Face and Digit Classification

submitted in fulfillment of the requirements

for

FALL 22 Project

in

CS520: INTRODUCTION TO ARTIFICIAL INTELLIGENCE

by

KOWNDINYA BOYALAKUNTLA	**219002814**
ANIRUDHA SARMA TUMULURI	**219000093**
SATYAM SAINI	**219005500**

Supervisor(s)

Dr. Abdeslam Boularias

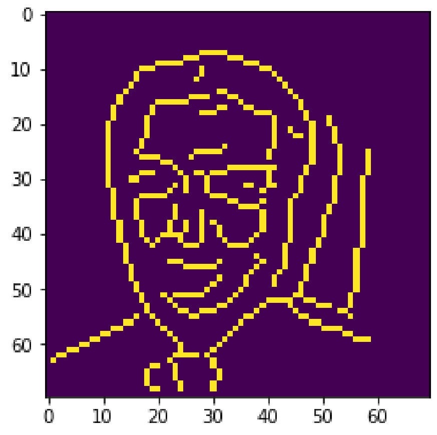
DEPARTMENT OF COMPUTER SCIENCE

RUTGERS UNIVERSITY - NEW BRUNSWICK

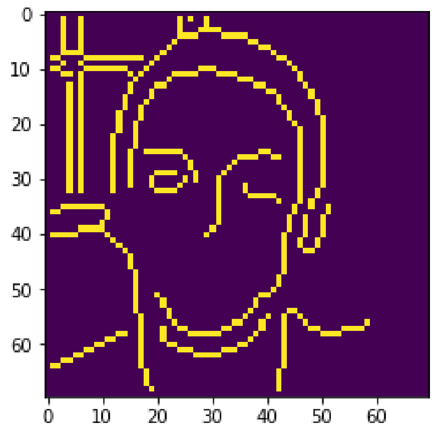
December 8, 2022

INTRODUCTION

In this project, we trained three classifiers for handwritten MNIST digits and face classification analysis. In the dataset, a face is 70 pixels by 70 pixels (refer to Figure 1), and a digit is 28 pixels by 28 pixels (refer to Figure 2).

