Programming Assignment 1

Multiclass Logistic Regression
Hayleigh Sanders - 661195735

Introduction

Multiclass logistic regression is a machine learning technique that seeks to classify a vector of input data as one of a number of discrete classes. The following project explores the use of multiclass logistic expression to classify images of handwritten digits from one to five using gradient descent to learn the parameters for each pixel.

Theory

For each vector of training data D = {X[m], y[m]} where m=1,2...M and multiple classes y $\in$ {1,2...K}, the parameters θ can be found by minimizing the negative conditional log likelihood given by,

$$L(D:\theta) = -\sum_{m=1}^{M}\sum_{k=1}^{K} t[m][k]\log\sigma_M(\theta^t{}_k X[m]) \quad (1)$$

Where σ_M is the softmax function given by,

$$\sigma_M(\theta^t{}_k X[m]) = \frac{exp(\theta^t{}_k X[m])}{\sum_{k'=1}^{K} exp(\theta^t{}_{k'} X[m])} \quad (2)$$

And t[m][k] is the one-of-k 5x1 encoded vector of the output with the index value equal to 1 and the other entries equal to zero, which corresponds to the correct class of the image vector. Given a Mx1 data vector and MxK matrix of parameters θ, where M is the length of the data vector and K is the number of classes, the softmax function will yield a Kx1 vector of the predicted probability that the data vector belongs to each class. The theta values can be solved iteratively through gradient descent, given in the following:

$$\theta_k{}^{t+1} = \theta_k{}^{t+1} - \eta\left[\nabla_{\theta_k} L(D:\theta)\right] \quad (3)$$

Where η is the learning rate and the gradient is equal to,

$$\nabla_{\theta_k} L(D:\theta) = -\sum_{m=1}^{M}\left\{t[m][k] - \left(\sigma_M X^t[m]\theta^k\right)\right\}X[m] \quad (4)$$

This is because the loss function is convex and differentiable, so a minimum point corresponding to the parameter value with the least error can be found by moving a tiny amount in the negative gradient direction after a number of iterations.

Experimental Results

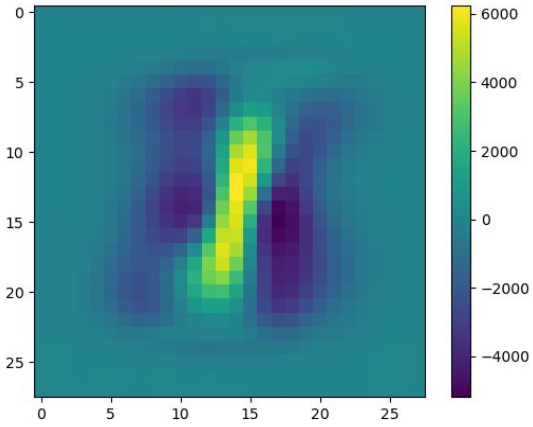
Given 25112 28x28 pixel training images of handwritten digits from 1 to 5, each image was flattened into a 1x784 vector, with 1 added to the end to make it a 1x785 vector so that the w0 bias values can be calculated. The pixels were normalized to [0,1] by dividing each by 255. Each corresponding output y was converted into a 5x1 binary one-of-k vector t, with the correct class being the index and the other values zero. A 785x5 weight matrix Θ was initialized to Gaussian random values (mean = 0, standard deviation = 1). Each of the 784 pixels in the data vector has five weights associated with each class.

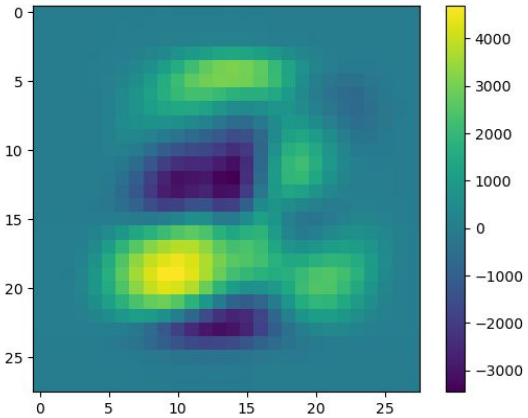
The gradient descent algorithm was performed on all 25112 data vectors in the set. The learning rate was set to .05 after trial and error. The weights for each class θ_k were computed with the following procedure:

