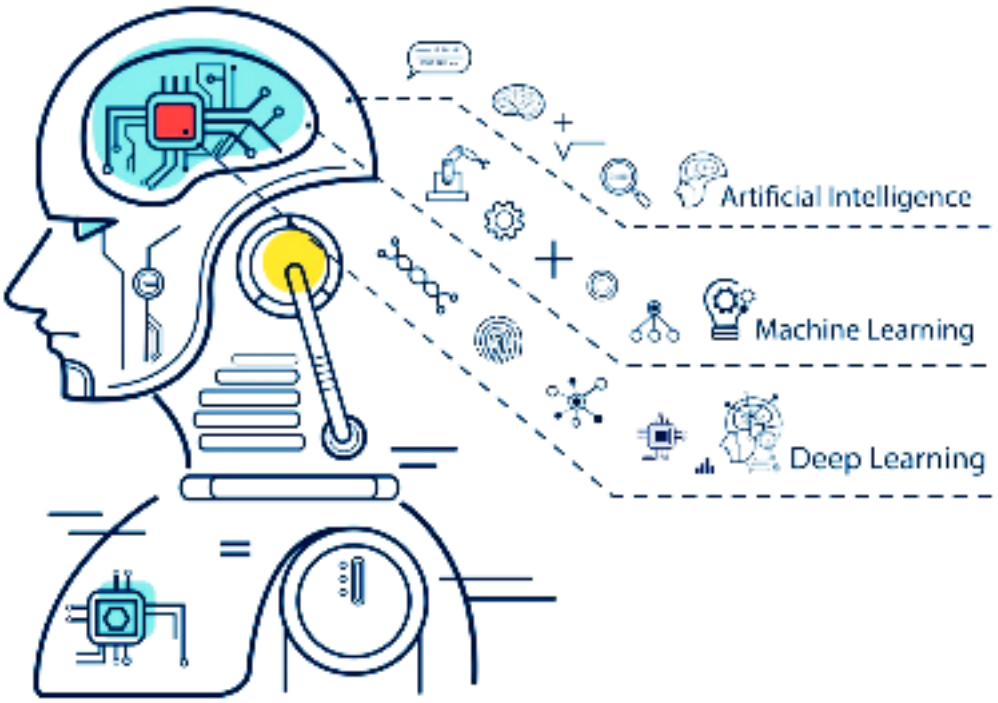
Introduction to Artificial Intelligence

CS440 - Final Assignment

Image Classification



Kevin Shah (kas665), Manav Patel (mpp124), and Ciera Fedell (clf124)

December 9, 2019

Features Used:

For every classifying algorithm, the image had to be broken down into features. These features give specific, individualized data for every section of the image. These features are then used to classify the images based on certain conditions and training.

In our case, we utilized a very simple feature extraction. We made each pixel into a feature, and the value of that pixel (0 if off, 1 if on) was used as the feature value. These features were then used to calculate and adjust weights as well as create conditional probability tables in the following algorithms.

Perceptron:

The perceptron algorithm is a simple, intuitive, but highly effective algorithm to classify things. In this case, it was used to detect faces and digits. The perceptron works by training the weights for each feature, for each label. The process of training the weights is the following: First the weights were initialized randomly to either a 0 or 1. This is because the features used took on those values and the computation of the weights depended on feature values. For each training image, it computed the expected label. This was done by multiplying each feature value by its corresponding weight value. This was done for each label. Once this dot product was computed, the max of all the labels was selected. This max label, was used as the expected label, the predicted label.

If the predicted label matched the actual label, nothing was done to the weights and it continued on to the next image. However, if the label did not match two things were done. The predicted, false, label had all of its corresponding weights and bias subtracted by their corresponding feature

values at each location. And the actual label had all of its weight values and bias incremented in the same manner. This intuitively makes sense because we are using the max values to assign labels. It would only make sense, to subtract from the wrongly predicted values and add to the actual weight vector.

