2022FA - INTRO TO ARTIFICIAL INTELLIGENCE

FINAL PROJECT

FACE AND DIGIT CLASSIFICATION

Submitted By:

Kanchu Kiran – kk1219

Shraddha Pattanshetti – sp2304

1. NAIVE BAYES ALGORITHM

1.1. Overview

A Naive Bayes classifier models a joint distribution over a label Y and a set of observed random variables or features (F1, F2, F3…FN) using the assumption that the entire joint distribution can be factored as follows (features are conditionally independent given the label):

$$P(F_1, \ldots, F_n, Y) = P(Y) \prod_i P(F_i|Y)$$

Naïve Bayes is based on the Bayes theorem that calculates the occurrence of one event while another has already occurred. Bayes theorem is derived with the help of the product rule and conditional probability of event A given the known event B.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where,

- P(A|B) is called Posterior, it is defined as updated probability after considering the evidence.
- P(B|A) is called the Likelihood. It is the probability of evidence when the hypothesis is true.
- P(A) is the Prior probability of the hypothesis before considering the evidence.
- P(B) is called Marginal probability. It is defined as the probability of evidence under any consideration.

Hence, Bayes Theorem can be written as:
Posterior = (Likelihood * Prior Probability) / Marginal Probability

1.2. Features

The feature set includes one feature for each pixel location, which can take values 0 or 1 (off or on). The features are encoded as a Counter where keys are feature locations (represented as (columns, rows)) and values are 0 or 1. The face recognition data set has value one only for those pixels identified by a Canny edge detector.

1.3. Smoothing

We need to ensure that no parameter receives an estimate of zero, but good smoothing can boost accuracy quite a bit by reducing overfitting.

In this project, we use *Laplace smoothing*, which adds *k* counts to every possible observation value:

$$P(F_i = f_i | Y = y) = \frac{c(f_i, y) + k}{\sum_{f_i' \in \{0,1\}} (c(f_i', y) + k)}$$

1.4. Training and Tuning

- Maintain a Dictionary with Legal labels as its key and the number of occurrences as values. Iterate over the dataset to get the count of labels.
- We now must calculate the probability and store it in a dictionary named feature counts which is further used for training purposes. In addition, calculated the conditional probability for each legal label, feature, and legal feature value.
- Now this collected information is used to train the classifier over the training data and stores the Laplace smoothed estimates so that they can be used to classify.
- Evaluate each value of the k grid to choose the smoothing parameter that gives the best accuracy on the held-out validation dataset.

1.5. Classify

Classify the data based on the posterior distribution over labels. Then take the log of the result to make comparison easier.

We compute the log probability for each one of the legal labels following the equation below.

$$\log P(y) + \sum_{i=1}^{m} \log P(f_i|y)$$

1.6. Data Processing:

We have the training and testing datasets and labels.

- The faces data contains 451 training data points and 150 test data points.
- The digits data includes 5000 training and 1000 test data points.

The training data points were picked randomly in groups of 10%, 20%, 30%,..100% to train, tune and compare the accuracy of the classifiers for each percentage used.

The command to run the Naïve Bayes algorithm is as follows:

The result gets stored in the resultsB.txt file.

python dataClassifier.py -c naiveBayes -d faces -t $amount -i 2 -s 150 >> resultsB.txt

where:

-c is the type of classifier (in this case, Naïve Bayes)

-d is the dataset to use (faces or digits)

-t is the size of the training dataset

-i is the number of iterations

-s is the number of test data points to use

1.7. Result and Analysis

Graphs are generated for the standard deviation, accuracy, and Time to train the dataset against the percentage of training data chosen below.

Epoch value = 5

1.7.1. Naïve Bayes Classification training time: Face

Naïve Bayes Face									
10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
0.19591999	0.38542295	0.58777404	0.78529096	0.98957992	1.20156193	1.40665102	1.70349097	1.7976532	2.00698805
0.18806195	0.40049195	0.61045718	0.80124283	0.9976778	1.19994187	1.37877703	1.58787107	1.74475479	2.00275207
0.19762683	0.39784789	0.59627819	0.80606103	0.98623991	1.21820617	1.39469314	1.55406213	1.78549695	1.99681091
0.19373298	0.38362813	0.60306406	0.77960896	0.99559593	1.19372606	1.38395691	1.57276392	1.82548809	2.07480407
0.19183397	0.40218306	0.59177899	0.78324103	1.01520157	1.19712472	1.40392184	1.54404902	1.79253793	2.02930212
0.193	0.394	0.598	0.791	0.997	1.202	1.394	1.592	1.789	2.022

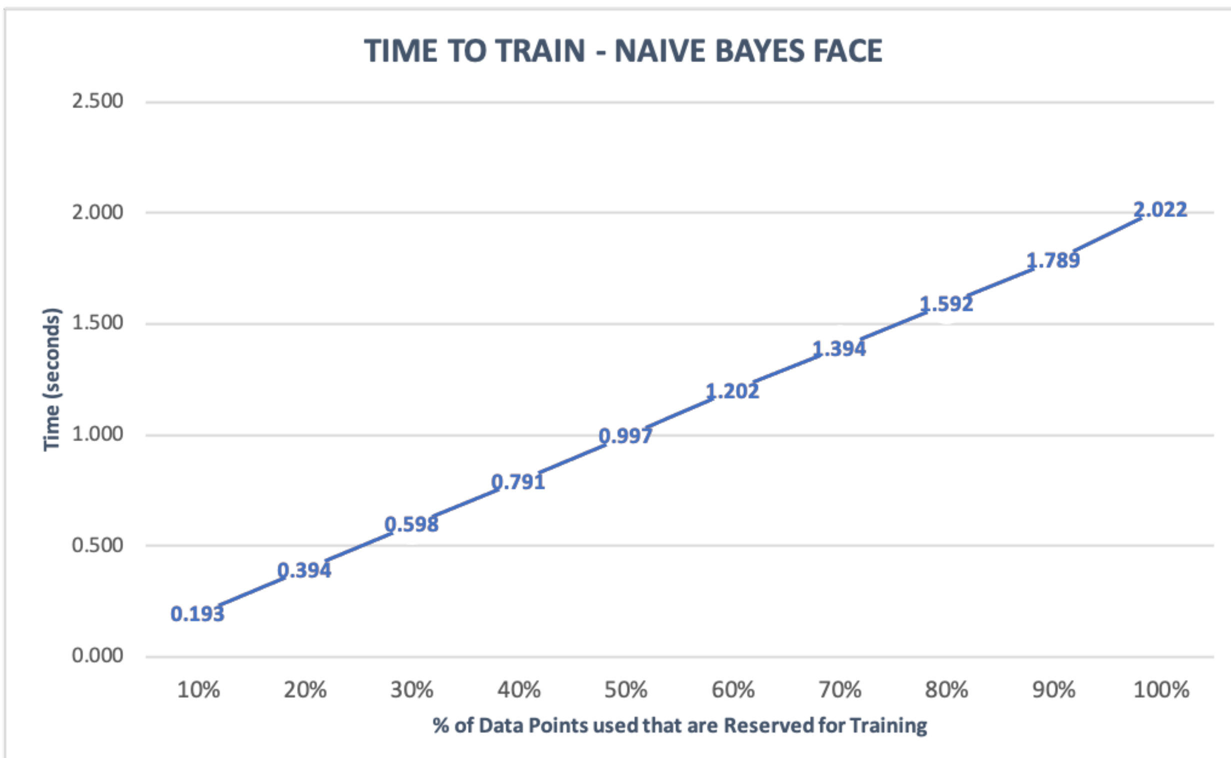

Fig1.1: Training time vs Percentage of training data points – Naïve Bayes Face

1.7.2. Naïve Bayes Classification training time: Digit

Naïve Bayes Digit									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0.41364717	0.87534475	1.25050974	1.66845393	2.02792597	2.48681712	2.81428933	3.25436592	3.7197001	4.02859116
0.42010212	0.85259795	1.27078104	1.63786292	2.00137115	2.42146778	2.95314717	3.75055695	3.58919096	4.08615804
0.40693402	0.81999993	1.32505608	1.61697793	2.06338	2.3858161	2.76331806	3.59880304	4.8704319	4.56809783
0.40523314	0.81519389	1.23487806	1.59065509	2.03851295	2.38307381	2.84135485	4.18173289	3.84205222	4.01947999
0.41718602	0.82474303	1.26310611	1.5999341	2.03811598	2.38347816	2.83940673	3.23043799	3.59071612	3.93252707
0.413	0.838	1.269	1.623	2.034	2.412	2.842	3.603	3.922	4.127

