Rutgers, The State University of New Jersey

Department of Computer Science

CS520: Intro to Artificial Intelligence

Project Report

Face & Digit Classification

Prof. Abdeslam Boularias

Team Members: -

Mansi Rajesh Borole (mb2208) 218002227

Hriday Anup Purohit (hp580) 218003087

Shreya Rajendra Pai (sp2305) 219001216

Contents

We have implemented 3 classification algorithms for detecting faces and classifying digits.

1. Naïve Bayes Algorithm

1.1 Algorithm Implementation:

a. Features: We have explored with the Basic Features Extractor which considers every pixel of the grid (digit/ face) as a 0 or a 1. And we have used the matrices forming the pixels as the features that we input to the model.

We have also explored with the Enhanced Feature Extractor which takes a block of 4 across the 28x28 grid of digits and a block of 5 across the 60x70 grid of faces; and counts the number of colored pixels in the block. We have used the resulting 7x7 matrix of digits and 12x14 matrix of faces with the number of pixels in each cell as the feature set to be inputted to the models. Each feature can have values from 0,1,2,…,16 for digits and 0,1,2,…,25 for faces.

b. Training & Tuning: We had the input as dictionaries with coordinates as keys and the values were either 0 or 1.

   a. Prior: We first calculate the prior probabilities wherein we initialize a dictionary which counts the probability of each of the digits/ face or not face (0 or 1). We normalize the frequency of the values to calculate the prior.

   For basic feature extractor:

$$P(y = true) = \frac{Number\ of\ times\ y_i = \ true\ in\ X_{train}}{Total\ number\ of\ training\ data}$$

For enhanced feature extractor:

$$P(y = 0,1,2, \dots 16) = \frac{Number\ of\ blocks\ of\ size\ 4x4\ with\ label\ y\ in\ X_{train}}{Total\ number\ of\ training\ data}$$

   b. Conditional: We maintain an "Occurrance_Counter" that counts the occurrence of digits/ 0 or 1 given the label.

   For basic feature extractor:

$$P(x \mid y = true) = \prod_{j=1}^{l} P\left(\emptyset_j(x) \middle| y = true\right)$$

$$\text{where } l = number\ of\ features\ (faces: 70x60;\ digits: 28x28)$$

For enhanced feature extractor:

$$P(x \mid y = 0,1,2,\ldots.,16) = \prod_{j=1}^{l} P\left(\emptyset_j(x) \middle| y = label\right)$$

$$\text{where } l = number\ of\ features\ (faces: 14x12;\ digits: 7x7)$$

$$and\ label = \{\text{faces: } 1,2,\ldots,24,25;\ \text{digits: } 1,2\ \ldots,15,16\}$$

We then smooth the occurrences by using a constant *k*, adding k to numerator and denominator while finding the conditional probability so that the joint probability product doesn't become 0. We then calculate the probability P(Y| label) and store this in a dictionary. We then calculate the conditional probability per feature and per label.

c. **Classify**: Here, we take the log of the joint probabilities for every label. The label with the maximum probability will be our final guess.
d. **Results**: We have observed that the enhanced feature extractor performs better as compared to the basic feature extractor. So, we will be using the enhanced feature extractor for the remainder of this report.

1.2 Digits Dataset Accuracies

The following are the results for digits on 10%, 20%, 30%, 40% … 100% **DIGIT** data using the Naïve Bayes algorithm with varying k (smoothing) values.