Naive Bayes Classifier:

The Naive Bayes classifier essentially just uses the Bayes theorem. The theorem is as follows:

$$
\begin{aligned}
P(y|f_1,\ldots,f_m) &= \frac{P(f_1,\ldots,f_m|y)P(y)}{P(f_1,\ldots,f_m)} \\
&= \frac{P(y)\prod_{i=1}^{m} P(f_i|y)}{P(f_1,\ldots,f_m)} \\
\arg\max_y P(y|f_1,\ldots,f_m) &= \arg\max_y \frac{P(y)\prod_{i=1}^{m} P(f_i|y)}{P(f_1,\ldots,f_m)} \\
&= \arg\max_y P(y)\prod_{i=1}^{m} P(f_i|y)
\end{aligned}
$$

What we are essentially trying to find is the conditional probability of a particular label given an image. The algorithm first computes P(y) which is simply the number of times a given label occurs in the training label data divided by the total number of training label data provided. Finding the conditional probability was the tricky part.

Because the image is composed of a bunch of features, we have to find the product of the conditional probability of all features given a particular label. We trained the algorithm by simply creating a table for the probability of a feature occurring given a label has occurred. So for each label, we check just from the pile of that label, how many times a given feature value occurs at the particular location of the feature.

Finally, we needed to be able to predict for a given test datum. To increase our accuracy for the Bayes classifier, we chose to implement log probabilities into the equation to get rid of the underflow:

$$\begin{aligned}\arg\max_y \log P(y|f_1,\ldots,f_m) &= \arg\max_y \log P(y,f_1,\ldots,f_m) \\ &= \arg\max_y \left\{\log P(y) + \sum_{i=1}^{m} \log P(f_i|y)\right\}\end{aligned}$$

There was a problem though that if we have a probability of 0 when we look up in the training table, we would get inaccurate predictions and errors. So we just changed the value of the probability to a very small value (0.0001).

Mira:

Mira is similar to perceptron, but tracks and updates weight vectors of expected and true values based on the difference between the weights. It knows the expected and true labels from scanning the data for each instance and compares the two. As in perceptron, the variable y prime is taken as the maximum label of this set. If y and y prime, the two labels, do not match, they are updated according to the following equations:

$$w^{y} = w^{y} + \tau f$$

$$w^{y'} = w^{y'} - \tau f$$

$$\tau = min\{C, \frac{(w^{y'} - w^{y})f + 1}{2\|f\|_2^2}\}$$

Above, y is the true label, w^{y} is the weight of the true label, y prime is the expected label, and $w^{y'}$ is the weight of the expected label. C is a cap which is added to prevent too large of an update from happening at once, as may happen when encountering an outlier or bad data point. If y and y prime match, the weights are not updated and the next point is tested, repeating the same process.

Performance:

