RUTGERS

# Final Project

## CS520

# Face and Digit Classification

---

**Presenters :**

**Jaini Patel jp1891 | Harsh Patel hkp49**

# Introduction:

## Abstract:

In Face and Digit Classification project, we have implemented three classification algorithms on two different data sets such that all of them generate a model to train the data from the known data with least cost function possible and then predict the output for unknown data set. The algorithms we have used here are Bayesian Network Classifier, Perceptron and MIRA(an updated and improved version of Perceptron).

We were two data set; one which had images of hand written digits and their corresponding labels and the other which had images of face and other random things. Our task for the first data set was to design a model which would predict the digit label (from 0 to 9) based on the image. For the other data set, we had to implement an algorithm which would accurately predict if the image was a face image or not (label 1 for face and 0 for non-face data).

During this process of implementing different algorithms, we learned the pros and the cons of these algorithms and many others which are used in the industry for classifying images and texts. We also studied the effect of change in size of training data on the cost function and the predicted accuracy. Here, we present the report on the working of the algorithms and our observations.

# Preprocessing:

The data was preprocessed before using. Four text files for training and testing data were given to us. Two of them had the images to be used for training the model and testing the model and the rest had the corresponding training and testing data labels.

The image files had images in the form of black and grey pixels, which we have considered as a single value(black pixel). First, we have transformed the images in each file into matrices of respective size(28x28 matrix for digit data and 70x60 matrix for face data), and then stored these matrices into list, making the list of images in the data file a list of matrices. The image matrix stores 1 for black and grey pixels (+ and # values in the original text file) and 0 for white spaces.

The selection of feature set affects the accuracy results and so, to get higher accuracy, we have used different feature sets depending on the type of the data and the algorithm used. These features are later used to train the model and similarly, the testing data is converted into feature set according to which the model is trained.

## Visualization:

In order to compare the accuracy of algorithms with respect to the varying training size set, we have plotted the accuracy graphs for each algorithm using matplotlib library. In case of Perceptron and MIRA algorithm, we have also plotted the accuracy after each epoch.

# Algorithms:

Here is a brief explanation about the algorithms that we have used.

## 1) Naive Bayes Classifier:

The Naive Bayes classifier is a simple classifier that is often used as a baseline for comparison with more complex classifiers. It is conceptually simple and works on the Bayes Rule.

According to Bayes Rule,

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

where, $x$ represents the image, or more precisely, the pixel values of the image formatted as a vector, and $c$ represents the digit or labels, which can be 0, 1, . . . , 9.

The main idea here is to find the probability of a label given some set of features of any image. For this, the image is divided into set of features and then these sets of features are used to calculate the probabilities. The largest probability in the list of calculated probabilities is considered to be the probability of predicted label value. The predicted label will be a digit in the case of digit recognition and face or not a face label in case of face classification system.

In the above equation of Bayes Rule, the left side $P(c|x)$ is read as "the probability that the class is $c$ given the data $x$". (this is called the "posterior") And, the right side $P(x|c)$ as "the probability that the data $x$ belongs to the class $c$". (this is called the "likelihood") The term $P(c)$ is read "the probability that a particular label $c$ occurs given the data $x$". (this is called the "prior") Here, we actually look for the value of c rather than $P(c|x)$ itself as $c$ tells us "which digit" the image belongs to. ($c$ is the label value) The class chosen is simply the one that yields the highest probability for that data:

$$c^* = argmax_c P(c|x) = argmax_c \frac{P(x|c)P(c)}{P(x)}$$

Here we can see that $P(x)$ will be constant for every computation of $c$, and hence it can be ignored while we take argmax of $\frac{P(x|c)P(c)}{P(x)}$.

## 2) Perceptron Classifier:

A Perceptron is a neural network unit (an artificial neuron) that does certain computations to detect features or business intelligence in the input data. A Perceptron is an algorithm for supervised learning of binary classifiers. This algorithm enables neurons to learn and processes elements in the training set one at a time. It is used when classes which we want to classify are linearly separable. When classes are not linearly separable (for example: XOR), we should use Neural Network.