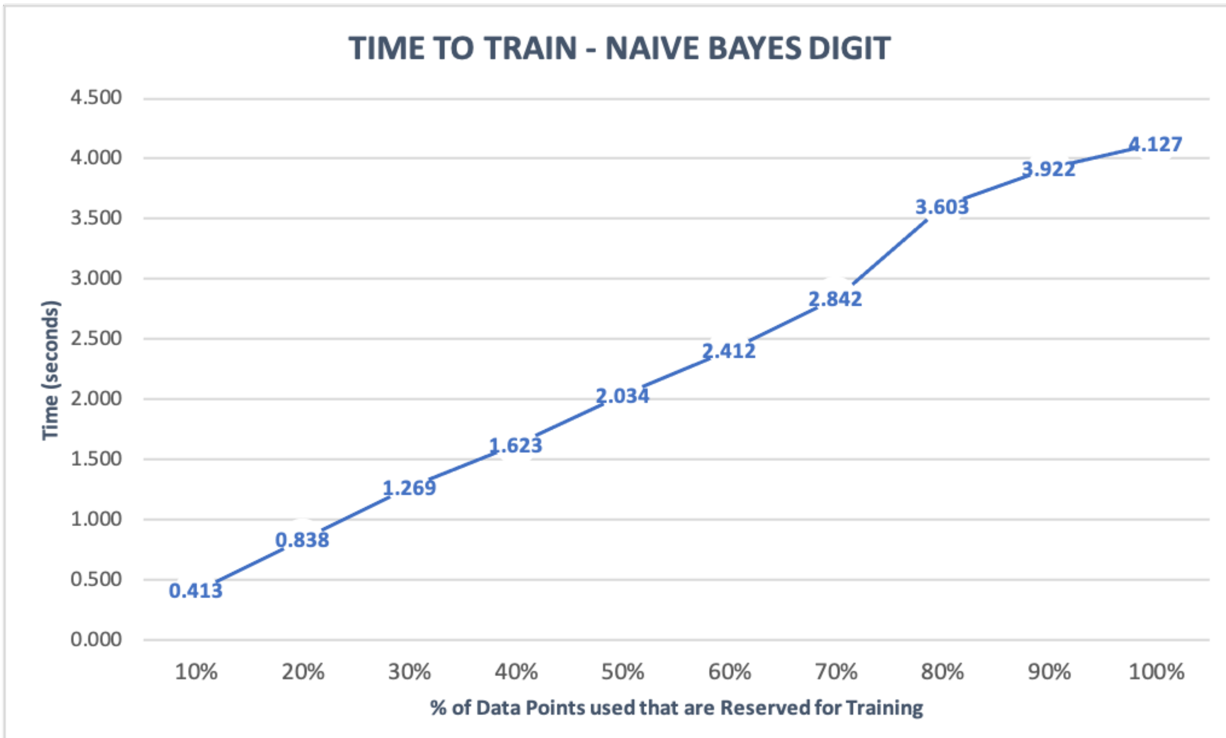

Fig1.2: Training time vs Percentage of training data points – Naïve Bayes Digit

1.7.3. Naïve Bayes Classification Accuracy: Face

Naive Bayes Face									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
78.00%	86.70%	85.30%	83.30%	87.30%	88.00%	88.70%	88.00%	90.00%	88.70%
75.30%	84.00%	83.30%	85.30%	88.00%	88.00%	89.30%	90.00%	88.70%	89.30%
74.70%	78.00%	81.30%	88.70%	88.70%	86.00%	88.00%	88.00%	88.00%	90.70%
70.70%	78.00%	88.00%	87.30%	87.30%	89.30%	89.30%	88.00%	87.30%	90.70%
80.70%	82.70%	87.30%	88.00%	88.00%	87.30%	90.00%	88.00%	88.70%	90.70%
0.759	0.819	0.850	0.865	0.879	0.877	0.891	0.884	0.885	0.900

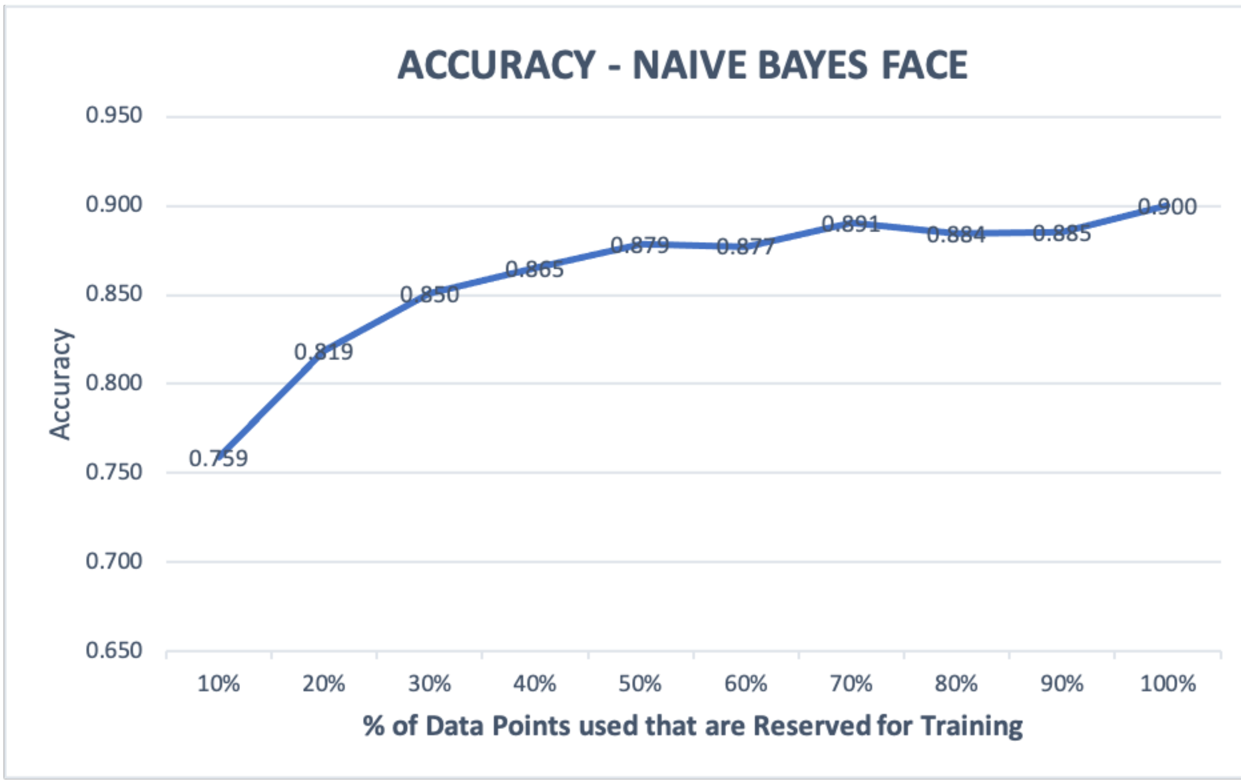

Fig1.3: Accuracy vs Percentage of training data points – Naïve Bayes Face

1.7.4. Naïve Bayes Classification Accuracy: Digit

Naïve Bayes Digit									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
74.00%	76.40%	78.10%	77.20%	76.80%	77.10%	77.60%	77.10%	77.60%	77.20%
73.60%	76.00%	77.40%	77.70%	77.00%	77.70%	77.30%	77.50%	77.20%	77.30%
76.00%	78.20%	75.90%	78.30%	78.30%	76.30%	77.10%	77.00%	77.10%	77.40%
76.10%	77.40%	76.20%	75.80%	76.60%	76.40%	76.50%	77.50%	77.00%	77.40%
74.50%	77.40%	77.90%	78.10%	76.50%	76.70%	77.60%	77.90%	77.30%	77.30%
0.748	0.771	0.771	0.774	0.770	0.768	0.772	0.774	0.772	0.773

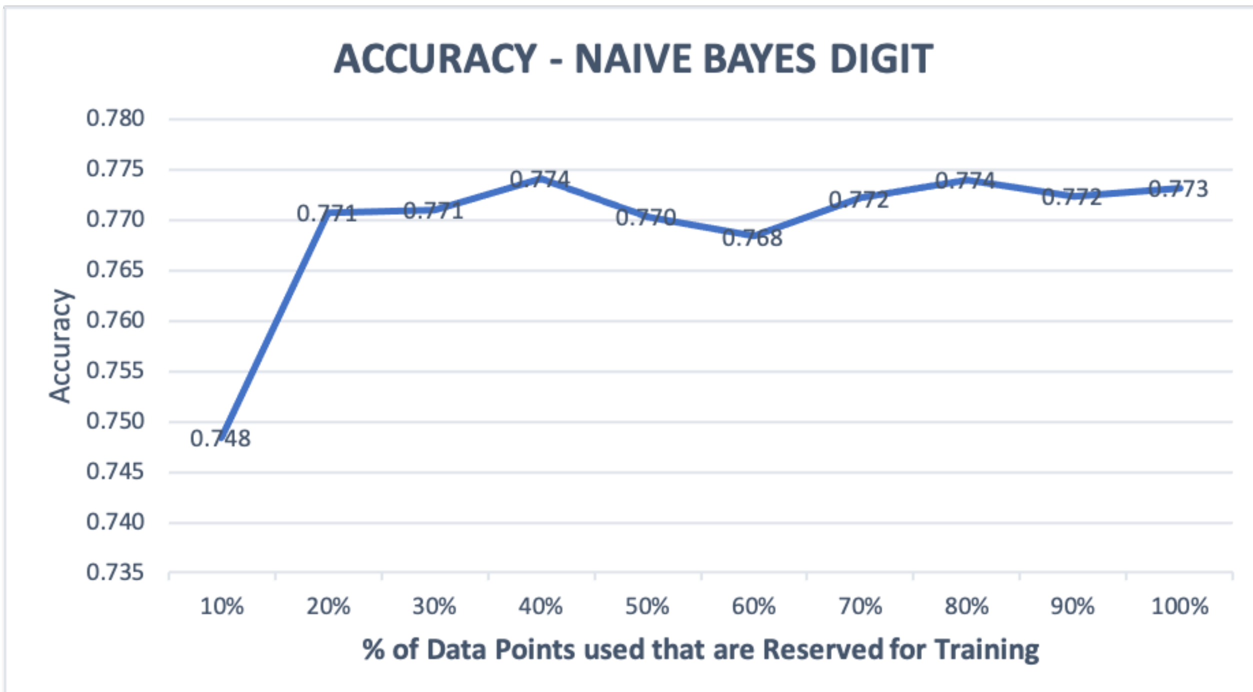

Fig1.4: Accuracy vs Percentage of training data points – Naïve Bayes Digit

1.7.5. Naïve Bayes Classification Standard Deviation: Face

Naïve Bayes Face									
10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
0.4142463	0.33993463	0.35377331	0.372678	0.33259919	0.32496154	0.31699982	0.32496154	0.3	0.31699982
0.43107102	0.36660606	0.372678	0.35377331	0.32496154	0.32496154	0.30868898	0.3	0.31699982	0.30868898
0.43492017	0.4142463	0.38964371	0.31699982	0.31699982	0.34698703	0.32496154	0.32496154	0.32496154	0.29089899
0.4552899	0.4142463	0.32496154	0.33259919	0.33259919	0.30868898	0.30868898	0.32496154	0.33259919	0.29089899
0.39491209	0.37853519	0.33259919	0.32496154	0.32496154	0.33259919	0.3	0.32496154	0.31699982	0.29089899
0.426	0.383	0.355	0.340	0.326	0.328	0.312	0.320	0.318	0.300