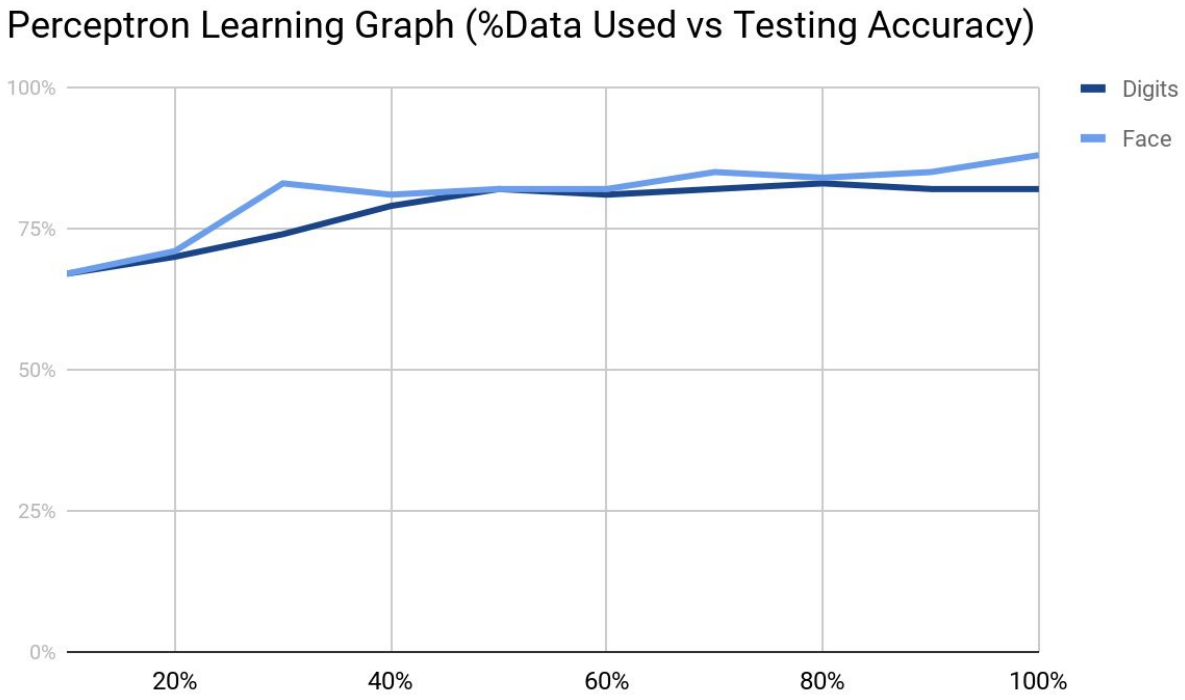
Data Utilized vs. Accuracy

Perceptron Classifier

Performance vs. Training Data

Perceptron-Training Learning Curve (With 3 Iterations of Training)

Percent Training Data Used	Tested Accuracy (Digits)	Tested Accuracy (Faces)	Std Dev (5 Iterations) (Digits)	Std Dev (5 Iterations) (Faces)
10%	67%	67%	7.11	3.84
20%	70%	71%	6.77	3.61
30%	74%	83%	6.54	3.51
40%	79%	81%	6.23	3.09
50%	82%	82%	5.98	2.87
60%	81%	82%	5.77	2.92
70%	82%	85%	5.69	2.61
80%	83%	84%	5.44	2.56
90%	82%	85%	5.12	2.44
100%	82%	88%	5.10	2.11

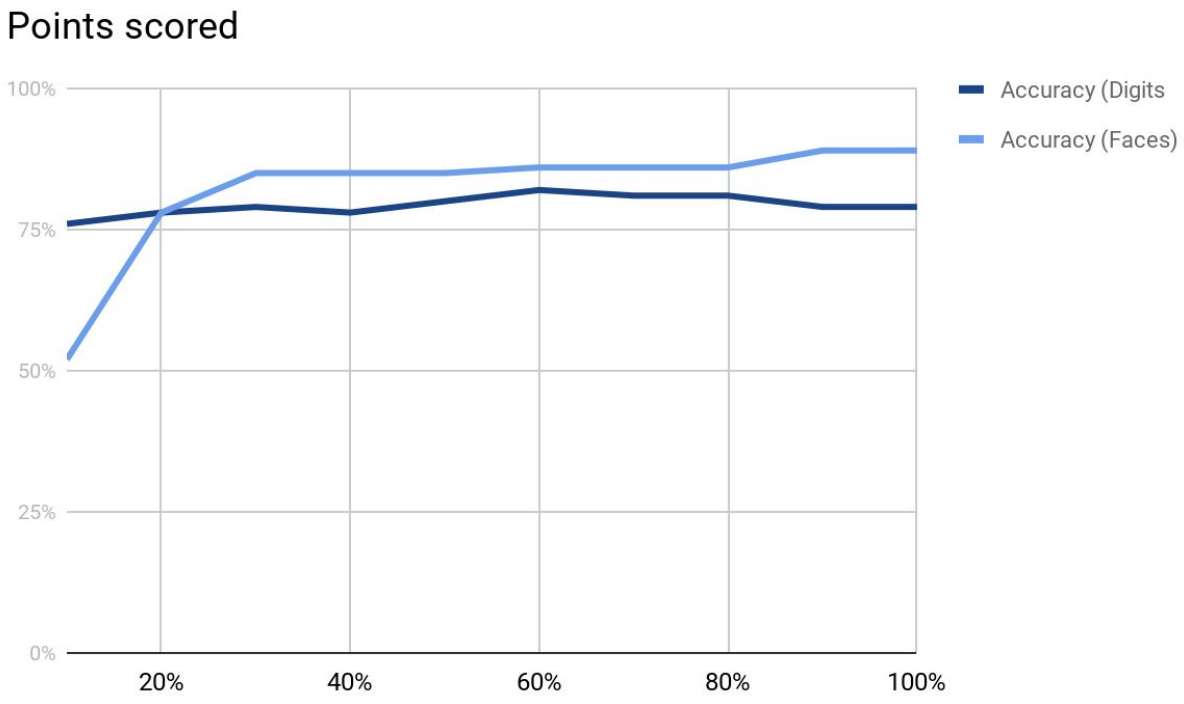


Naive Bayes Classifier

Performance vs. Training Data

Percent Training Data Used	Tested Accuracy (Digits)	Tested Accuracy (Faces)	Std Dev (5 Iterations) (Faces)
10%	76%	52%	5.62
20%	78%	78%	3.01
30%	79%	85%	2.74
40%	78%	85%	2.72

