Kacper Murdzek

Intro to AI

Final Project

Face and Digit Classification

Overview:

In this project, we designed three classifiers: a Naive Bayes classifier, a Perceptron classifier, and a MIRA classifier. We tested these classifiers on two separate data sets, a set of scanned handwritten digit images and a set of face images in which the edges have already been detected.

Perceptron Algorithm

For our Perceptron Algorithm we used the distinct pixels as our features. The algorithm first initializes the weights for each feature associated with each label to 0. The algorithm can run multiple iterations of training the data, but we used three iterations as the basis. During training, the algorithm attempts to classify the training data with the weights that it knows up to that point in the training data. The classifier iterates through the legal labels and multiplies each weight associated with the label by the features of the data, this is the prediction that the data is that label, we store each prediction in an array and then return the max prediction along with its label. Once the label is returned it checks against the training label to distinguish if the classifier made the correct prediction. If the classifier was incorrect the weights of the predicted label are decreased by the training data features and the weights of the actual label are increased by the training data features. This demonstrates the following equations, $w^{y} = w^{y} + F$ and $w^{y'} = w^{y'} - F$. The more iterations the algorithm does and the more data the model trains on the more accurate the model becomes until it plateaus.

Naive Bayes Algorithm

For our Naive Bayes algorithm, we used each pixel as a distinct feature to start with. After, we initialized a probability array for each label where each array is initially filled with values 0.001, and the probability array is updated once we start iterating through our training data. We also have a counter dictionary that stores the amount of each label we have in our training data in which it's updated as we iterate through the training labels. We use both the probability array and the counter dictionary when calculating the joint probabilities for test images. Then, we iterate through the training data and set every index in the probability array for the respective training label, every time we encounter a feature value is 1 at an index we add 1/(count of that label in training data) to the probability array index where the feature was encountered. This sets the probability that a feature exists at an index given the label of the data. Now we have everything we need to start classifying, so we calculate the joint probabilities by iterating through all our labels and getting the probabilities that the test data is that label, which is just 1/(count of that label in training data), simultaneously we iterate through the test data and multiply each feature by the probability of that feature given its label which is the probability we stored in the probability array associated with the label. We sum the product of the probability of the label * the feature * the probability of that feature given its label and storing it in an array, we do this for every label. The label that has the max probability value in the array is the prediction we will make on the test data.

MIRA

The MIRA algorithm follows the same process as the Perceptron algorithm but attains a higher accuracy. The algorithm initializes the same exact way as the Perceptron algorithm does,

but the primary distinction is that if the classifier predicts incorrectly we scale the features by the minimum of the following:

$$\tau = min(0.001, \frac{(weights^{prediction} - weights^{actual}) * features + 1}{2(features)^2})$$

The algorithm scales the features of the guess and the features of the actual label by the variable τ. Once we have scaled our features we proceed to adjust the weights of the guess by subtracting the scaled features from the weights of the guess, and by adding the scaled features to the weights of the actual label. This could be described by the following equation: $w^y = w^y + \tau F$ and $w^{y'} = w^{y'} - \tau F$. The MIRA algorithm introduces a variable step size that keeps the weights from adjusting drastically after each incorrect prediction, as a result, we have better prediction accuracy.

Results:

Out of all three classifiers, the most accurate for both face and digits was the MIRA classifier. For faces, the average prediction error for MIRA was 19.4% as opposed to Perceptron(21.6%) and Naive Bayes(33.9%). While MIRA proved to be the most consistent in accuracy, producing a high of 87% accuracy, it retained the same run time as Perceptron and a higher run time compared to Naive Bayes. Both Perceptron and MIRA train the same way by performing multiple iterations of adjusting weights for the training labels in either a variable step size or a fixed step size, which explains the similar runtime while Naive Bayes generalizes with conditional probability. Generally, each classifier ran faster when identifying the set of face images, and overall for both data sets, Naive Bayes had the fastest run time and the other two performed at around the same speed (MIRA might be a couple of seconds slower than Perceptron but they are very close in comparison).