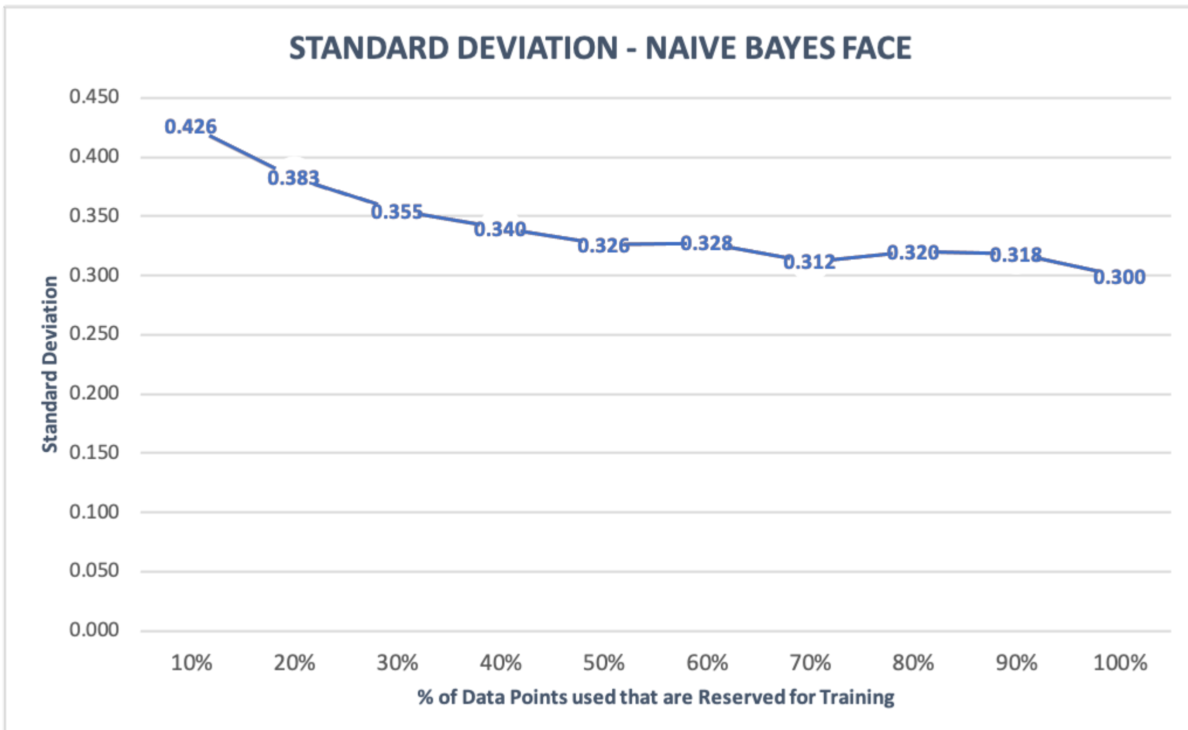

Fig1.5: Standard Deviation vs Percentage of training data points – Naïve Bayes Face

1.7.6. Naïve Bayes Classification Standard Deviation: Digit

Naïve Bayes Digit									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0.43863424	0.42462219	0.41356862	0.41954261	0.42210899	0.42018924	0.41692206	0.42018924	0.41692206	0.41954261
0.44079927	0.42708313	0.41823917	0.41625833	0.42083251	0.41625833	0.41889259	0.41758233	0.41954261	0.41889259
0.42708313	0.41288739	0.42769031	0.41220262	0.41220262	0.42524228	0.42018924	0.42083251	0.42018924	0.41823917
0.42647274	0.41823917	0.42585913	0.42829429	0.42337218	0.42462219	0.42399882	0.41758233	0.42083251	0.41823917
0.43586122	0.41823917	0.41492047	0.41356862	0.42399882	0.42274224	0.41692206	0.41492047	0.41889259	0.41889259
0.434	0.420	0.420	0.418	0.421	0.422	0.419	0.418	0.419	0.419

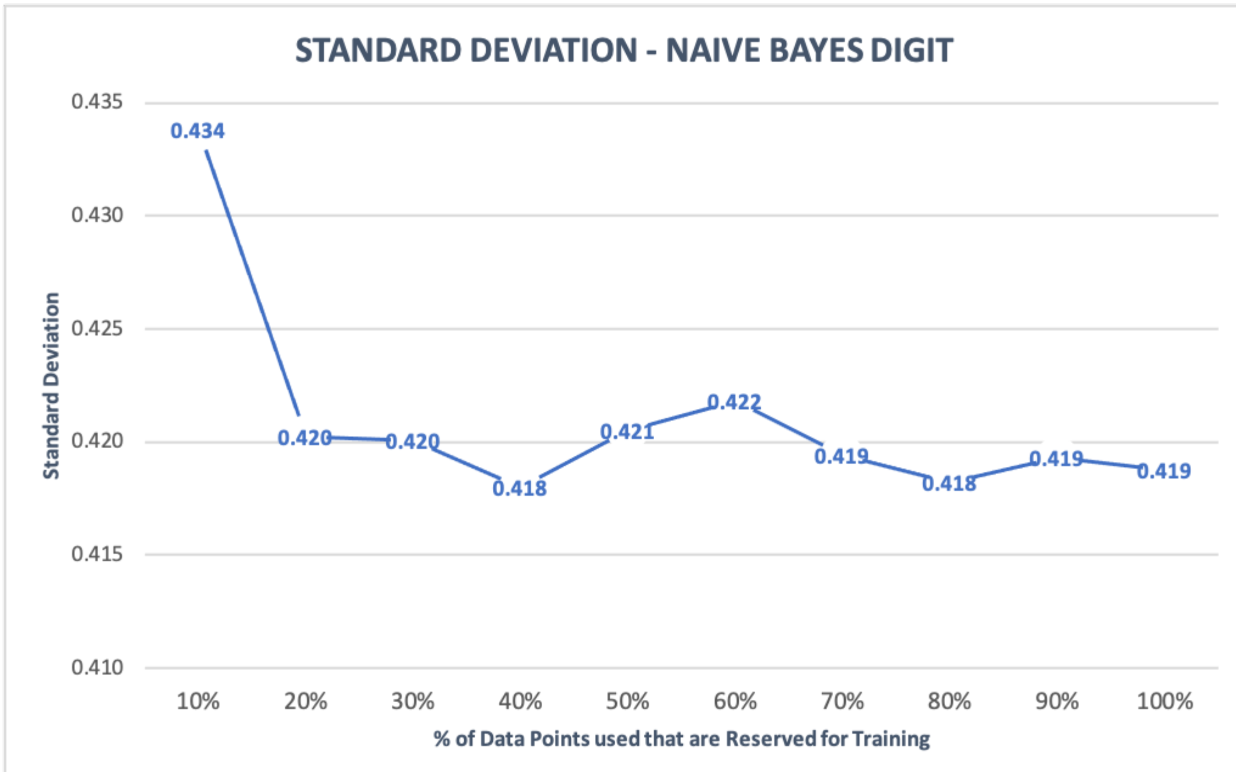

Fig1.6: Standard Deviation vs Percentage of training data points – Naïve Bayes Digit

1.8. Inference

1.8.1. Naive Bayes – FACE

- Training Time
    - From the graph, we can observe that time required to train the model with 10% of the training data is 0.193 seconds. When given 100 % of the training data for model training, the time required is 2.022 seconds.
    - Furthermore, the graph generated is a straight line of the form Y = MX + C, which indicates that as the percentage increases, the time required to train the model also increases.

- Accuracy
    - From the graph, we can observe that the model's accuracy, when given 10% of the training data, is 75 percent. If 100% of the training data is utilized for training the data model, then accuracy escalates to 90 percent.
    - Furthermore, the graph generated is almost straight with slight curves, which indicates that as the percentage increases, the accuracy of the model increases steadily.

- Standard Deviation
    - From the graph, we can observe that the standard deviation of the model with 10% of the training data is 0.426. When given 100 % of the training data for model training, the standard deviation of the model is 0.300.
    - Clearly, we can see that as the percentage increases, the standard deviation decreases. This indicates that the percentage of training data fed is inversely proportional to the standard deviation.

1.8.2. Naive Bayes - DIGIT

- Training Time
  - From the graph, we can observe that time required to train the model with 10% of the training data is 0.413 seconds. When given 100 % of the training data for model training, the time required is 4.127 seconds.
  - Furthermore, the graph generated is a straight line of the form Y = MX + C, which indicates that as the percentage increases, the time required to train the model also increases.

- Accuracy
  - From the graph, we can observe that the model's accuracy, when given 10% of the training data, is 74.8 percent. If 100% of the training data is utilized for training the data model, then the accuracy increases gradually to 77.3 percent.
  - Furthermore, the graph generated has variations; initially, the chart grows linearly. The accuracy drops slightly from 77.4 percent to 76.8 percent when the model is trained with 40% to 60% of data. Thereafter, the graph climbs gradually and finally reaches 77.3 percent accuracy.

- Standard Deviation
  - From the graph, we can observe that the model's standard deviation, when given 10% of the training data, is 0.434. If 100% of the training data is utilized for training the data model, then the standard deviation decreases gradually to 0.419.
  - Furthermore, the graph generated has variations; initially, the chart declines linearly. The standard deviation upsurges slightly from 0.418 to 0.422 when the model is trained with 40% to 60% of data. Thereafter, the graph falls slowly and finally reaches 0.419.

2. PERCEPTRON

2.1. Overview

The perceptron classifier uses a binary classifier to determine if the input's features match the class's characteristics. There is a binary classifier for each class (one class per digit and one for face). The image will be classified in the category that produces that maximum score when the feature vector is multiplied by that class's weight vector.

It keeps a weight vector of each class y (y is an identifier, not an exponent). Given a feature list f, the perceptron computes the class y, whose weight vector is most like the input vector f. Formally, given a feature vector f (in our case, a map from pixel locations to indicators of whether they are on), we score each class with the following:

$$score(f, y) = \sum_i f_i w_i^y$$

We choose the class with the highest score as the predicted label for that data instance.

2.2. Features

The feature set includes one feature for each pixel location, which can take values 0 or 1 (off or on). The features are encoded as a Counter where keys are feature locations (represented as (columns, rows)) and values are 0 or 1. The face recognition data set has value one only for those pixels identified by a Canny edge detector.

2.3. Initializing weights

When we come to the instance (f, y), we calculate the label with the highest score using the formula below.

$$y' = \arg\max_{y''} score(f, y'')$$

Now, compare the new label y` with the actual label y. If both are equal, then do nothing. Otherwise, update the weights based on the values of y and y` as given in the figure below.

$$w^y = w^y + f$$

$$w^{y'} = w^{y'} - f$$

2.4. Training

The first step is to set the value of iterations, and the training of the dataset will take for the specified no of iterations. The training loop for the perceptron passes through the training data several times, and the weight vector for each label is updated based on the classification errors as below.

$$\begin{aligned}
&fori = 1, 2, ..., j : weights[label][i] += \phi_i(datum) \\
&w_0 += 1 \\
&fori = 1, 2, ..., j : weights[guess][i] -= \phi_i(datum) \\
&w_0 -= 1
\end{aligned}$$

We repeat this process until nothing is changed in one iteration or the max iteration is reached.