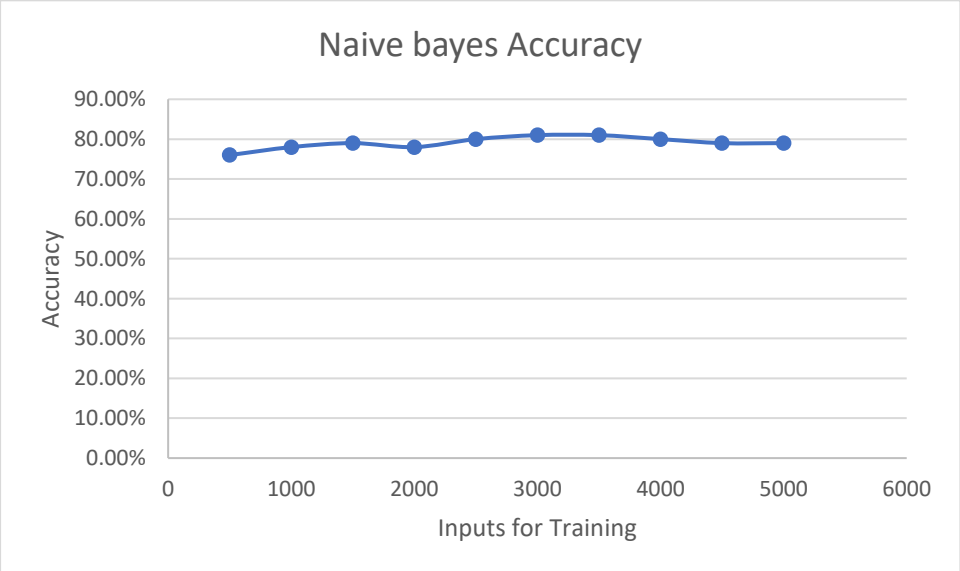

Fig1: Digits Dataset: Accuracy vs Inputs using Naïve Bayes Algorithm, smoothing with hyper parameters

1.3 Faces Dataset Accuracies

The following are the results for digits on 10%, 20%, 30%, 40% … 100% **FACE** data using the Bayesian algorithm with varying k (smoothing) values.

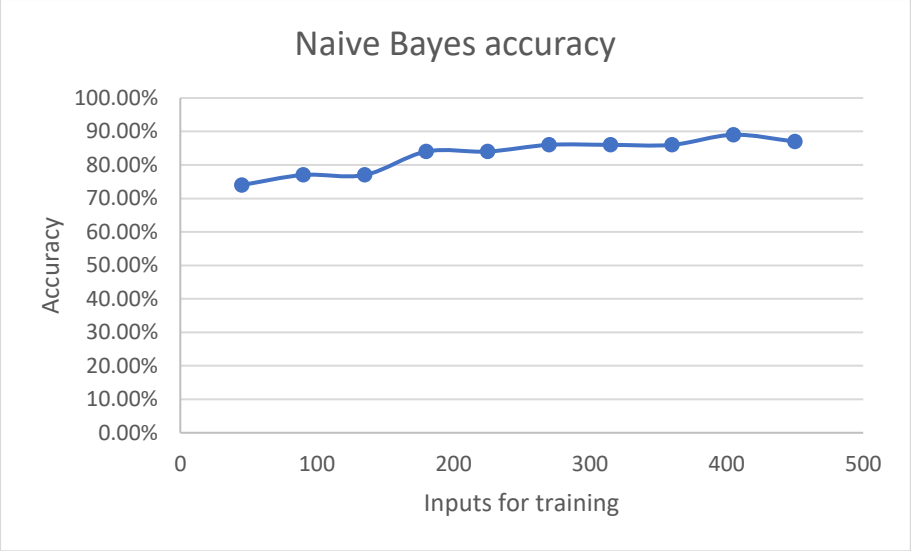

Fig2: Faces Dataset: Accuracy vs Inputs using Naïve Bayes Algorithm, smoothing with hyper parameters

2. Perceptron

2.1 Algorithm Implementation:

a. Features: Here, we used the same features as that in Naïve Bayes. We have initialized the weights to be -999999999.

b. Training: The max iterations are set as 3 and we stop the training of our program when we reach the maximum iterations. The dictionary "weights" stores the label as the key with the weights as the value. We compute a list of numbers corresponding to all the legal labels. We do this by using the equation:

$$f(x_i, w) = w_0 + w_0 \emptyset_1(x_i) + w_2 \emptyset_2(x_i) + \cdots + w_l \emptyset_l(x_i)$$

Guess: We pick the number with the highest $f(x_i, w)$ as out guess.

If $f(x_i, w) >= 0$ and y_i is true or if If $f(x_i, w) < 0$ and y_i is false; just move to the next x_{i+1} input and y_{i+1} label.