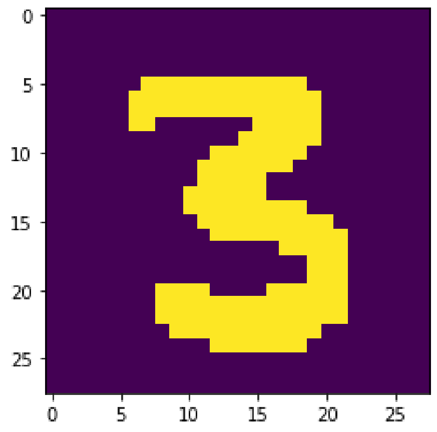


(a) MNIST FACE

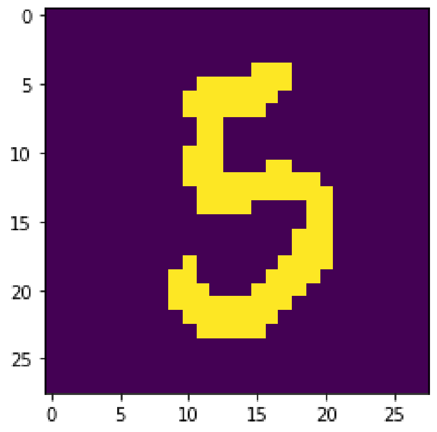


(b) MNIST FACE

Figure 1: MNIST FACES in the dataset



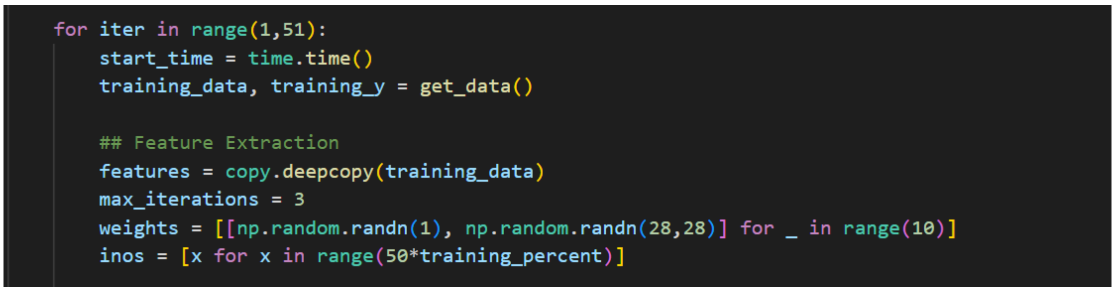
(a) MNIST DIGIT 3



(b) MNIST DIGIT 5

Figure 2: MNIST DIGITS in the dataset

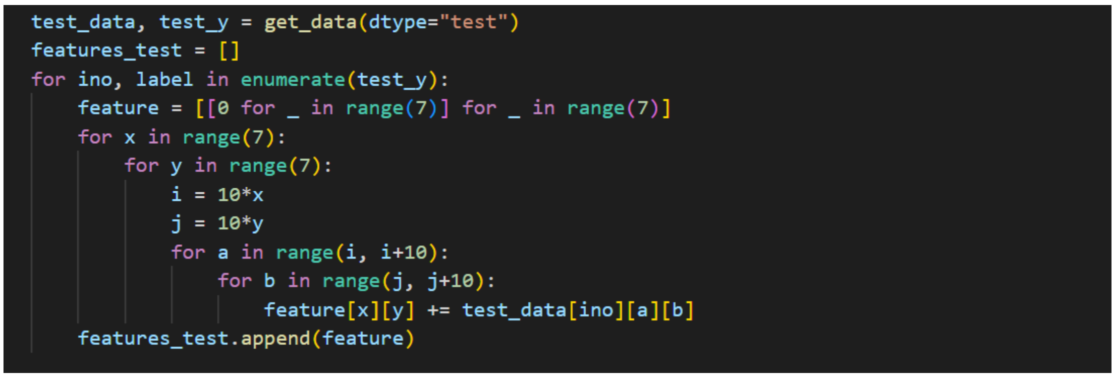
For the Perceptron classifier, we are considering each pixel as a feature (refer to Figure 3). For the Naive Bayes classifier, we consider a 10x10 grid as a feature, totaling 49 features for the face (refer to Figure 4). Perceptron and Naive Bayes classifiers were built from scratch. Concerning the third classification algorithm, we use the off-the-shelf Random Forest classifier from the *sklearn*[1] python library. Classifier training is performed on ten increasing chunks of training data (from 10% to 100%). During training, the input data is randomized, and the simulation is carried out 50 times. Correspondingly, the average prediction error and the standard deviation were reported.



```
    for iter in range(1,51):
        start_time = time.time()
        training_data, training_y = get_data()

        ## Feature Extraction
        features = copy.deepcopy(training_data)
        max_iterations = 3
        weights = [[np.random.randn(1), np.random.randn(28,28)] for _ in range(10)]
        inos = [x for x in range(50*training_percent)]
```

Figure 3: Feature Extraction for Perceptron



```
test_data, test_y = get_data(dtype="test")
features_test = []
for ino, label in enumerate(test_y):
    feature = [[0 for _ in range(7)] for _ in range(7)]
    for x in range(7):
        for y in range(7):
            i = 10*x
            j = 10*y
            for a in range(i, i+10):
                for b in range(j, j+10):
                    feature[x][y] += test_data[ino][a][b]
    features_test.append(feature)
```

Figure 4: Feature Extraction for Naive Bayes

From an overall perspective, for digits classification, the Perceptron classifier reported accuracies between 67.38% to 81.4%, the Naive Bayes classifier between 62.14% to 75.36%, and the Random Forest classifier between 82.82% to 90.89% for training sizes ranging between 10% to 100%. Similarly, for face classification, the Perceptron classifier reported accuracies between 70.32% to 86.67%, the Naive Bayes classifier between 60.69% to 88%, and the Random Forest classifier between 82.7% to 91.1% for training sizes ranging between 10% to 100%.