2.5. Classify

Classifies each datum as the label that closely matches the prototype vector for that label.

This is done using the below equation. We have to pick one with the highest value.

$$f(x_i, w) = w_0 + w_1\phi_1 + ... + w_j\phi_j$$

2.6. Data Processing:

We have the training and testing datasets and labels.

- The faces data contains 451 training data points and 150 test data points.
- The digits data includes 5000 training and 1000 test data points.

The training data points were picked randomly in groups of 10%, 20%, 30%, .. 100% to train, tune and compare the accuracy of the classifiers for each percentage used.

The result gets stored in the resultsP file.

The command to run the Perceptron algorithm is as follows:

python dataClassifier.py -c perceptron -d faces -t $amount -i 2 -s 150 >> resultsP.txt

where:

-c is the type of classifier (in this case, Perceptron)

-d is the dataset to use (faces or digits)

-t is the size of the training dataset

-i is the number of iterations

-s is the number of test data points to use

2.7. Inference

2.7.1. Perceptron – FACE

- Training time
  - From the graph, we can observe that time required to train the model with 10% of the training data is 0.701 seconds. When given 100 % of the training data for model training, the time required is 6.762 seconds
  - Furthermore, the graph generated is a straight line of the form Y = MX + C, which indicates that as the percentage increases, the time required to train the model also increases.

- Accuracy
  - From the graph, we can observe that the model's accuracy, when given 10% of the training data, is 71.7 percent. If 100% of the training data is utilized for training the data model, then the accuracy increases to 85.1 percent.
  - Furthermore, the graph generated is almost straight with slight curves, which indicates that as the percentage increases, the accuracy of the model increases steadily.

- Standard Deviation
  - From the graph, we can observe that the standard deviation of the model with 10% of the training data is 0.445. When given 100 % of the training data for model training, the standard deviation of the model is 0.352.
  - Clearly, we can see that as the percentage increases, the standard deviation decreases. This indicates that the percentage of training data fed is inversely proportional to the standard deviation.

2.7.2. Perceptron - DIGIT

- Training time
    - From the graph, we can observe that time required to train the model with 10% of the training data is 9.071 seconds. When given 100 % of the training data for model training, the time required is 87.296 seconds.
    - Furthermore, the graph generated is a straight line of the form Y = MX + C, which indicates that as the percentage increases, the time required to train the model also increases.

- Accuracy
    - From the graph, we can observe that the model's accuracy, when given 10% of the training data, is 72.2 percent. If 100% of the training data is utilized for training the data model, then the accuracy increases to 79.7 percent.
    - Furthermore, the graph has slight variations, which indicates that as the percentage increases, the accuracy of the model increases steadily.

- Standard Deviation
    - From the graph, we can observe that the model's standard deviation, when given 10% of the training data, is 0.447. If 100% of the training data is utilized for training the data model, then the standard deviation decreases gradually to 0.402.
    - Furthermore, the graph generated has variations; initially, the chart declines linearly. The standard deviation upsurges slightly from 0.417 to 0.434 when the model is trained with 40% to 70% of data. Thereafter, the graph falls slowly and finally reaches 0.402.

2.8. **Result and Analysis**

Graphs are generated for the standard deviation, accuracy, and Time to train the dataset against the percentage of training data chosen below.

Epoch value = 5

2.8.1. Perceptron Classification Training time: Face

Perceptron Face									
10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
0.68038321	1.39950705	2.03531194	2.77660108	3.44474196	4.31187415	4.91760683	5.9269073	6.01631474	6.66825795
0.72754788	1.47175598	2.0340271	2.70291376	3.38778114	4.24047303	4.73359609	5.37870908	6.20598912	6.94880795
0.69924092	1.4009881	1.99107599	2.85083699	3.47512031	4.50656319	5.29596806	5.28308988	5.97709703	6.67463088
0.71783996	1.37568498	2.10250998	2.83654881	3.58250284	4.24095893	5.26553392	5.46777511	6.07648921	6.75349188
0.68187094	1.3262639	2.13522983	2.74512506	3.53055906	4.11433411	4.78176117	5.83931208	6.05445385	6.76716614
0.701	1.395	2.060	2.782	3.484	4.283	4.999	5.579	6.066	6.762

Fig2.1: Training time vs Percentage of training data points – Perceptron Face

2.8.2. Perceptron Classification Training time: Digit

Perceptron Digit									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
8.96419907	18.0011299	26.890135	35.202209	44.7496309	54.8439021	64.539552	74.0686131	80.7286427	88.3861589
9.07778478	18.0097249	27.5689399	35.7816463	46.410079	54.2027411	63.458961	72.855752	81.055409	86.8557158
9.28772902	18.603595	26.8979461	35.6774871	44.2474391	54.145102	64.6054981	71.3862991	81.3223233	86.7701759
8.9718132	18.0920568	26.5885172	36.6685729	45.4337647	53.747021	64.3423541	71.7171121	78.5374439	87.5339341
9.05559683	17.733439	26.6900599	35.7466049	46.6054499	55.9350531	63.0717566	72.4104619	79.0213737	86.9350591
9.071	18.088	26.927	35.815	45.489	54.575	64.004	72.488	80.133	87.296

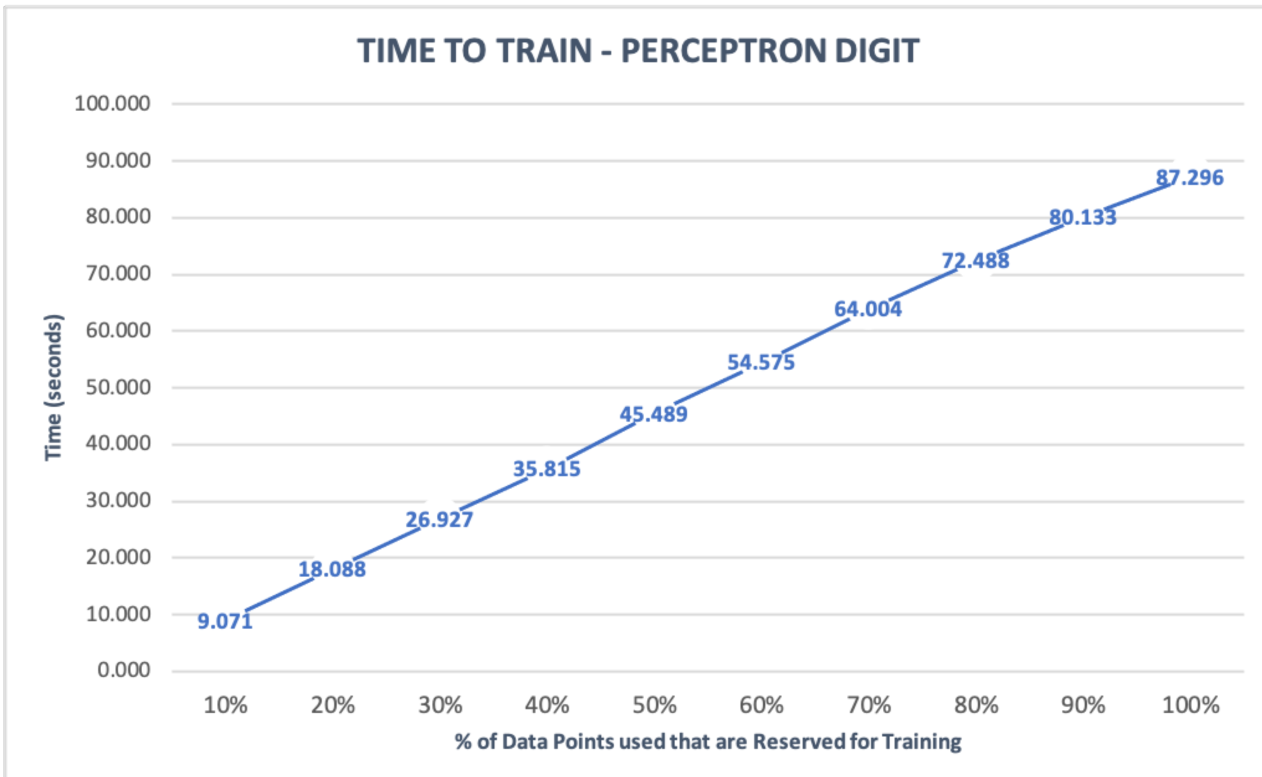

Fig2.2: Training time vs Percentage of training data points – Perceptron Digit

2.8.3. Perceptron Classification Accuracy: Face

Perceptron Face									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
74.70%	80.70%	74.00%	82.70%	79.30%	81.30%	84.70%	82.70%	88.70%	78.70%
68.00%	73.30%	72.00%	82.70%	83.30%	81.30%	86.70%	82.00%	86.70%	83.30%
77.30%	79.30%	73.30%	80.00%	80.70%	84.00%	68.00%	81.30%	78.00%	88.70%
61.30%	60.00%	74.00%	77.30%	83.30%	86.00%	85.30%	84.00%	77.30%	84.70%
77.30%	83.30%	84.00%	80.00%	81.30%	84.70%	82.70%	82.70%	86.70%	90.00%
0.717	0.753	0.755	0.805	0.816	0.835	0.815	0.825	0.835	0.851

