Ethan Chang, ec778
Patrick Meng

# Project 2

CS 440
August 8, 2019

## Naive Bayes

The Naive Bayes Algorithm is implemented using a node structure which holds the original image, its associated label, and a feature vector which is calculated based on the algorithm and the type of object to be learned later on. Nodes were generated at random from the complete set of training images and training labels.

The random selection was designed to include duplicates, meaning that even if a node is already in the selected set, it may be chosen again. This is to emulate instances where digits may be extremely similar due to being the same font, or faces being extremely alike, and due to the minor details lost in the preprocessing. For testing purposes, the full distinct set was also tested and its results fell in line with that of the random selection process.

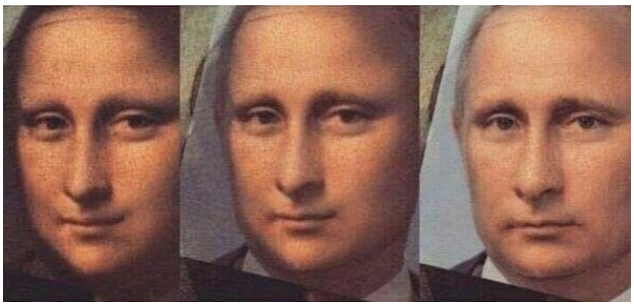


Figure 1: Two (supposedly) different people with similar faces

Training is done by computing the featureset for all selected training instances, which will be discussed in a later section. In this time frame the labels were indexed by counting all distinct values using a dictionary. Most of the compute-time is spent performing the prediction.

The prediction algorithm is based off of Bayes Rule where x is the image and y is the label:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

$p(x)$ is unknown during the prediction, so $p(y|x)$ is converted into a likelihood ratio:

$$L(x) = \frac{p(y|x)}{p(\neg y|x)} = \frac{p(x|y)p(y)}{p(x|\neg y)p(\neg y)}$$

$p(x|y)$ and $p(y)$ is still unknown, but can be estimated. $p(y)$ can be derived from the number of times the value for key y appears in our dictionary divided by the total amount of labels.

$p(x|y)$ is found in part by calculating the feature vector, $\phi(x)$, for all instances in the test set.

After calculating the vector values, $p(\phi_j(x) = v|y)$ is calculated for all j in the vector, then factored together to find $p(x|y)$. The denominator of $L(x)$ is found using the set complement of y. $L(x)$ is derived to be:

$$L(x) = \frac{(\prod_{j=1}^{l} p(\phi_j(x) = v|y))p(y)}{(\prod_{j=1}^{l} p(\phi_j(x) = v|\neg y))p(\neg y)}$$

This process is applied to all instances in the test set, where the instance is passed into the function, along with a hypothetical label that is iterated for all possible values to find the highest likelihood ratio. The training instance with the highest likelihood ratio's label is returned and compared with the test instance's actual label.

The predictions are not stored as it is not in the scope of this project, but an accuracy was computed. The algorithm was run 5 times for training rates in increments of 0.1 and the average accuracy and sample standard deviation was recorded.