In the following image, architecture of perceptron network with single layer of perceptrons is illustrated. As shown, input of network is features of a particular image. Each feature has its own weight. Hence, we have $n$ weights for $n$ features. The output of network will be summation of multiplication of each feature with its corresponding weight.

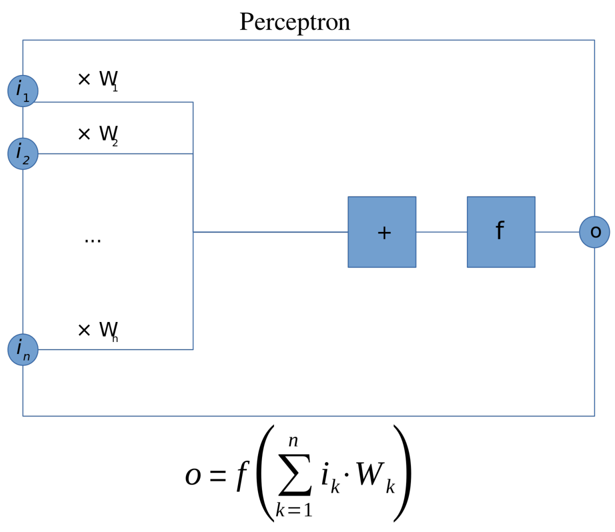

Figure 1: Perceptron

In other words,

$$Y = \sum_{i=1}^{n} w_i \cdot f_i$$

As we stated earlier perceptron is used when classes are linearly separable, we can predict label based on the value of $Y$.

$$Y = \begin{cases} 1, & if \sum_{i=1}^{n} w_i \cdot f_i > 0 \\ 0, & \text{otherwise} \end{cases}$$

Now, if the algorithm predicts the correct label, we won't make any changes to the weights associated with the features but if it identifies incorrect label, then we will have to make appropriate changes to the weights. The weights will be updated as follows.

Let's consider $Y$ as the predicted label and $Y^*$ as the actual label;

If $Y < 0$ and $Y^* = True$:

$$W(Y) = W(Y) + f(j),\ where\ j \in 1, ..., n$$

If $Y > 0$ and $Y^* = False$:

$$W(Y) = W(Y) - f(j),\ where\ j \in 1, ..., n$$

**Weight updates in case of multiclass perceptron:**

In digit classification we have 10 classes *i.e.* from digits 0 to 9 so we will have 10 perceptrons. We do not make changes in weights if algorithm predicts the label correctly. We only update the weights if the algorithm makes any mistake.

If actual label is $Y^*$ and we have predicted $Y$ then:

We increase weight of actual label:

$$W(Y^*) = W(Y^*) + f(j),\ where\ j \in 1, ..., n$$

We decrease weight of wrongly identified label:

$$W(Y) = W(Y) - f(j),\ where\ j \in 1, ..., n$$

The weights are randomly initialize. When we update the weights, it slowly gets converged where it will make very minimal training error. Irrespective of initial weights, it gets converged. However, if initialization is not done properly, then it may get longer time to get converged.

## 3) MIRA Classifier:

The Perceptron suffers from the below issues:

- Due to the presence of some noise or anomalous data, the weights might thrash.
- Sometimes the Perceptron model finds barely separable solutions.
- Overtraining of training data may result in test accuracy falling down.

The main aim of Margin Infused Relaxed Algorithm(MIRA) is to mitigate the above issues by choosing an update size and minimizing the updates on the weights by introducing a variable named $\tau$.

$\tau$ is defined as below:

$$\tau = \frac{(w'_y - w'_{y^*}) \cdot f + 1}{2(f \cdot f)}$$

where, $w'_y$ = Weights of Predicted Label

$w'_{y^*}$ = Weights of True Label

f = Features

Unlike perceptron, where the weights of the predicted label are updated by simply subtracting themselves with the features, in MIRA they are updated as below:

$$w_y = w'_y - \tau \cdot f(x)$$

$$w_{y^*} = w'_{y^*} + \tau \cdot f(x)$$

Here, $w_y$ = Updated weights of predicted labels

$w_{y^*}$ = Updated weights of true labels

## Observations:

We selected different percentages of training data for each data sets and algorithms, ranging from 10% to 99% testing data (at interval of 10) and plotted different accuracy and error rates with respect to the size of training data selected. This study shows the importance of training data on the accuracy and the time taken for execution. We came to conclusion that if you decrease the size of training data, accuracy of the model gets decreased. However, if model tries to overfit then increasing the training data can help improve the accuracy of the testing data.

