# Face and Digit Classification

Ridhima Biswas

## I. How to Run the Project

The tar file FinalExam.zip provided with the report is to be downloaded. Ensure that the following code files are in the directory.

- dataClassifier.py
- samples.py
- perceptron.py
- naiveBayes.py
- classificationMethod.py
- data.zip

The program is run through dataClassifier and follows this structure:

Python dataClassifier -c **<classifier_name>** -d **<type_of_data>** -t **<number_of_training_data_to_use>** -s **<number_test_data_to_be_used>**

The fields with the form <argument> can be changed as desired. The code can be run with each of the following arguments. These command line arguments will be utilizing all the images in the training and test sets in data.zip.

- Naive Bayes Faces: python dataClassifier.py -c naiveBayes -d faces -t 451 -s 150
- Naive Bayes Digits: python dataClassifier.py -c naiveBayes -d digits -t 5000 -s 1000
- Perceptron Faces: python dataClassifier.py -c perceptron -d faces -t 451 -s 150
- Perceptron Digits:  python dataClassifier.py -c perceptron -d digits -t 5000 -s 1000

## II. Implementation Approach for Both Classifiers

The approach in the implementation was heavily inspired by the UC Berkeley Introduction To Artificial Intelligence code files provided in Project 5 on their website.

The operation starts with the dataClassifier where the command line arguments are read in the readCommand function and processed to run the desired classifier with chosen data. The classifier can either be run using an informal method, runClassifier, or with runClassifierWithRandomSubsets which uses random sampling of the image data and gives more structured observations.

The method dataClassifer is a crucial code file because after the command lines are read and organized, the process of setting up parameters to the classifier begins. Feature extraction starts when the raw data is loaded into the program to be processed as a pixel image and then turned into a feature vector to be computed for the Naive Bayes and Perceptron algorithms. These feature vectors are represented as lists where the length equals the total number of pixels in an image and binary values depending on whether it is on or off (if there is a marking or not).

The pixel data is flattened row by row (or column by column depending on implementation), meaning that all pixel values from the first row are appended to the list, followed by values from the second row, and so on. Both the training and test data are converted into feature vectors that are fed into the classifier class.

The method runClassifierWithRandomSubsets in dataClassifier allows a more observational perspective of the training. Given the input arguments's desire for a chosen data and classifier to train with, the algorithm with the feature vectors goes through each subset percentages (10%, 20%, …, 100% of training data). The training subset size is then calculated and used for random sampling from the total training data set. With this random sample the algorithm trains the model (Naive Bayes or Perceptron) on the desired data while keeping track of time. After the training is finished, the method will run the test set and calculate the amount of correct guesses. This process is repeated five times due to time limitations, though more trials could be conducted to get more accurate results.

The two classifiers in perceptron.py and naiveBayes.py are crucial to the training stage. When runClassifier starts the training process, the process will be conducted in either one of those classes. The implemented approach in naiveBayes.py uses the prior probability table and conditional probability table in order to eventually run a calculated probability for each class. In order to build the parameters of the naiveBayes, it was necessary to count the label and feature frequencies and normalize the counts to obtain a probability estimate for the prior and conditional tables. In addition, Laplacian smoothing was utilized to handle unseen feature-label combinations. This approach ensures the model can withstand when it encounters features or labels that are rare in the training data. The classification approach uses the Bayes formula where for each label, the equation below is computed. The label with the highest probability will be the predicted label.

$$\hat{y} = \underset{y}{argmax}\, P(y) \prod_{i=1}^{n} P(x_i | y)$$

The Perceptron class is also another classifier that can be used to train the data and focuses on using weight vectors instead of probability. The weights for each label are initially zero and the size equals the feature vector dimension. For a fixed number of iterations, the algorithm loops through each training datum and the associated labels and adjusts the weights. For each training datum, the scores are computed for each label with the dot product between the feature vector and the weight weight. The highest score is picked as the predicted label. If the predicted label is wrong then the weight is adjusted by adding or subtracting the feature vector from the weight of correct and incorrect labels. For classification, for each input datum, the dot product between the datum and weight vectors is calculated for all labels. The predicted will be the highest score label.

