Prasidh Sriram, Akash Ram & Ahmed Aboukoura

Professor Rui Wang

Artificial Intelligence (198:440)

August 10, 2020

Group 27 Project 2: Face and Digit Classification

Part 0:

In this project, we were given two tasks:

1) Given an image containing a digit from 0-9, determine which digit is displayed.
2) Given an image, determine whether the image contains a face.

To accomplish these tasks, we implemented two different classification algorithms, called Perceptron and Naive-Bayes. Each classifier has its own benefits and drawbacks, and we will be analyzing them in detail in this report.

Part 1:

In our first implementation of feature extraction, we assumed each pixel could have only 2 values: on (black/gray) or off (empty). Once this assumption was made, then we could simply treat the entire image as a matrix of boolean values. This method worked, but yielded low accuracy rates during our initial testing phase. To improve it, we assigned a larger numerical value to the black pixels and a smaller value to the gray pixels. By doing this, the black pixels had a larger influence on the weights of different regions when the image was broken down. Additionally, we added code to count the number of regions of empty pixels. Once we implemented these upgrades, our accuracy rates rose dramatically for digit classification. We did not run into these problems for the face classification, so we chose to use the simple extraction for face classification.

Part 2:

Usage:

`python dataClassifier.py` - Executes driver module with default flags

Flags:

`-c [perceptron/naiveBayes]` - Choose which classifier to use

`-d [digits/faces]` - Choose which dataset to use

`-t [training_size]` - Choose the size of the training set (max 5000 for digits, max 450 for faces)

`-i [num_iterations]` - Choose the maximum number of iterations for the training algorithm

`-s [testing_size]` - Choose the size of the testing set (max 1000 for digits, max 150 for faces)

Our implementation of this project used the skeleton code provided by Berkeley. In order to run the code, the user must run dataClassifier.py, which is the driver module for both classifiers. Based on the command-line input passed to dataClassifier,

Unlike the Naive-Bayes classifier, a perceptron does not use probabilities to make its decisions. Instead, it generates a weight vector that is dependent on the density of black pixels on various parts of the image. As mentioned earlier, black pixels are given a higher weight so the perceptron favors regions with a high density of black/gray pixels. A perceptron is also known as a "binary classifier" because of how it always splits the image into two at each iteration, and then compares the two regions' pixel densities.

Naive Bayes, on the other hand, is based on probabilistic theory. It attempts to find a theory based on previous observations. The main difference between Naive Bayes and Perceptron is that in Naive Bayes, the training and testing data is separate from each other. However, in Perceptron the training data also serves the purpose of test data. As we found after

testing, this difference played a large role in how the classifiers performed, accuracy-wise and runtime-wise.

Part 3:

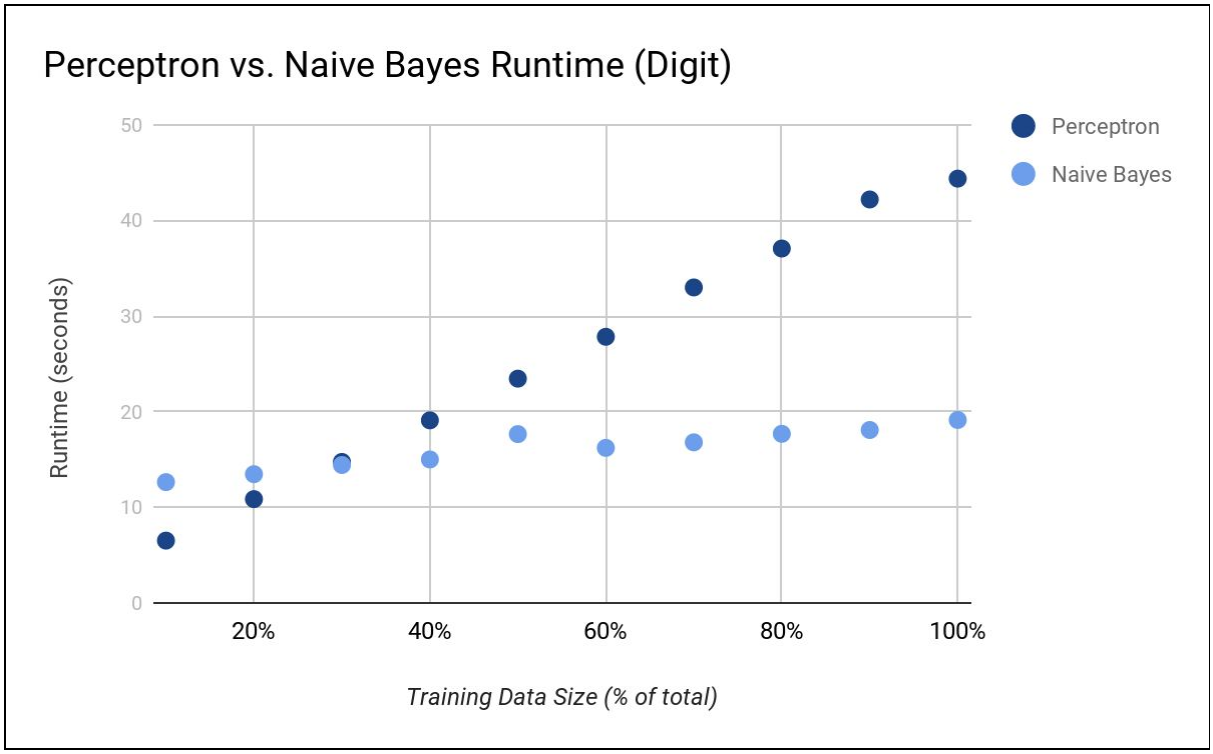
1. Perceptron (Digit)