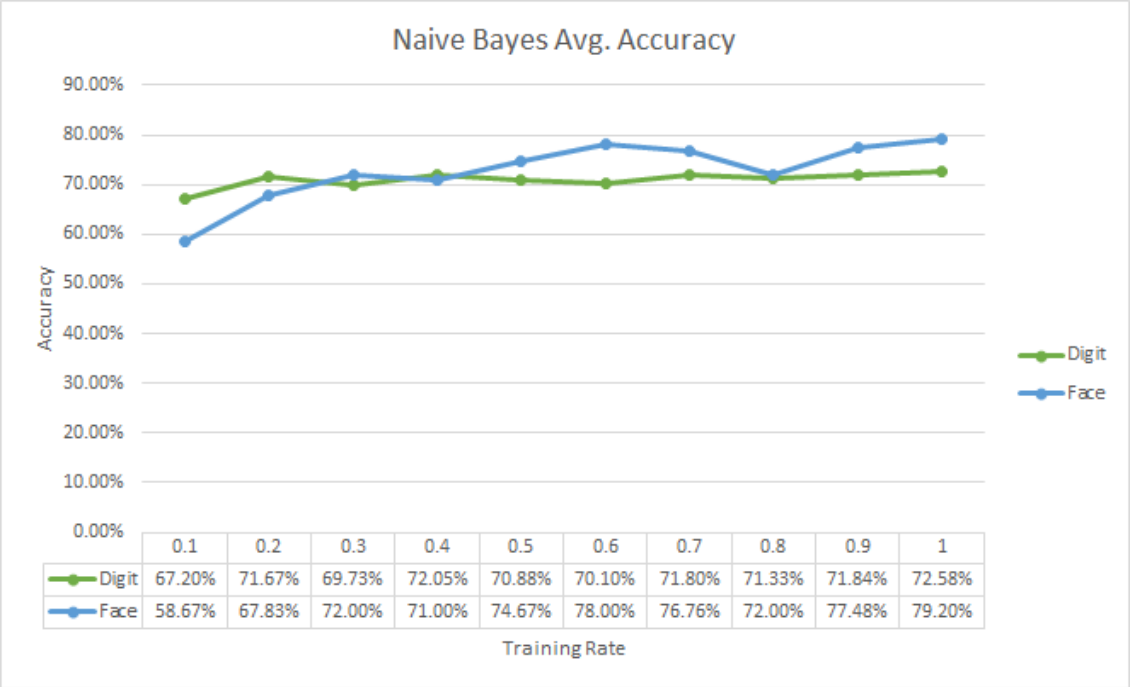
# Features

The first feature implemented was a regression slope that treated the image as a cartesian plane and the filled pixels as points. This slope regression resulted in 22% accuracy on digits and 55% on faces. Next, the gap between filled pixels was calculated per row and taken as a ratio of the amount of filled pixels. This is done for each non-blank row and averaged. Multiple gaps in the same row were aggregated. This process was repeated for columns as well. The combination of regression slope, horizontal gap, and vertical gap only amounted to 38% accuracy on digits, but was more effective on faces, guessing correctly 72% of the time. The regression line is theorized to be a good summary for all the points in a low cost number, while the two gaps were seen as values which were unique to each different digit.

Digits required a different feature vector than faces. A method where the image was scaled down to a 25% version was implemented, and each pixel in this scaled version was a feature. This was 70% accurate, but required more than 60 minutes to compute, as there were 49 features (7 x 7) in the set. Taking each row as a feature reduces the featureset size, but still retains its accuracy, therefore turning a 1 hour compute into a 3-5 minute compute time. The process of finding features was a loop of estimated guesses followed by trial and error.

# Results

The feature sets for digit and face recognition were different, which is why their respective accuracy is different. Even both feature sets were the same the data types are different enough that they may yield inherently different reactions. Accuracy did seem to rise marginally for face detection as it was trained on more data, however digit recognition stayed at around the same level for most of the training levels. One distinct note is that the accuracy for both datasets drop off significantly at 0.1 and lower, indicating that there is a threshold where simply feeding more data hits a point of diminishing returns. In this case, the algorithm is most likely bottleneck by its feature extraction.