[1]https://scikit-learn.org/stable/modules/ensemble.html#random-forests

IDEA OF OUR ALGORITHM

Perceptron

Concerning the objective function

$$f(X) = sign(W.X)$$

, where X is the input data vector, and W is the weight vector for the perceptron classifier. The objective function is fairly differentiable. However, the derivative is zero everywhere, and the derivative is largely uninformative to apply a gradient descent algorithm to reduce the number of misclassifications (or loss function). Hence, to that end, we used the idea of PLA (Perceptron Learning Algorithm). We started with a random guess for W_0 at time t_0. At time t_k, for the input data point i, $W_{k+1} = W_k - (f(X^{\mathrm{i}}) - Y_i)X^{\mathrm{i}}$.

Naive Bayes

The main working part of the Naive Bayes Classifier is the calculation of various probabilities of an instance belonging to certain classes based on various features of the instance.

In our case, for the MNIST images, we use each pixel as an individual feature and for the Face classifier due to the larger size of the image, we divide the image into 49 10x10 pixel chunks and calculate the number of pixels with a value 1 and consider that as a feature.

We first calculate the Prior Probabilities of each class which is the probability of an instance of a class occurring. Then we use the features mentioned above to calculate the Likelihood Probabilities for each of the features and the class.

Whenever there is an image that has to be classified, we extract features from that image and find the product of prior probability and likelihood probabilities for each class and find the class which has the maximum probability.

Random Forest

A random forest is made up of a large number of independent decision trees called estimators that work together as an ensemble. For our use case, we have used 100 estimators. Each tree (of this random forest) predicts a class, and the class with the most votes becomes the Forest's forecast. Random Forest's core principle is simple, A large number of relatively uncorrelated decision tree models acting as a committee outperforms each of the individual constituent tree models.

Each pixel is a feature for us. For each image, We concatenated the pixels into a 1D array and passed the array to the Random Forest model. Whenever there's an image to be classified, we convert it into a 1D array and pass it to the trained Random Forest.

CLASSIFIERS PERFORMANCE ON DIGITS

Training time for Digits Classification

As shown in Figure 5, we observe an increasing trend in the amount of time taken for training as the size of the training data increases. Random Forest classifier appeared to be taking the maximum training time among the three across all training sizes and Naive Bayes the least, while Perceptron almost the same time as the Random Forest classifier. Reading from the graph, the average training time (over 50 simulations) consumed by the classifiers (for training sizes ranging from 10% to 100%) is as follows:

- Perceptron ≈ 0.53 sec to 1.54 sec.
- Naive Bayes ≈ 0.12 sec to 1.21 sec.
- Random Forest ≈ 0.55 sec to 1.59 sec.

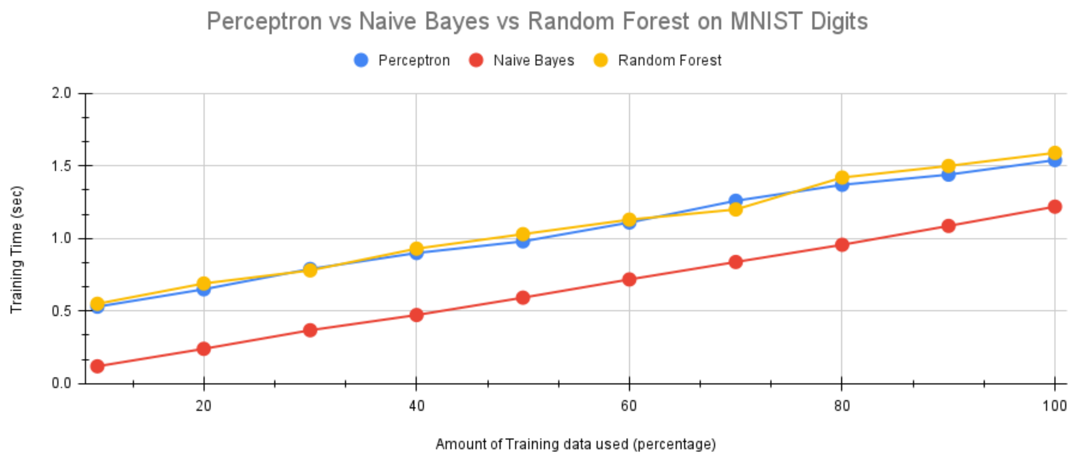


Figure 5: Training time observed for Perceptron, Naive Bayes, and Random Forest Classifiers