50%	80%	85%	2.74
60%	82%	86%	2.65
70%	81%	86%	2.68
80%	81%	86%	2.63
90%	79%	89%	2.43
100%	79%	89%	2.39

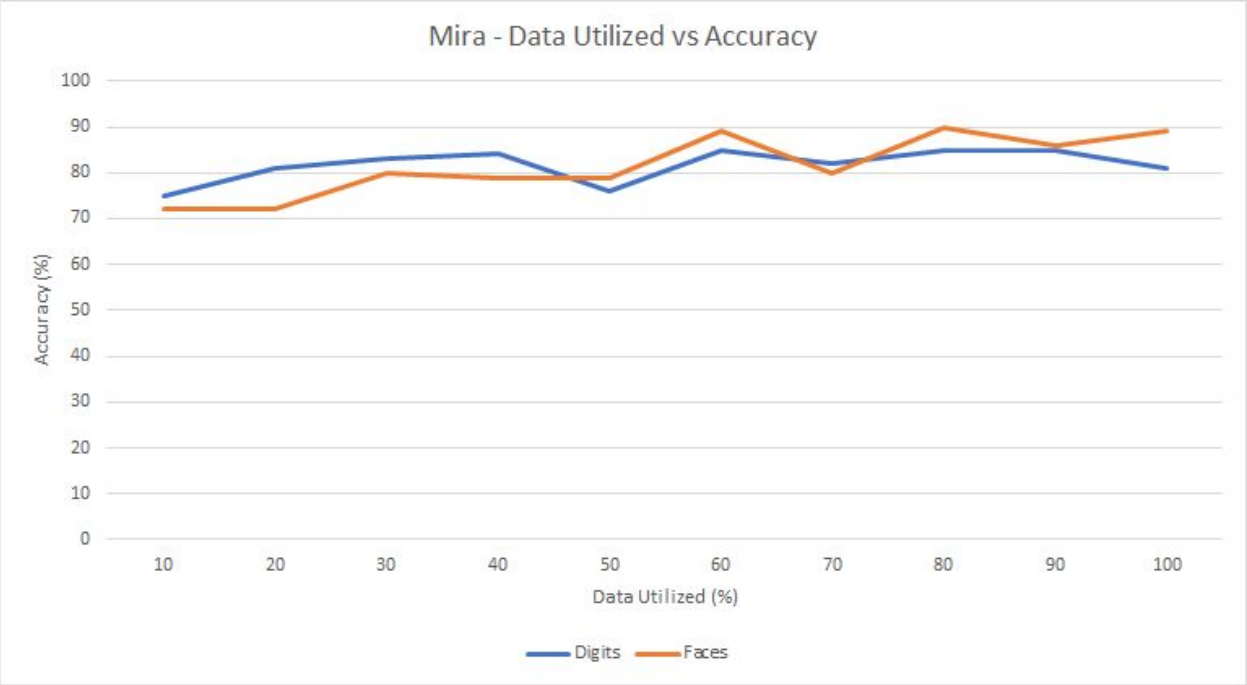


Mira Classifier

Mira - Data Utilized vs Accuracy

Percent Training	Tested Accuracy	Tested Accuracy	Std Dev (5

Data Used	(Digits)	(Faces)	Iterations) (Faces)
10%	75%	72%	3.54
20%	81%	72%	3.61
30%	83%	80%	2.87
40%	84%	79%	3.15
50%	76%	79%	3.12
60%	85%	89%	2.06
70%	82%	80%	2.84
80%	85%	90%	1.98
90%	85%	86%	2.31
100%	81%	89%	2.08