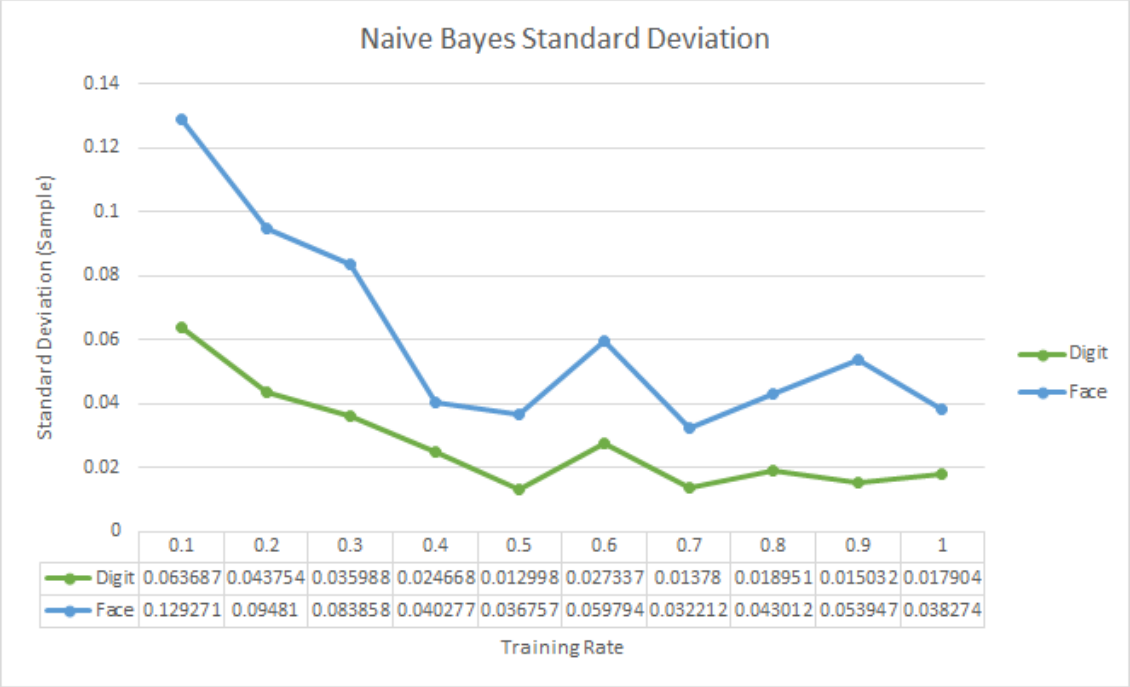



| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| Digit | 67.20% | 71.67% | 69.73% | 72.05% | 70.88% | 70.10% | 71.80% | 71.33% | 71.84% | 72.58% |
| Face | 58.67% | 67.83% | 72.00% | 71.00% | 74.67% | 78.00% | 76.76% | 72.00% | 77.48% | 79.20% |

Figure 2: Accuracy of predictions with different training rates

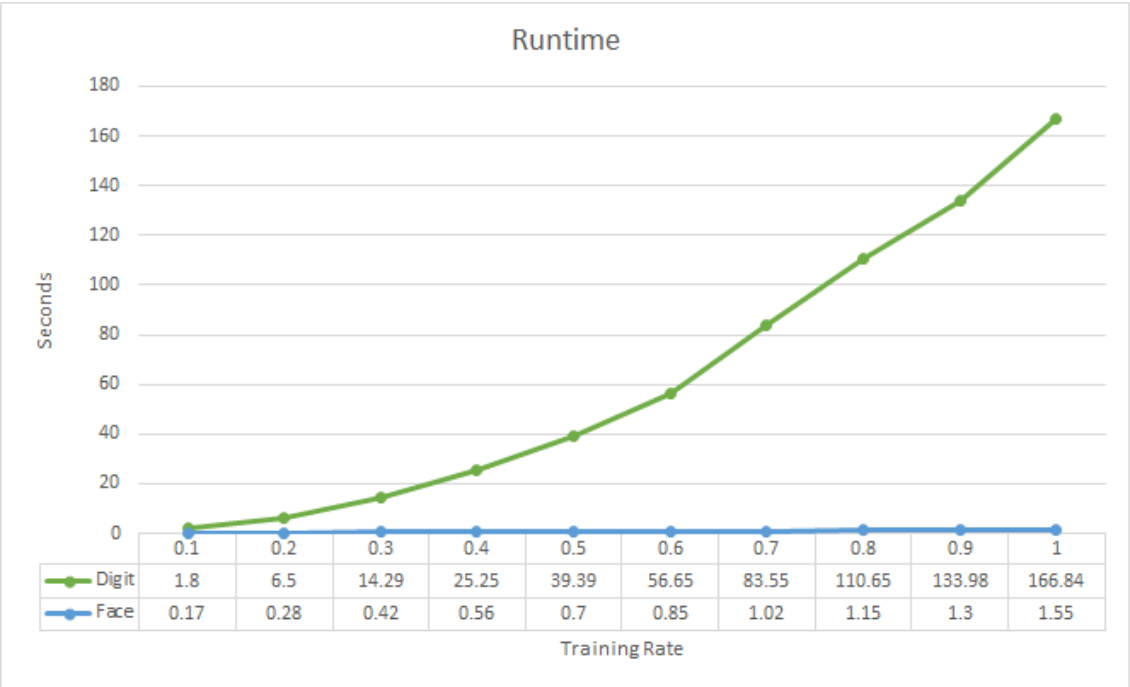
Standard deviation was inversely correlated with a rise in training rate for both datasets, but tapered off once a certain training rate was reached. It can be inferred that more training data leads to more consistent, or precise results, but not necessarily more accurate.