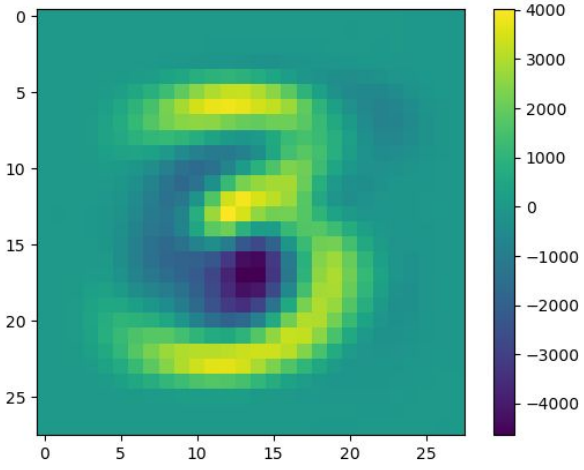
For each θ_k:
    For N iterations:
        For each data vector X[m] in the training data set:
            Compute gradient sum from equation (4)
        Update theta with equation (3)

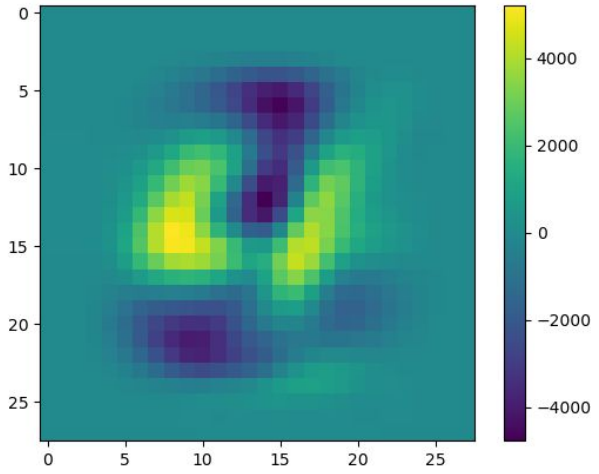
The accuracy was evaluated using the learned weights on each of the 4982 test data vectors with the softmax equation (2). The index of the largest value of the vector produced by the softmax function is equivalent to the predicted class. Total accuracy was calculated to be the number of correctly classified images over the total number of images. Accuracy for each digit was calculated to be the total number of correctly classified instances of the digit over the total number of instances of the digit. The error was calculated to be 1 - percent accuracy.

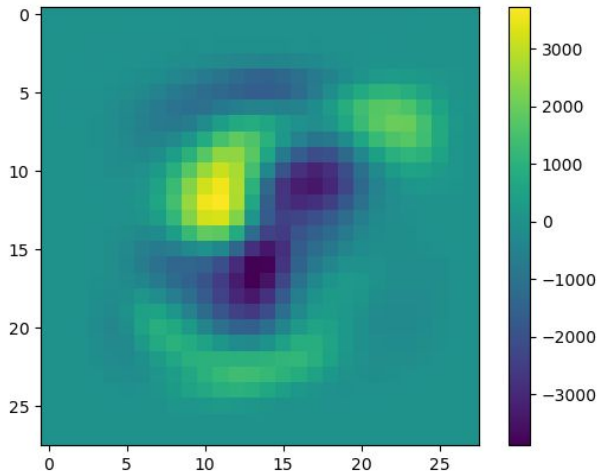
The trained weights were converted to an image and plotted for each class, shown below from one to five in consecutive order:


0
5
10
15
20
25
0
5
10
15
20
25
6000
4000
2000
0
−2000
−4000


0
5
10
15
20
25
0
5
10
15
20
25
4000
3000
2000
1000
0
−1000
−2000
−3000

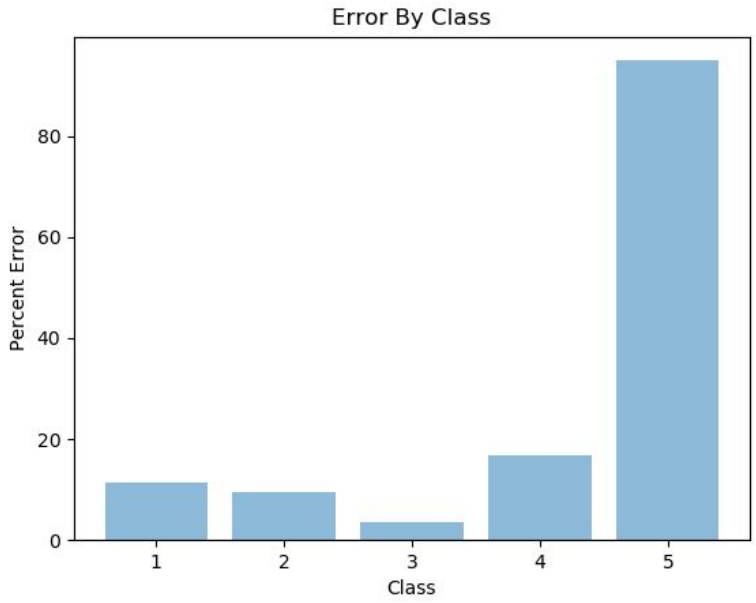

0
5
10
15
20
25
0
5
10
15
20
25
4000
3000
2000
1000
0
−1000
−2000
−3000
−4000





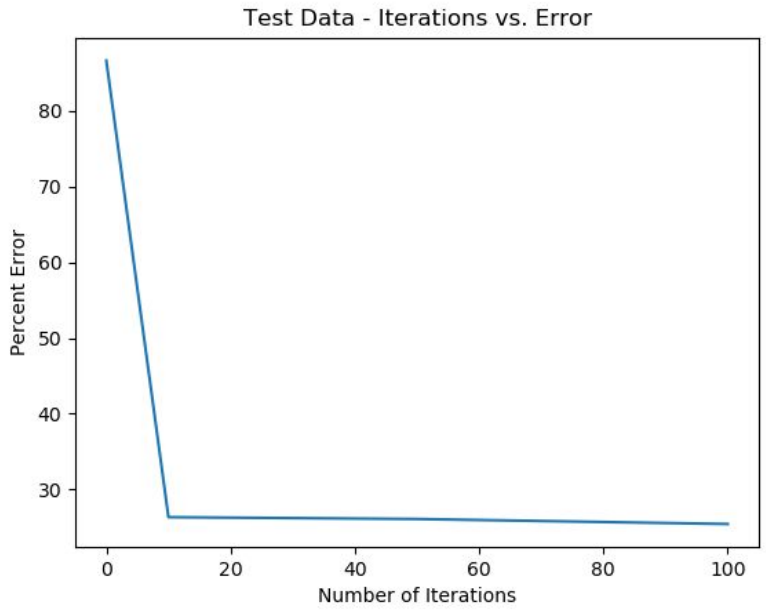
The weights all resemble their respective classes, with the most positive values falling along the structure of the correct digit.

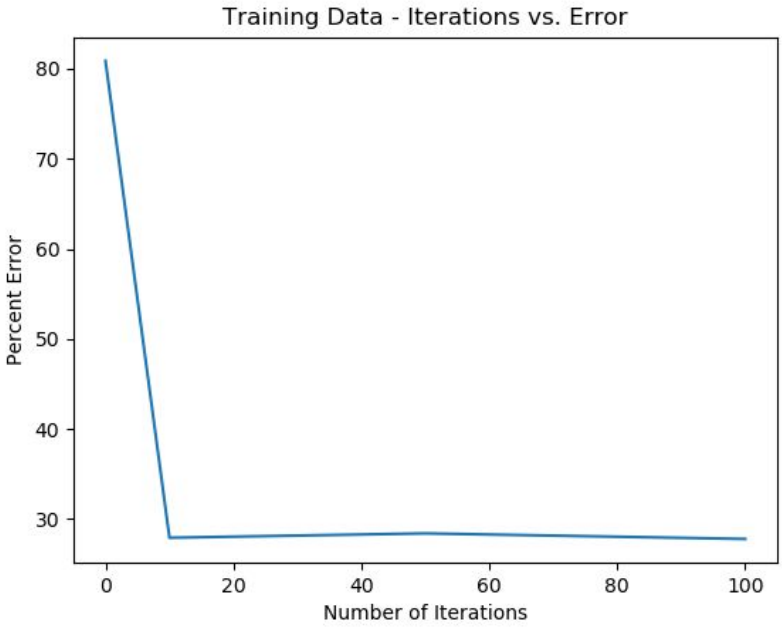
The classification error for each digit is shown below:



For some reason the digit five had a very high classification error rate, and was most often misclassified as three.

The overall training and testing error vs the iterations is plotted below:





As expected, the error rate dropped sharply with the number of iterations as the theta values were updated. When the weights were initialized to Gaussian noise, the softmax function yielded a prediction of approximately 20% (random guess) for each class. As the iterations increased, the weights were learned to produce an output with less error. The error hit a plateau around 25% due to high error in the five classification rate.

Conclusion

The weights were able to be computed relatively accurately except for the digit five. This may be due to it resembling three in most of the training examples.