For basic feature extractor:

If $f(x_i, w) < 0$ and y_i = label;

$$w_j = w_j + \emptyset_j(x_i) \; for \; j = 1,2, \ldots, l$$

and

$$w_0 = w_0 + 1$$

If f(x_i , w) >= 0 and y_i = label;

$$w_j = w_j - \emptyset_j(x_i) \; for \; j = 1,2,\dots,l$$

and

$$w_0 = w_0 - 1$$

The label is 1,2,…,8,9 for digits and true/ false for faces. There will be different f(x_i , w) computations for each digit.

c. Classify: We compute a list of numbers corresponding to all the legal labels. We do this by using the equation : -

$$f(x_i, w) = w_0 + w_0\emptyset_1(x_i) + w_2\emptyset_2(x_i) + \cdots + w_l\emptyset_l(x_i)$$

We pick the label with the highest f(x , w) as our guess.

e. Results: As with the Naïve Bayes algorithm, we received very poor results when we ran the models with the basic feature extractor. So, we are showing results for only the enhanced feature extractor.

2.2 Digits Dataset Accuracies

The following are the results for digits on 10%, 20%, 30%, 40% … 100% **DIGIT** data using the Perceptron algorithm with iterations=10;

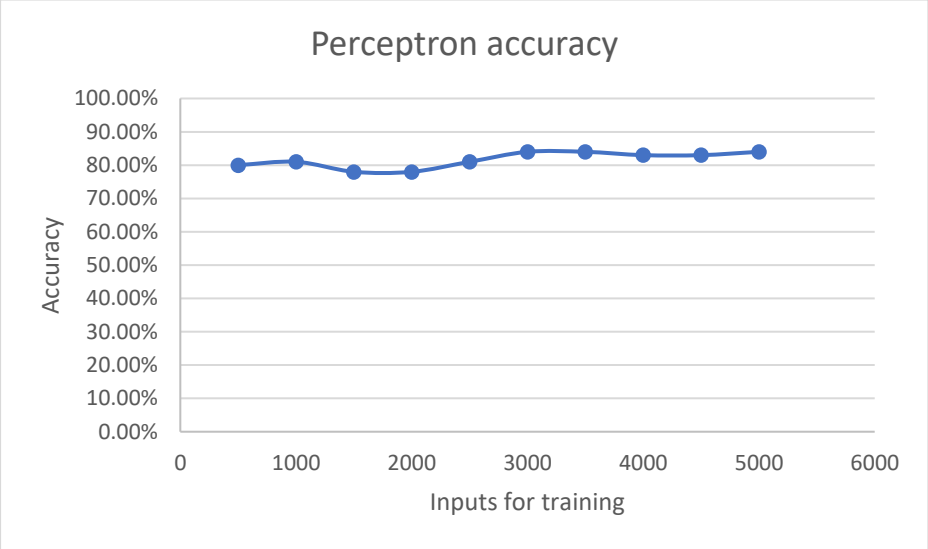

Fig3: Digit Dataset: Accuracy vs Inputs using Perceptron, iterations =10

2.3 Face Dataset Accuracies

The following are the results for digits on 10%, 20%, 30%, 40% … 100% **FACE** data using the Perceptron algorithm with 10 iterations;

