Project 4

Gaurav Shilpakar

Pranesh Shrestha

Raju Khadka

April 2020

Part A

Description:

Image classification of Handwritten Digits using Logistic Regression.

Dataset:

MNIST dataset consists of 4 files which can be subdivided into two categories:

a) Training file: 60000 28\*28 pixels images and its labels

b) Testing file: 10000 28\*28 pixels Images and its labels

it is in idx file format so idx2numpy library was used to import the dataset as numpy array. These images are handwritten digit from 0 to 9 which means this problem require multinomial classification with 10 classes.

Training and Hyperparameter Selection:

Training image is reshaped from 60000\*28\*28 to 60000\*784, then normalized to value range (0 to 1) from (0 to 255) and bias term is added which results the design matrix of size 60000\*785. Linear Classification model has been used i.e. no (mapping function). A weight matrix of size 785\*10 is initialized in value randomly drawn from uniform distribution. Since this problem require multi-class classification, the probability distribution score for multi-class is done using softmax function. This function gives the probability for the data in the class in the range from 0 to 1. Log loss function is used as cost function as this is a classification problem and gradient from this function is used to update the weight.

y = predicted score, w = weight, j = number of classes, n = number of samples

t = Target label, y = predicted label

Batch Gradient Descent is suitable optimization technique for this training size and Its update equation can be presented as:

Hyperparameter selection (learning rate and lambda) is done through trail and error method. After running above update equation with stopping criteria at 1e-4 using learning rate of 1e-5 and lambda value 5, the function converged after approximately 496 epochs and the resulted plot is in the below figure:

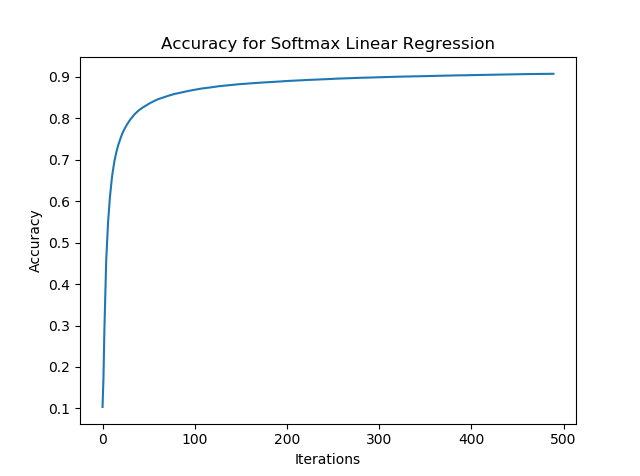
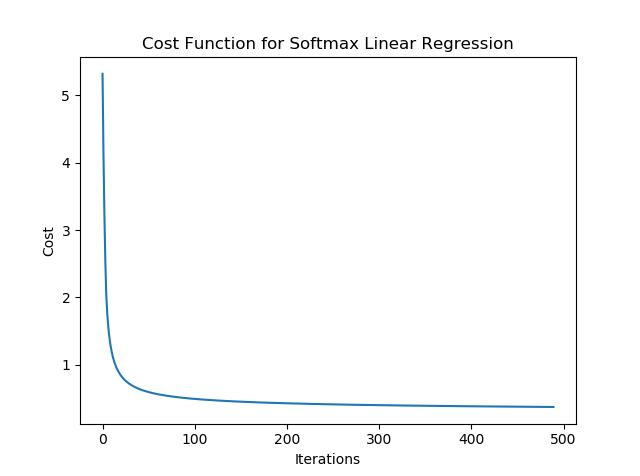


Figure Cost Function Figure Training Accuracy

From the above figure we can conclude that model is performing good enough as a linear classifier. For the given number of iterations, the model is not showing signs of overfitting and accuracy for both training and testing data is tolerable considering training data. For further evaluation, confusion matrix is generated using Scikit-Learn and the cost, accuracy and time required to run the code for training and testing data is in figure 2 and figure 3.