## III. Analysis of Observations and Data on Both Classifiers

This section focused on the output results that showcase detailed observations of the two datasets on each classifier provided in Section V. The key metrics that are examined are the accuracy rate, prediction error, standard deviation, and training time. From the data, some important analysis of results are as follows:

In the faces dataset, perceptron outperformed Naive Bates across all training sizes generally which indicates a better ability to learn in the perceptron algorithm with this dataset. This is likely due to the assumptions of feature independence in Naive Bayes, which may hold more for pixel data in faces. Error rates for Perceptron are also lower than Naive Bayes across all training sizes. The standard deviation for accuracy is generally higher in Naive Bayes, suggesting less stable performance compared to Perceptron.

For the digits dataset, both classifiers perform better on the dataset compared to faces. This is likely due to digit data being more structured and less noisy making it more stable to work with. Perceptron still slightly outperforms Naive Bayes in most cases but not as significantly as in faces. In addition, error rates are closer between Naive Bayes and Perceptron on this dataset, indicating that Naive Bayes' assumptions are better suited for digits. Similar to faces, Naive Bayes trains significantly faster. Perceptron's performance had a lower standard deviation and thus digits is more stable, reflecting its ability to generalize better on structured datasets.

Next, the relationship between improvement of accuracy and training data size was examined. Both classifiers had higher accuracy rates with more training data. Perceptron especially expressed more significant gains in accuracy with more training data. For example when using the faces dataset, it was more capable to use the data more effectively. However, as the training data increased, there is also diminishing returns. For instance Naive Bayes reaches a plateau in accuracy as training size increases, indicating that it cannot extract additional patterns beyond a certain limit. Error rates decrease with more training data for both classifiers, but the rate of decrease slows over time, showing diminishing returns with very large datasets.

From the observation data, some conclusions can also be made about each of the two data sets. Faces data is likely more complex and noisy, which explains Naive Bayes' lower performance. Perceptron's ability to model relationships between features gives it a significant edge. Perceptron might benefit even more from techniques like feature scaling or additional regularization for Faces. Digits are simpler and more structured, making it easier for both classifiers to perform well. Naive Bayes performs reasonably well here, as digit pixel data may align better with its independence assumption.

## IV. Learned Lessons

During the implementation of the perceptron algorithm, it was apparent that it performed well with the faces dataset. Perceptron generally performs well with complex datasets. This is due to perceptron being a linear classifier, meaning it finds a decision boundary (hyperplane) to separate data points into classes. Even though Faces datasets are complex, it is still linearly separable which allows for decision boundaries. However, this classifier also has longer training times which can be a downside as training data is increased.

On the other hand, Naive Bayes was very effective for simpler datasets or limited resources. Naive Bayes operates on the assumption that all features are conditionally independent given the class label. While this assumption rarely holds true in complex datasets, it is often reasonable for simpler datasets where features have little communication and can be modeled independently without losing too much accuracy. Naive Bayes is highly efficient because there are no iterative updates like in the Perceptron algorithm.

When it comes to classification problems, the conclusion made was that it is important to balance training time and accuracy requirements. For real-time applications, Naive Bayes is preferable, while Perceptron suits high-accuracy needs.

## V. Detailed Observations and Data

Observations and data was noted for both classifiers and their performance on both databases. Each section starts with the data, classifier, total data size used to train, and the size of the test data. For each percentage of the data (10%, 20%, and so on), five trials were conducted to ensure consistent results and to account for variability in classifier performance using random sampling. For example, in performance using 10% of the training data, each of the five trials was recorded in terms of its training time, percentage of accuracy, and percentage of error (100 - accuracy percentage). When five trials were undertaken, the average training time was displayed. In addition, the mean over the accuracy and error was calculated along with standard deviation across the five trials which accounts for variability.