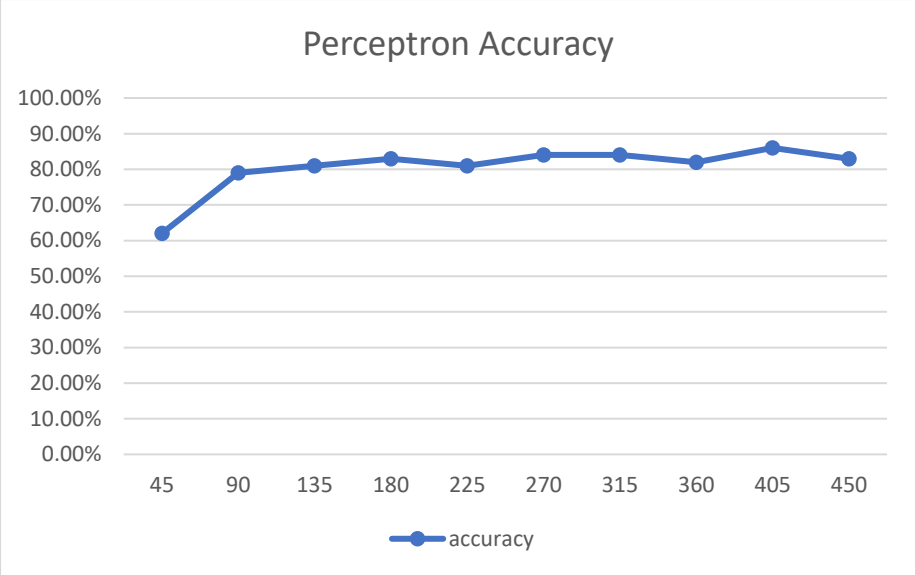

Fig4: Faces Dataset: Accuracy vs Inputs using Perceptron, iterations =10

3. KNN

3.1 Algorithm Implementation:

a. Data Pre-processing: We first pre-process the data and we are storing the dictionaries of lists of coordinates (marked as pixels 0 and 1), as a list of matrices of size 28x28 (representing each digit). We are also storing the labels as lists.

b. Features: We consider every pixel of the grid (digit/ face) as a 0 or a 1. And we have used the matrices forming the pixels as the features that we input to the model.

c. Training & Testing: We have taken different k values from 3 to 100 with a step size of 2. We fit the KNN model to the X_{train} and y_{train} dataset. We then predict the accuracy of the model. We store the k value of the model which showed the highest accuracy. Finally, we calculate the accuracy of the model with the highest k value.