The runtime of both datasets differed vastly because their featureset, distinct labels, and training set size were different. The algorithm loops through all distinct labels and all features in the feature vector. Faces has 20% the distinct labels and 43% the features as digits, as well as 20% the training set size. The runtime to predict faces was negligable and closer to linear than digits, which resembled exponential runtime in the chart.




| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| Digit | 0.063687 | 0.043754 | 0.035988 | 0.024668 | 0.012998 | 0.027337 | 0.01378 | 0.018951 | 0.015032 | 0.017904 |
| Face | 0.129271 | 0.09481 | 0.083858 | 0.040277 | 0.036757 | 0.059794 | 0.032212 | 0.043012 | 0.053947 | 0.038274 |

Figure 3: Standard sample deviation with different training rates




| | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|
| Digit | 1.8 | 6.5 | 14.29 | 25.25 | 39.39 | 56.65 | 83.55 | 110.65 | 133.98 | 166.84 |
| Face | 0.17 | 0.28 | 0.42 | 0.56 | 0.7 | 0.85 | 1.02 | 1.15 | 1.3 | 1.55 |

Figure 4: Runtime as a function of training rates

# Project 2

## Perceptron Digits

For this algorithm we create lists of weights for each feature initialized to small random numbers. We then made a master list that contained the lists of weights for each number. During training we created equations multiplying each weight to its corresponding feature and then summing the results up.

$$equation = weight() * feature(0) + weight(1) * feature(1) + \ldots + weight(k) * feature(k)$$

Then the algorithm traverses through the equation finding the highest one. If this equation corresponds to the same digit as the label, then it is ignored. If they are different: For the correct label each feature is added to its corresponding weight. For the largest equation, each feature is subtracted by its corresponding weight. This step is repeated until either all of the equations found correspond to the correct label or until a set number of times it is run.

## Perceptron Faces

This version of Perceptron was implemented significantly differently from Digits. There was only one equation and the first weight served as a "y-intercept" and not multiplied with a feature.

$$equation = weight(0) + weight(1) * feature(1) + \ldots + weight(k) * feature(k)$$

If the equation is positive and the label is true or the equation is negative and the label is false, it is ignored. If it is negative and the label is true, the features are added to each weight. If it is positive and the label is false, the features are subtracted from each weight. This step is also repeated until either all equations pass or a set number of runs.

## Lessons Learned

The project taught us some valuable lessons, such as how both algorithms work, and the importance of writing efficient code in machine learning algorithms with large data sets. It also taught more abstract things such as time management and the importance of communication. The most impressionable lesson is that sometimes a certain feature will work really well and you don't know why (scaledown on digits, regression on faces), and more importantly sometimes a feature won't work well at all and you don't know why (scaledown on faces, regression on digits), but the most important thing is to keep trying different ideas.