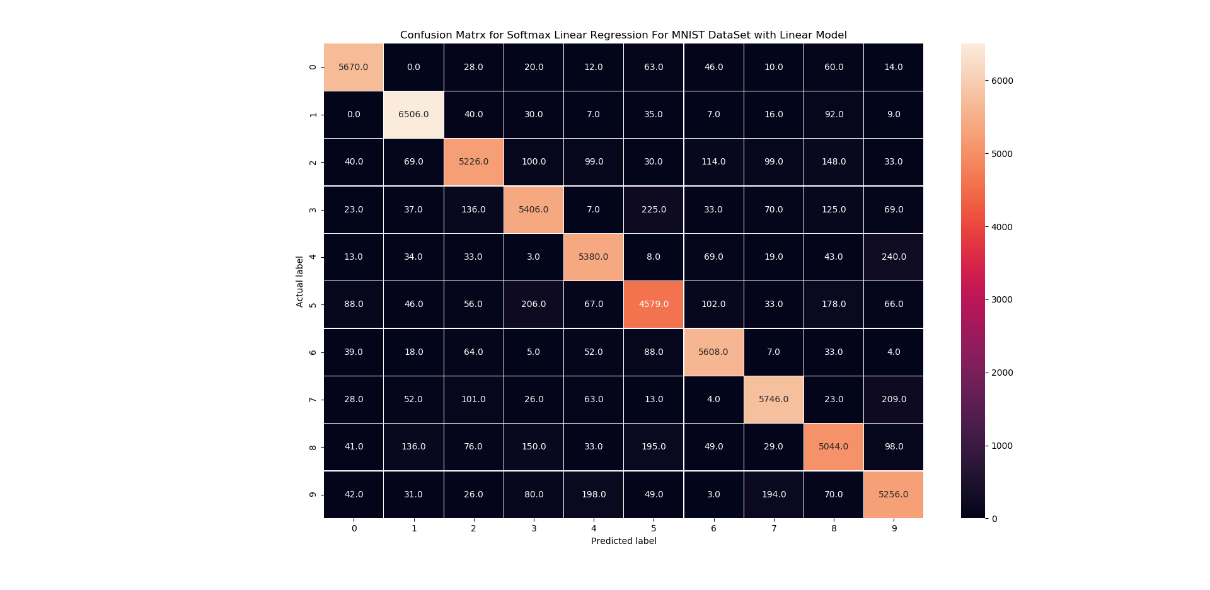


Figure Confusion Matrix

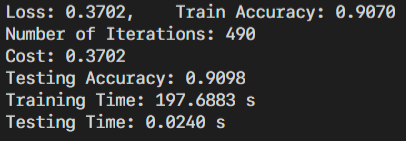


Figure Model Evaluation

Part B

Dataset Generation

Global Binarization (OpenCV)

Initially, the C. Elegans images were binarized using global binarization. For this, a global value for threshold was used. It was set at 200 such that and value below 200 was set to 0 and the rest was set to 255. This value remained constant throughout all the images. But the total images had variations in lighting conditions so differences in brightness and contrast was visible. Because of this, the global binarization technique failed on most of the images.

Adaptive Binarization (OpenCV)

Because of the lighting differences, there were irregularities in the binarized images and in finding the contours. Thus, adaptive binarization was used. For this, the threshold value is calculated for smaller regions of the image. This causes variable threshold with change in luminance values of the image. Gaussian Adaptive method was used to select the threshold value. In this method, a constant value is subtracted from the weighted sum of a certain block size from a point to return the threshold separator. For our purposes, the following values were used:

BlockSize = 99

ConstantValue = -35

Even after adaptive thresholding, variable lighting caused numerous binarization issues. So, method of contrast stretching was further utilized.