### 1) Naive Bayes Classifier:

The below plots show how the accuracy of Naive Bayes Classifier Algorithm change with respect to the size of training data.

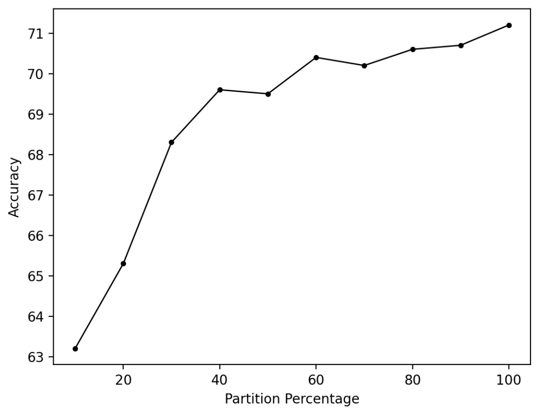

Figure 2: Accuracy of Digit Classification Model

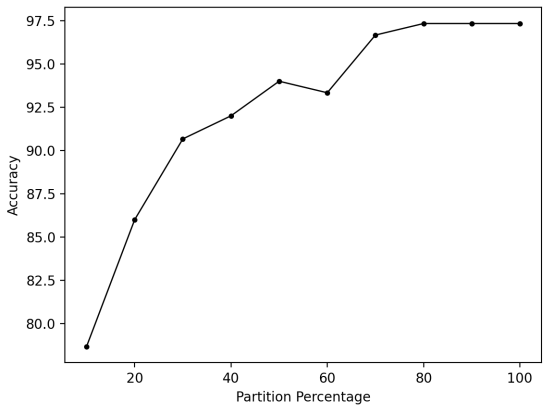

Figure 3: Accuracy of Face Classification Model

Here, the partition percentage in the images shows the percentage of training data used. We can clearly see that the accuracy is a function of training data set. The accuracy is the lowest when the model uses only 10% of the training data for training, this is because very less amount of data is provided to the model to train. And hence, the many features are missed in such small data set leading to failing of the working of the algorithm.

## 2) Perceptron Classifier:

The below plots show how the accuracy of Perceptron Classifier Algorithm change with respect to the size of training data.

