Kenneth Augusto (213008718), Julian Grande (203003045), Anthony Mathis (200007361)

Final Project

DIGITS & FACES

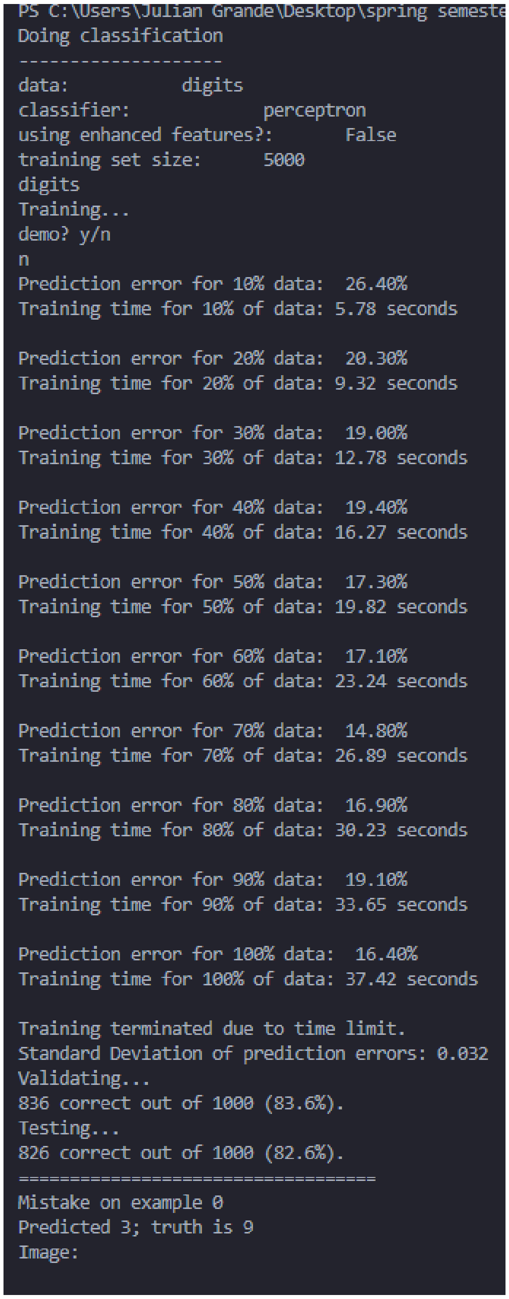
The ‘perceptron’ algorithm using ‘dataClassifier’, to run use the commands:

“python(3) dataClassifier.py -c perceptron -t 5000 -s 1000” : for digits
“python(3) dataClassifier.py -c perceptron -d faces -t 451” : for faces

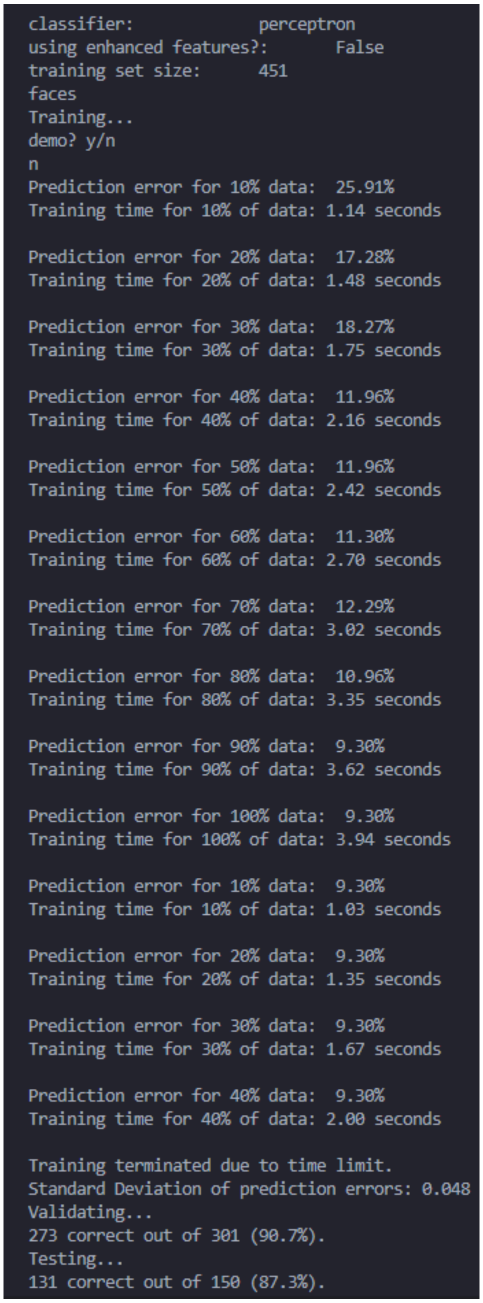
- Perceptron makes predictions using a linear predictor function that combines a set of weights with the feature vector.
    - The feature vector contains the raw pixels of the image converted to 1’s and 0’s
- Training function runs for ~3.5 mins and continually selects random samples of 10%,20%,30%,...100% of the training data
    - It updates weights during training to minimize classification errors.
    - calculates prediction error for that training subset %
    - calculates training time for that training subset %
- classify function utilizes classify_single function, in order to take the list of datums and use dot product of label and weight vector to predict what the image is
- Once training is finished, validation and testing data is put through the classify function, alongside their respective labels, and the accuracy of validation and testing data is printed
- Analysis of the first misclassified face/digit along with the image is printed