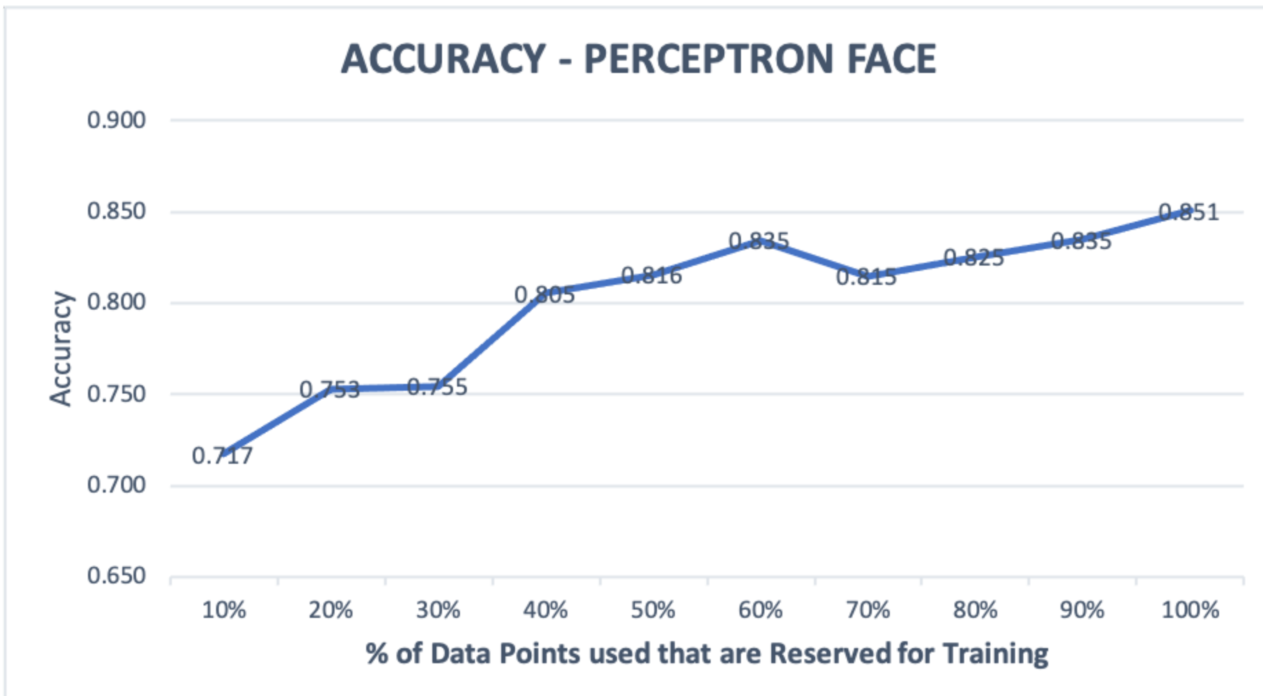

Fig2.3: Accuracy vs Percentage of training data – Perceptron Face

2.8.4. Perceptron Classification Accuracy: Digit

Perceptron Digit									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
73.00%	76.50%	76.90%	75.60%	74.30%	82.00%	76.80%	70.10%	73.90%	80.50%
68.90%	71.20%	72.50%	77.60%	71.50%	74.20%	70.90%	75.10%	78.30%	78.20%
73.20%	74.20%	73.20%	80.90%	82.00%	76.90%	75.00%	75.50%	78.10%	77.70%
70.60%	73.00%	78.10%	77.30%	74.50%	79.90%	71.30%	80.20%	76.10%	80.00%
75.30%	70.00%	76.50%	75.90%	74.30%	76.50%	78.80%	81.40%	77.20%	82.10%
0.722	0.730	0.754	0.775	0.753	0.779	0.746	0.765	0.767	0.797

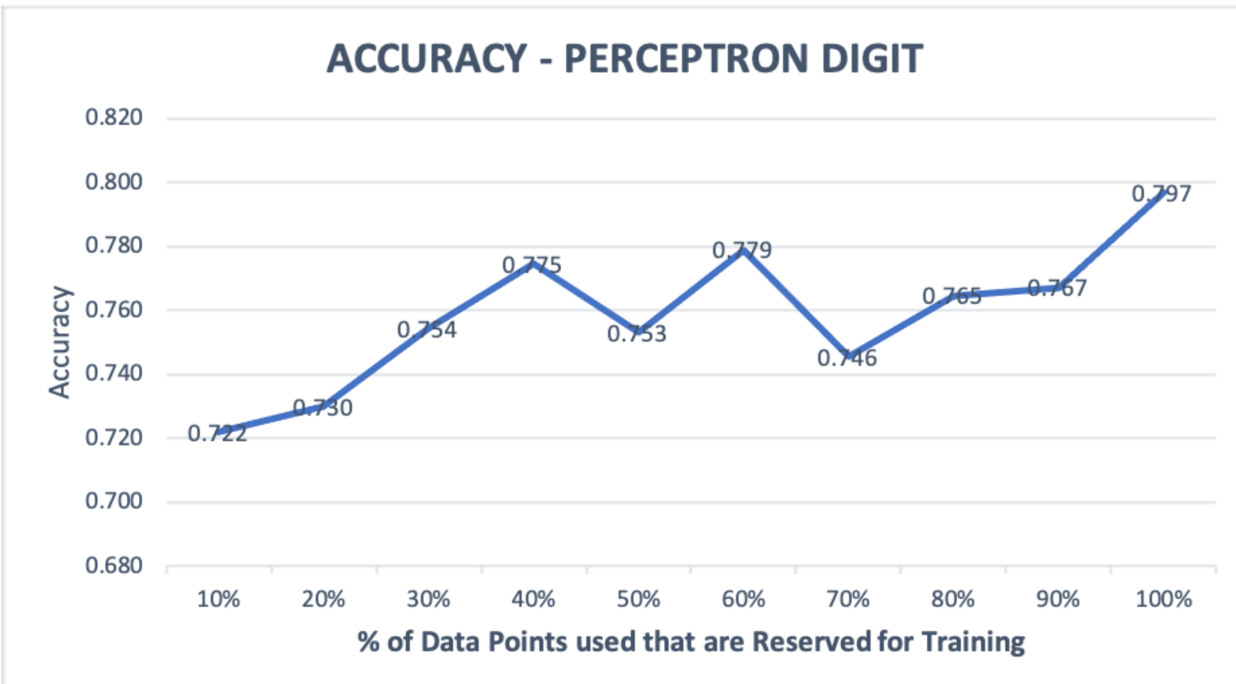

Fig2.4: Accuracy vs Percentage of training data – Perceptron Digit

2.8.5. Perceptron Classification Standard Deviation: Face

Perceptron Face									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0.43492017	0.39491209	0.43863424	0.37853519	0.40491426	0.38964371	0.36030851	0.37853519	0.31699982	0.40966111
0.46647615	0.44221664	0.44899889	0.37853519	0.372678	0.38964371	0.33993463	0.38418745	0.33993463	0.372678
0.41867516	0.40491426	0.44221664	0.4	0.39491209	0.36660606	0.46647615	0.38964371	0.4142463	0.31699982
0.4869862	0.48989795	0.43863424	0.41867516	0.372678	0.34698703	0.35377331	0.36660606	0.41867516	0.36030851
0.41867516	0.372678	0.36660606	0.4	0.38964371	0.36030851	0.37853519	0.37853519	0.33993463	0.3
0.445	0.421	0.427	0.395	0.387	0.371	0.380	0.380	0.366	0.352

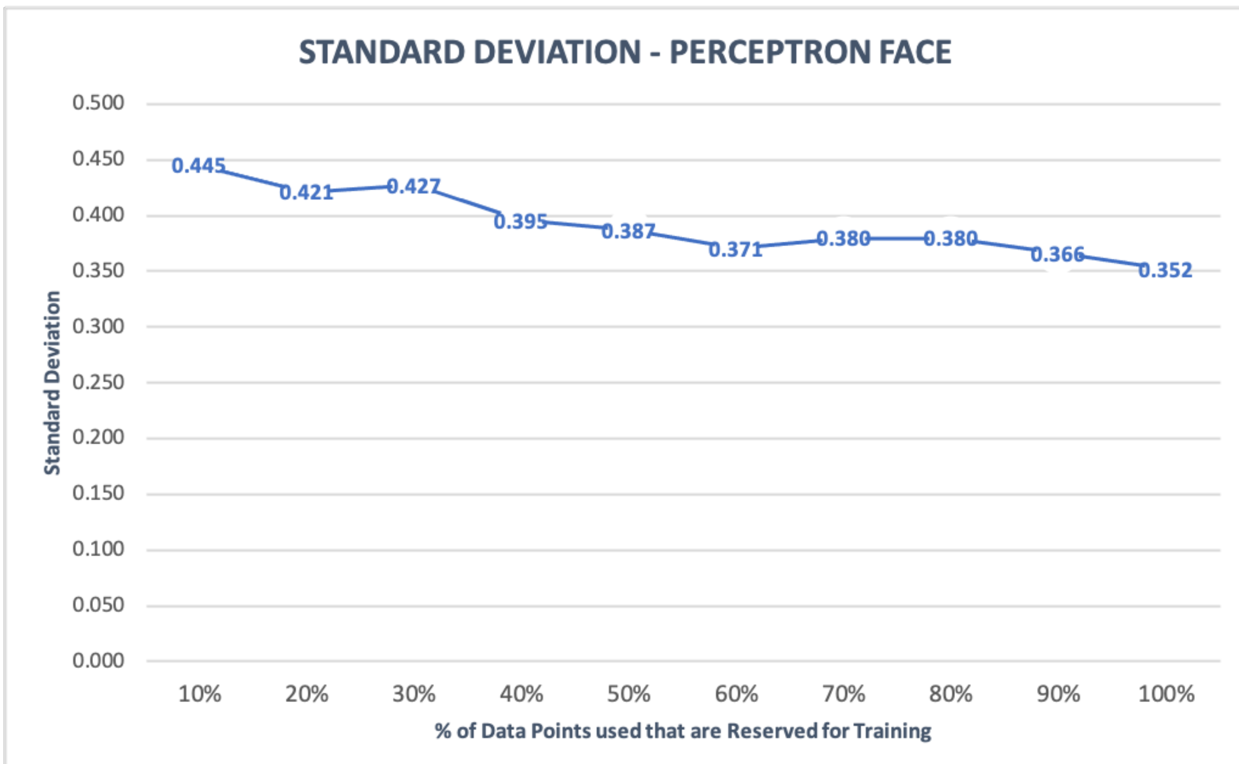

Fig2.5: Standard Deviation vs Percentage of training data – Perceptron Face

2.8.6. Perceptron Classification Standard Deviation: Digit

Perceptron Digit									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0.44395946	0.42399882	0.42147242	0.42949272	0.4369794	0.38418745	0.42210899	0.45781983	0.43917992	0.39620071
0.4629028	0.45283109	0.44651428	0.41692206	0.45141444	0.437534	0.45422351	0.43243381	0.41220262	0.41288739
0.4429176	0.437534	0.4429176	0.39308905	0.38418745	0.42147242	0.4330127	0.4300872	0.41356862	0.41625833
0.45559192	0.44395946	0.41356862	0.41889259	0.43586122	0.40074805	0.45236158	0.39849216	0.42647274	0.4
0.43126674	0.45825757	0.42399882	0.42769031	0.4369794	0.42399882	0.40872485	0.38910667	0.41954261	0.38335232
0.447	0.443	0.430	0.417	0.429	0.414	0.434	0.422	0.422	0.402