### 1. Naive Bayes: Faces

```
--------------------
Data: faces
Classifier: naiveBayes
```

```
Training Data Size: 451
Test Data Size: 150

[10% - Run 1] Training Time: 0.01s | Accuracy: 74.0% | Error: 26.0%
[10% - Run 2] Training Time: 0.01s | Accuracy: 76.7% | Error: 23.3%
[10% - Run 3] Training Time: 0.01s | Accuracy: 88.0% | Error: 12.0%
[10% - Run 4] Training Time: 0.01s | Accuracy: 71.3% | Error: 28.7%
[10% - Run 5] Training Time: 0.01s | Accuracy: 76.7% | Error: 23.3%
[10%] Mean Accuracy: 77.3% | Mean Error: 22.7% | Std Dev: 6.36 | Avg Training Time: 0.01s

[20% - Run 1] Training Time: 0.01s | Accuracy: 84.0% | Error: 16.0%
[20% - Run 2] Training Time: 0.01s | Accuracy: 80.0% | Error: 20.0%
[20% - Run 3] Training Time: 0.01s | Accuracy: 85.3% | Error: 14.7%
[20% - Run 4] Training Time: 0.01s | Accuracy: 85.3% | Error: 14.7%
[20% - Run 5] Training Time: 0.01s | Accuracy: 81.3% | Error: 18.7%
[20%] Mean Accuracy: 83.2% | Mean Error: 16.8% | Std Dev: 2.42 | Avg Training Time: 0.02s

[30% - Run 1] Training Time: 0.02s | Accuracy: 84.0% | Error: 16.0%
[30% - Run 2] Training Time: 0.02s | Accuracy: 88.7% | Error: 11.3%
[30% - Run 3] Training Time: 0.02s | Accuracy: 83.3% | Error: 16.7%
[30% - Run 4] Training Time: 0.02s | Accuracy: 86.0% | Error: 14.0%
[30% - Run 5] Training Time: 0.02s | Accuracy: 86.0% | Error: 14.0%
[30%] Mean Accuracy: 85.6% | Mean Error: 14.4% | Std Dev: 2.09 | Avg Training Time: 0.03s

[40% - Run 1] Training Time: 0.02s | Accuracy: 88.7% | Error: 11.3%
[40% - Run 2] Training Time: 0.02s | Accuracy: 88.7% | Error: 11.3%
[40% - Run 3] Training Time: 0.02s | Accuracy: 85.3% | Error: 14.7%
[40% - Run 4] Training Time: 0.02s | Accuracy: 85.3% | Error: 14.7%
[40% - Run 5] Training Time: 0.02s | Accuracy: 88.0% | Error: 12.0%
[40%] Mean Accuracy: 87.2% | Mean Error: 12.8% | Std Dev: 1.73 | Avg Training Time: 0.06s

[50% - Run 1] Training Time: 0.03s | Accuracy: 88.7% | Error: 11.3%
[50% - Run 2] Training Time: 0.03s | Accuracy: 87.3% | Error: 12.7%
[50% - Run 3] Training Time: 0.03s | Accuracy: 86.7% | Error: 13.3%
[50% - Run 4] Training Time: 0.03s | Accuracy: 86.7% | Error: 13.3%
[50% - Run 5] Training Time: 0.03s | Accuracy: 90.7% | Error: 9.3%
[50%] Mean Accuracy: 88.0% | Mean Error: 12.0% | Std Dev: 1.70 | Avg Training Time: 0.08s

[60% - Run 1] Training Time: 0.03s | Accuracy: 86.7% | Error: 13.3%
[60% - Run 2] Training Time: 0.03s | Accuracy: 88.7% | Error: 11.3%
```

[60% - Run 3] Training Time: 0.03s | Accuracy: 88.0% | Error: 12.0%
[60% - Run 4] Training Time: 0.03s | Accuracy: 90.0% | Error: 10.0%
[60% - Run 5] Training Time: 0.03s | Accuracy: 88.0% | Error: 12.0%
**[60%]** Mean Accuracy: 88.3% | Mean Error: 11.7% | Std Dev: 1.21 | Avg Training Time: 0.12s

[70% - Run 1] Training Time: 0.04s | Accuracy: 88.0% | Error: 12.0%
[70% - Run 2] Training Time: 0.04s | Accuracy: 89.3% | Error: 10.7%
[70% - Run 3] Training Time: 0.04s | Accuracy: 87.3% | Error: 12.7%
[70% - Run 4] Training Time: 0.04s | Accuracy: 87.3% | Error: 12.7%
[70% - Run 5] Training Time: 0.04s | Accuracy: 89.3% | Error: 10.7%
**[70%]** Mean Accuracy: 88.3% | Mean Error: 11.7% | Std Dev: 1.01 | Avg Training Time: 0.15s

[80% - Run 1] Training Time: 0.04s | Accuracy: 87.3% | Error: 12.7%
[80% - Run 2] Training Time: 0.04s | Accuracy: 88.7% | Error: 11.3%
[80% - Run 3] Training Time: 0.04s | Accuracy: 87.3% | Error: 12.7%
[80% - Run 4] Training Time: 0.04s | Accuracy: 88.0% | Error: 12.0%
[80% - Run 5] Training Time: 0.04s | Accuracy: 88.7% | Error: 11.3%
**[80%]** Mean Accuracy: 88.0% | Mean Error: 12.0% | Std Dev: 0.67 | Avg Training Time: 0.20s