2.2 Digits Dataset Accuracies

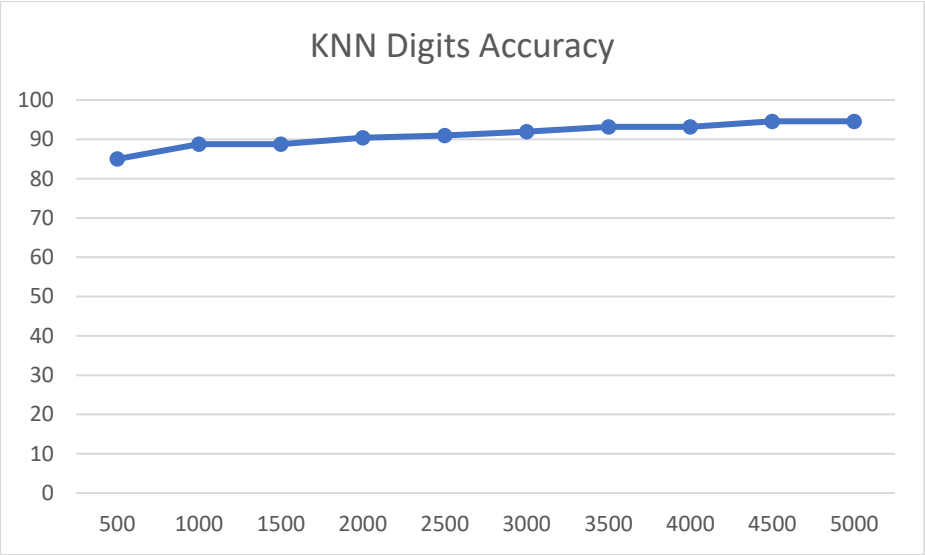

Fig5: Digits Dataset: Accuracy vs Inputs using KNN

2.3 Face Dataset Accuracies

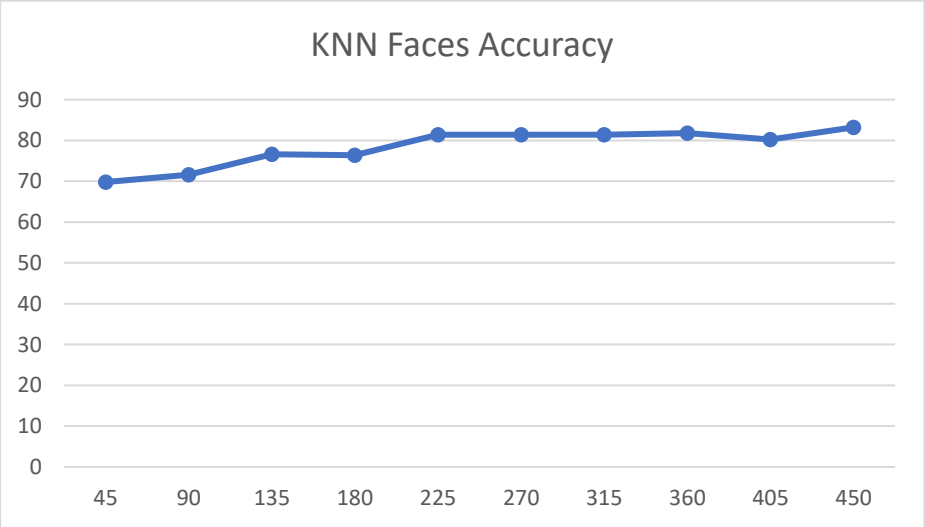

Fig6: Faces Dataset: Accuracy vs Inputs using KNN

4. Algorithms Performance Comparison

We have selected random data points for 10%, 20%, 30% .. and run the algorithms 5 times. Took the mean and standard deviation for each iteration of the input data size.

4.1 Faces Mean, Standard Deviation, Time Comparison

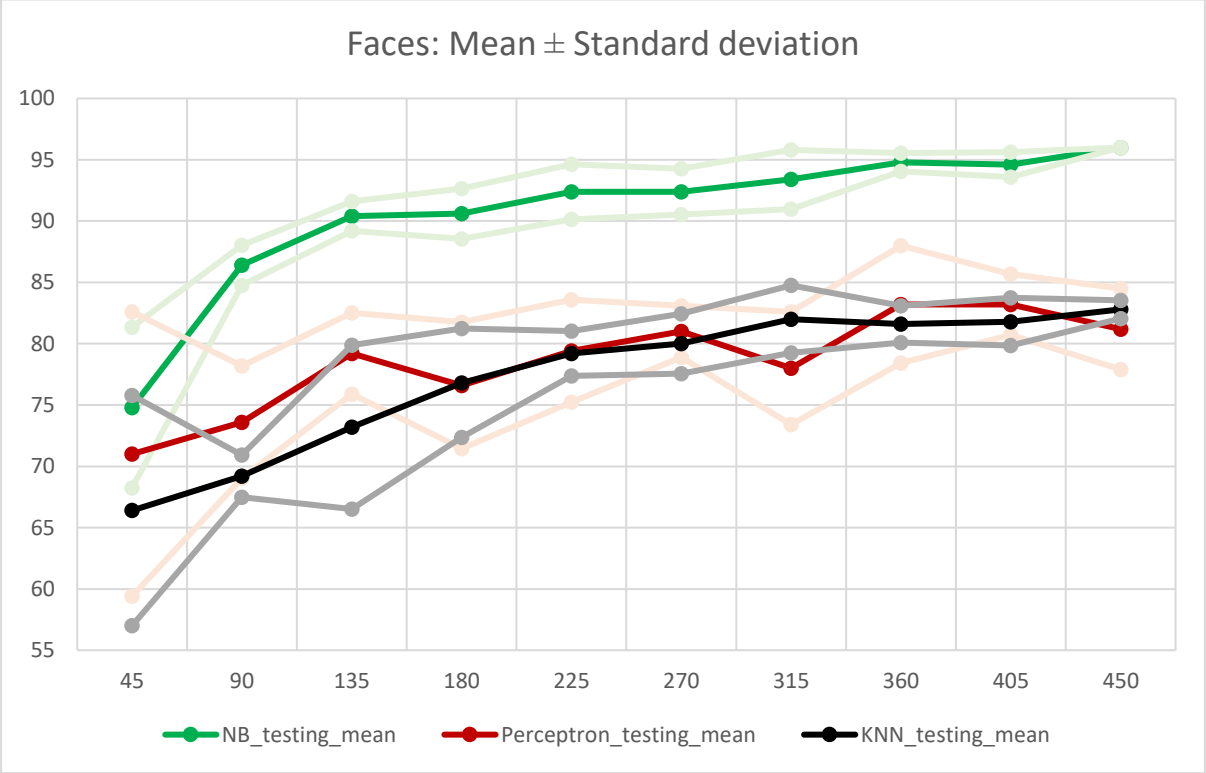

Fig7 : Faces Dataset Mean Accuracy comparison

	NB_mean	NB_std	Perceptron _mean	Perceptron_ testing_std	KNN_mean	KNN_std
45	74.8	6.554388	71	11.5931	66.4	0.093936
90	86.4	1.624808	73.6	4.586938	69.2	0.017205
135	90.4	1.2	79.2	3.310589	73.2	0.066753
180	90.6	2.059126	76.6	5.161395	76.8	0.044452
225	92.4	2.244994	79.4	4.176123	79.2	0.01833
270	92.4	1.854724	81	2.097618	80	0.024495
315	93.4	2.416609	78	4.604346	82	0.027568
360	94.8	0.748331	83.2	4.791659	81.6	0.014967
405	94.6	1.019804	83.2	2.481935	81.8	0.019391
450	96	0	81.2	3.310589	82.8	0.007483

