Explanation - Colin Nies & Eric Zimmerman

Image Retrieval

Getting the images from the text files was the first challenge we encountered. After looking over the image sets I found out that all the digits images were no more than 20 pixels high and 28 pixels wide and the face images were no more than 70 pixels high and 60 pixels wide. While scanning a text file with digit images the program will ignore whitespace until it encounters a line with something in it besides whitespace and then grab the next 20 lines and convert it into a boolean 2D array that I call the pixels matrix. This method didn't exactly work for faces because the images were split up evenly every 70 lines. So grabbing face images just meant I had to convert every 70 lines into a 70 x 60 pixel matrix. Once the program grabs all the images it must also grab the associated labels which can easily be extracted line by line. I bundled the true label and pixels of an image together into my own Image object. The raw pixels of an image are the features we are using for classification. The classification algorithms use 2 ArrayLists of image objects corresponding to the training and testing sets. The classification algorithms also use an ArrayList of LabelData objects to hold important data on each label. So if you want to classify faces there will be 2 LabelData objects you need to work with, one for face and one for not-face.

Naive Bayes

To classify an image with NaiveBayes you must first create a NaiveBayes object which will train on the given training set. It trains in the initialize(...) method by iterating through all the training images keeping track of how many times each pixel is true for each label. This information is held in a 2D int array called trueFeatures which is a field of the LabelData object. Each label will have its own trueFeatures matrix whose size is the same as the pixel matrix. So for the sake of clarity, each element of the trueFeatures matrix indicates how many times the corresponding pixel is true for a particular label. The trueFeatures matrix of each label makes a lot of the necessary calculations easier. To classify an image the program must determine the most likely label for the image by iterating through each possible label and determining which one has the highest probability. I got how to determine the probability of a label from the berkeley resource the professor posted. The equation for the probability of label y can be written as log(P(Y)) + summation[i:m] {log (P(fi | y))} where P(Y) is the "prior distribution" of label y and P(fi | y) is the probability of feature fi given the label y. The prior distribution for a label is easy to compute since it is the the number of times the label shows up in the training images divided by the total number of training images. The probability of feature fi given label y can be computed by dividing the [number of times fi takes on the value it does in the image for each image in training set labeled as y] by [the number of times label y appears in the training data]. So to determine the error of classifying with naive bayes the program iterates through all the testing images, choosing the most likely label and incrementing a counter every time the classification is wrong. After iterating through all the images the number of failed classifications over the total number of training images is the percent error.

Perceptron

We implemented the Perceptron algorithm, computing the class with highest score as the predicted label for that data instance with $score(f,y) = \sum_i f_i w_i^y$. Using a weight matrix to give weight to each feature, given a feature list f, the perceptron computes the class y whose weight vector is most similar to the input vector f. We scan over the data, one instance at a time. When we come to an instance (f,y), we find the label with highest score:

$y' = \arg\max_{y''} score(f, y'')$. We then compare it to the true label, if equivalent we do nothing; if it is not equal we update the matrices accordingly w=w+f, w=w-f. This trains our data to prevent error in the future. To determine the error of classifying with Perceptron the program iterates through all the testing images, choosing the most likely label and incrementing a counter every time the classification is wrong. After iterating through all the images the number of failed classifications over the total number of training images is the percent error.

MIRA

The MIRA classification algorithm is very similar to the Perceptron algorithm, where We scan over the data, one instance at a time. When we come to an instance (f,y), we find the label with highest score:

$y' = \arg\max_{y''} score(f, y'')$. We then compare it to the true label, if equivalent we do nothing; if it is not equal we update the matrices (this is where MIRA classification differs) according to the function

