CS520 Project: Face and Digit Classification

Sehej Jain	Ganesh Rohit Basam	Gayathri Ravipati
sj1030	gb555	gr485

December 2022

Acknowledgement:
This project is based on the one created by Dan Klein and John DeNero that was given as part of the programming assignments of Berkeley's CS188 course.

Problem Statement:
In this project, you will design three classifiers: a Naive Bayes classifier, a perceptron classifier and a classifier of your choice. You will test your classifiers on two image data sets: a set of scanned handwritten digit images and a set of face images in which edges have already been detected. Even with simple features, your classifiers will be able to do quite well on these tasks when given enough training data.

Introduction:
We have implemented three different classification algorithms to detect handwritten digit images and a set of face images. We have implemented the following algorithms: (1) Naive Bayes (2) Perceptron and (3) Support Vector Classification. We have implemented this entire project in python3. We have written the entire algorithms for the Naive Bayes Classifier and the Perceptron classifier and we used the inbuilt SVC classifier from the Scikit-learn machine learning library for Python.

In this report, we first give a brief description of each of these algorithms and then we talk about their implementations in each section respectively. This is followed by the Results section which explains the various results observed during the experimentation using the three different Classification Algorithms. We end this report with the observed conclusions based on the results obtained.

Data Used:

We have used two different data sets containing images of : (1) a set of scanned and processed handwritten digit images and (2) a set of face images in which edges have already been detected.
We divided each image dataset into training data and test data. We have tested our algorithms on the test data after training them over the training set.

Feature extraction:

We have used two different types of feature extractors to get the image data.

(1) **Basic Feature Extractor**: We have used the basic feature extraction to get each pixel from the image and used them to train our Models. So, using this we had 784 pixels or features for each image of digits and 4200 pixels/features for each image of the face.

(2) **Square Feature Extractor**: We have written a new feature extraction algorithm which clubs multiple pixels of the image to make up a single feature. Each feature is made up of 'n*n' pixels. This value of n can be changed too. We have used 2*2 and 4*4 dimensions for the digit classification and used 2*2 and 5*5 dimensions for the face classification.

Algorithms

Perceptron

Perceptron is considered as a supervised learning classification algorithm. The important parameters include initialized weights, extracted feature values, bias, sum and activation function. The perceptron model initially multiplies all the initialized weights with the corresponding feature values. Sum of these individual products gives us the weighted sum. We also consider adding the bias to the weighted sum. The weighted sum is being evaluated with the activation function and by this we get the output. The role of this activation function is to categorize the value between 0 and 1 / -1 and 1. In case if the algorithm classifies incorrectly then we update the weights by incrementing or incrementing it and repeat the same until there are no wrong classifications.

Naive Bayes

Naive Bayes is a classification algorithm based on Bayes theorem. All the features are considered to be independent of each other. Every feature has equal weightage which means we cannot exclude any feature when we calculate the probabilities.

Bayes theorem is used to determine the probability of hypothesis given prior experience.

$P(H \mid E) = P(E \mid H) * P(H) / P(E)$

$P(H \mid E)$- Posterior probability - Probability of a given hypothesis to happen after we have witnessed an evidence E
$P(E \mid H)$ - Likelihood probability - Probability of the evidence given that the probability of a hypothesis is true.
$P(H)$- Prior probability - Probability of Hypothesis before the evidence is noticed
$P(E)$- Marginal probability - Probability of Evidence

Since Naive Bayes is a classification algorithm we can consider all the independent features as a feature matrix and the output as class variable. The class variable will be a binary variable taking variables as yes/no or 1/0.

Support vector Classifier:
Support vector classifier is a supervised machine learning algorithm used for analysis or classification of data. It does the classification by drawing a hyperplane. This hyperplane separated the categories of data. After finding the hyperplane tries to maximize the distance from the plane to the support vectors which is the margin.It is extensively used for digit recognition and image classification in real world applications as well. Support vector classifier helps to perform binary and multiple class classification on the dataset.

Naive Bayes:

Face detection

We consider image 'X' as an input and 'y' as a label which says if the image 'X' is a face or not a face. We compute the probability for X being a face and X not being a face and take the highest value as the output.

For computing the probability to check if y is a face we apply the bayes formula as:

$P(y = face \mid X) = P(X \mid y = face) * P(y = face) / P(X)$

$P(y = face \mid X)$ - Posterior probability
$P(X \mid y = face)$ - Likelihood
$P(y = face)$ - Prior probability

$P(X)$ - Marginal probability - We can ignore this component as it doesn't change the prediction of our decision.

Calculating the values -
P(y=face) = Total number of faces in the given dataset with the label y / Total number of images in the dataset.

$$P(y) = c(y) / n$$

$$P(X \mid y = face) = P(feature\_1(x) \mid y = face) * P(feature\_2(x) \mid y = face) * P(feature\_3(x) \mid y = face) * P(feature\_4(x) \mid y = face) * \ldots * P(feature\_m(x) \mid y = face)$$

$$P(y \mid feature\_1, \ldots feature\_m) = \frac{P(feature\_1, \ldots feature\_m \mid y) * P(y)}{P(feature1, \ldots feature\_m)}$$

How do we compute $P(feature\_1(x) \mid y = face)$?
Consider, in our dataset, if our total possible $feature\_1(x)$ values are $0, 1, 2, 3$ then we first find what is our $feature_1(x)$value for the given image and from the value obtained we find its probability from the dataset.