Prediction Error for Digits Classification

As the size of the training data increases, the prediction error decreased for all three classifiers, and this is shown in Figure 6. The Naive Bayes classifier was found less accurate among the three across all training sizes. Perceptron Classifier, although close to Naive Bayes, reported fewer misclassifications resulting in a marginally better accuracy over Naive Bayes. Random Forest classifier stands out among the three with a much lower prediction error (of about 10%).

Reading from the graph, the average prediction error $\pm$ standard_deviation (over 50 simulations) reported by the classifiers (for training sizes ranging from 10% to 100%) is as follows:

- Perceptron $\approx 32.62\% \pm 0.035$ to $19.62\% \pm 0.015$.
- Naive Bayes $\approx 37.86\% \pm 0.017$ to $24.64\% \pm 0.004$.
- Random Forest $\approx 17.18\% \pm 0.0103$ to $9.11\% \pm 0.004$.

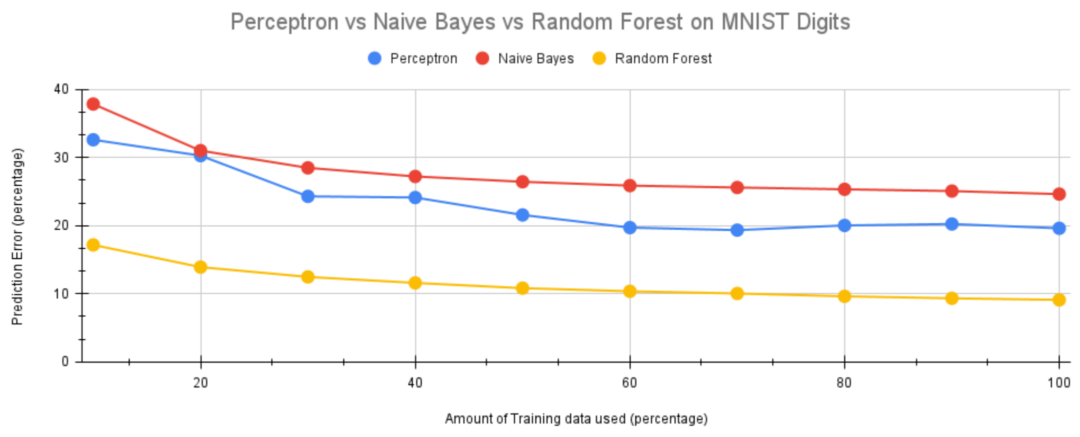


Figure 6: Prediction error observed for Perceptron, Naive Bayes, and Random Forest Classifiers

Alongside the prediction error, we have also plotted the graph for standard deviation across different training sizes for all three classifiers. The figure is shown below in Figure: 7.