MIRA(Faces)	Training Data Amount	Prediction Accuracy (%)	Runtime	MIRA(Digits)	Training Data Amount	Prediction Accuracy (%)	Runtime
	45	77%	0:04		500	65%	0:14
	90	72%	0:06		1000	74%	0:27
	135	77%	0:07		1500	83%	0:38
	180	80%	0:09		2000	84%	0:50
	225	79%	0:11		2500	82%	1:00
	270	86%	0:13		3000	86%	1:14
	315	82%	0:14		3500	83%	1:29
	360	80%	0:15		4000	86%	1:44
	405	86%	0:17		4500	87%	1:59
	450	87%	0:19		5000	85%	2:14

Perceptron(Faces)	Training Data Amount	Prediction Accuracy (%)	Runtime	Perceptron(Digits)	Training Data Amount	Prediction Accuracy (%)	Runtime
	45	62%	0:04		500	75%	0:13
	90	76%	0:05		1000	75%	0:24
	135	72%	0:07		1500	75%	0:36
	180	81%	0:08		2000	82%	0:42
	225	79%	0:09		2500	85%	0:54
	270	87%	0:10		3000	85%	1:06
	315	80%	0:12		3500	83%	1:19
	360	80%	0:13		4000	82%	1:33
	405	82%	0:15		4500	85%	1:46
	450	85%	0:16		5000	81%	2:02

Naive Bayes(Faces)	Training Data Amount	Prediction Accuracy (%)	Runtime	Naive Bayes(Digits)	Training Data Amount	Prediction Accuracy (%)	Runtime
	45	59%	0:02		500	53%	0:01
	90	55%	0:03		1000	59%	0:02
	135	60%	0:04		1500	69%	0:03
	180	68%	0:05		2000	67%	0:04
	225	67%	0:06		2500	69%	0:04
	270	70%	0:06		3000	64%	0:05
	315	66%	0:06		3500	68%	0:05
	360	72%	0:07		4000	73%	0:06
	405	72%	0:07		4500	70%	0:06
	450	72%	0:07		5000	72%	0:06

Training Data at Random

Below is the data we derived from training a percentage of the training data at random for each classifier, and displayed is the average accuracy and standard deviation for each trial. The way this method of training works is described in the following way: start with 100% of the training data but make the model choose 10% of the training data at random, the model would train on this random 10% and then test on the testing data and return an accuracy. It repeats the above process for three iterations for each percentage of training data (10%, 20%, … 100%), after testing is concluded, for each percentage of training data the mean is computed of the three trials on that data as well as the standard deviation.

Perceptron(Faces) Training Data %	Standard Deviation	Mean Prediction %	Perceptron(Digits) Training Data %	Standard Deviation	Mean Prediction %
10%	3.74	78%	10%	4.02	76.66%
20%	1.63	79%	20%	3.55	83%
30%	2.62	85.66%	30%	3.39	84.66%
40%	2.05	83.33%	40%	2.16	82%
50%	0.94	85.66%	50%	0.94	85.33%

60%	0	85%	60%	2.45	82%
70%	0	85%	70%	3.29	84.66%
80%	0	85%	80%	2.62	84.33%
90%	0	85%	90%	0	85%
100%	0	85%	100%	1.24	85.33%

Naive Bayes(Faces) Training Data %	Standard Deviation	Mean Prediction %	Naive Bayes(Digits) Training Data %	Standard Deviation	Mean Prediction %
10%	10.2	67.33%	10%	5.71	50%
20%	8.95	59.33%	20%	8.52	53%
30%	11.31	56%	30%	1.63	55%
40%	3.29	49.33%	40%	9.84	57.66%
50%	2.82	49%	50%	6.18	63.66%
60%	5.88	77%	60%	4.32	65%
70%	6.54	79.33%	70%	6.16	64%
80%	6.59	76.66%	80%	3.26	66%
90%	9.1	61.33%	90%	5.18	71.66%
100%	6.59	71.66%	100%	2.62	70.66%

MIRA(Faces) Training Data %	Standard Deviation	Mean Prediction %	MIRA(Digits) Training Data %	Standard Deviation	Mean Prediction %
10%	2.94	77%	10%	1.41	75%
20%	4.49	83.33%	20%	5.55	84.66%
30%	1.24	87.33%	30%	2.62	85.66%
40%	0	89%	40%	0.47	85.33%
50%	0	89%	50%	0.94	85.33%
60%	0	89%	60%	0	86%
70%	0	89%	70%	0	86%
80%	0	89%	80%	0	86%
90%	0	89%	90%	0	86%
100%	0	89%	100%	0	86%

Conclusion:

Through our analysis, the MIRA algorithm performed the best when we trained data sequentially and even when we trained data at random. The Naive Bayes algorithm performed more poorly when training on random data and my hypothesis for this is because when training randomly there could be a bias, where one label gets trained more than the others since we are picking our training data at random and we could randomly pick more of one label than another. Perceptron is a very solid algorithm that produces a high accuracy when trained on random data and data in sequential order, all with very simple calculations. The MIRA algorithm takes the Perceptron algorithm a step further to increase accuracy and uses variable step sizes opposed to Perceptrons fixed step sizes for every incorrect prediction. Overall I was surprised with how accurate the MIRA algorithm was able to get without dropping in accuracy. If I were to take this project a step further I would try to train the MIRA model on more training data to see with how much data the accuracy would begin to plateau.