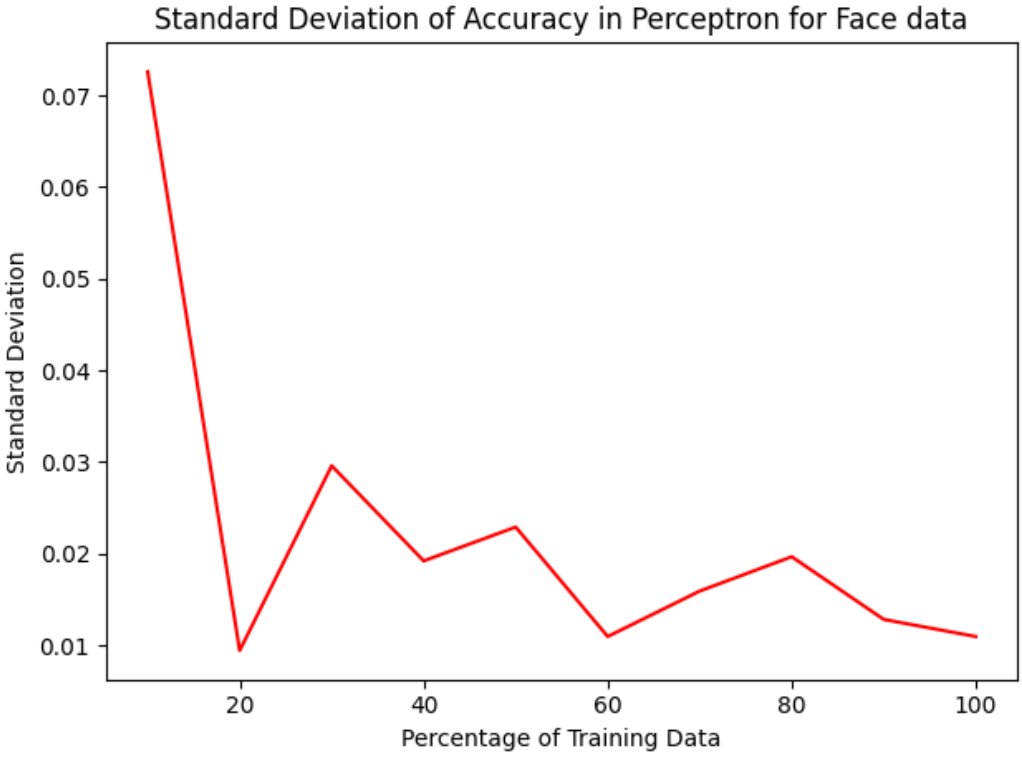
2. Perceptron

Again for perceptron algorithm we see how training data influences the time and accuracy. We see a similar trend to Naive-Bayes where both the time and accuracy increases as the amount of data points increases. We see again that standard deviation generally decreases for the face data. For the digit data, however, there are multiple peaks, but we notice that the standard deviation is quite low regardless.

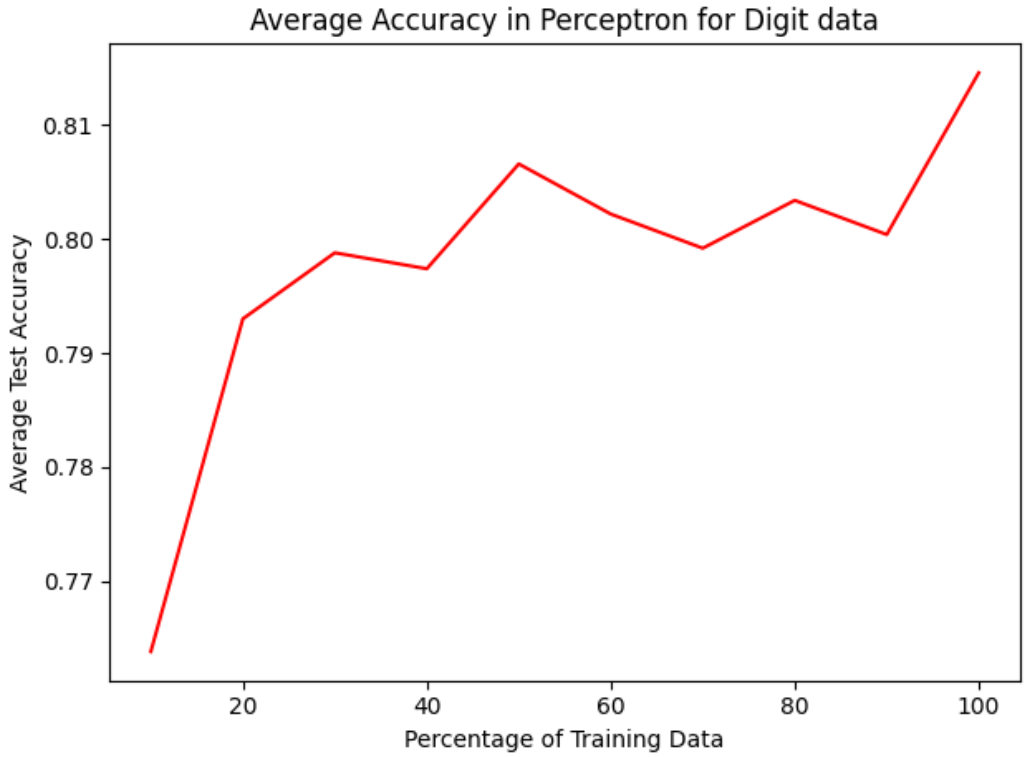
For face data:

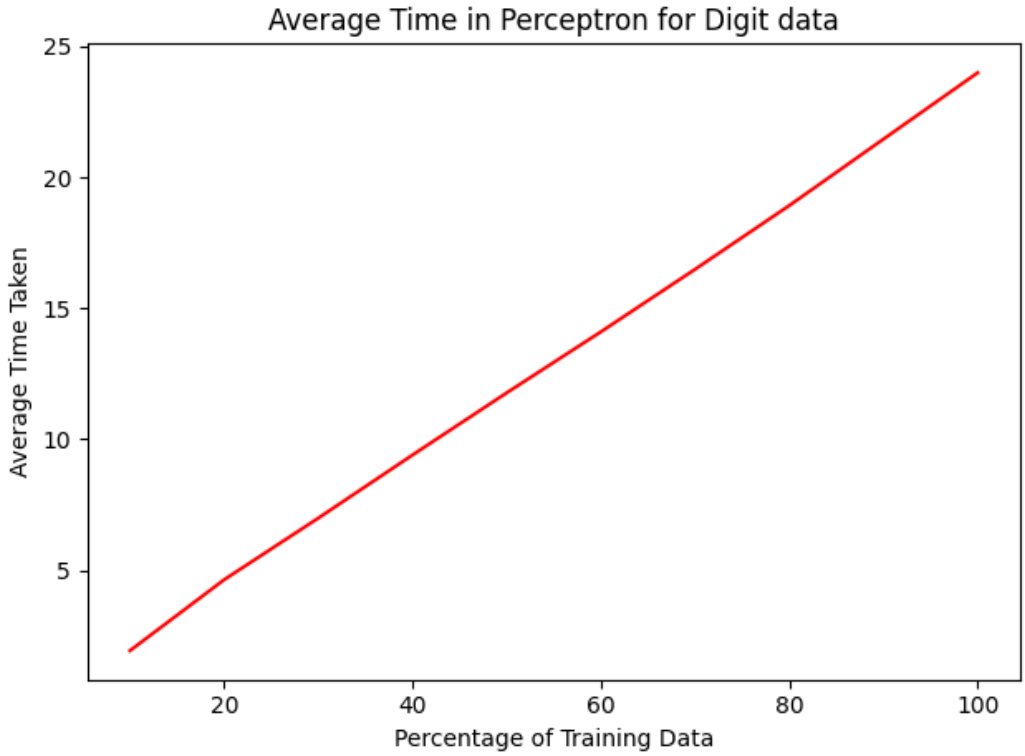


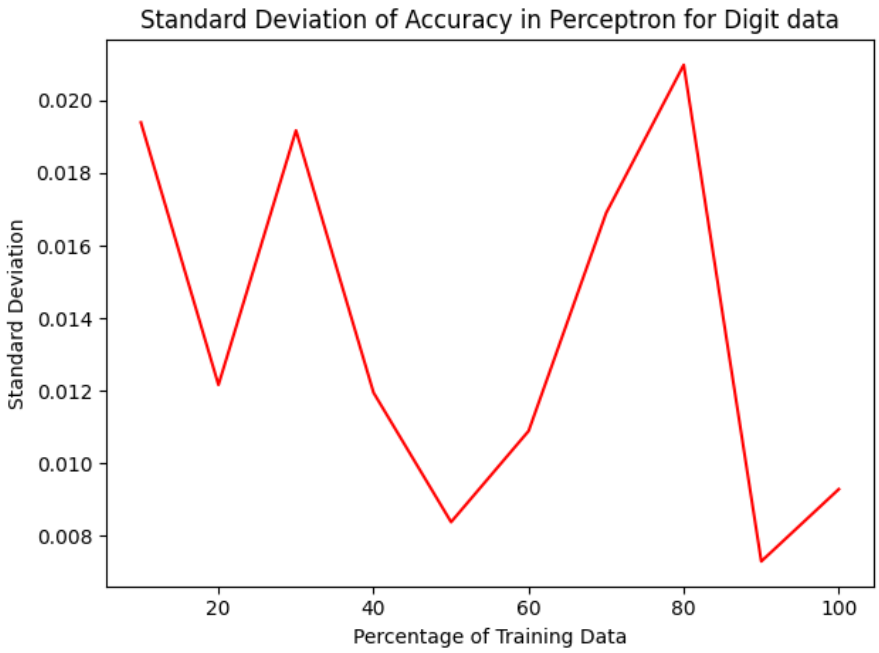

Average Time in Perceptron for Face data
Average Time Taken
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
20
40
60
80
100
Percentage of Training Data


Standard Deviation of Accuracy in Perceptron for Face data
Standard Deviation
0.01
0.02
0.03
0.04
0.05
0.06
0.07
20
40
60
80
100
Percentage of Training Data

For digit data:



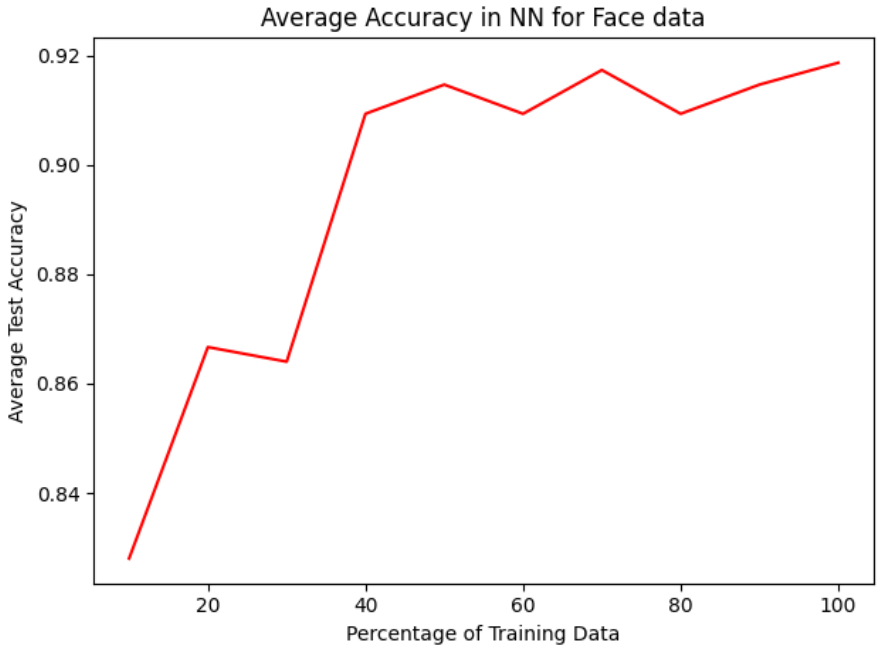


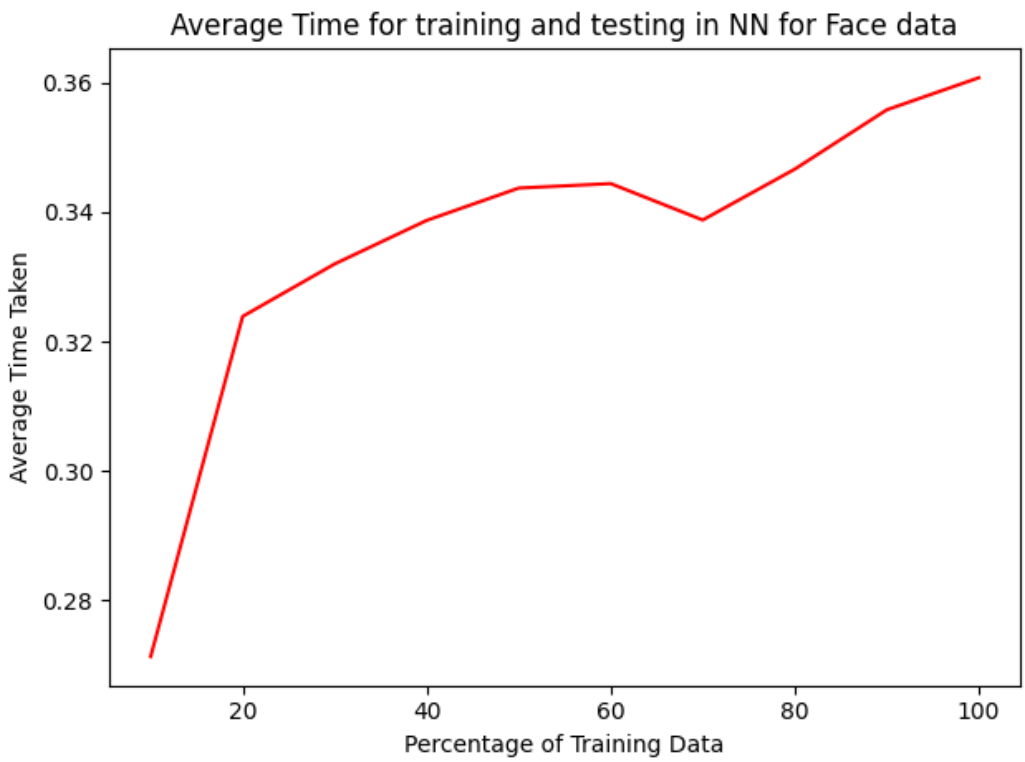


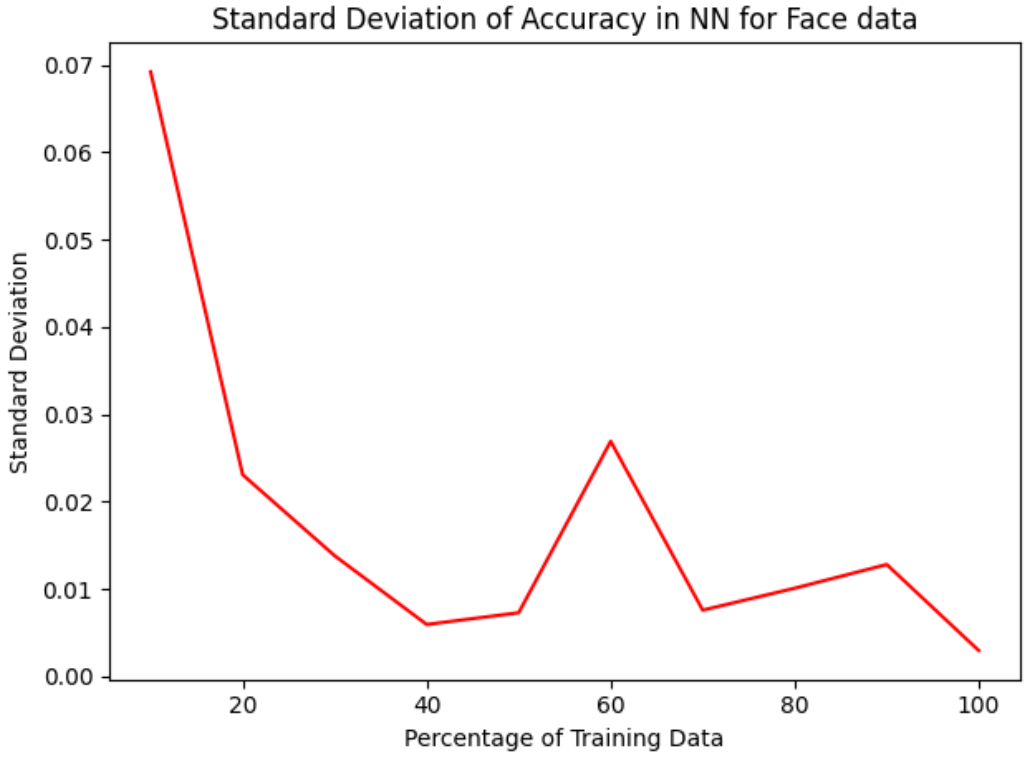
3. Neural Networks

For neural networks, the relationship between number of data points used for training and time and accuracy is the same. We see once again that as the number of training data points increases, the time needed for training as well as the accuracy of the prediction also increase. The standard deviation also generally decreases as the data points increase. We once again see that for the digit data, similar to perceptron, there is an interesting pattern for the standard deviation. But we note again that the standard deviation is low regardless.