Computing the probability for y is not a face given the same input image.

$$P(y = \sim face \mid X) = P(X \mid y = \sim face) * P(y = \sim face) / P(X)$$

$$P(y = \sim face) = \frac{Total\ number\ of\ faces\ in\ the\ given\ dataset}{Total\ number\ of\ images\ in\ the\ dataset}$$

Multiplying all the probabilities together and applying log function to it gives

$$argmax_y log\ P(y \mid f1, \ldots, fm) = argmax_y\ P(y, f1, \ldots, fm)$$

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

The conditional probabilities of our features given each label y are the additional parameters to be estimated:

P($F_i \mid Y = y$) . This is computed for every possible feature value $f_i \in \{0, 1\}$

$$P(F_i = f_i \mid Y = y) = \frac{c(f_i, y)}{\sum_{i \in \{0,1\}}^{'} c(f_i^{'}, y)}$$

$c(f_i, y)$ = number of times pixel F_i took value f_i for the training examples with label 'y'.

Smoothing:

By using Laplace smoothing, we add constant k to every possible observational value. If the value of k is zero then the probabilities are unsmoothed. As the k value grows we get better probabilities. The value of k can be chosen from the validation dataset.

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

Number classification:

For the number classification, the calculation of likelihood value for the features would be the same. We calculate the bayes formulae for each class in this for each digit starting from 0..9 and based on the higher value we get, we recognise the image.

For example,
The bayes formula to find the probability for the given image to be 1 would be

$$P(y = 1 \mid X) = \frac{P(X \mid y{=}1) * P(y{=}1)}{P(X)}$$

Perceptron:

Face Dataset:

Given an input image X, we identify if the image is face or not a face by doing the following:
f(X) - linear function
If the f(X) > 0 then we categorize it as a face
If the f(X) < 0 then we don't categorize it as a face

We retrieve the feature values for a given image. The result of a feature value from a pixel indicates if we have an edge present or not. Number of features of an image is equal to the number of pixels of an image. Then we also initialize a weight vector with 10 random values. Now we multiply each feature vector value with the corresponding weights and a bias added to this product.

$$f(X) = (w1 * feature\_1) + (w2 * feature\_2) + (w3 * feature\_3) + \ldots\ldots + bias\ value$$

If the output is greater than zero then we consider that the given image is a face and if it is less than zero then we don't consider it as a face.

In case if there is any wrong classification, then the weights of the image will be updated by the following way:

1) When f(x) >=0 and the y is not a face then we decrement the weights as,
   Wi = Wi - feature_1(Xi) and decrement the bias value by -1.
2) When f(x) < 0 and c then we increment the weights as,
   Wi = Wi + feature_1(Xi) and increment the bias value by -1.

In our implementation of the Perceptron Algorithm, we have used 10 iterations to train the model.

Digit dataset:

In the classification for digits, we have multiple classes, starting from $0, 1, 2, 3 \ldots 9$. The image can be anything from $0 \ldots 9$.

We will compute ten different $f(X)$values say from $f(X0), f(X1), f(X2), \ldots f(X9)$

$$f(X0) = b0 + (w1 * feature\_1(Xi)) + (w2 * feature\_2(Xi)) + \ldots + (wn * feature\_n(Xi))$$

Predicting the digit:
We predict the digit based on which function gives the highest score.

In case if there is a wrong prediction then we update weights the following way.

Let us assume that 3 is categorized as 1 then we update the weights of both 1 and 3. We reduce the weights of f(X1) and increase the weights of f(X3).

$$f(X1) = b0 - 1 + (w1 - feature\_1(Xi)) + (w2 - feature\_2(Xi)) + \ldots + (wn - feature\_n(Xi))$$

$$f(X3) = b0 + 1 + (w1 + feature\_1(Xi)) + (w2 + feature\_2(Xi)) + \ldots + (wn + feature\_n(Xi))$$

w_y - weight vector for each class 'y'

For a given feature list 'f' the score is computed as

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

For every instance (f, y) we find the label with the highest score

$$y' = argmaxscore_{y''}(f, y'')$$

Incase if the prediction is wrong, we update the weights in the following way:

y is not a face and we predict it as a face, then we decrement the weight as:

$$w_y = w_y + f$$

y is a face and the prediction is not a face, then we increment the weight as:

$$w_y = w_y - f$$

Results

For each dataset and classifier, we calculate the average prediction error and the standard deviation of the accuracy. We ran the classification problem five times for each classifier and sampled the dataset chosen.

We increased the percentage of the training data used for the actual training in increments of 10%. We started with 10% and increased to 100% resulting in 10 experiments for each combination of dataset and classifier.

All three classifiers attain an accuracy greater than 70% when we train them with all the training data.

Faces Dataset

We observe that SVC starts with the worst accuracy when we train with just 10% of the training data. As we increase the training data size, all three algorithms gradually increase their mean accuracy.

For this dataset the Naive Bayes classifier performs the best consistently.
All three classifiers attain an accuracy greater than 70% when we train them with all the training data. The standard deviation at the end is 0 for 2 algorithms, and significantly small for the Perceptron algorithm

Table 1 depicts our results on the three chosen classifiers with the faces dataset.