[90% - Run 1] Training Time: 0.05s | Accuracy: 86.7% | Error: 13.3%
[90% - Run 2] Training Time: 0.05s | Accuracy: 86.0% | Error: 14.0%
[90% - Run 3] Training Time: 0.05s | Accuracy: 86.7% | Error: 13.3%
[90% - Run 4] Training Time: 0.05s | Accuracy: 87.3% | Error: 12.7%
[90% - Run 5] Training Time: 0.05s | Accuracy: 89.3% | Error: 10.7%
**[90%]** Mean Accuracy: 87.2% | Mean Error: 12.8% | Std Dev: 1.28 | Avg Training Time: 0.25s

[100% - Run 1] Training Time: 0.05s | Accuracy: 88.7% | Error: 11.3%
[100% - Run 2] Training Time: 0.05s | Accuracy: 88.7% | Error: 11.3%
[100% - Run 3] Training Time: 0.05s | Accuracy: 88.7% | Error: 11.3%
[100% - Run 4] Training Time: 0.05s | Accuracy: 88.7% | Error: 11.3%
[100% - Run 5] Training Time: 0.06s | Accuracy: 88.7% | Error: 11.3%
**[100%]** Mean Accuracy: 88.7% | Mean Error: 11.3% | Std Dev: 0.00 | Avg Training Time: 0.30s

**Overall Mean Accuracy: 86.2% | Overall Std Dev: 4.0%**

## 2. Naive Bayes: Digits

--------------------
Data: digits

Classifier: naiveBayes
Training Data Size: 5000
Test Data Size: 1000

[10% - Run 1] Training Time: 0.01s | Accuracy: 71.4% | Error: 28.6%
[10% - Run 2] Training Time: 0.01s | Accuracy: 73.3% | Error: 26.7%
[10% - Run 3] Training Time: 0.01s | Accuracy: 72.1% | Error: 27.9%
[10% - Run 4] Training Time: 0.01s | Accuracy: 74.5% | Error: 25.5%
[10% - Run 5] Training Time: 0.01s | Accuracy: 69.4% | Error: 30.6%
**[10%]** Mean Accuracy: 72.1% | Mean Error: 27.9% | Std Dev: 1.93 | Avg Training Time: 0.01s

[20% - Run 1] Training Time: 0.02s | Accuracy: 74.8% | Error: 25.2%
[20% - Run 2] Training Time: 0.02s | Accuracy: 74.2% | Error: 25.8%
[20% - Run 3] Training Time: 0.02s | Accuracy: 73.3% | Error: 26.7%
[20% - Run 4] Training Time: 0.02s | Accuracy: 74.2% | Error: 25.8%
[20% - Run 5] Training Time: 0.02s | Accuracy: 72.7% | Error: 27.3%
**[20%]** Mean Accuracy: 73.8% | Mean Error: 26.2% | Std Dev: 0.83 | Avg Training Time: 0.03s

[30% - Run 1] Training Time: 0.03s | Accuracy: 73.2% | Error: 26.8%
[30% - Run 2] Training Time: 0.03s | Accuracy: 73.5% | Error: 26.5%
[30% - Run 3] Training Time: 0.03s | Accuracy: 75.2% | Error: 24.8%
[30% - Run 4] Training Time: 0.03s | Accuracy: 73.5% | Error: 26.5%
[30% - Run 5] Training Time: 0.03s | Accuracy: 73.7% | Error: 26.3%
**[30%]** Mean Accuracy: 73.8% | Mean Error: 26.2% | Std Dev: 0.79 | Avg Training Time: 0.07s

[40% - Run 1] Training Time: 0.04s | Accuracy: 74.8% | Error: 25.2%
[40% - Run 2] Training Time: 0.04s | Accuracy: 74.8% | Error: 25.2%
[40% - Run 3] Training Time: 0.04s | Accuracy: 73.8% | Error: 26.2%
[40% - Run 4] Training Time: 0.04s | Accuracy: 74.9% | Error: 25.1%
[40% - Run 5] Training Time: 0.04s | Accuracy: 73.9% | Error: 26.1%
**[40%]** Mean Accuracy: 74.4% | Mean Error: 25.6% | Std Dev: 0.54 | Avg Training Time: 0.11s