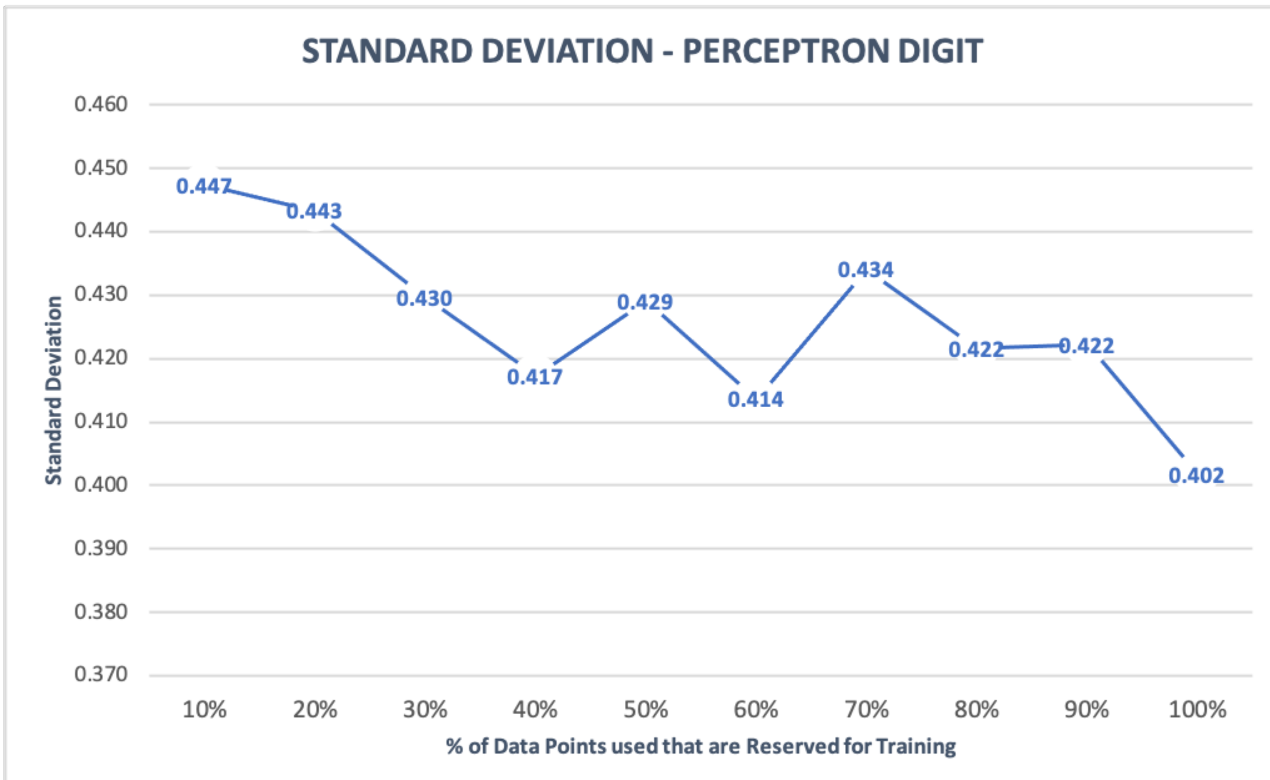

Fig2.6: Standard Deviation vs Percentage of training data – Perceptron Digit

3. K NEAREST NEIGHBOR

3.1. Overview

The **k-nearest neighbors (KNN) algorithm** is a data classification method for estimating the likelihood that a data point will become a member of one group or another based on what group the data points nearest to it belong to.

KNN classifier identifies the class of a data point using the majority voting principle. If k is set to 5, the classes of 5 nearest points are examined. Prediction is made according to the predominant class. Similarly, KNN regression takes the mean value of 5 nearest locations.

It's called a lazy learning algorithm or lazy learner because it doesn't perform any training when you supply the training data. Instead, it just stores the data during training and doesn't perform any calculations. It doesn't build a model until a query is performed on the dataset.

3.2. Features

The feature set includes one feature for each pixel location, which can take values 0 or 1 (off or on). The features are encoded as a Counter where keys are feature locations (represented as (columns, rows)) and values are 0 or 1. The face recognition data set has value one only for those pixels identified by a Canny edge detector.

3.3. Training

Initialize the value of K. To get the predicted class, iterate over the training data and calculate the distance between the test data and each row of the training data. Sort the calculated distances in ascending order based on distance values. Get the top K rows from the sorted array and then get the most frequent class of these rows. Return the predicted class.

3.4. Classify

Find the K closest neighbors of the test image in the training data and then return the label which appeared the most. Pick the training label with the lowest distance if there is a tie.

3.5. Data Processing

We have the training and testing datasets and labels.

- The faces data contains 451 training data points and 150 test data points.
- The digits data includes 5000 training and 1000 test data points.

The training data points were picked randomly in groups of 10%, 20%, 30%,..100% to train, tune and compare the accuracy of the classifiers for each percentage used.

The result gets stored in the resultsK file.

The command to run the K Nearest neighbors' algorithm is as follows:

python dataClassifier.py -c KNN -d faces -t $amount -i 2 -s 150 >> resultsK.txt

where:

-c is the type of classifier (in this case, KNN)

-d is the dataset to use (faces or digits)

-t is the size of the training dataset

-i is the number of iterations

-s is the number of test data points to use

3.6. **Result and Analysis**

Graphs are generated for the standard deviation, accuracy, and Time to train the dataset against the percentage of training data chosen below.

Epoch value = 5

3.6.1. KNN Classification Training time: Face

K-Nearest Face									
10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
0.05812693	0.11184001	0.17135692	0.22787809	0.29724169	0.34467268	0.40985823	0.51904297	0.56127906	0.60962105
0.05330992	0.11891413	0.17491007	0.25169897	0.28666615	0.369838	0.42613721	0.46290302	0.54127908	0.60222077
0.05583096	0.11927795	0.17576408	0.24457097	0.30230594	0.35970497	0.40939999	0.46162224	0.55550694	0.62160683
0.05938482	0.11230206	0.18360114	0.23429179	0.33181024	0.34322524	0.43895698	0.51657271	0.53344274	0.59893799
0.05395699	0.11443996	0.18378997	0.23965693	0.28627896	0.36743927	0.40969491	0.48421383	0.54109788	0.60615587
0.056	0.115	0.178	0.240	0.301	0.357	0.419	0.489	0.547	0.608

Fig3.1: Training time vs Percentage of training data – KNN Face

3.6.2. KNN Classification Training time: Digit

K-Nearest Digit									
10%	**20%**	**30%**	**40%**	**50%**	**60%**	**70%**	**80%**	**90%**	**100%**
0.17756391	0.39387274	0.58042693	78.85%	0.99994111	1.13098884	1.42614508	1.4806602	1.76304197	1.85389709
0.19710732	0.4719708	0.57483912	76.23%	1.11122918	1.10557699	1.27251577	1.69273496	1.75088692	1.77532697
0.18277407	0.36985397	0.59575796	75.89%	1.09645391	1.11727929	1.28307509	1.88873315	1.72667766	1.94458199
0.18710709	0.4041543	0.59107828	76.65%	1.02046418	1.08594394	1.25152302	1.54047894	1.70394778	2.03464198
0.19402194	0.37070823	0.56342816	74.51%	0.95731592	1.14672804	1.34832001	1.64179206	1.7152741	1.99365401
0.188	0.402	0.581	0.764	1.037	1.117	1.316	1.649	1.732	1.920