Contrast Stretching

To uniformly distribute the contrast throughout the whole set of images, contrast stretching was considered. First the mean value of all the individual images were calculated and stored. Then the absolute mean of these mean values was calculated. This was obtained as 163.03

Then, each image was processed again. If the mean of the individual image was higher than that of the absolute mean, difference between the means was subtracted and vice versa. After contrast stretching, the adaptive binarization algorithm worked effectively. So, it was adapted, and contours were calculated from the binarized image.

Finding Contours (OpenCV)

The findContours function from OpenCV library returned a numpy array of X-coordinate, Y-coordinate, width and height of the bounding box. This fuction was adapted on the binarized image for optimum results. Then, if the area of the bounding box was less than 400 pixels or greater than 4900 pixels, the box was disregarded.

A small set of bounded images with classification for NoWorm and Worm were recorded first for an initial test. The mean values of each image from these classes were recorded. The mean value for the Worm class was observed to be between 40 and 85. This was used as a basis for classifying the data between two different labels.

Finally, the boxed images were padded or cropped to a constant size of 40x40 using a recursive algorithm. It checked if the return image was of the desired dimensions. This made the learning of the images during the training process easier.

Loading data and Training:

We have decided to do binary classification for this problem i.e. Worm and no worm considering the amount of data generated. For that reason, preprocessed image has been saved in two separate folders for easily labeling and generating training and testing data. The images are loaded to a numpy array using scikit-learn in grayscale from the both the folders and concatenated. Image label equaling the size of no Worm and size of Worm are concatenated in similar fashion. After that images are shuffled along with the image label using numpy random permutation then it is divided into training data and testing data in the ratio of 8:2. Then the data is passed into the Logistic Regression class in the same manner as with Part A. Model similar to Part A was used for training.

a) Training file: 60000 40\*40 pixels images and its labels

b) Testing file: 10000 40\*40 pixels Images and its labels

Since our implementation of Logistic regression uses softmax function for calculating probability distribution for each class, it is compatible for any number of input classes and same applies for the loss function. Considering the size and features of training data, there is no need to change the optimizer and design matrix as well. But the update equation was having difficulty in converging at learning rate above 0.2 and lambda was increased to 8. Although cost function was decreasing linearly and accuracy was showing gradual progress, it these metrics failed to improve 400 iteration. After 500 iteration, training loss was observed to be 1.252 with accuracy of 70.935% then testing accuracy was 69.56%.

Training took approximately 40.57s and testing took 0.02s which differs every time because of randomized weight initialization and random shuffling.

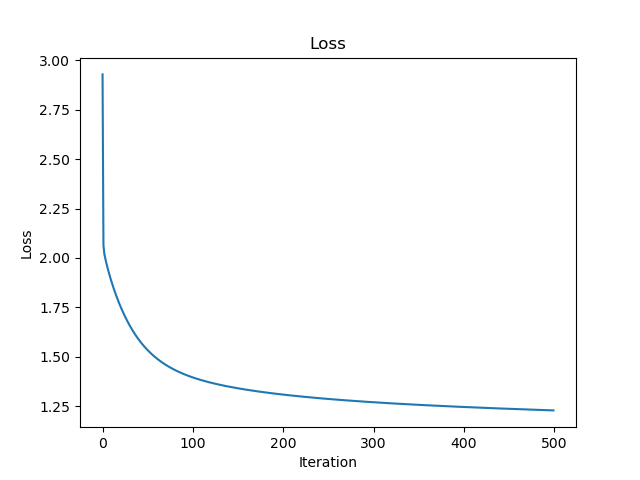


Figure Loss

A screenshot of a cell phone

Description automatically generated

Figure Training Accuracy

Conclusion:

This part performed poorly in comparison to part A. we can conclude from the figure above that the model has started to overfit the data. The reason is the amount of training data and its quality. The process of generating the data is mediocre with little to no prior information about the dataset and image manipulation techniques. For improving the performance of this model data augmentation techniques can be implemented to generate more data with variation.