[50% - Run 1] Training Time: 0.05s | Accuracy: 75.1% | Error: 24.9%
[50% - Run 2] Training Time: 0.05s | Accuracy: 74.6% | Error: 25.4%
[50% - Run 3] Training Time: 0.05s | Accuracy: 73.8% | Error: 26.2%
[50% - Run 4] Training Time: 0.05s | Accuracy: 74.9% | Error: 25.1%
[50% - Run 5] Training Time: 0.05s | Accuracy: 73.3% | Error: 26.7%
**[50%]** Mean Accuracy: 74.3% | Mean Error: 25.7% | Std Dev: 0.76 | Avg Training Time: 0.17s

[60% - Run 1] Training Time: 0.07s | Accuracy: 74.7% | Error: 25.3%

[60% - Run 2] Training Time: 0.07s | Accuracy: 74.7% | Error: 25.3%
[60% - Run 3] Training Time: 0.07s | Accuracy: 74.5% | Error: 25.5%
[60% - Run 4] Training Time: 0.07s | Accuracy: 75.0% | Error: 25.0%
[60% - Run 5] Training Time: 0.07s | Accuracy: 74.9% | Error: 25.1%
**[60%]** Mean Accuracy: 74.8% | Mean Error: 25.2% | Std Dev: 0.19 | Avg Training Time: 0.23s

[70% - Run 1] Training Time: 0.08s | Accuracy: 74.6% | Error: 25.4%
[70% - Run 2] Training Time: 0.08s | Accuracy: 74.7% | Error: 25.3%
[70% - Run 3] Training Time: 0.08s | Accuracy: 75.1% | Error: 24.9%
[70% - Run 4] Training Time: 0.08s | Accuracy: 74.5% | Error: 25.5%
[70% - Run 5] Training Time: 0.08s | Accuracy: 75.2% | Error: 24.8%
**[70%]** Mean Accuracy: 74.8% | Mean Error: 25.2% | Std Dev: 0.31 | Avg Training Time: 0.31s

[80% - Run 1] Training Time: 0.09s | Accuracy: 74.9% | Error: 25.1%
[80% - Run 2] Training Time: 0.09s | Accuracy: 75.4% | Error: 24.6%
[80% - Run 3] Training Time: 0.09s | Accuracy: 74.8% | Error: 25.2%
[80% - Run 4] Training Time: 0.09s | Accuracy: 75.7% | Error: 24.3%
[80% - Run 5] Training Time: 0.09s | Accuracy: 75.5% | Error: 24.5%
**[80%]** Mean Accuracy: 75.3% | Mean Error: 24.7% | Std Dev: 0.39 | Avg Training Time: 0.40s

[90% - Run 1] Training Time: 0.10s | Accuracy: 75.3% | Error: 24.7%
[90% - Run 2] Training Time: 0.10s | Accuracy: 75.3% | Error: 24.7%
[90% - Run 3] Training Time: 0.10s | Accuracy: 75.4% | Error: 24.6%
[90% - Run 4] Training Time: 0.10s | Accuracy: 75.0% | Error: 25.0%
[90% - Run 5] Training Time: 0.10s | Accuracy: 75.3% | Error: 24.7%
**[90%]** Mean Accuracy: 75.3% | Mean Error: 24.7% | Std Dev: 0.15 | Avg Training Time: 0.50s

[100% - Run 1] Training Time: 0.11s | Accuracy: 75.4% | Error: 24.6%
[100% - Run 2] Training Time: 0.11s | Accuracy: 75.4% | Error: 24.6%
[100% - Run 3] Training Time: 0.11s | Accuracy: 75.4% | Error: 24.6%
[100% - Run 4] Training Time: 0.11s | Accuracy: 75.4% | Error: 24.6%
[100% - Run 5] Training Time: 0.11s | Accuracy: 75.4% | Error: 24.6%
**[100%]** Mean Accuracy: 75.4% | Mean Error: 24.6% | Std Dev: 0.00 | Avg Training Time: 0.61s