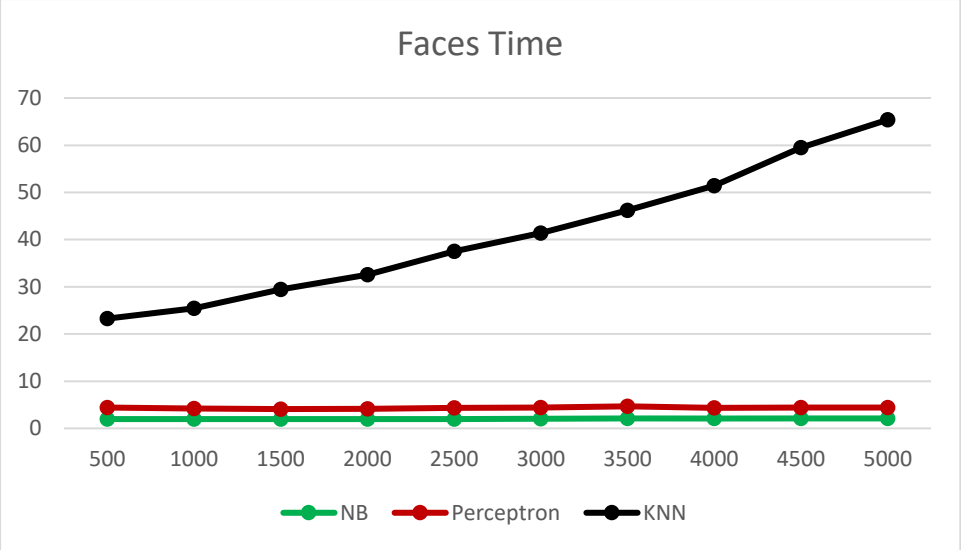

Fig8: Faces Dataset: Time Comparison

Inference:

Naïve Bayes performs best for faces dataset with 96% accuracy.

The Standard Deviation perceptron is not 0 when we consider the full dataset due the weights assigned. It is 0 for others because we are considering random datapoints in the full dataset which is nothing but the full dataset.