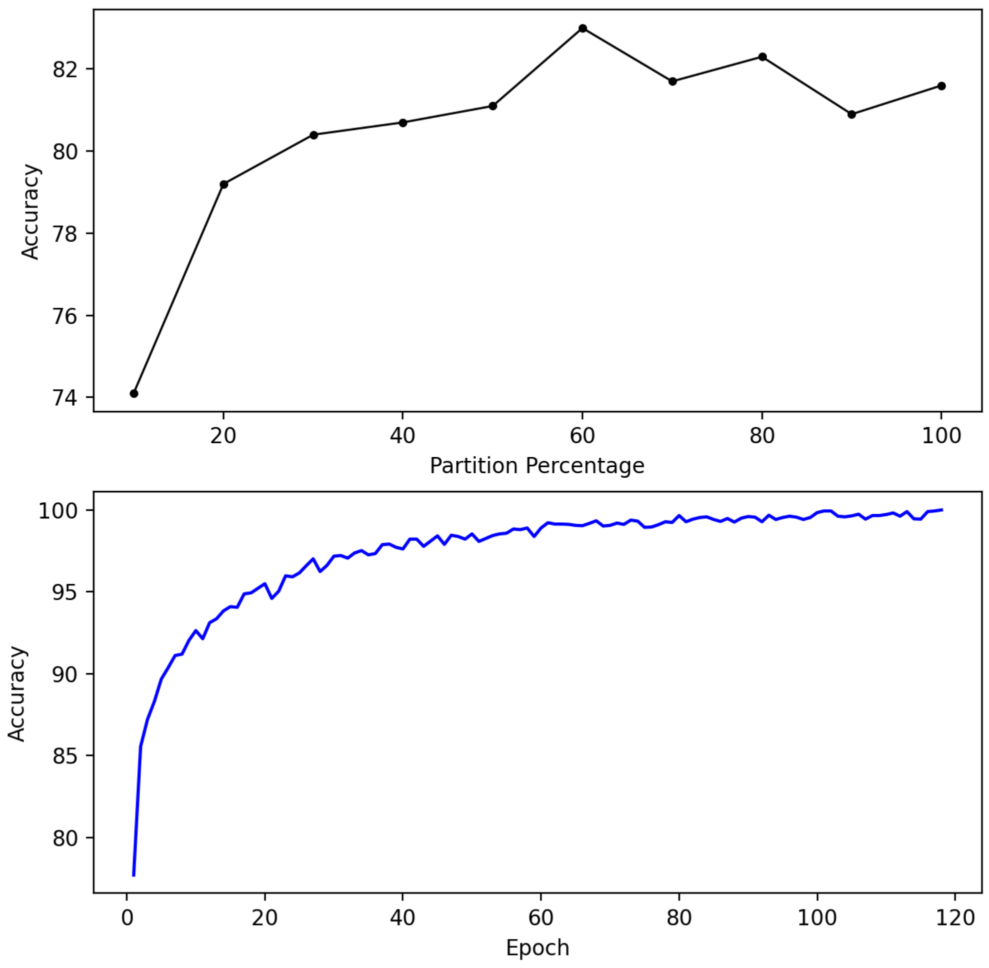

Figure 4: Accuracy of Digit Classification Model

Here, there are two graphs for Perceptron digit classification algorithm; one shows the accuracy vs partition percentage and the other shows the accuracy vs epoch graph for last training iteration.

One epoch is when an entire dataset is passed through the network only once. In Bayesian Network we only iterate training data once and we calculate our prior probability and likelihood probability.So, multiple epochs are not required. However, in Perceptron Neural Network and MIRA, we do need multiple epochs in order to learn weights of each feature.