**Overall Mean Accuracy: 74.4% | Overall Std Dev: 1.2%**

## 3. Perceptron: Faces

--------------------
Data: faces

```
Classifier: perceptron
Training Data Size: 451
Test Data Size: 150

[10% - Run 1] Training Time: 0.10s | Accuracy: 67.3% | Error: 32.7%
[10% - Run 2] Training Time: 0.10s | Accuracy: 72.0% | Error: 28.0%
[10% - Run 3] Training Time: 0.10s | Accuracy: 75.3% | Error: 24.7%
[10% - Run 4] Training Time: 0.10s | Accuracy: 77.3% | Error: 22.7%
[10% - Run 5] Training Time: 0.10s | Accuracy: 77.3% | Error: 22.7%
[10%] Mean Accuracy: 73.9% | Mean Error: 26.1% | Std Dev: 4.25 | Avg Training Time: 0.10s

[20% - Run 1] Training Time: 0.20s | Accuracy: 77.3% | Error: 22.7%
[20% - Run 2] Training Time: 0.19s | Accuracy: 79.3% | Error: 20.7%
[20% - Run 3] Training Time: 0.19s | Accuracy: 84.0% | Error: 16.0%
[20% - Run 4] Training Time: 0.20s | Accuracy: 84.0% | Error: 16.0%
[20% - Run 5] Training Time: 0.20s | Accuracy: 86.7% | Error: 13.3%
[20%] Mean Accuracy: 82.3% | Mean Error: 17.7% | Std Dev: 3.82 | Avg Training Time: 0.29s

[30% - Run 1] Training Time: 0.29s | Accuracy: 86.7% | Error: 13.3%
[30% - Run 2] Training Time: 0.31s | Accuracy: 85.3% | Error: 14.7%
[30% - Run 3] Training Time: 0.30s | Accuracy: 84.7% | Error: 15.3%
[30% - Run 4] Training Time: 0.29s | Accuracy: 88.0% | Error: 12.0%
[30% - Run 5] Training Time: 0.29s | Accuracy: 88.0% | Error: 12.0%
[30%] Mean Accuracy: 86.5% | Mean Error: 13.5% | Std Dev: 1.52 | Avg Training Time: 0.59s

[40% - Run 1] Training Time: 0.39s | Accuracy: 88.0% | Error: 12.0%
[40% - Run 2] Training Time: 0.39s | Accuracy: 88.0% | Error: 12.0%
[40% - Run 3] Training Time: 0.39s | Accuracy: 88.0% | Error: 12.0%
[40% - Run 4] Training Time: 0.39s | Accuracy: 88.0% | Error: 12.0%
[40% - Run 5] Training Time: 0.39s | Accuracy: 87.3% | Error: 12.7%
[40%] Mean Accuracy: 87.9% | Mean Error: 12.1% | Std Dev: 0.30 | Avg Training Time: 0.98s

[50% - Run 1] Training Time: 0.48s | Accuracy: 87.3% | Error: 12.7%
[50% - Run 2] Training Time: 0.48s | Accuracy: 87.3% | Error: 12.7%
[50% - Run 3] Training Time: 0.48s | Accuracy: 87.3% | Error: 12.7%
[50% - Run 4] Training Time: 0.48s | Accuracy: 87.3% | Error: 12.7%
[50% - Run 5] Training Time: 0.48s | Accuracy: 87.3% | Error: 12.7%
[50%] Mean Accuracy: 87.3% | Mean Error: 12.7% | Std Dev: 0.00 | Avg Training Time: 1.46s

[60% - Run 1] Training Time: 0.58s | Accuracy: 87.3% | Error: 12.7%
```

```
[60% - Run 2] Training Time: 0.59s | Accuracy: 87.3% | Error: 12.7%
[60% - Run 3] Training Time: 0.59s | Accuracy: 87.3% | Error: 12.7%
[60% - Run 4] Training Time: 0.64s | Accuracy: 87.3% | Error: 12.7%
[60% - Run 5] Training Time: 0.59s | Accuracy: 87.3% | Error: 12.7%
[60%] Mean Accuracy: 87.3% | Mean Error: 12.7% | Std Dev: 0.00 | Avg Training Time: 2.06s

[70% - Run 1] Training Time: 0.69s | Accuracy: 87.3% | Error: 12.7%
[70% - Run 2] Training Time: 0.69s | Accuracy: 87.3% | Error: 12.7%
[70% - Run 3] Training Time: 0.70s | Accuracy: 87.3% | Error: 12.7%
[70% - Run 4] Training Time: 0.68s | Accuracy: 87.3% | Error: 12.7%
[70% - Run 5] Training Time: 0.68s | Accuracy: 87.3% | Error: 12.7%
[70%] Mean Accuracy: 87.3% | Mean Error: 12.7% | Std Dev: 0.00 | Avg Training Time: 2.75s

[80% - Run 1] Training Time: 0.78s | Accuracy: 87.3% | Error: 12.7%
[80% - Run 2] Training Time: 0.78s | Accuracy: 87.3% | Error: 12.7%
[80% - Run 3] Training Time: 0.79s | Accuracy: 87.3% | Error: 12.7%
[80% - Run 4] Training Time: 0.78s | Accuracy: 87.3% | Error: 12.7%
[80% - Run 5] Training Time: 0.78s | Accuracy: 87.3% | Error: 12.7%
[80%] Mean Accuracy: 87.3% | Mean Error: 12.7% | Std Dev: 0.00 | Avg Training Time: 3.53s

[90% - Run 1] Training Time: 0.88s | Accuracy: 87.3% | Error: 12.7%
[90% - Run 2] Training Time: 0.88s | Accuracy: 87.3% | Error: 12.7%
[90% - Run 3] Training Time: 0.87s | Accuracy: 87.3% | Error: 12.7%
[90% - Run 4] Training Time: 0.88s | Accuracy: 87.3% | Error: 12.7%
[90% - Run 5] Training Time: 0.88s | Accuracy: 87.3% | Error: 12.7%
[90%] Mean Accuracy: 87.3% | Mean Error: 12.7% | Std Dev: 0.00 | Avg Training Time: 4.41s

[100% - Run 1] Training Time: 0.98s | Accuracy: 87.3% | Error: 12.7%
[100% - Run 2] Training Time: 0.97s | Accuracy: 87.3% | Error: 12.7%
[100% - Run 3] Training Time: 0.97s | Accuracy: 87.3% | Error: 12.7%
[100% - Run 4] Training Time: 0.97s | Accuracy: 87.3% | Error: 12.7%
[100% - Run 5] Training Time: 0.98s | Accuracy: 87.3% | Error: 12.7%
[100%] Mean Accuracy: 87.3% | Mean Error: 12.7% | Std Dev: 0.00 | Avg Training Time: 5.38s
```