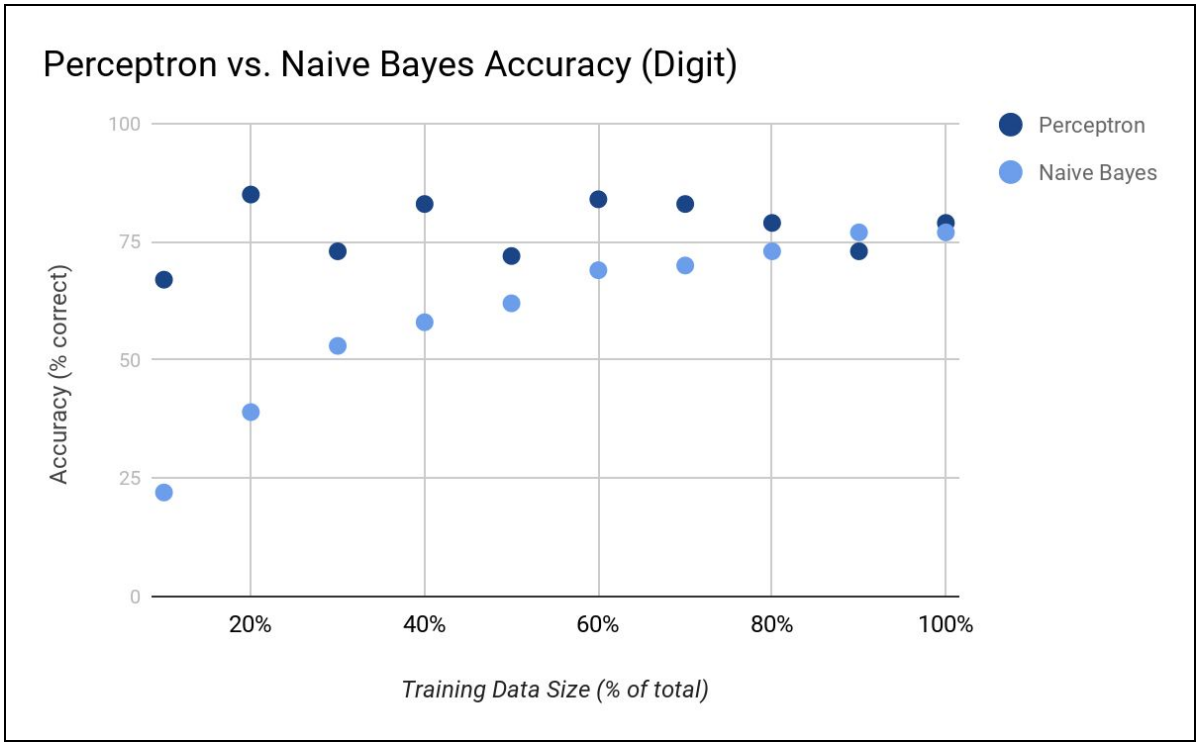
Percentage of Training Set (%)	Training Set Size (# images)	Accuracy of Test Output (%)	Total Runtime (s)
10%	500	67.0	6.52
20%	1000	85.0	10.86
30%	1500	73.0	14.75
40%	2000	83.0	19.10
50%	2500	72.0	23.46
60%	3000	84.0	27.84
70%	3500	83.0	33.00
80%	4000	79.0	37.07
90%	4500	73.0	42.20
100%	5000	79.0	44.38

2. Naive Bayes (Digit)

Percentage of Training Data (%)	Training Set Size (# of images)	Accuracy of Test Output (%)	Total Runtime (s)
10%	500	22.0	12.64
20%	1000	39.0	13.47
30%	1500	53.0	14.43
40%	2000	58.0	15.01
50%	2500	62.0	17.66
60%	3000	69.0	16.22
70%	3500	70.0	16.80