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

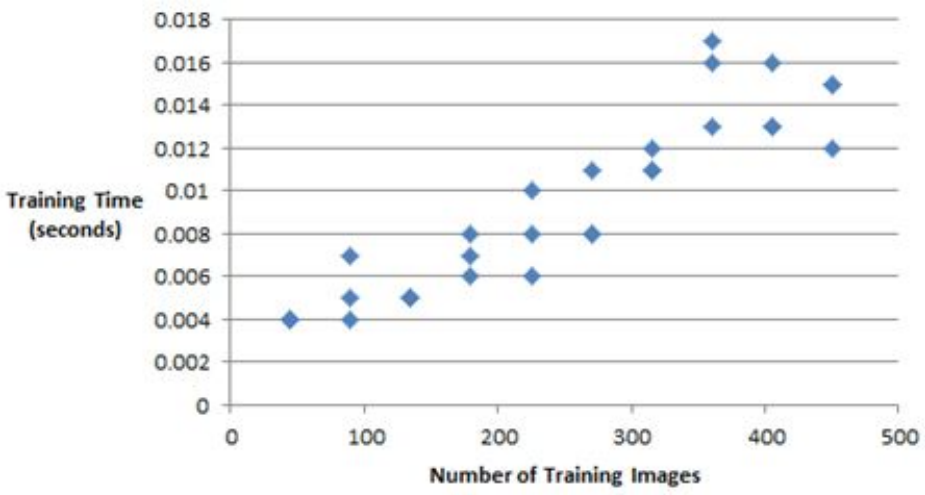
where $\tau = \min\left\{C, \frac{(w^{y'} - w^{y})f + 1}{2||f||_2^2}\right\}$. Every time we run through our training images, we alter our weight Matrix according to these algorithms to better classify future images. To determine the error of classifying with MIRA the program iterates through all the testing images, choosing the most likely label and incrementing a counter every time the classification is wrong. After iterating through all the images the number of failed classifications over the total number of training images is the percent error. The training times of MIRA and Perceptron are very similar because they both have the same time limit on updates.

Performance

This is the data of the time needed for training on face images with the Naïve Bayes Algorithm. I did 3 trials for each group of data points.

Data Points	Training Time (seconds)
45	0.004
45	0.004
45	0.004
90	0.007
90	0.005
90	0.004
135	0.005
135	0.005
135	0.005
180	0.007
180	0.008
180	0.006
225	0.008

225	0.01
225	0.006
270	0.008
270	0.011
270	0.008
315	0.011
315	0.011
315	0.012
360	0.016
360	0.017
360	0.013
405	0.016
405	0.013
405	0.013
451	0.015
451	0.015
451	0.012

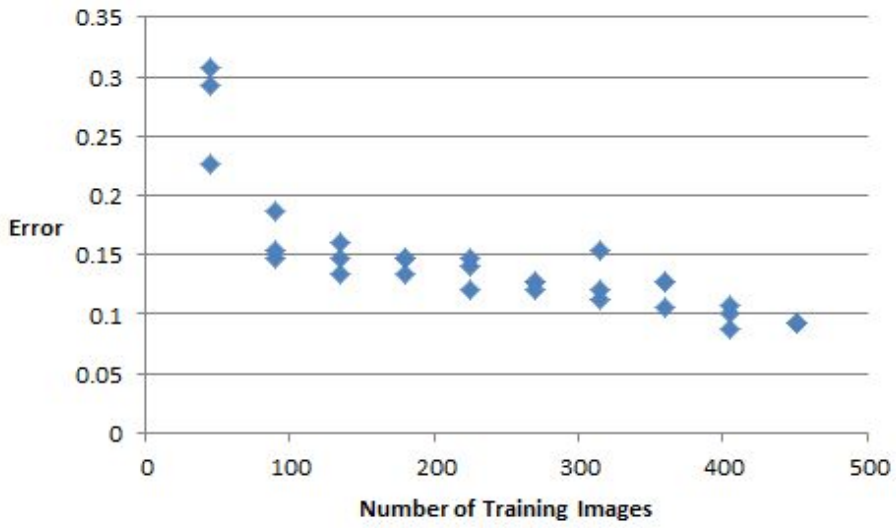


This is the data for the prediction error on faces in relation to the number of data points used.