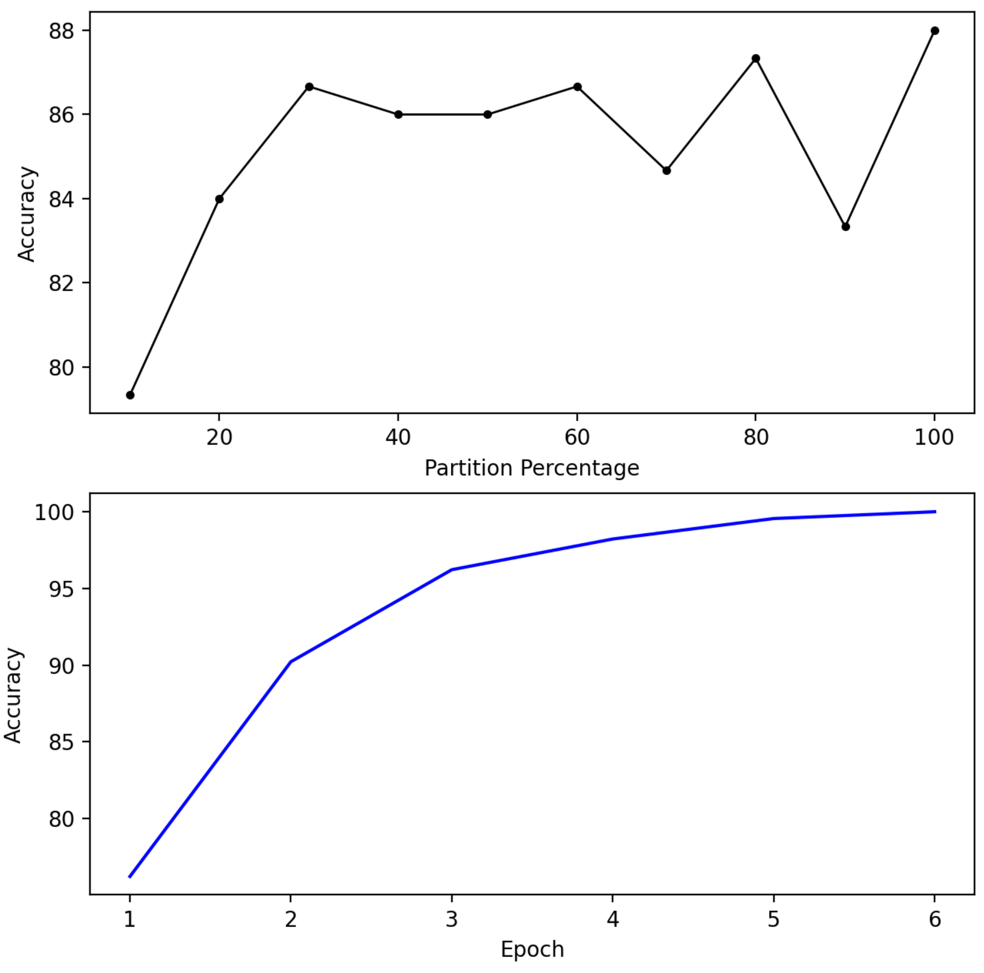

Figure 5: Accuracy of Face Classification Model

Here, there are two graphs for Perceptron face classification algorithm; one shows the accuracy vs partition percentage and the other shows the accuracy vs epoch graph for last training iteration.

We can see from accuracy vs partition percentage graphs for each data set that the accuracy does not follow an increasing trend with the increase in training data size. For training data size, there is a decrease in the accuracy rate compared to the previous training data size. Argument for this is that the model tries to overfit the training data, hence we get minimal error on training data but more error on testing data for those training data set.

## 3) MIRA Classifier:

As we know MIRA works similarly as Perceptron, the only difference is how we update weights in both cases. Hence, we get results almost similar to that of perceptron in this case. The below plots show how the accuracy of MIRA Classifier change with respect to the size of training data.

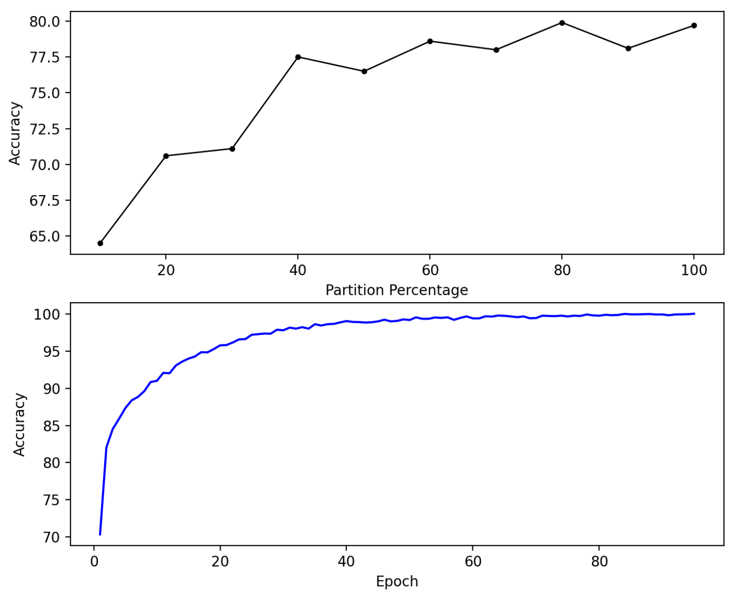

Figure 6: Accuracy of Digit Classification Model

Here, there are two graphs for MIRA digit classification algorithm; one shows the accuracy vs partition percentage and the other shows the accuracy vs epoch graph for last training iteration.