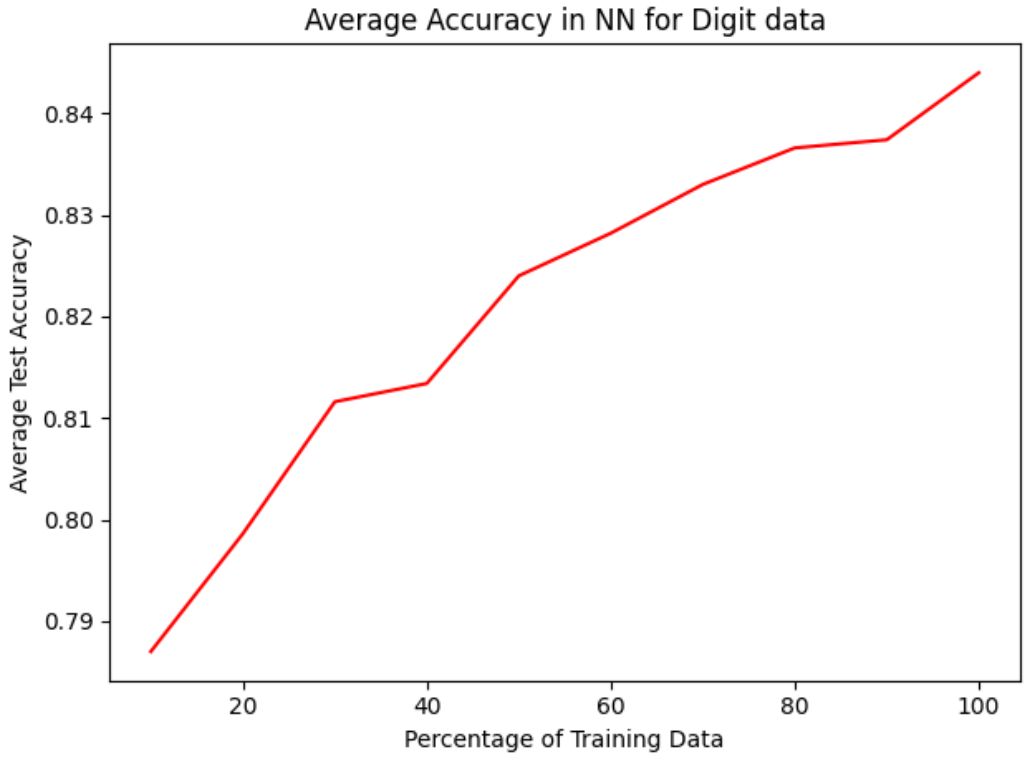
For face data:

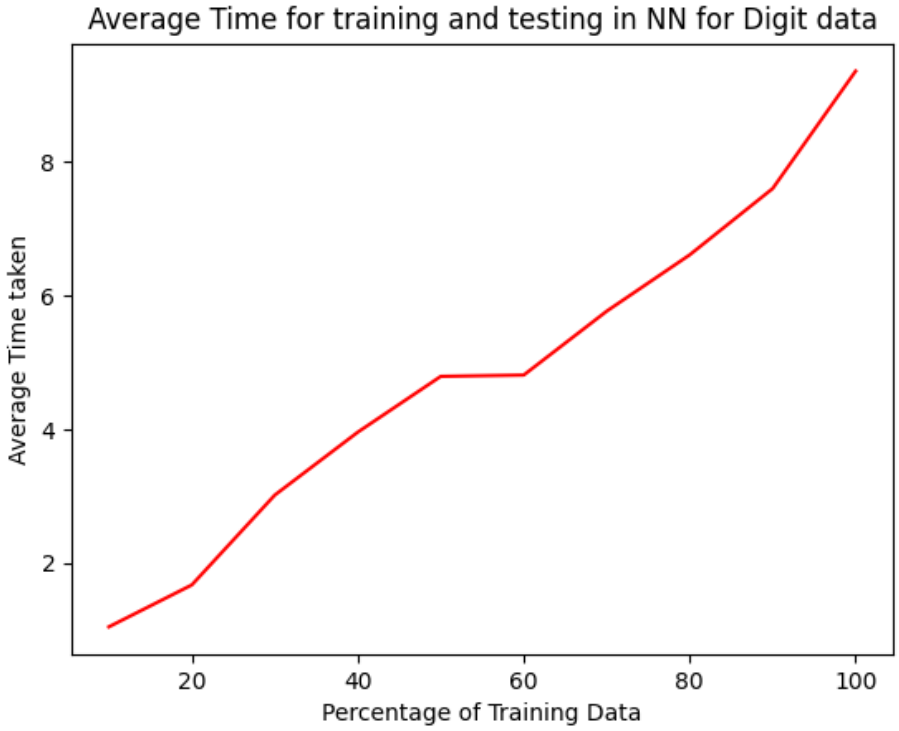


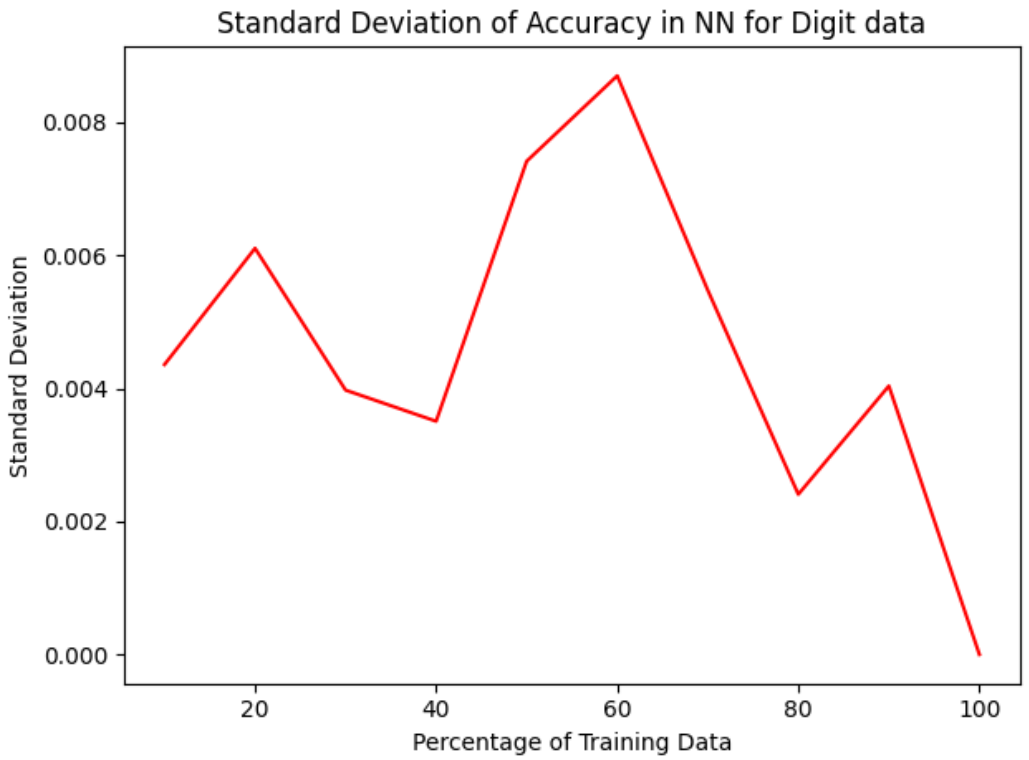

Average Time for training and testing in NN for Face data
Average Time Taken
0.36
0.34
0.32
0.30
0.28
20
40
60
80
100
Percentage of Training Data


Standard Deviation of Accuracy in NN for Face data
Standard Deviation
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
20
40
60
80
100
Percentage of Training Data

For digit data:







Conclusion/Lessons Learned

In general, for all 3 algorithms, we conclude that the time for training and the accuracy of the results increases as the number of data points used for training increases, while the standard deviation of accuracy decreases as the number of data points increases. This inherently makes sense, as it should take more time for the models to consider more training points. Furthermore, the models should be better at predicting given more data to learn from. Also, as they get better, the difference between predictions should be lesser as well. So our results follow what basic intuition would suggest. When using 100% of training data, the average accuracy is as follows:

Algorithm	Average Accuracy Face Data	Average Accuracy Digit Data
Naive Bayes	92.9%	76.3%
Perceptron	90.6%	81.2%
NN	92.13%	84.4%