Table 1: Results on Faces Dataset(All values in percentages)

	Perceptron		Naive Bayes		SVC	
Training Percentage	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
10%	71.20	3.07	73.73	3.45	57.33	11.98
20%	76.40	6.87	84.13	3.31	69.73	12.18
30%	82.40	2.77	85.73	1.38	78.53	10.22
40%	83.73	1.38	87.07	1.53	80.13	4.41
50%	87.07	0.37	88.00	1.33	88.00	1.56
60%	85.20	2.08	87.07	1.01	87.73	2.03
70%	83.73	2.61	87.87	0.73	88.13	1.59
80%	86.80	2.51	88.67	1.05	88.40	0.60
90%	86.27	1.38	89.87	0.99	90.00	1.25
100%	86.93	1.38	90.67	0.00	89.33	0.00

Figure 1 shows the learning curves for this dataset.

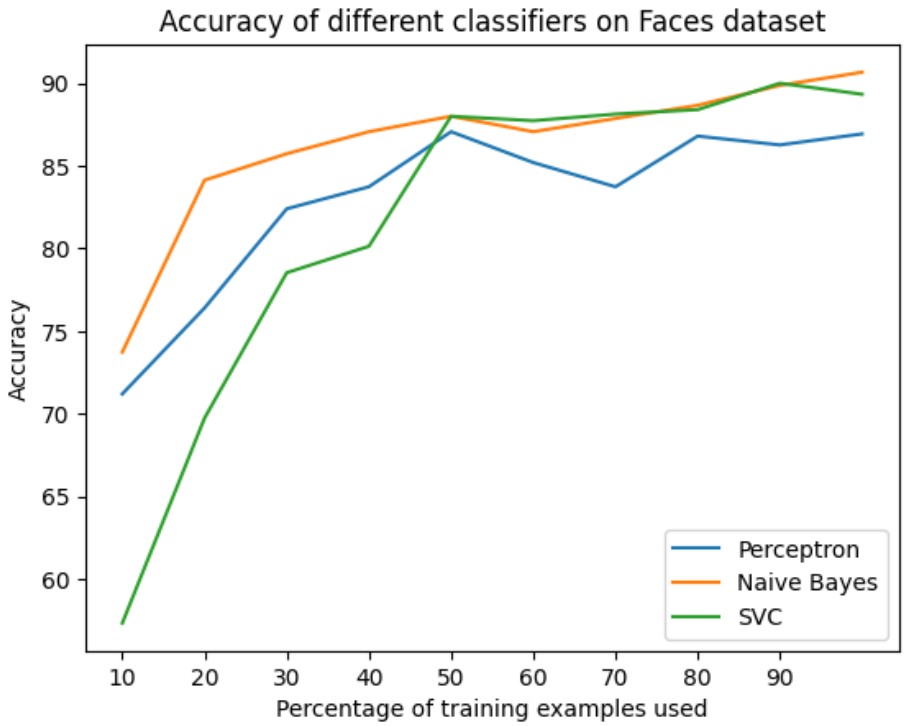

Figure 1: Mean Accuracy on the Faces Dataset

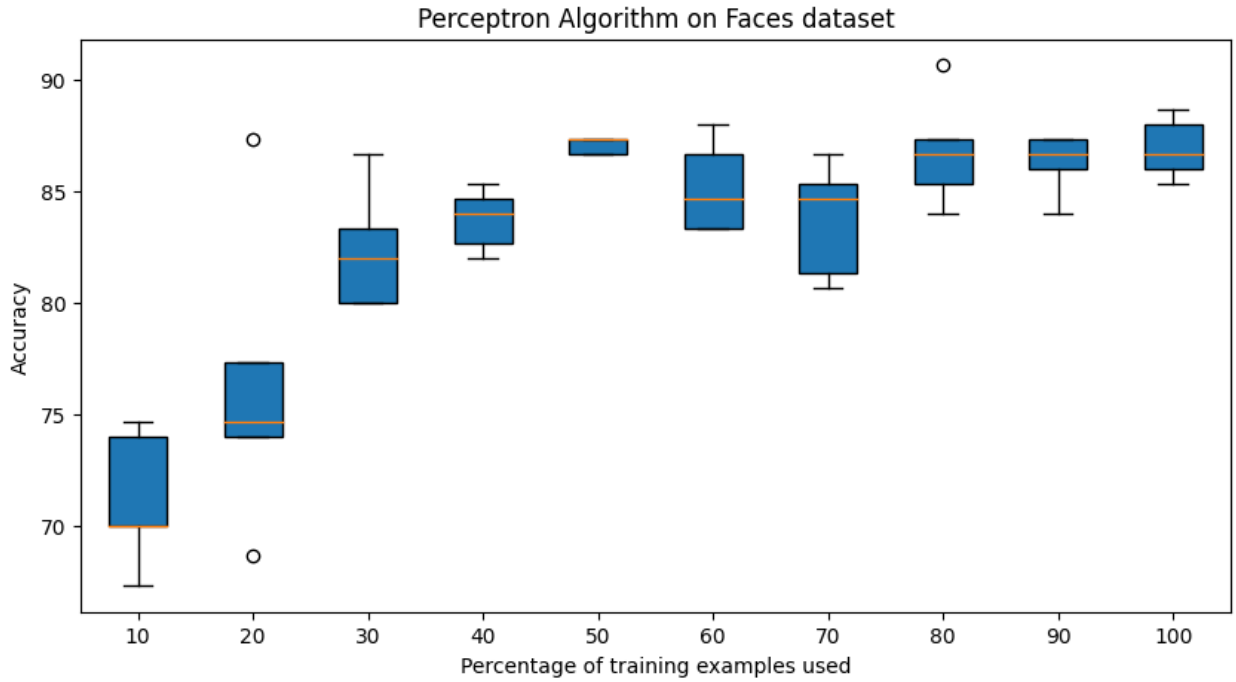

Figure 2: Box plot for Perceptron Algorithm on Faces dataset

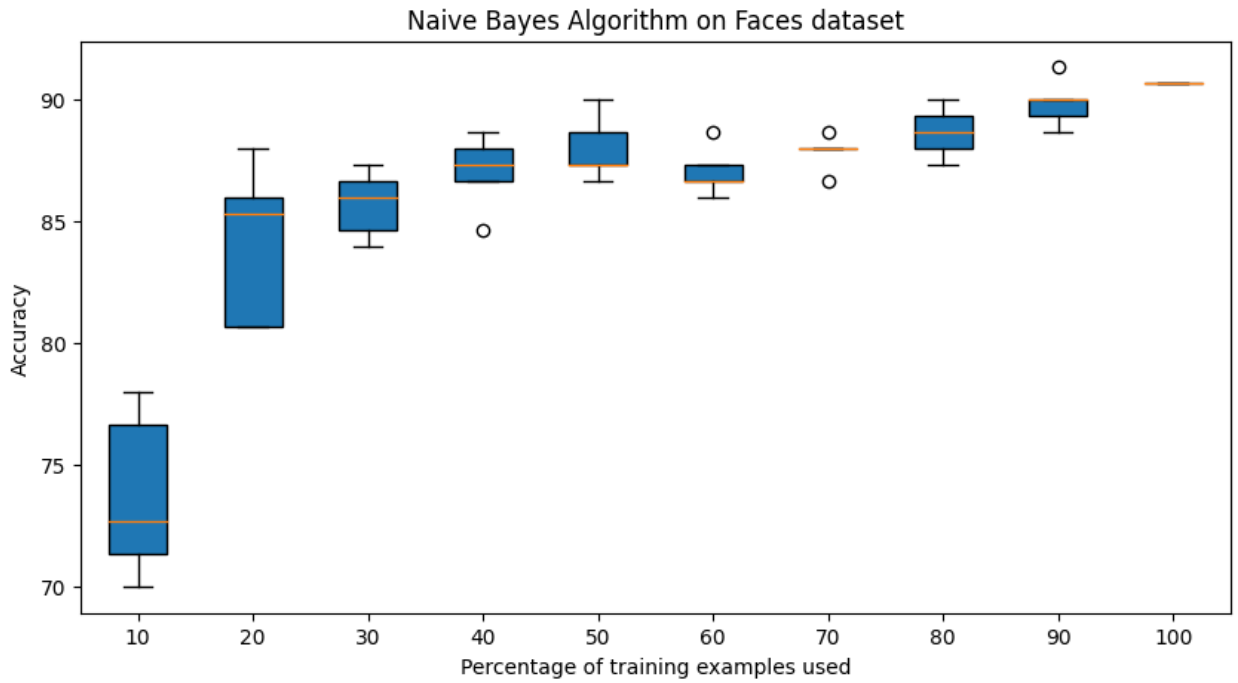

Figure 3: Box plot for Naive Bayes on Faces dataset

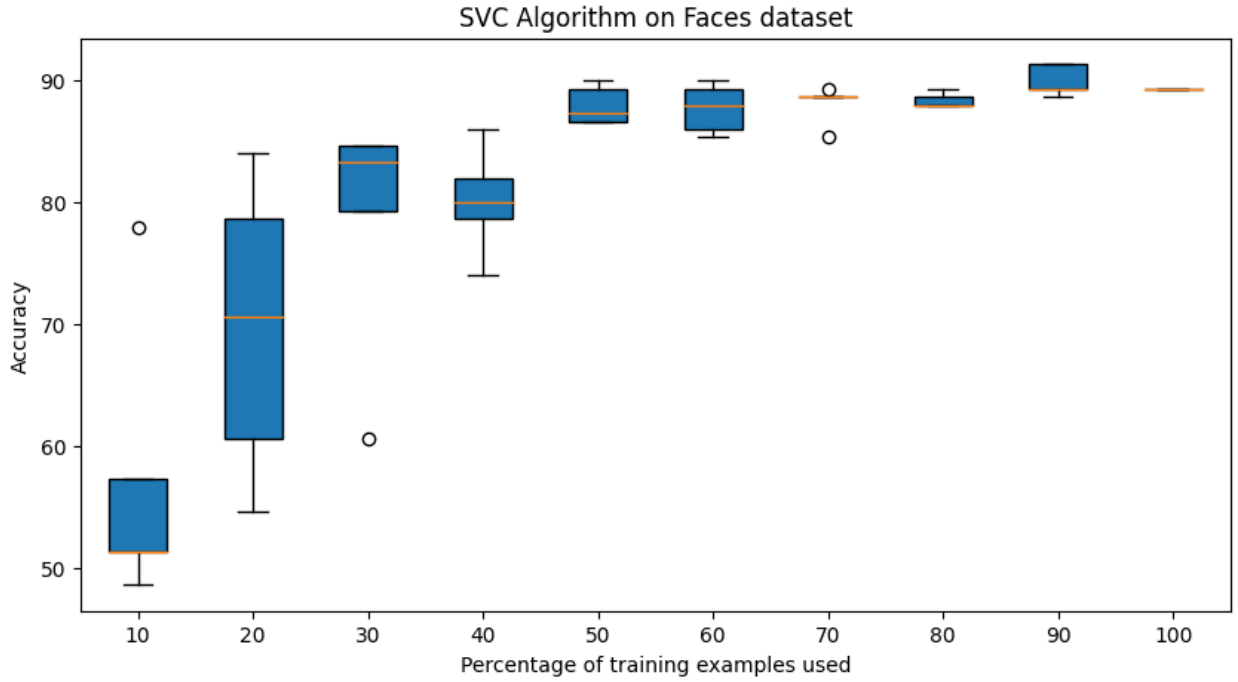

Figure 4: Box plot for SVC on Faces dataset

Figures 2, 3, and 4 show boxplots that indicate their mean, median, and standard deviation for all the three classifiers. We note that the standard deviation decreases as we increase the number of training images. This is the expected behavior of a machine learning model.