In terms of difficulty, the biggest challenge in implementing the perceptron algorithm at first was reading the data correctly. Using the provided Berkeley code, it was using an outdated version of python so it took us a while to figure out how we could implement it again, we ended up reimplementing the datums it reads and rolling them into lists and repurposing the classify function. When implementing the train algorithm itself, the sample code gave us a good starting point, along with the notes on the berkeley website. Our implementation trains the perceptron for 3 minutes for digits, 30 seconds for faces, as there are less data points in faces so it takes much less time to learn, during which it continually trains over a random sample of 10%,20%,30%,...100% the data, until time is up and if the prediction was incorrect we update the weights and labels accordingly. One slight adjustment we made to the Berkeley code was the way it would read the number of validation data and labels, it was doing so by using the same number of testing data and labels and was not fully utilizing the validation set. For simplicity, we hard coded the values of the number of validation labels and raw faces, as this was strictly an issue for the faces data set.

Time and Accuracy output for digits(left) and faces(right) as a function of % of data points used for training



```
PS C:\Users\Julian Grande\Desktop\spring semeste
Doing classification
--------------------
data:           digits
classifier:             perceptron
using enhanced features?:       False
training set size:      5000
digits
Training...
demo? y/n
n
Prediction error for 10% data:  26.40%
Training time for 10% of data: 5.78 seconds

Prediction error for 20% data:  20.30%
Training time for 20% of data: 9.32 seconds

Prediction error for 30% data:  19.00%
Training time for 30% of data: 12.78 seconds

Prediction error for 40% data:  19.40%
Training time for 40% of data: 16.27 seconds

Prediction error for 50% data:  17.30%
Training time for 50% of data: 19.82 seconds

Prediction error for 60% data:  17.10%
Training time for 60% of data: 23.24 seconds

Prediction error for 70% data:  14.80%
Training time for 70% of data: 26.89 seconds

Prediction error for 80% data:  16.90%
Training time for 80% of data: 30.23 seconds

Prediction error for 90% data:  19.10%
Training time for 90% of data: 33.65 seconds

Prediction error for 100% data:  16.40%
Training time for 100% of data: 37.42 seconds

Training terminated due to time limit.
Standard Deviation of prediction errors: 0.032
Validating...
836 correct out of 1000 (83.6%).
Testing...
826 correct out of 1000 (82.6%).
===================================
Mistake on example 0
Predicted 3; truth is 9
Image:
```



```
classifier:             perceptron
using enhanced features?:       False
training set size:      451
faces
Training...
demo? y/n
n
Prediction error for 10% data:  25.91%
Training time for 10% of data: 1.14 seconds

Prediction error for 20% data:  17.28%
Training time for 20% of data: 1.48 seconds

Prediction error for 30% data:  18.27%
Training time for 30% of data: 1.75 seconds

Prediction error for 40% data:  11.96%
Training time for 40% of data: 2.16 seconds

Prediction error for 50% data:  11.96%
Training time for 50% of data: 2.42 seconds

Prediction error for 60% data:  11.30%
Training time for 60% of data: 2.70 seconds

Prediction error for 70% data:  12.29%
Training time for 70% of data: 3.02 seconds

Prediction error for 80% data:  10.96%
Training time for 80% of data: 3.35 seconds

Prediction error for 90% data:  9.30%
Training time for 90% of data: 3.62 seconds

Prediction error for 100% data:  9.30%
Training time for 100% of data: 3.94 seconds

Prediction error for 10% data:  9.30%
Training time for 10% of data: 1.03 seconds

Prediction error for 20% data:  9.30%
Training time for 20% of data: 1.35 seconds

Prediction error for 30% data:  9.30%
Training time for 30% of data: 1.67 seconds

Prediction error for 40% data:  9.30%
Training time for 40% of data: 2.00 seconds

Training terminated due to time limit.
Standard Deviation of prediction errors: 0.048
Validating...
273 correct out of 301 (90.7%).
Testing...
131 correct out of 150 (87.3%).
```