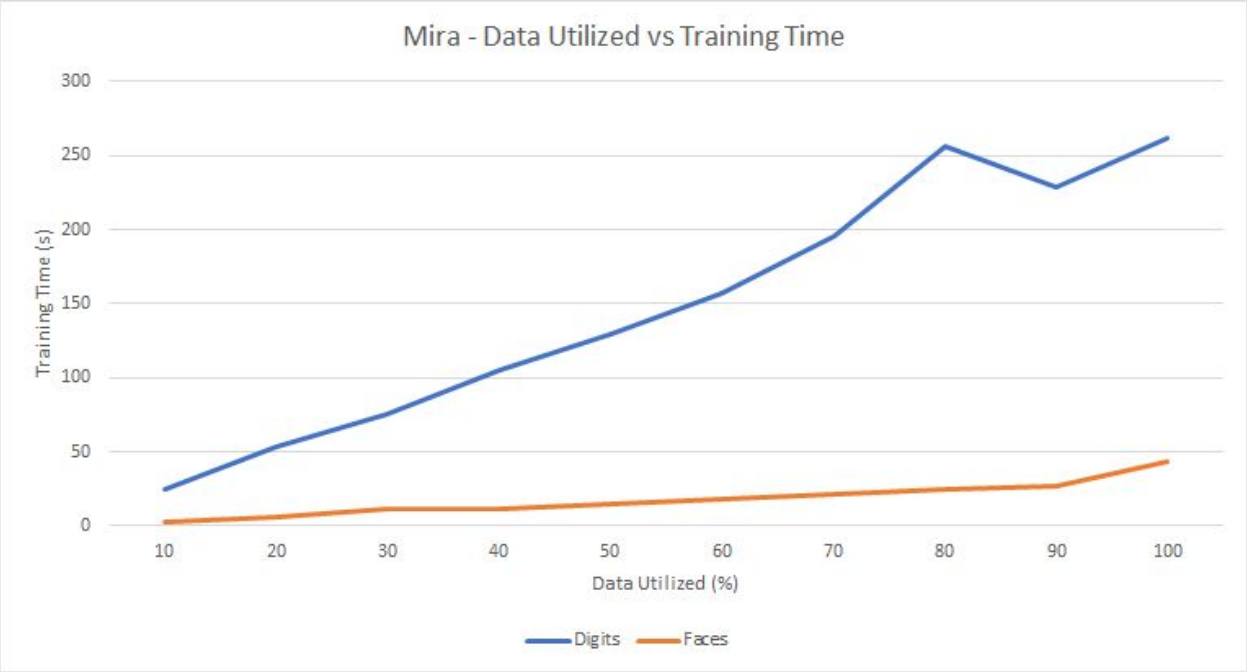


Training Time vs. Training Data

Mira - Data Utilized vs Training Time

Data Utilized	Digits - Training Time (s)	Faces - Training Time (s)
10%	24.69	3.25
20%	52.95	6.52
30%	75.24	11.86
40%	105.1	12.06
50%	129.34	15.08
60%	157.04	18.51
70%	195.29	21.75

80%	255.8	25.17
90%	229.13	26.94
100%	261.72	43.8



Prediction Error vs. Training Data

Mira - Data Utilized vs Prediction Error

Data Utilized	Digits - Prediction Error	Faces - Prediction Error
10%	25%	28%
20%	19%	28%
30%	17%	20%
40%	16%	29%

50%	24%	29%
60%	15%	11%
70%	18%	20%
80%	15%	10%
90%	15%	14%
100%	19%	11%

DATA-ANALYSIS:

PERCEPTRON:

For the perceptron, it was evident that the weights for the face-recognition were trained slightly faster than the training of digit-recognition. Also, the training-testing accuracies converged faster for the face recognition compared to digits. This was expected as recognizing something as a face or not is a binary decision to make, while digits had ten labels to choose from. Also standard deviation was better for digits than perceptron because it utilized a larger data set.

NAIVE BAYES:

Accuracy was of course better faces over digits because faces had binary features while the digit recognition had many more features. Automatically, with binary features, the lowest probability one should be getting is 50% which means automatically has the advantage.

MIRA:

Mira was able to train faces much more quickly than digits. While both training times increased linearly with the percentage of training data utilized,

training digits resulted in a much steeper slope for time increase. Faces showed more accuracy improvement based on the training size, although both were roughly linear again and were very similar in accuracy. It therefore follows that the prediction error decreased more overall for faces, though both showed the same downward trend with the amount of data used. Because there is much more data in the digits training set (5000) vs faces (451), it would be expected that given the same number of training points, faces would become reasonably more accurate than digits with the same algorithm.