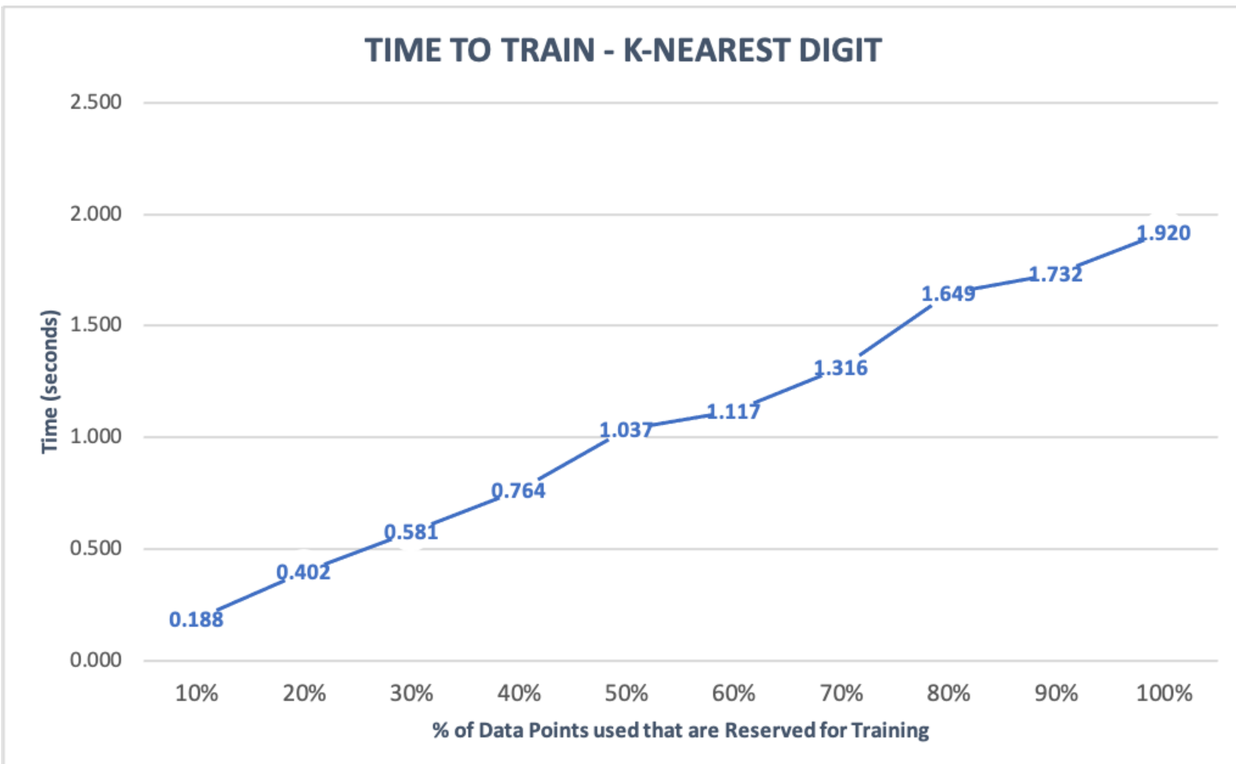

Fig3.2: Training time vs Percentage of training data – KNN Digit

3.6.3. KNN Classification Accuracy: Face

K-Nearest Face									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
60.00%	76.00%	76.70%	79.30%	77.30%	82.00%	82.70%	84.70%	83.30%	82.00%
75.30%	76.00%	80.70%	79.30%	78.70%	78.00%	78.00%	79.30%	79.30%	81.30%
78.00%	74.70%	76.00%	78.70%	79.30%	78.00%	82.70%	78.00%	79.30%	81.30%
74.00%	75.30%	77.30%	79.30%	78.70%	80.00%	80.00%	81.30%	80.70%	82.00%
68.70%	75.30%	74.70%	80.00%	78.70%	82.00%	80.70%	79.30%	80.00%	81.30%
0.712	0.755	0.771	0.793	0.785	0.800	0.808	0.805	0.805	0.816

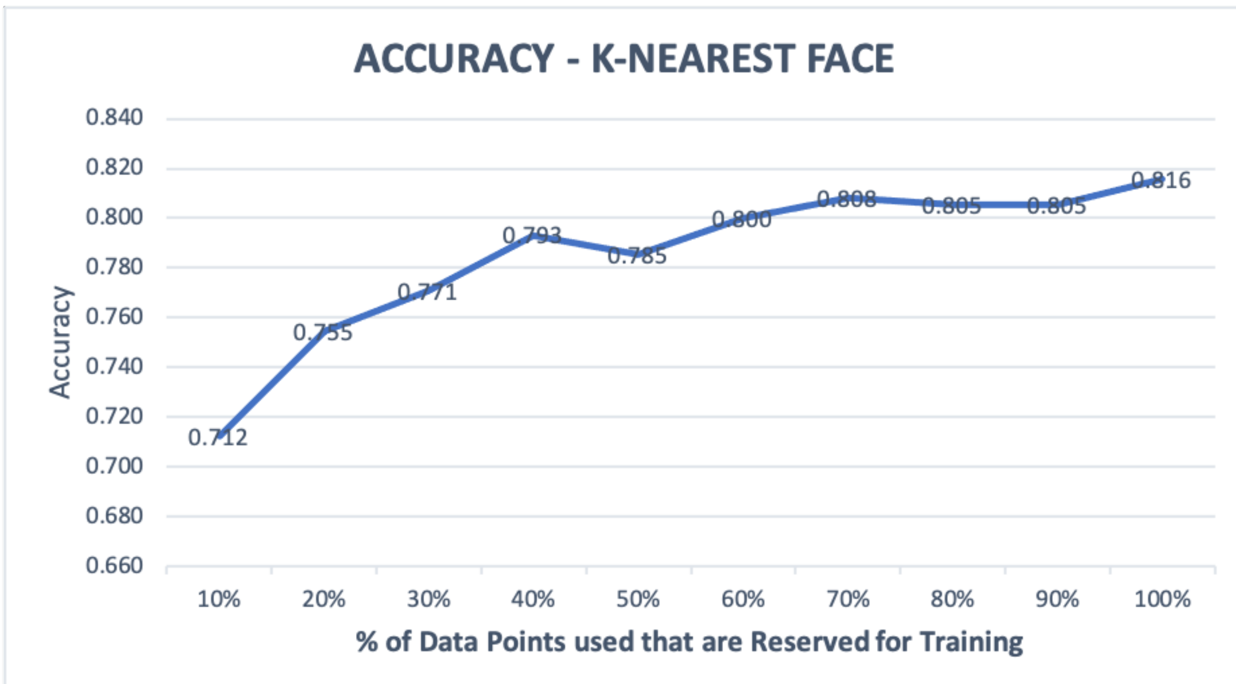

Fig3.3: Accuracy vs Percentage of training data – KNN Face

3.6.4. KNN Classification Accuracy: Digit

K-Nearest Digit									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
62.60%	63.90%	67.10%	67.50%	69.90%	67.90%	68.90%	70.50%	70.80%	70.30%
59.50%	65.30%	65.50%	67.70%	69.90%	68.90%	70.10%	70.80%	70.80%	69.80%
64.20%	66.00%	67.10%	67.70%	69.60%	71.00%	69.80%	70.10%	71.20%	71.10%
62.20%	65.70%	67.10%	70.90%	67.50%	69.60%	69.50%	69.60%	71.40%	71.50%
59.40%	64.50%	68.10%	66.50%	67.90%	68.60%	70.10%	71.40%	69.20%	70.90%
0.616	0.651	0.670	0.681	0.690	0.692	0.697	0.705	0.707	0.707

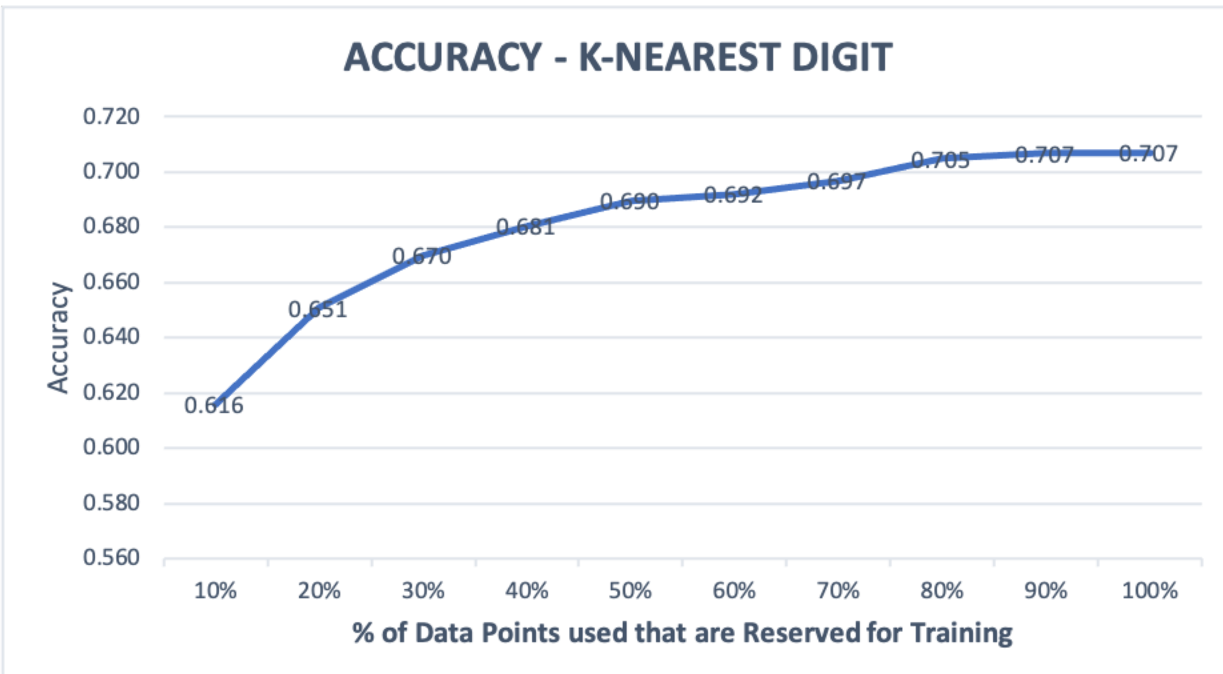

Fig3.4: Accuracy vs Percentage of training data – KNN Digit

3.6.5. KNN Classification Standard Deviation: Face

K-Nearest Face									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0.48989795	0.42708313	0.42295258	0.40491426	0.41867516	0.38418745	0.37853519	0.36030851	0.372678	0.38418745
0.43107102	0.42708313	0.39491209	0.40491426	0.40966111	0.4142463	0.4142463	0.40491426	0.40491426	0.38964371
0.4142463	0.43492017	0.42708313	0.40966111	0.40491426	0.4142463	0.37853519	0.4142463	0.40491426	0.38964371
0.43863424	0.43107102	0.41867516	0.40491426	0.40966111	0.4	0.4	0.38964371	0.39491209	0.38418745
0.46384863	0.43107102	0.43492017	0.4	0.40966111	0.38418745	0.39491209	0.40491426	0.4	0.38964371
0.448	0.430	0.420	0.405	0.411	0.399	0.393	0.395	0.395	0.387