**Overall Mean Accuracy: 85.5% | Overall Std Dev: 4.5%**

## 4. Perceptron: Digits

```
--------------------
Data: digits
```

```
Classifier: perceptron
Training Data Size: 5000
Test Data Size: 1000

[10% - Run 1] Training Time: 0.30s | Accuracy: 76.9% | Error: 23.1%
[10% - Run 2] Training Time: 0.30s | Accuracy: 79.4% | Error: 20.6%
[10% - Run 3] Training Time: 0.30s | Accuracy: 79.9% | Error: 20.1%
[10% - Run 4] Training Time: 0.30s | Accuracy: 80.9% | Error: 19.1%
[10% - Run 5] Training Time: 0.30s | Accuracy: 80.0% | Error: 20.0%
[10%] Mean Accuracy: 79.4% | Mean Error: 20.6% | Std Dev: 1.51 | Avg Training Time: 0.30s

[20% - Run 1] Training Time: 0.59s | Accuracy: 82.9% | Error: 17.1%
[20% - Run 2] Training Time: 0.59s | Accuracy: 81.1% | Error: 18.9%
[20% - Run 3] Training Time: 0.59s | Accuracy: 81.8% | Error: 18.2%
[20% - Run 4] Training Time: 0.60s | Accuracy: 80.0% | Error: 20.0%
[20% - Run 5] Training Time: 0.59s | Accuracy: 80.0% | Error: 20.0%
[20%] Mean Accuracy: 81.2% | Mean Error: 18.8% | Std Dev: 1.24 | Avg Training Time: 0.89s

[30% - Run 1] Training Time: 0.89s | Accuracy: 80.5% | Error: 19.5%
[30% - Run 2] Training Time: 0.89s | Accuracy: 80.9% | Error: 19.1%
[30% - Run 3] Training Time: 0.95s | Accuracy: 81.5% | Error: 18.5%
[30% - Run 4] Training Time: 1.01s | Accuracy: 80.2% | Error: 19.8%
[30% - Run 5] Training Time: 0.90s | Accuracy: 80.6% | Error: 19.4%
[30%] Mean Accuracy: 80.7% | Mean Error: 19.3% | Std Dev: 0.49 | Avg Training Time: 1.82s

[40% - Run 1] Training Time: 1.19s | Accuracy: 81.9% | Error: 18.1%
[40% - Run 2] Training Time: 1.20s | Accuracy: 81.5% | Error: 18.5%
[40% - Run 3] Training Time: 1.20s | Accuracy: 82.9% | Error: 17.1%
[40% - Run 4] Training Time: 1.20s | Accuracy: 82.5% | Error: 17.5%
[40% - Run 5] Training Time: 1.19s | Accuracy: 82.7% | Error: 17.3%
[40%] Mean Accuracy: 82.3% | Mean Error: 17.7% | Std Dev: 0.58 | Avg Training Time: 3.02s

[50% - Run 1] Training Time: 1.50s | Accuracy: 82.2% | Error: 17.8%
[50% - Run 2] Training Time: 1.49s | Accuracy: 82.9% | Error: 17.1%
[50% - Run 3] Training Time: 1.49s | Accuracy: 81.7% | Error: 18.3%
[50% - Run 4] Training Time: 1.50s | Accuracy: 82.1% | Error: 17.9%
[50% - Run 5] Training Time: 1.49s | Accuracy: 82.5% | Error: 17.5%
[50%] Mean Accuracy: 82.3% | Mean Error: 17.7% | Std Dev: 0.45 | Avg Training Time: 4.51s

[60% - Run 1] Training Time: 1.79s | Accuracy: 82.4% | Error: 17.6%
```