80%	4000	73.0	17.69
90%	4500	77.0	18.09
100%	5000	77.0	19.14



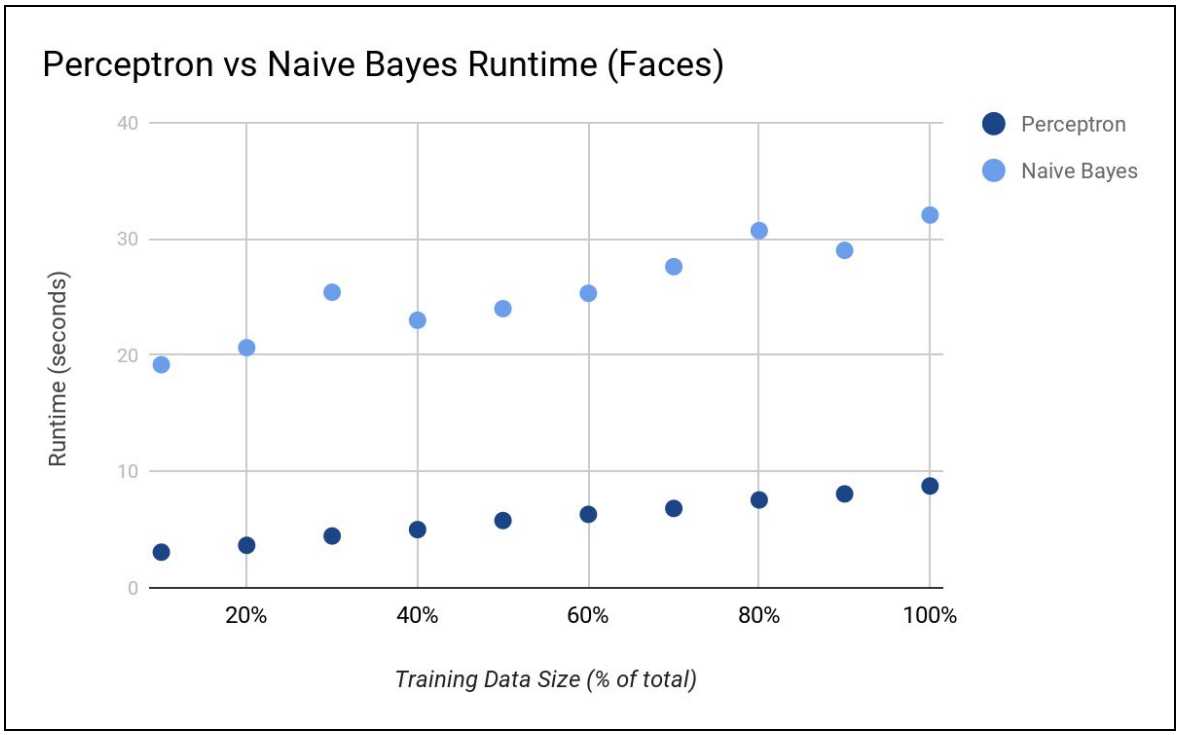


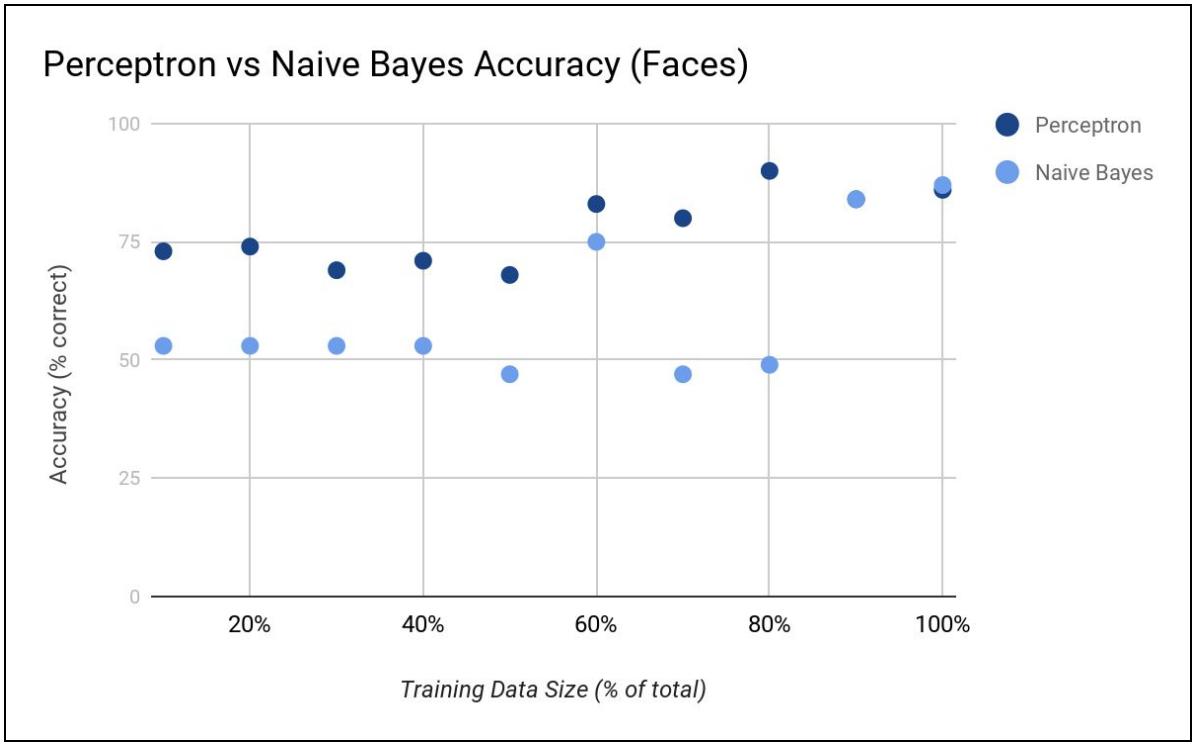
3. Perceptron (Face)