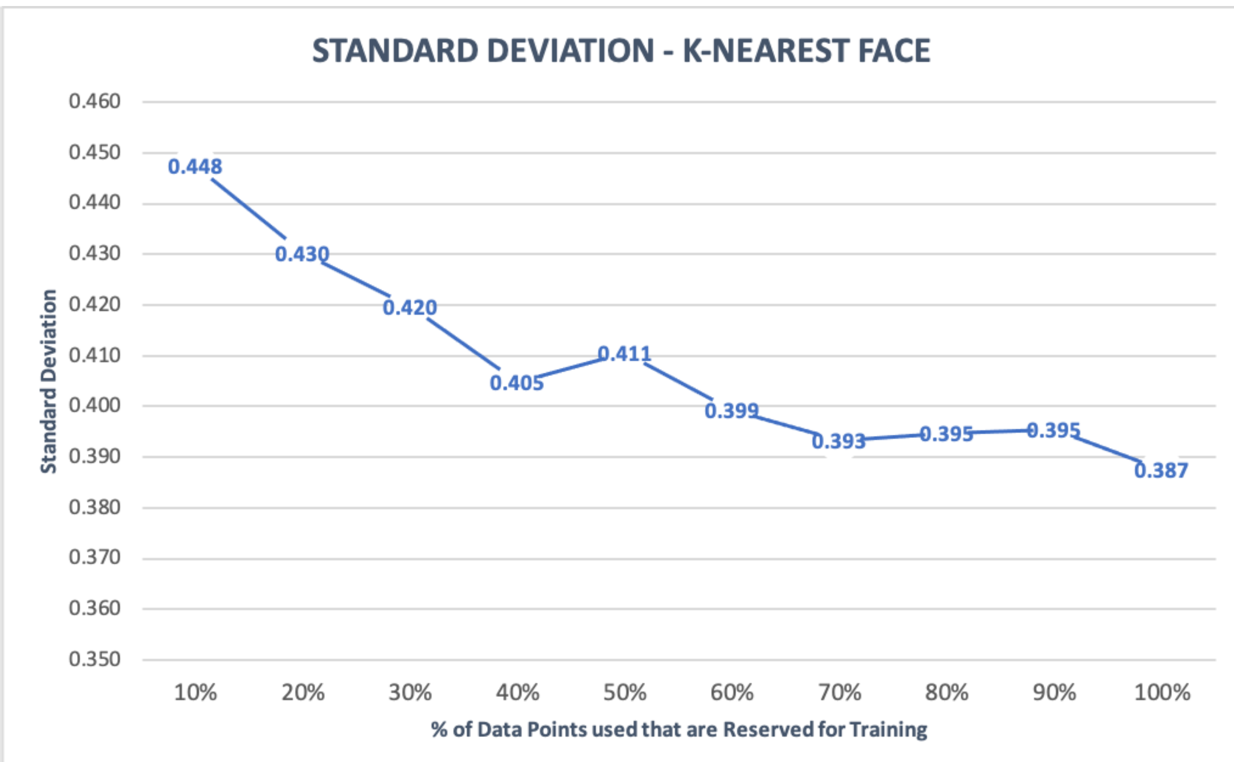

Fig3.5: Standard Deviation vs Percentage of training data – KNN Face

3.6.6. KNN Classification Standard Deviation: Digit

K-Nearest Digit									
10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0.48386362	0.48029054	0.46984998	0.46837485	0.45869271	0.46686079	0.4629028	0.45604276	0.45468231	0.45693654
0.49089205	0.47601576	0.47536828	0.46762271	0.45869271	0.4629028	0.45781983	0.45468231	0.45468231	0.45912526
0.47941214	0.47370877	0.46984998	0.46762271	0.45998261	0.45376205	0.45912526	0.45781983	0.45283109	0.45329792
0.48488762	0.47471149	0.46984998	0.45422351	0.46837485	0.45998261	0.46040743	0.45998261	0.45188937	0.45141444
0.49108451	0.47851332	0.46608905	0.47199047	0.46686079	0.46411636	0.45781983	0.45188937	0.46166655	0.45422351
0.486	0.477	0.470	0.466	0.463	0.462	0.460	0.456	0.455	0.455

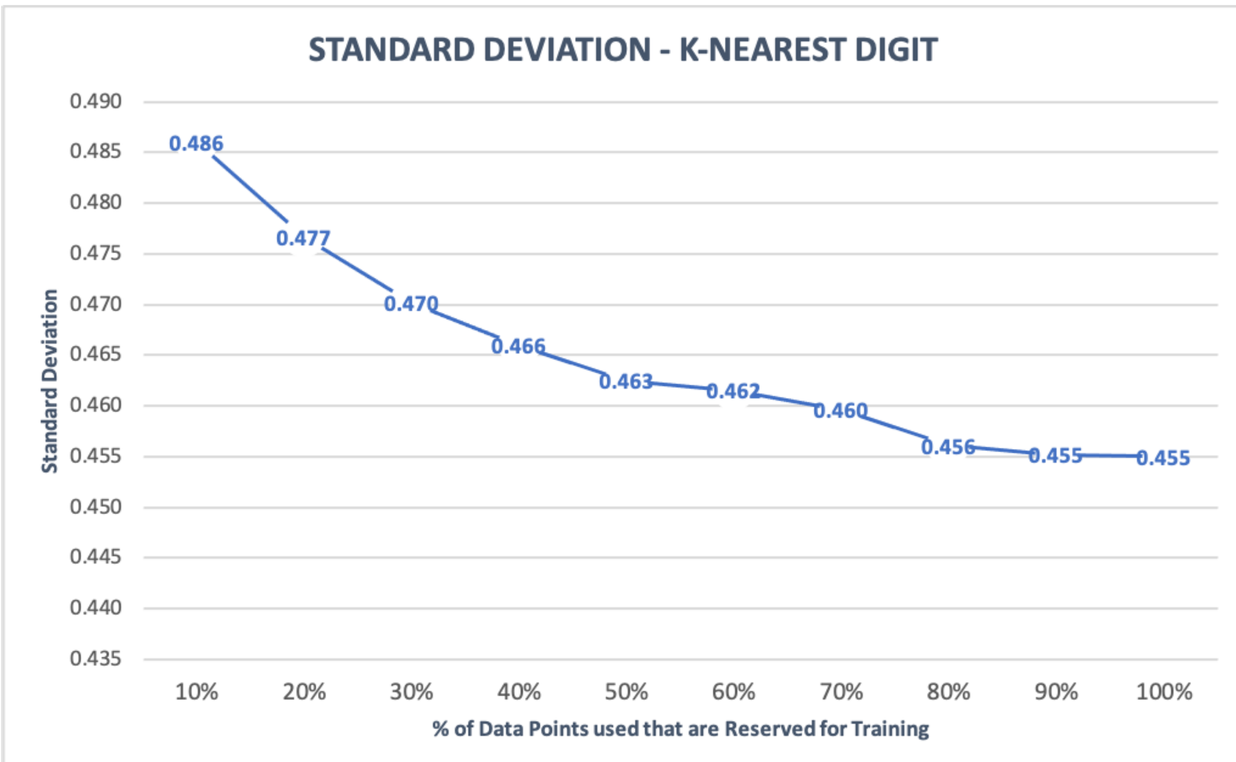

Fig3.6: Standard Deviation vs Percentage of training data – KNN Digit

3.7. **Inference**

3.7.1. KNN - FACE

- Training Time
    - From the graph, we can observe that time required to train the model with 10% of the training data is 0.056 seconds. When given 100 % of the training data for model training, the time required is 0.608 seconds
    - Furthermore, the graph generated is a straight line of the form Y = MX + C, which indicates that as the percentage increases, the time required to train the model also increases.

- Accuracy
    - From the graph, we can observe that the model's accuracy, when given 10% of the training data, is 71.2 percent. If 100% of the training data is utilized for training the data model, then the accuracy increases to 81.6 percent.
    - Furthermore, the graph generated is almost straight with slight curves, which indicates that as the percentage increases, the accuracy of the model increases steadily.

- Standard Deviation
    - From the graph, we can observe that the standard deviation of the model with 10% of the training data is 0.448. When given 100 % of the training data for model training, the standard deviation of the model is 0.387.
    - Clearly, we can see that as the percentage increases, the standard deviation decreases. This indicates that the percentage of training data fed is inversely proportional to the standard deviation.

3.7.2. KNN – DIGIT

- Time to Train
  - From the graph, we can observe that time required to train the model with 10% of the training data is 0.188 seconds. When given 100 % of the training data for model training, the time required is 1.920 seconds
  - Furthermore, the graph generated is a straight line of the form Y = MX + C, which indicates that as the percentage increases, the time required to train the model also increases

- Accuracy
  - From the graph, we can observe that the model's accuracy, when given 10% of the training data, is 61.6 percent. If 100% of the training data is utilized for training the data model, then the accuracy increases to 70.7 percent.
  - Furthermore, the graph has slight variations, which indicates that as the percentage increases, the accuracy of the model increases.

- Standard Deviation
  - From the graph, we can observe that the standard deviation of the model with 10% of the training data is 0.486. When given 100 % of the training data for model training, the standard deviation of the model is 0.455
  - Clearly, we can see that as the percentage increases, the standard deviation decreases. This indicates that the percentage of training data fed is inversely proportional to the standard deviation.

4. Comparison and Analysis

4.1. FACE

- Training Time
  - For the face’s dataset, the time required to train the model in the case of naive Bayes, perceptron, and k nearest neighbor is 2.02, 6.762, and 0.608 seconds, respectively, when 100% of the training data is used to train the model.
  - Comparatively, Time required to train the model for KNN is less than the other two algorithms. On the other hand, Perceptron takes more time to train the data. Here KNN outperforms other algorithms

- Accuracy
  - For the face’s dataset, the model's accuracy in the case of naive Bayes, perceptron, and k nearest neighbor is 90%, 85.1%, and 81.6%, respectively, when 100% of the training data is used to train the model.
  - Comparatively, the model's accuracy for Naive Bayes is more than the other two algorithms. On the other hand, KNN has least accuracy. Here Naive Bayes outperforms other algorithms

- Standard deviation
  - For the face’s dataset, the standard deviation of the model in the case of naive Bayes, perceptron, and k nearest neighbor is 0.3, 0.352, and 0.387, respectively, when 100% of the training data is used to train the model.
  - Comparatively, the standard deviation of the model for naive Bayes is less than the other two algorithms. On the other hand, k's nearest neighbor has a more standard deviation. Here Naive Bayes outperforms other algorithms

4.2. DIGIT

- Training Time
  - For the digit's dataset, the time required to train the model in the case of naive Bayes, perceptron, and k nearest neighbor is 4.127, 87.296, and 1.920 seconds, respectively, when 100% of the training data is used to train the model.
  - Comparatively, Time required to train the model for KNN is less than the other two algorithms. On the other hand, Perceptron takes more time to train the data. Here KNN outperforms other algorithms

- Accuracy
  - For the face's dataset, the model's accuracy in the case of naive Bayes, perceptron, and k nearest neighbor is 77.3%, 79.7%, and 70.7%, respectively, when 100% of the training data is used to train the model.
  - Comparatively, the model's accuracy for perceptron is more than the other two algorithms. On the other hand, the KNN algorithm least accuracy among the three algorithms. Here Perceptron outperforms other algorithms

- Standard deviation
  - For the face's dataset, the standard deviation of the model in the case of naive Bayes, perceptron, and k nearest neighbor is 0.419, 0.402, and 0.455, respectively, when 100% of the training data is used to train the model.
  - Comparatively, the Standard deviation of the model for Perceptron is less than the other two algorithms. On the other hand, KNN has a more standard deviation than others. Here Perceptron outperforms other algorithms