Data Points	error
45	0.307
45	0.227
45	0.293
90	0.147
90	0.153
90	0.186
135	0.147
135	0.133

135	0.16
180	0.147
180	0.147
180	0.133
225	0.14
225	0.147
225	0.12
270	0.127
270	0.127
270	0.12
315	0.113
315	0.12
315	0.153
360	0.127
360	0.127
360	0.106
405	0.087
405	0.1
405	0.107
451	0.093
451	0.093
451	0.093

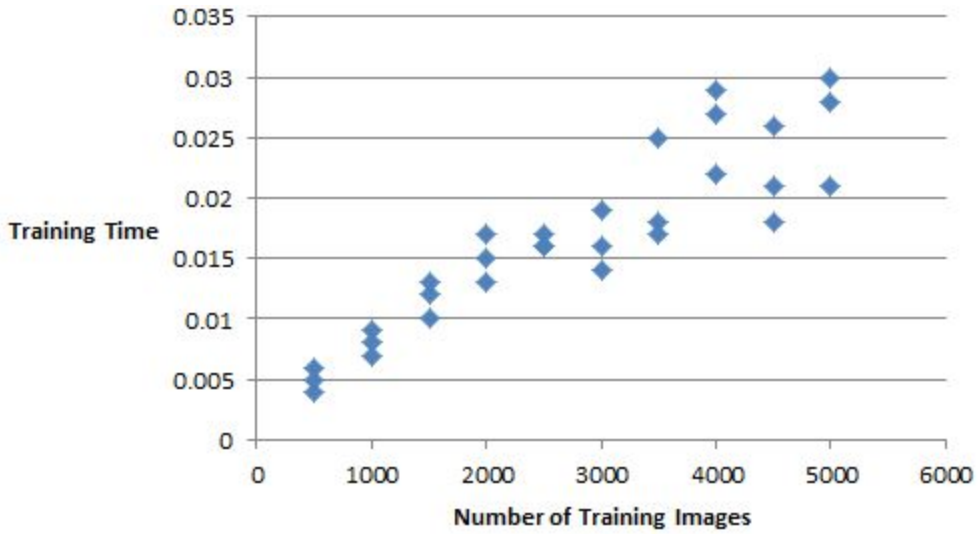
The standard deviation is 0.051774267



This is the data for the time needed to train on digit images with the Naïve Bayes Algorithm. I did 3 trials for each group of data points.

Data Points	Training Time (seconds)
500	0.004
500	0.006

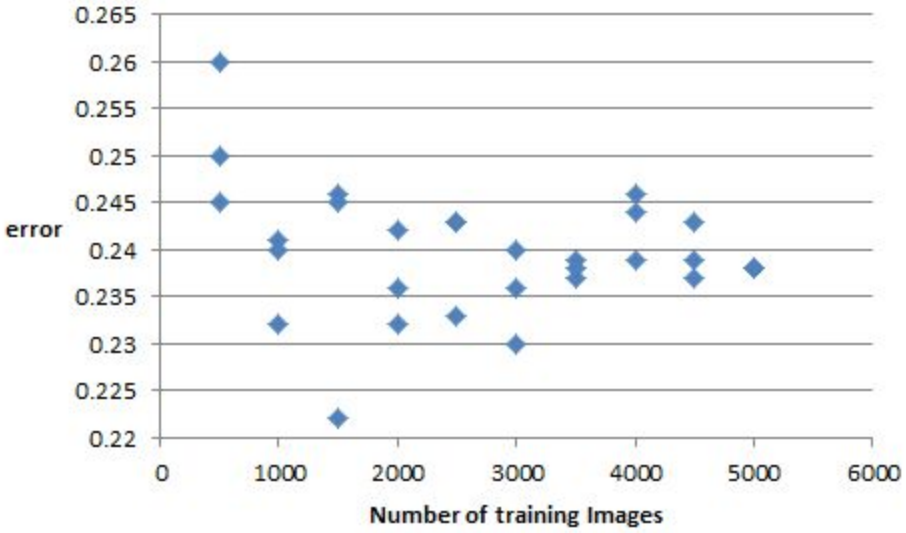
500	0.005
1000	0.009
1000	0.008
1000	0.007
1500	0.012
1500	0.013
1500	0.01
2000	0.017
2000	0.013
2000	0.015
2500	0.017
2500	0.016
2500	0.016
3000	0.019
3000	0.016
3000	0.014
3500	0.018
3500	0.025
3500	0.017
4000	0.027
4000	0.029
4000	0.022
4500	0.026
4500	0.021
4500	0.018
5000	0.028
5000	0.03
5000	0.021



This is the data for the prediction error on digits in relation to the number of data points used.