Table 2 contains the time taken to train each classifier with the sampled datasets

Table 2: Time for training on Face Dataset

Training Percentage	Perceptron	Naive Bayes	SVC
10%	2.502950907	2.032702923	0.02364015579
20%	4.297045946	2.091068983	0.03506994247
30%	6.345251083	2.285584211	0.04596614838
40%	8.059120893	2.359499931	0.06794071198
50%	10.07936478	2.479486942	0.09413814545
60%	11.89078403	2.683092833	0.1231496334
70%	13.82015419	2.788743019	0.1697878838
80%	15.88848519	2.899127007	0.1990158558
90%	17.40052795	2.784038067	0.2957360744
100%	19.22606897	2.979080915	0.2904200554

Digits Dataset

We observe that SVC starts with the best accuracy when we train with just 10% of the training data. It is consistently better than the other two algorithms.
As we increase the training data size, all three algorithms gradually increase their mean accuracy.

We increased the percentage of the training data used for the actual training in increments of 10%. We started with 10% and increased to 100% resulting in 10 experiments for each combination of dataset and classifier.

All three classifiers attain an accuracy greater than 70% when we train them with all the training data. The standard deviation at the end is 0 for 2 algorithms, and significantly small for the Perceptron algorithm

Table 3 depicts our results on the three chosen classifiers with the digits dataset.

Table 3: Results on Digits Dataset (All values in percentages)

	Perceptron		Naive Bayes		SVC	
Training Percentage	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
10%	75.36	1.25	73.16	2.06	84.60	1.58
20%	75.76	2.73	75.68	0.98	87.96	0.58
30%	77.66	1.82	76.02	1.03	89.70	0.17
40%	77.40	2.39	75.90	0.25	89.96	0.40
50%	78.84	2.09	77.10	0.77	91.28	0.79
60%	79.20	1.07	76.56	0.49	91.70	0.44
70%	80.94	0.97	76.26	0.56	92.06	0.43
80%	79.64	1.17	76.76	0.43	92.24	0.34
90%	77.70	4.06	77.06	0.39	92.56	0.25
100%	79.10	3.01	77.10	0.00	92.70	0.00

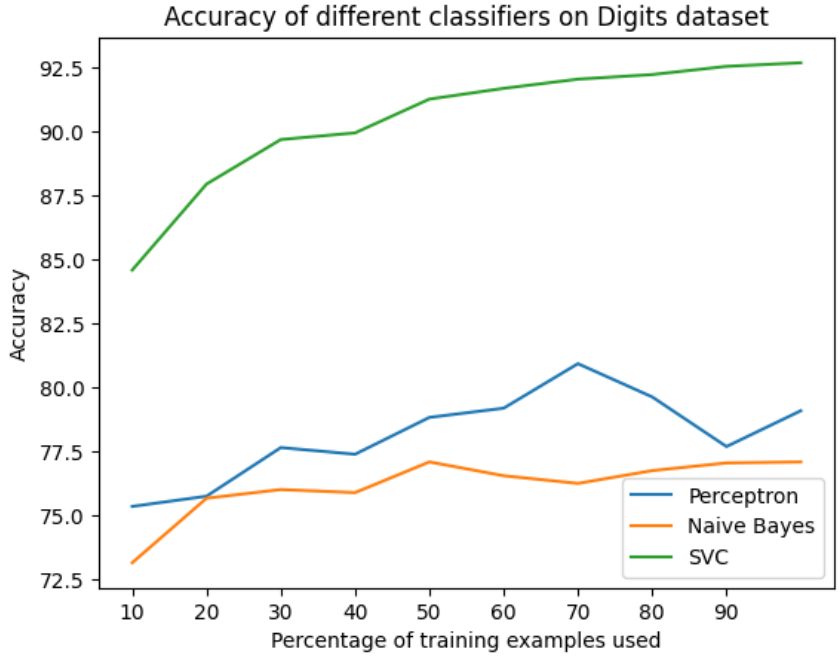

Figure 5: Mean Accuracy for the digits dataset

Figures 6, 7, and 8 represent the box plots with the mean, median and the standard deviations of the 3 algorithms on the digits datasets.