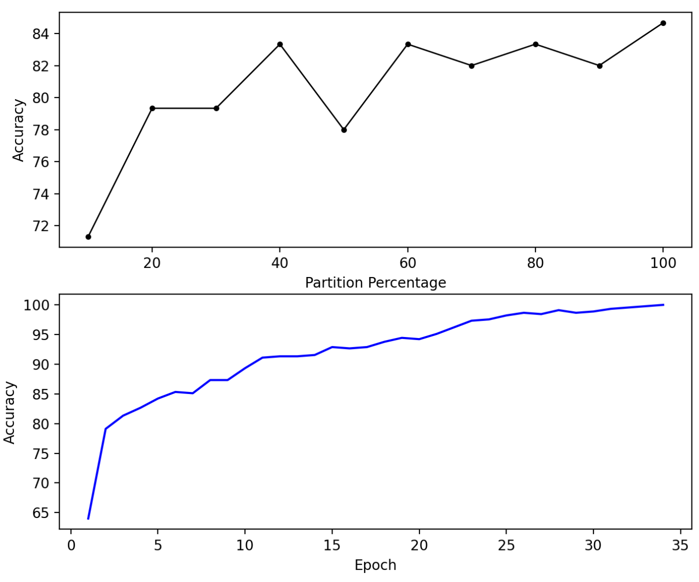

Figure 7: Accuracy of Face Classification Model

Here, there are two graphs for MIRA face classification algorithm; one shows the accuracy vs partition percentage and the other shows the accuracy vs epoch graph for last training iteration.

The epoch curves in all the cases for Perceptron and MIRA classifier show a similar trend. The epoch curve for Perceptron face classifier algorithm and MIRA face classification algorithm slowly increase in the beginning, but later remain almost flat till accuracy becomes 100%. Similar for digit classification, there is a sudden increase in the curve after certain epoch and then the curve remains almost flat till accuracy becomes 100%.

## Conclusion:

| Accuracy Table | | |
|---|---|---|
| Algorithm Name | Digit Data | Face Data |
| Naive Bayes Classifier | 71.9% | 97.33% |
| Perceptron Classifier | 81.56% | 88% |
| MIRA Classifier | 81.23% | 84.33% |

| Total Time Taken(in sec) Table | | |
|---|---|---|
| Algorithm Name | Digit Data | Face Data |
| Naive Bayes Classifier | 14.33 | 21.91 |
| Perceptron Classifier | 710.54 | 31.62 |
| MIRA Classifier | 566.87 | 19.28 |

Above mentioned are the result tables showing accuracy for each algorithm and total time taken to train and test the data for each algorithm, for each dataset is measured when we fed 100% of training data. We tried different feature size for both data and have we got the best results with following

- Naive Bayes Classifier:
  - Digit Data : 28 rows as features, i.e., feature size is 28.
  - Face Data : 5x5 (sliding window)
- Perceptron Classifier:
  - Digit Data : 1x1 (for each pixel)
  - Face Data : 1x1 (for each pixel)
- MIRA Classifier:
  - Digit Data : 1x1 (for each pixel)
  - Face Data : 5x5 (sliding window)

In Bayesian Network the final accuracy does not change. However, for Perceptron and MIRA it changes every time you run the algorithm because the weights used in the program are initialized randomly. This random assignment makes it subject to change even if the training data used is same.

From the above experiments and results, we can observe how the portion of training data decides the learning of the model and how it effects the accuracy on the testing and unknown data.

In Perceptron and MIRA algorithms, we can see the effect of number of neurons and the random initialization of weights on the accuracy of algorithms. By simply initializing all the weights as zero will always lead to the same results and the model may get stuck in the local minima of the cost function plot. However, by randomly initializing weights we may find other local minimas which may be actually better or if we are lucky, we may get to the global minimum. The other important observation we can see is how the change in number of features changes the working of the model. By simply changing few features used, the model can show vast differences in the training data and testing data accuracy.

## References:

- https://lazyprogrammer.me/bayes-classifier-and-naive-bayes-tutorial-using/
- https://en.wikipedia.org/wiki/Perceptron
- http://www.cs.rpi.edu/∼xial/Teaching/2018S/slides/IntroAI_21.pdf