4.2 Digits Mean, Standard Deviation, Time

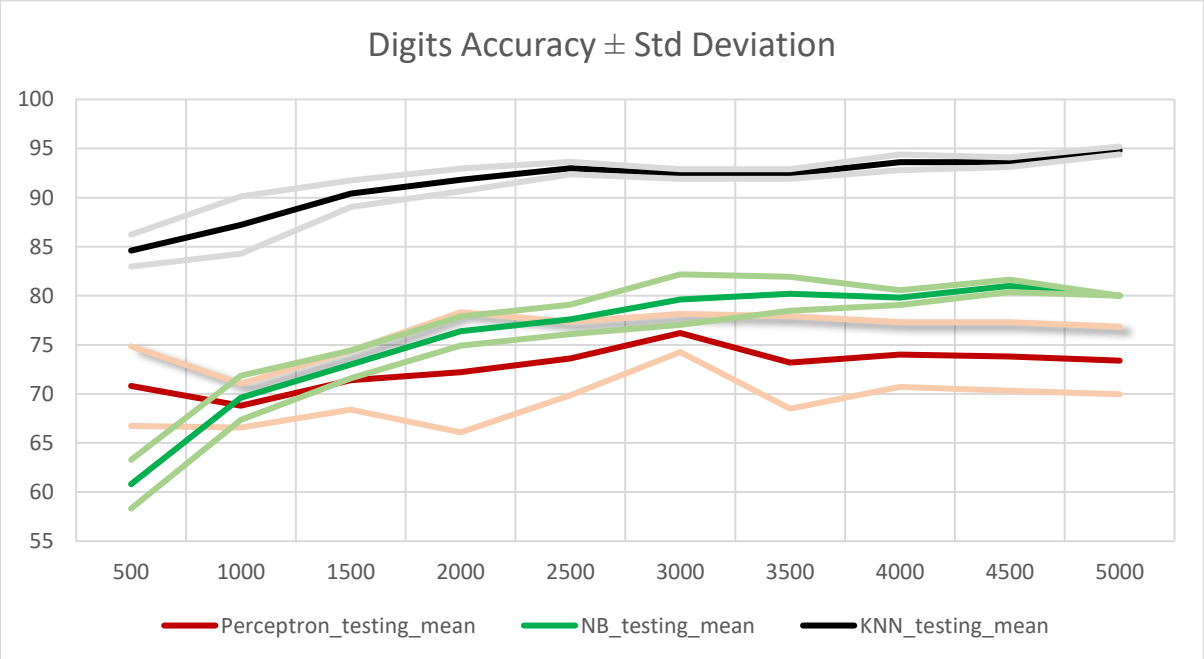

Fig9: Digits Dataset Mean Accuracy comparison

set size	Perceptron_testing_mean	Perceptron_testing_std	NB_testing_mean	NB_testing_std	KNN_testing_mean	Perceptron_testing_std
500	70.8	4.069398	60.8	2.481935	84.6	4.069398
1000	68.8	2.227106	69.6	2.244994	87.2	2.227106
1500	71.4	3.006659	73	1.414214	90.4	3.006659
2000	72.2	6.112283	76.4	1.496663	91.8	6.112283
2500	73.6	3.720215	77.6	1.496663	93	3.720215
3000	76.2	1.939072	79.6	2.57682	92.4	1.939072
3500	73.2	4.707441	80.2	1.720465	92.4	4.707441
4000	74	3.286335	79.8	0.748331	93.6	3.286335
4500	73.8	3.487119	81	0.632456	93.6	3.487119
5000	73.4	3.44093	80	0	94.8	3.44093

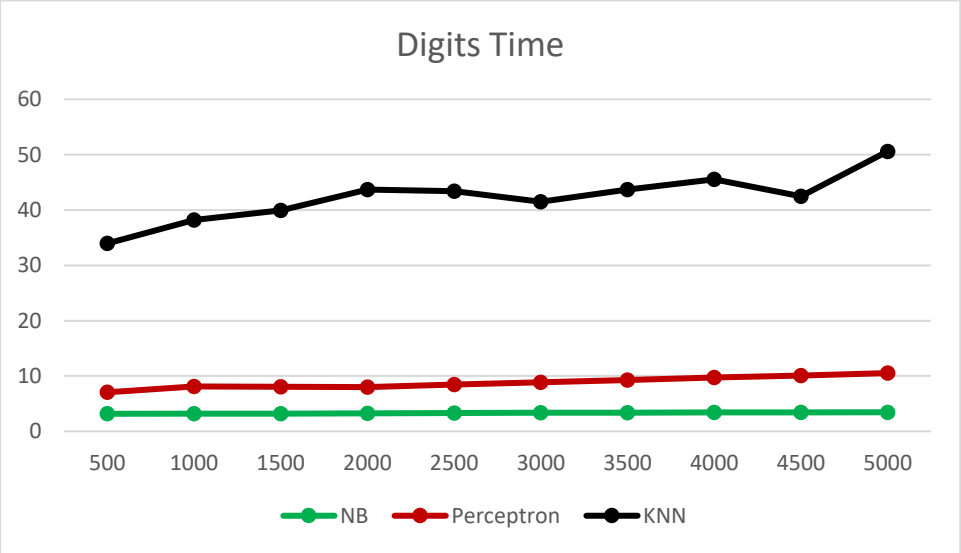

Fig10: Digits Dataset: Time Comparison

Inference:

KNN performs best for faces dataset with 94% accuracy.

The Standard Deviation perceptron is not 0 when we consider the full dataset due the weights assigned. It is 0 for others because we are considering random datapoints in the full dataset which is nothing but the full dataset.

Citations:

Wrapper Code and Datasets
http://inst.eecs.berkeley.edu/~cs188/sp11/projects/classification/classification.html

Naive Bayes:
https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Perceptron:
https://towardsdatascience.com/perceptron-learning-algorithm-d5db0deab975

KNN:
https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761