Nicholas Nasta
Arvind Vasudevan
CS440 Project 2

Project 2 Report: Image Classification

Overview:

Perceptron:

The Perceptron algorithm is a method for classifying inputs into predetermined categories, based on how closely weights calculated from training data match, and classify, live test data. The Perceptron is said to be a single layer neural network, meaning its decision making is based on the results of whether, or not the input training data satisfies the calculated set of booleans. Using Perceptron, for each image that is processed, the classifier parses every pixel in the image, line-by-line, and determines which calculated set of weights, which correspond to the expected label for the image, most closely correspond to the actual label.

Naive Bayes:

The Naive Bayes algorithm is a method for classifying inputs into independent categories, based on a calculated probability distribution. When using the Naive Bayes algorithm, it is assumed that all features are independent effects of the label, meaning they are mutually exclusive events. To classify a given image, the Naive Bayes algorithm finds which label is most probable, given the feature values for each pixel. To prevent underflow, logarithms are used to represent the probabilities in computations, but this was unnecessary, and inefficient, in our implementation. Additional parameters in this algorithm come from the prior distribution of over labels(P(Y)), and the conditional probabilities of the features, given each label ($F_i | Y = y$), which are estimated. These estimates can be refined to better fit the system through a process called Laplace Smoothing. In our implementation, we experienced difficulty when implementing Laplace smoothing. The smoothed estimate values generally resulted in a reduction in accuracy, so we could not maintain our current accuracy levels while also implementing smoothing.

Implementation Details:

Feature Design:

Our approach to feature design was based in checking adjacent pixels. Our initial thought was to assign weights based on checking 2 pixels at a time from each image. The thought was that if the adjacent pixel is colored, and the present picture is colored, it would receive a higher weight(2). If the present pixel is colored and the adjacent pixel is not, or if the present pixel is not colored, and the adjacent pixel is colored, the pixel receives a middling weight(1). If neither the current pixel or the adjacent pixel are colored, the pixel receives a low weight(0). In doing so, we hoped to create a kind of weight gradient, where unpopulated sections have no weight, and heavily populated sections have higher weights, with the transition parts being weighted in between. This actually resulted in a noticeable drop in accuracy for faces, and a slight drop in accuracy for digits.

This may be because of the nature of the inputs. With faces, if a section is more heavily populated with pixels, it does not necessarily confirm or deny that an image is a face. In the case of the digits data, while pixel density in specific sections may be common among images with the same label, the vertical orientation is not necessarily the same, meaning that the images are not necessarily parallel to the y-axis. This can cause problems when just checking horizontal occupancy, similar occupancy patterns can correspond to different labels. Examples of these are “1” and “7”, or “6” and “8”, respectively.

This led us to abandon the idea of adjacency, in favor of the concept of continuity. Rather than checking adjacency just vertically, we also check diagonally and horizontally, to see if the present pixel is colored, and also neighboring any colored pixel. In doing so, the feature extractor looks for non-disrupted, continuous chains of colored pixels, and sets the weight to 5 any time it perceives the current pixel to be part of a continuous chain. Functionally, the feature is designed to assign higher weights to the pixels in the image that form lines. This makes more sense in the context of faces, where non-specific continuous line are more indicative of the image being a face.

In the case of digits, our feature looked for pixel density rather than continuity. The idea was that there is more of an emphasis on pixels being dense and continuous in specific regions, than just being continuous. This is to say that as the classifier progresses to a given pixel, it checks the 8 pixels immediately surrounding it. If all the surrounding pixels are colored, and the pixel is colored, the weight is changed to 5. If the pixel is colored, and is surrounded by any uncolored pixels, the weight is set to 1. Otherwise, the weight is set to 0. In doing so, The classifier sets the weights of the heavily populated regions far higher than the unpopulated sections. The transition areas have a weight of 1, and effectively creates a gradient of weights across the regions corresponding to the images colored pixels.

Data:

Percent of Training Data Used	Perceptron Accuracy (Out of 149 Test Images)	Standard Deviation For Perceptron	Naive Bayes Accuracy (Out of 149 Test Images)	Standard Deviation Naive Bayes
10%	112	18	76	12
20%	114	18	75	12
30%	126	20	75	12
40%	127	20	75	12
50%	130	21	75	12
60%	127	20	75	12
70%	124	20	75	12
80%	128	21	75	12
90%	125	20	75	12
100%	130	21	75	12

Observations:

When classifying the Faces data set, we observed that the Perceptron algorithm was more accurate than Naive Bayes. This may be because the Perceptron is designed to use its feature weights to distinguish between mutually exclusive groups. As a linear classifier, Perceptron essentially models the lower bound of acceptable values. Meaning, the Perceptron value function, for the faces set, models the minimum required set of features for an image to qualify as a face.

Digits:

Percent of Training Data Used	Naive Bayes Digits Accuracy (Out of 1000 Test Images)	Standard Deviation Naive Bayes	Perceptron Digits Accuracy (Out of 1000 Test Images)	Standard Deviation Pereptron
10	684	40	753	47
20	714	45	744	47
30	731	46	781	49
40	745	47	762	48
50	740	46	786	49
60	750	47	796	50
70	745	47	800	50
80	757	47	789	49
90	753	47	804	50
100	753	47	819	51

Observations:

When classifying digits, we saw that the Perceptron algorithm was more accurate than the Naive Bayes algorithm. We were rather surprised by this, as we expected Naive Bayes, which accounts for multiple independent events in a probability distribution, to be more accurate. A reason for this unexpected outcome may be

Project 2 Report: Image Classification

that it is assumed that all features are independent effects of the label. This assumption could be false, if features are vague and non descriptive enough to correspond to multiple expected labels. In this scenario,

Run Time:

Percent of Training Data Used	Runtime of Naive Bayes Digits (In seconds)	Runtime of Perceptron Digits (In seconds)	Runtime of Naive Bayes Faces (In seconds)	Runtime of Perceptron Faces (In seconds)
10	32	382	44	553
20	41	398	64	568
30	44	414	84	586
40	50	429	104	606
50	60	445	130	633
60	59	462	148	664
70	62	480	167	698
80	65	498	185	773
90	67	518	225	770
100	71	538	246	810

Observations:

Perceptron has a consistently longer runtime than the Naive Bayes Algorithm. This is because of the inherent design of the algorithms. Perceptron requires multiple iterations through the data set, so as to optimize the weights. Naive Bayes only requires one one iteration to calculate the conditional probabilities.