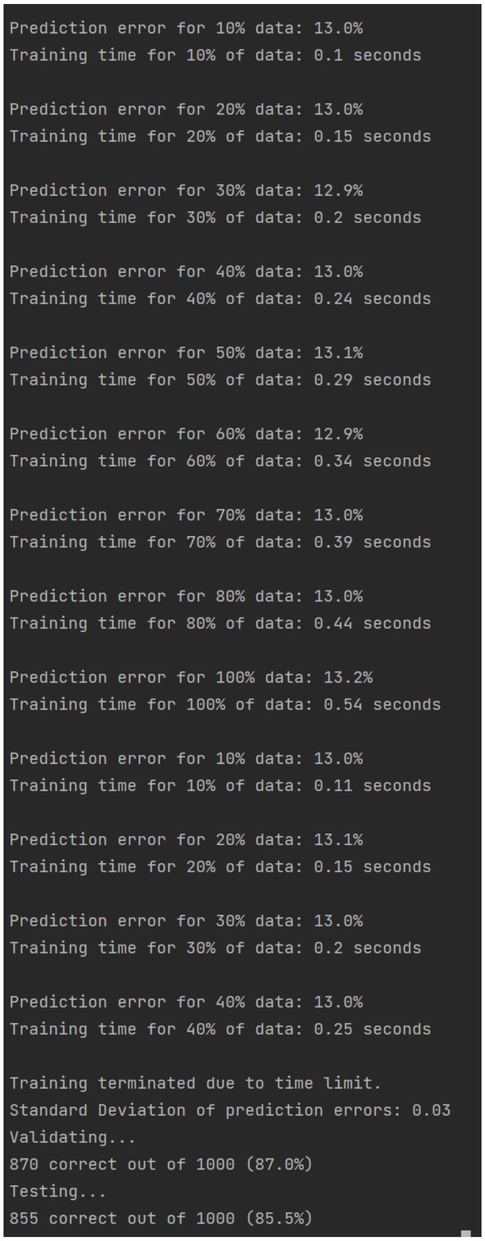
The ‘perceptron’ algorithm using ‘dataClassifier’, to run use the commands:

“python(3) dataClassifier.py -c nn -t 5000 -s 1000” : <u>for digits</u>
“python(3) dataClassifier.py -c nn -d faces -t 451” : <u>for faces</u>

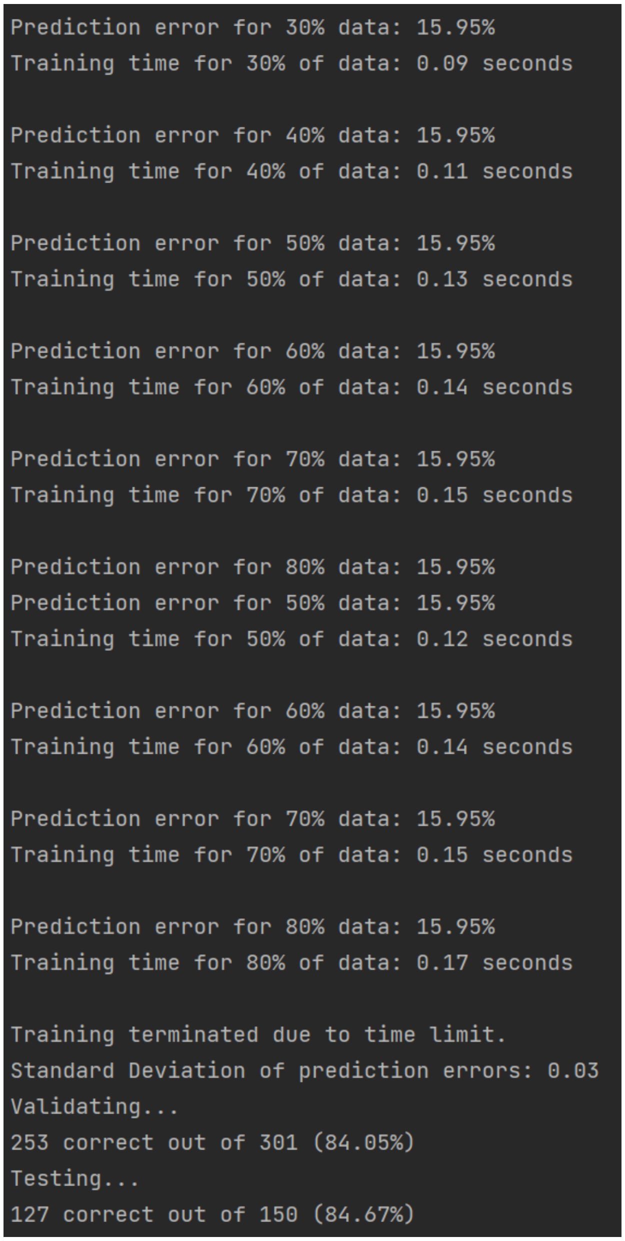
- The neural network uses a 1d vector composed of the raw pixels of each image. The pixels are represented as numerical values (0 if white, 1 if gray/black/edge) and are the inputs to the neural network.
- The training function runs for a time limit of 3 minutes, it trains on random samples of the training data. The samples are chosen in increasing sizes from 10%, 20%, …, 100% of the data. During training, the neural network adjusts its weights to minimize classification errors. This adjustment is done using backpropagation, where the network learns from the errors made in predictions by updating weights in the direction that decreases the prediction error.
- For each subset of the data, the neural network calculates prediction error, showing how accurate the neural network works in classifying the images. It also calculates training time for processing each subset, to show the efficiency of the program.
- After training, the neural network classifies new images using a method called forward propagation. Essentially, we take the dot product of the input vector and the trained weight vectors, and then pass it through the sigmoid function (we do this because we can take any value from -inf to inf and represent it as a value from 0 to 1).
- The neural network tests, computes, and displays its accuracy on separate validation and training data sets.