Percentage of Training Data (%)	Training Data Size (# of images)	Accuracy of Test Output (%)	Total Runtime (s)
10%	45	73.0	3.07
20%	90	74.0	3.66
30%	135	69.0	4.46
40%	180	71.0	5.01
50%	225	68.0	5.79
60%	270	83.0	6.32
70%	315	80.0	6.83
80%	360	90.0	7.56
90%	405	84.0	8.08
100%	450	86.0	8.76

4. Naive Bayes (Face)

Percentage of Training Data (%)	Training Data Size (# of images)	Accuracy of Test Output (%)	Total Runtime (s)
10%	45	53.0	19.19
20%	90	53.0	20.65
30%	135	53.0	25.42
40%	180	53.0	23.01
50%	225	47.0	24.01
60%	270	75.0	25.33
70%	315	47.0	27.62
80%	360	49.0	30.72
90%	405	84.0	29.02
100%	450	87.0	32.06


Perceptron vs Naive Bayes Runtime (Faces)
Perceptron
Naive Bayes
Runtime (seconds)
40
30
20
10
0
20%
40%
60%
80%
100%
Training Data Size (% of total)


Perceptron vs Naive Bayes Accuracy (Faces)
Perceptron
Naive Bayes
Accuracy (% correct)
100
75
50
25
0
20%
40%
60%
80%
100%
Training Data Size (% of total)

Part 4:

While obtaining our data, we noticed several differences between the algorithms observed. Our main observation was how the Perceptron and Naive Bayes algorithms reacted differently to the changes in the training set size.

While performing the classification, we observed that the runtime of Perceptron had a linear relationship with the size of the training set. Also, we observed that the runtime of Naive Bayes was linear as well but with a smaller slope. From this, we can conclude that Naive Bayes would be more efficient for large datasets. With regards to the accuracy of both algorithms, we found that Perceptron had a relatively high accuracy regardless of the training set size. However, the accuracy of Naive Bayes rose significantly as the training set size grew, and the Naive Bayes algorithm even beat Perceptron when the training set was at 100%. From this, we were able to infer that Perceptron delivers more consistent results, while Naive Bayes requires more training data to produce accurate predictions.

Using our data, we computed regressions for the runtime/accuracy of each classifier, where X represents the percentage of the training set used and Y represents seconds elapsed/% correct:

Classifier	Runtime (Digit)	Accuracy (Digit)	Runtime (Faces)	Accuracy (Faces)
Perceptron	Y = 0.44X + 1.99	Y = 0.04X + 75.6	Y = 0.06X + 2.5	Y = 0.2X + 66.67
Naive-Bayes	Y = 0.07X + 12.41	Y = 0.55X + 29.87	Y = 0.13X + 18.5	Y = 0.31X + 43

Part 5:

While working on this project, our group gained considerable insight in Machine Learning Algorithms. If our group was to repeat this project, there are a number of ways in which we could have improved our development and testing procedure:

1) All of our data points were computed with only a single iteration of the classifier for each data point. As a result, there were a number of data points that were outliers and did not make sense within the context of the data. We should have performed more trials for each point to produce more accurate results.
2) Linear regression was used for all the graphs displayed in the report. In reality, not all of these relationships are linear. Instead, we should have tried different regression models such as logarithmic regression, polynomial regression, or exponential regression to see which one best represents our data.
3) Our code was not fully optimized and the runtime could have been improved by reducing the number of redundant variables and data structures used.
4) While writing our code, we used Google Documents to share code, which was very inefficient. In the future, we should implement a more advanced version control system like GitHub to share code with each other seamlessly.