Data Points	error
500	0.25
500	0.245
500	0.26
1000	0.241
1000	0.232
1000	0.24
1500	0.246
1500	0.222
1500	0.245
2000	0.242
2000	0.232
2000	0.236
2500	0.243
2500	0.233
2500	0.243
3000	0.24
3000	0.23
3000	0.236
3500	0.237
3500	0.239
3500	0.238
4000	0.246
4000	0.239
4000	0.244
4500	0.239
4500	0.243
4500	0.237
5000	0.238
5000	0.238
5000	0.238

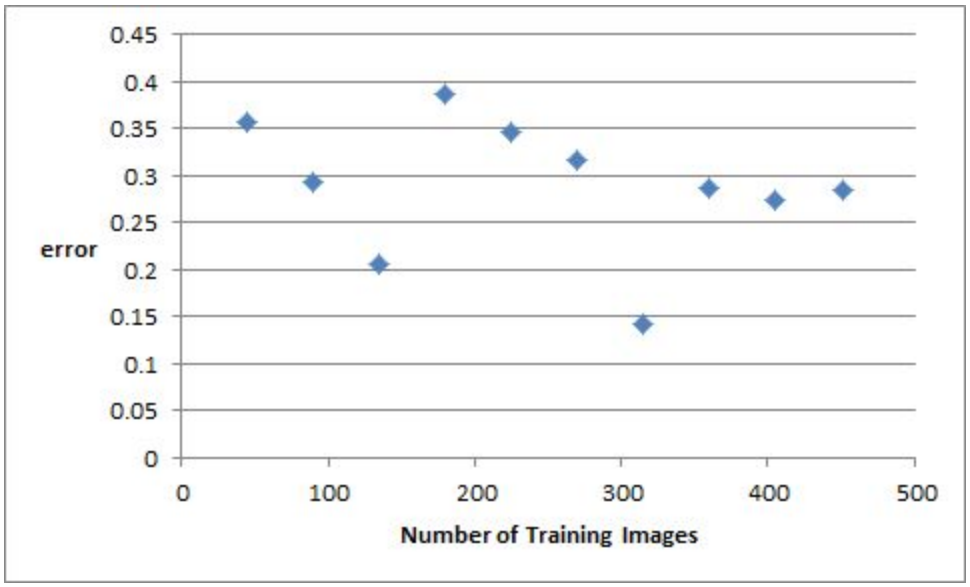
The standard deviation was 0.006792152



Training Times & Accuracy of Perceptron-Faces

Perceptron-Faces **Data Points**	**Training Time (s)**	**Accuracy/ Prediction Error @5s an update**
45	180s	.303
90	360s	.314
135	540s	.487
180	720s	.387
225	901s	.114
270	1080s	.387
315	1260s	.143
360	1440s	.287
405	1620s	.274
451	1805s	.285

Mean Accuracy of 70.19% across data sets or error of .2981



Training Times & Accuracy of Perceptron-Digits

Perceptron-Digits **Data Points**	**Training Time (s)**	**Accuracy/ Prediction Error @5s an update**
500	2500s	.461
1000	5002s	.3
1500	7504s	.314
2000	10006s	.338
2500	12515s	.303
3000	51298s	.291
3500	17500	.267

4000	20020s	.27
4500	22550s	.134
5000	25008s	.114

Mean Accuracy of 72.08% across data sets or error of .2792

Training Times & Accuracy of Mira-Faces

MIRA-Faces **Data Points**	**Training Time(s)**	**Accuracy/ Prediction Error @5s an update**
45	190s	.487
90	400s	.387
135	540s	.36
180	750s	.387
225	931s	.134
270	1180s	.387
315	1360s	.35
360	1480s	.114
405	1635s	.187
451	1895s	.16

Mean Accuracy of 70.47% across data sets or error of .2953

Training Times & Accuracy of MIRA-Digits

MIRA-Digits **Data Points**	**Training Time(s)**	**Accuracy/ Prediction Error @10s an update**
500	2550s	.487
1000	5102s	.314
1500	7574s	.238
2000	10106s	.32
2500	12564s	.314
3000	53298s	.287
3500	17500	.134
4000	20020s	.36

4500	22550s	.187
5000	25008s	.134

Mean Accuracy of 72.2% across data sets or error of .278