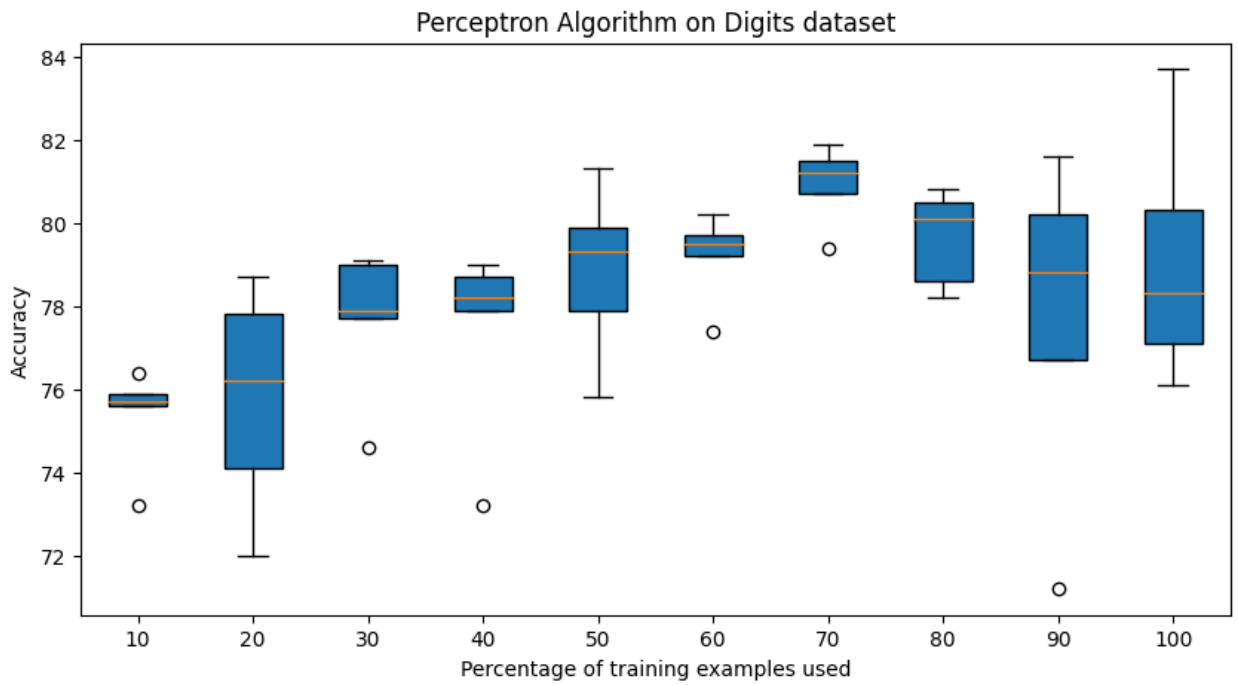

Figure 6: Box plot for the Perceptron Algorithm

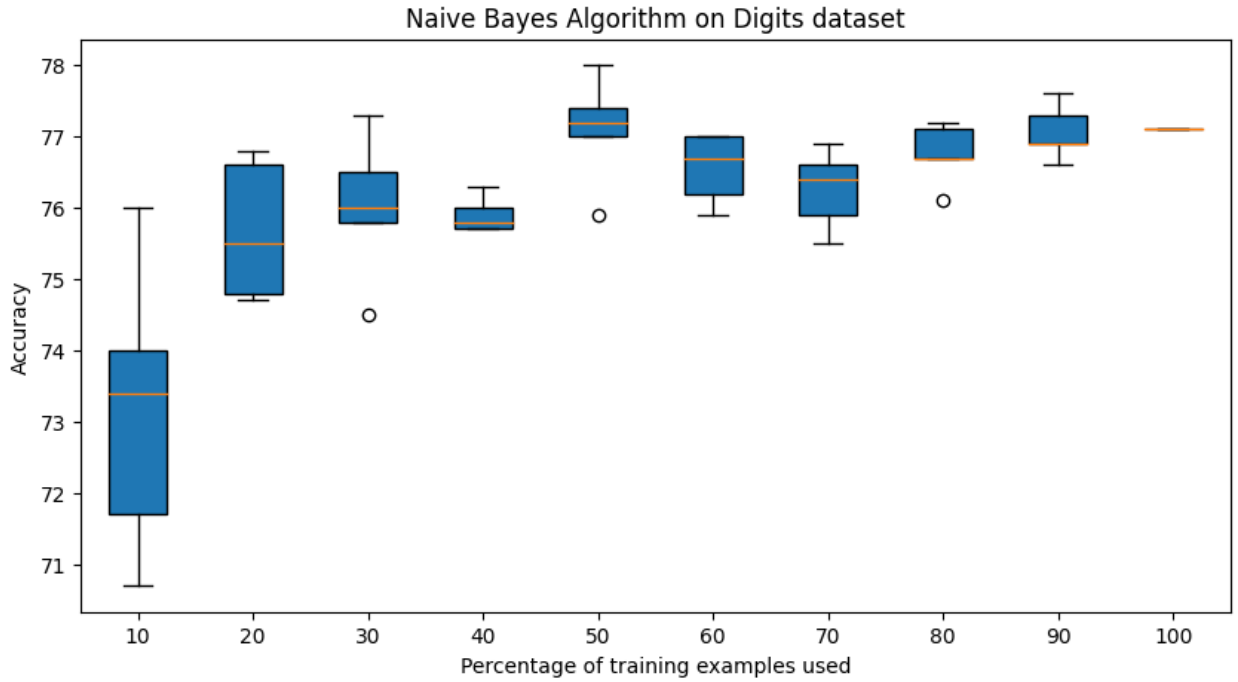

Figure 7: Box plot for the Naive Bayes Algorithm

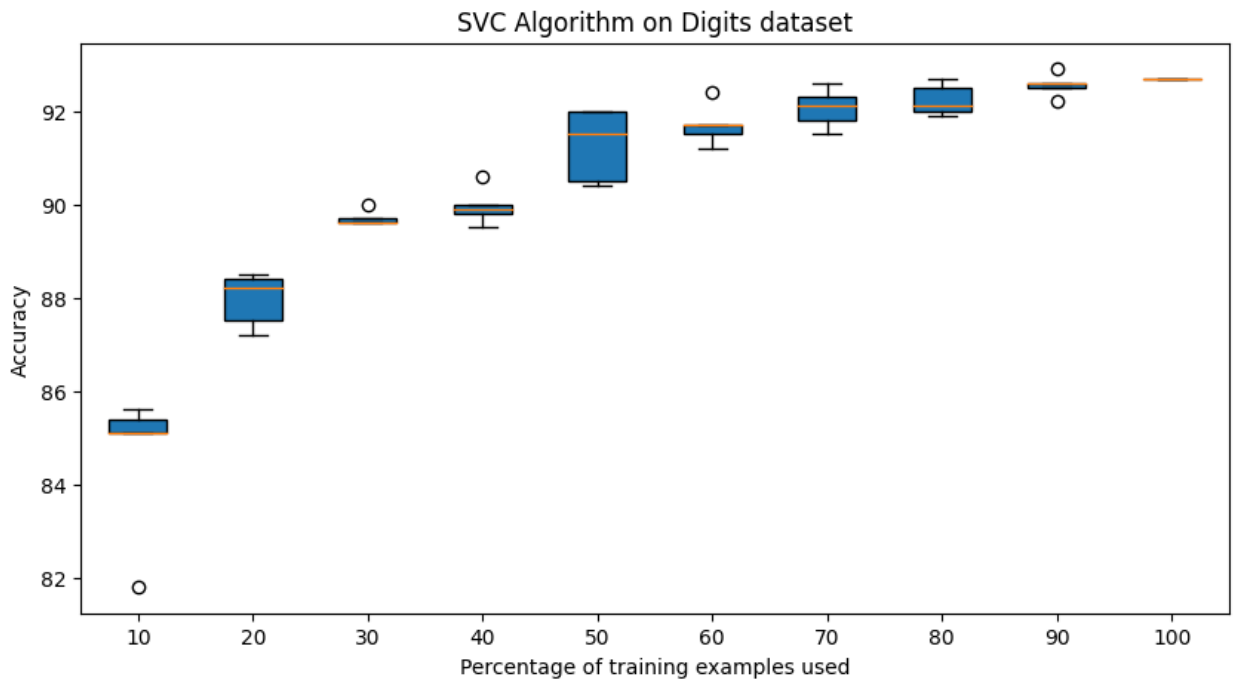

Figure 8: Box plot for the SVC

Again, we observe that the standard deviation converges to 0 as we increase the training data size.

Time taken by three algorithms to train is given in Table 4.

Table 4: Time for training on Digits Dataset

Training Percentage	Perceptron	Naive Bayes	SVC
10%	21.74240208	11.07502627	0.1904349327
20%	40.15223789	11.43460822	0.3997988701
30%	59.14528298	12.15867591	0.6218948364
40%	77.69736195	12.1724782	0.8223211765
50%	95.50636601	12.27854109	1.043995142
60%	113.56917	12.42151809	1.287755966
70%	133.015347	12.6150682	1.549170256
80%	154.1270001	13.51919103	1.856565952
90%	172.4727399	13.06520867	2.138153076
100%	191.6064909	13.57697105	2.635164976

Results using the **Square Feature Extractor**

2*2 dimensions for the digit classification

	Perceptron		Naive Bayes		SVC	
Training Percentage	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
10	64.27	13.65	80.13	2.08	62.13	6.08
20	65.87	15.20	84.80	3.18	84.67	4.55
30	64.67	9.25	83.73	2.09	84.53	3.78
40	84.00	6.53	86.67	2.26	87.07	1.46
50	76.80	13.12	85.47	1.73	88.27	1.80
60	76.00	6.24	86.93	1.30	87.33	2.83
70	81.60	9.61	87.20	0.30	87.47	0.99
80	80.53	8.50	86.13	0.99	87.33	1.33
90	81.20	10.83	86.80	1.66	88.27	1.01
100	76.53	13.31	87.33	0.00	88.00	0.00