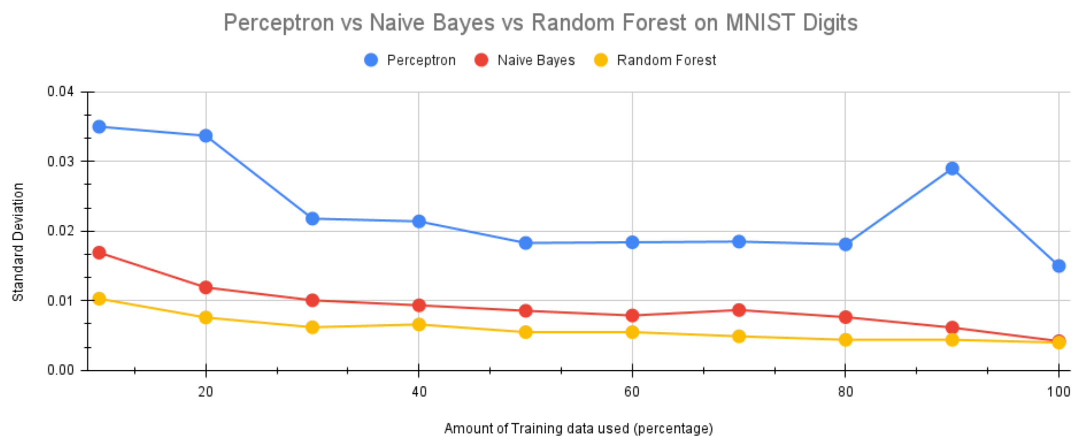


Figure 7: Observed Standard Deviation for Perceptron, Naive Bayes, and Random Forest Classifiers

CLASSIFIERS PERFORMANCE ON FACES

Training time for Faces Classification

As shown in Figure 8, we observe an increasing trend in the time taken for training as the size of the training data increases. Random Forest classifier appeared to be taking the maximum training time among the three across all training sizes and Perceptron the least, and Naive Bayes almost the same time as the Perceptron classifier. Reading from the graph, the average training time (over 50 simulations) consumed by the classifiers (for training sizes ranging from 10% to 100%) is as follows:

- Perceptron ≈ 0.2 sec to 0.26 sec.
- Naive Bayes ≈ 0.041 sec to 0.418 sec.
- Random Forest ≈ 0.53 sec to 1.67 sec.

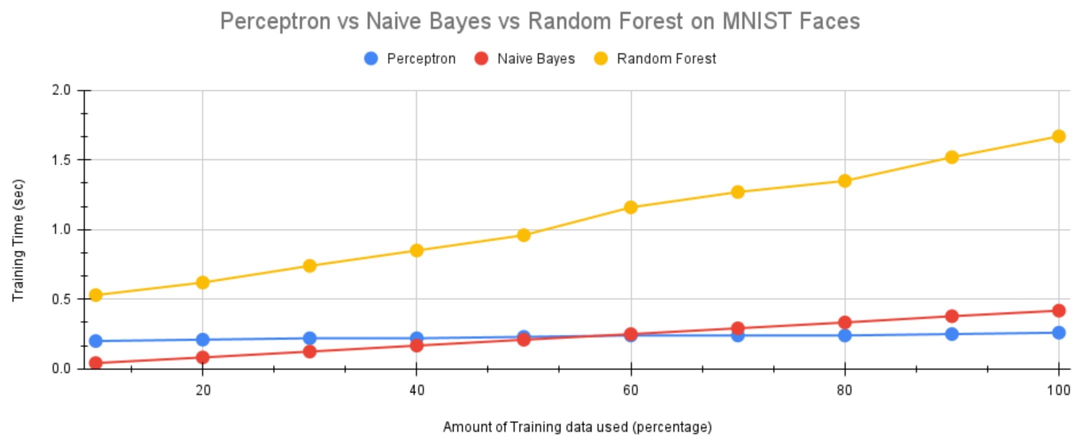


Figure 8: Training time observed for Perceptron, Naive Bayes, and Random Forest Classifiers

Prediction Error for Faces Classification

As the size of the training data increases, the prediction error decreases for all three classifiers, and that is shown in Figure 9. Although initial prediction errors for the three classifiers differed vastly (Naive Bayes the worst and Random Forest the best). As the size of the training increased, all the classifiers performed almost equally on face recognition. However, Random Forest still provides the least prediction error among the three, followed by Naive Bayes and Perceptron classifiers. Reading from the graph, the average prediction error ± standard_deviation (over

50 simulations) reported by the classifiers (for training sizes ranging from 10% to 100%) is as follows:

- Perceptron $\approx$ 29.68%±0.056 to 14.62%±0.021.
- Naive Bayes $\approx$ 39.31% $\pm$0.088 to 12.02% $\pm$0.001.
- Random Forest $\approx$ 17.3% $\pm$0.011 to 8.9% $\pm$0.0044.