Like perceptron, the biggest challenge in implementing the neural network algorithm was reading the data. We used the framework of the neural network from a YouTube video (https://www.youtube.com/watch?v=9RN2Wr8xvro) but had to revise much of it because our input data was processed differently. The YouTube video used a 2d array of input data but our data was stored in dictionary (Counter) structures, so the way we processed our input data was just 1d. Knowing this, we had to revise substantial portions of the code from the video so that all data is suitable for the neural network processing including changing the logic for forward propagation and backwards propagation. All of these changes were tedious but necessary to enable the network to function with our dataset.

Time and Accuracy output for digits(left) and faces(right) as a function of % of data points used for training



```
Prediction error for 10% data: 13.0%
Training time for 10% of data: 0.1 seconds

Prediction error for 20% data: 13.0%
Training time for 20% of data: 0.15 seconds

Prediction error for 30% data: 12.9%
Training time for 30% of data: 0.2 seconds

Prediction error for 40% data: 13.0%
Training time for 40% of data: 0.24 seconds

Prediction error for 50% data: 13.1%
Training time for 50% of data: 0.29 seconds

Prediction error for 60% data: 12.9%
Training time for 60% of data: 0.34 seconds

Prediction error for 70% data: 13.0%
Training time for 70% of data: 0.39 seconds

Prediction error for 80% data: 13.0%
Training time for 80% of data: 0.44 seconds

Prediction error for 100% data: 13.2%
Training time for 100% of data: 0.54 seconds

Prediction error for 10% data: 13.0%
Training time for 10% of data: 0.11 seconds

Prediction error for 20% data: 13.1%
Training time for 20% of data: 0.15 seconds

Prediction error for 30% data: 13.0%
Training time for 30% of data: 0.2 seconds

Prediction error for 40% data: 13.0%
Training time for 40% of data: 0.25 seconds

Training terminated due to time limit.
Standard Deviation of prediction errors: 0.03
Validating...
870 correct out of 1000 (87.0%)
Testing...
855 correct out of 1000 (85.5%)
```



```
Prediction error for 30% data: 15.95%
Training time for 30% of data: 0.09 seconds

Prediction error for 40% data: 15.95%
Training time for 40% of data: 0.11 seconds

Prediction error for 50% data: 15.95%
Training time for 50% of data: 0.13 seconds

Prediction error for 60% data: 15.95%
Training time for 60% of data: 0.14 seconds

Prediction error for 70% data: 15.95%
Training time for 70% of data: 0.15 seconds

Prediction error for 80% data: 15.95%
Prediction error for 50% data: 15.95%
Training time for 50% of data: 0.12 seconds

Prediction error for 60% data: 15.95%
Training time for 60% of data: 0.14 seconds

Prediction error for 70% data: 15.95%
Training time for 70% of data: 0.15 seconds

Prediction error for 80% data: 15.95%
Training time for 80% of data: 0.17 seconds

Training terminated due to time limit.
Standard Deviation of prediction errors: 0.03
Validating...
253 correct out of 301 (84.05%)
Testing...
127 correct out of 150 (84.67%)
```