2*2 dimensions for the face classification

	Perceptron		Naive Bayes		SVC	
Training Percentage	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
10	71.87	3.03	82.67	3.68	61.60	14.35
20	74.67	12.37	85.33	2.49	77.07	9.75
30	76.00	9.19	87.20	3.81	86.13	2.47
40	78.93	5.16	86.93	2.39	89.33	2.45
50	80.27	2.39	88.40	1.74	89.87	1.91
60	85.33	1.49	88.53	1.10	90.13	1.19
70	82.67	3.30	88.00	0.47	89.87	0.87
80	80.80	5.15	89.47	0.73	90.80	0.87
90	81.73	7.45	89.73	0.89	90.13	1.10
100	82.80	3.44	89.33	0.00	90.67	0.00

4*4 dimensions for the digit classification

	Perceptron		Naive Bayes		SVC	
Training Percentage	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
10	44.02	4.98	61.56	1.20	66.28	1.97
20	52.92	3.62	63.02	1.54	69.74	1.22
30	55.74	5.48	62.48	0.69	72.16	0.43
40	58.24	2.86	63.20	0.58	72.96	0.65
50	58.30	3.54	63.38	0.62	73.10	0.47
60	58.88	3.55	63.08	0.87	73.86	0.45
70	58.48	3.11	62.94	0.36	74.18	0.88
80	57.96	4.21	63.30	0.86	74.50	0.33
90	59.74	3.34	63.18	0.29	74.58	0.24
100	60.6	2.99165506	63.1	0	75.3	0

5*5 dimensions for the face classification

	Perceptron		Naive Bayes		SVC	
Training Percentage	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
10	64.27	13.65	80.13	2.08	64.27	13.65
20	65.87	15.20	84.80	3.18	65.87	15.20
30	64.67	9.25	83.73	2.09	64.67	9.25
40	84.00	6.53	86.67	2.26	84.00	6.53
50	76.80	13.12	85.47	1.73	76.80	13.12
60	76.00	6.24	86.93	1.30	76.00	6.24
70	81.60	9.61	87.20	0.30	81.60	9.61
80	80.53	8.50	86.13	0.99	80.53	8.50
90	81.20	10.83	86.80	1.66	81.20	10.83
100	76.53	13.31	87.33	0.00	76.53	13.31

Conclusions

For Classification of Images of Faces:

- On an average The Naive Bayes Classifier performs better than the Perceptron Classifier for the Classification of Faces. The Naive Bayes Classifier and the SVC Classifier perform similarly for the classification of faces when we use more than 50% of the training data. However, the Naive Bayes Classifier performs slightly better than SVC when 100% training data is used. The SVC Classifier does very poorly when compared to the other two when we use less than 40% of the training data.
- The mean accuracy values of the three Classifiers keep on increasing as we increase the percentage of training data. The standard deviations of the three algorithms keep on decreasing and almost reach to the value of zero, as we increase the training data. The standard deviation is quite high for SVC when compared to the other two classifiers, when the training data used is less.
- SVC takes the least amount of time out of all the three classifiers to train, less than 1 second. Naive Bayes Classifier takes around 2-3 seconds to train. The perceptron algorithm takes the highest amount of time to get trained out of all three algorithms. Perceptron takes 2.5 seconds to train on 10% of the training data and 19.2 seconds to get trained on 100% of the trained data. Increasing the training data points increases the training time for all the three algorithms.

For Classification of Images of Digits:

- SVC outperformed both the Naive Bayes Classifier and the Perceptron Classifier for the classification of digits. SVC consistently performed better than the other two over all the number of data points used for training. SVC reaches an accuracy of 92% when 100% of the training data is used. Naive Bayes and Perceptron perform similar to each other, however we cannot conclude that the perceptron algorithm does better than the Naive Bayes Classifier based on their mean accuracy values, because the Standard deviation of the perceptron algorithm is quite high and does not decrease as we increase the training data.
- Similar to the classification of faces, the mean accuracy values of the three Classifiers keep on increasing as we increase the percentage of training data.The The standard

deviations of both the Naive Bayes and the SVC algorithms keep on decreasing and reach zero value, as we increase the training data. However, the Standard deviation of the perceptron does not decrease and remains significantly higher than the other two classifiers.

- SVC takes the least amount of time out of all the three classifiers to train, whereas the perceptron algorithm takes the highest amount of time to get trained out of all three algorithms. Increasing the training data points increases the training time for all the three algorithms.

For both digits and Faces, with the increase in the number of Iterations for the perceptron algorithm, the accuracy of the model increases.

When the New Square Feature Extractor is used:

- When we use the new Feature Extractor with '2*2' dimensions (that is using 4 pixels for one feature) for both the data sets, then the accuracies of all the 3 models are lower than the accuracies of the models when we use the basic feature extractor. However, the training time is significantly reduced.
- The accuracy of models is further reduced if we increase the size of dimensions to '4*4' for digits and '5*5' for faces. However, the training time reduces significantly as we have way lesser number of features for each image now.
- The mean accuracies of Naive Bayes and the SVC models keep on increasing in general as we increase the training data. However, for the case of Perceptron, it does not follow a strictly increasing trend for accuracy as we keep on increasing the training data.