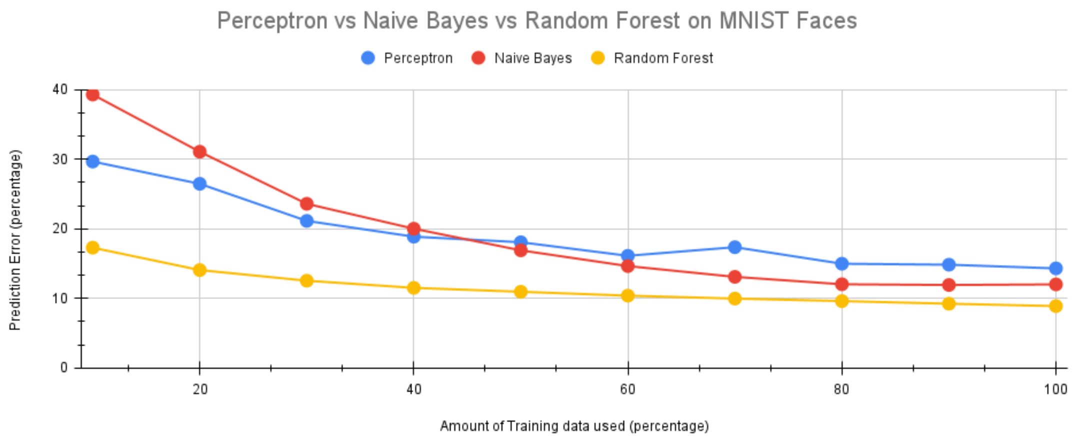


Figure 9: Prediction error observed for Perceptron, Naive Bayes, and Random Forest Classifiers

Alongside, the prediction error, we have also plotted the graph for standard deviation across different training sizes for all three classifiers. The figure is shown below in Figure: 10.

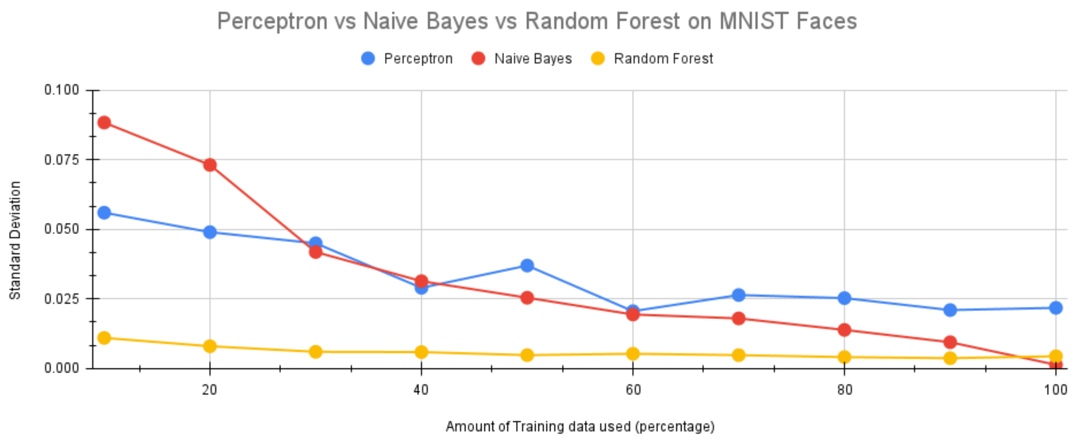


Figure 10: Observed Standard Deviation for Perceptron, Naive Bayes, and Random Forest Classifiers

ACKNOWLEDGMENTS

- Picture credits for Rutgers Logo is taken from online search[2].
- Some LATEX libraries have been used from the IIT Madras Overleaf template available here[3].

[2] https://logos-world.net/wp-content/uploads/2022/01/Rutgers-University-Emblem.png

[3] https://www.overleaf.com/latex/templates/iitmdissertation-the-official-latex-template-f pbrrtcygybkj