[60% - Run 2] Training Time: 1.79s | Accuracy: 82.0% | Error: 18.0%
[60% - Run 3] Training Time: 1.79s | Accuracy: 81.6% | Error: 18.4%
[60% - Run 4] Training Time: 1.79s | Accuracy: 81.7% | Error: 18.3%
[60% - Run 5] Training Time: 1.80s | Accuracy: 82.5% | Error: 17.5%
**[60%]** Mean Accuracy: 82.0% | Mean Error: 18.0% | Std Dev: 0.40 | Avg Training Time: 6.30s

[70% - Run 1] Training Time: 2.09s | Accuracy: 81.4% | Error: 18.6%
[70% - Run 2] Training Time: 2.09s | Accuracy: 81.4% | Error: 18.6%
[70% - Run 3] Training Time: 2.09s | Accuracy: 82.2% | Error: 17.8%
[70% - Run 4] Training Time: 2.09s | Accuracy: 82.2% | Error: 17.8%
[70% - Run 5] Training Time: 2.09s | Accuracy: 82.2% | Error: 17.8%
**[70%]** Mean Accuracy: 81.9% | Mean Error: 18.1% | Std Dev: 0.44 | Avg Training Time: 8.39s

[80% - Run 1] Training Time: 2.39s | Accuracy: 82.3% | Error: 17.7%
[80% - Run 2] Training Time: 2.39s | Accuracy: 82.3% | Error: 17.7%
[80% - Run 3] Training Time: 2.39s | Accuracy: 82.3% | Error: 17.7%
[80% - Run 4] Training Time: 2.39s | Accuracy: 82.3% | Error: 17.7%
[80% - Run 5] Training Time: 2.39s | Accuracy: 82.3% | Error: 17.7%
**[80%]** Mean Accuracy: 82.3% | Mean Error: 17.7% | Std Dev: 0.00 | Avg Training Time: 10.78s

[90% - Run 1] Training Time: 2.69s | Accuracy: 82.3% | Error: 17.7%
[90% - Run 2] Training Time: 2.69s | Accuracy: 82.3% | Error: 17.7%
[90% - Run 3] Training Time: 2.69s | Accuracy: 82.3% | Error: 17.7%
[90% - Run 4] Training Time: 2.69s | Accuracy: 82.3% | Error: 17.7%
[90% - Run 5] Training Time: 2.69s | Accuracy: 82.3% | Error: 17.7%
**[90%]** Mean Accuracy: 82.3% | Mean Error: 17.7% | Std Dev: 0.00 | Avg Training Time: 13.47s

[100% - Run 1] Training Time: 3.00s | Accuracy: 82.3% | Error: 17.7%
[100% - Run 2] Training Time: 3.00s | Accuracy: 82.3% | Error: 17.7%
[100% - Run 3] Training Time: 3.00s | Accuracy: 82.3% | Error: 17.7%
[100% - Run 4] Training Time: 3.01s | Accuracy: 82.3% | Error: 17.7%
[100% - Run 5] Training Time: 3.00s | Accuracy: 82.3% | Error: 17.7%
**[100%]** Mean Accuracy: 82.3% | Mean Error: 17.7% | Std Dev: 0.00 | Avg Training Time: 16.47s

**Overall Mean Accuracy: 81.7% | Overall Std Dev: 1.1%**

## References

“Project 5: Classification.” *Berkeley Ai Materials*, University of Berkeley, 26 Aug. 2014, ai.berkeley.edu/classification.html.