Assignment #1

Yutae Lee – 626005947

CSCE 636 - 600

PART 1 : Answers for the non-programming part

1)

a) train_valid_split in takes a dataset, y-values for the dateset, and the length of train set then splits dataset and the y-values into training and validation sets according to the user's choice of the splitting index.

We need this step to split the training and validation dataset. The main reason that we are separating original training set into training and validation dataset is to prevent the problem of overfitting. In other words, if there are only training dataset the model would be very accurate in predicting the values within training dataset itself, but still may not be able to generalize and be able to predict result of the new data.

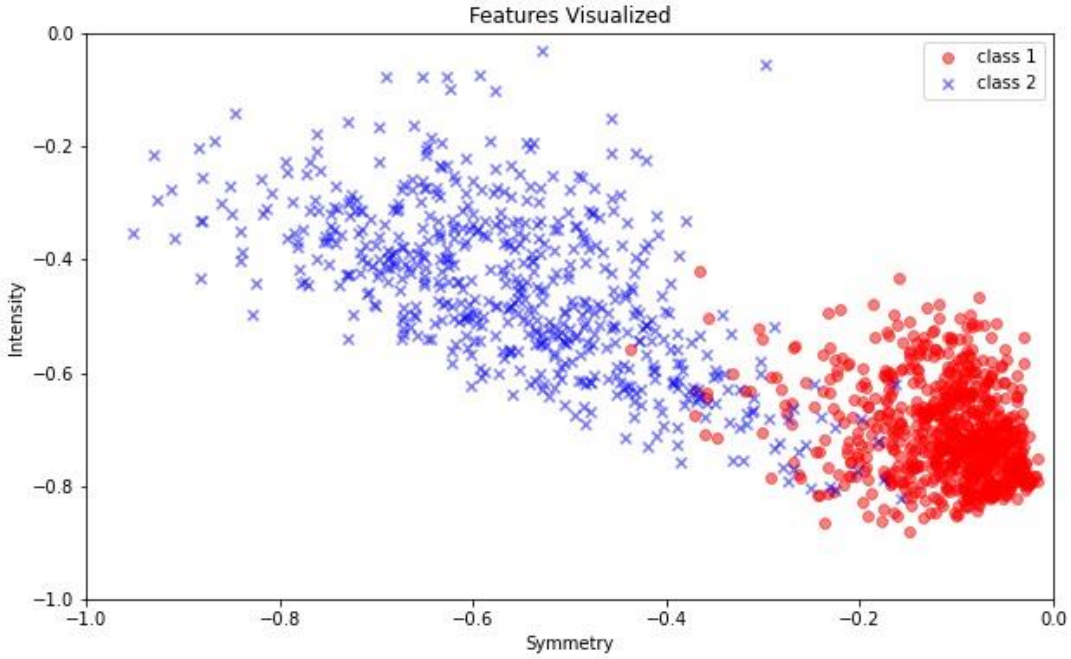
b) No, the whole purpose of splitting the training dataset is to make sure to avoid the overfitting problem. However, if we re-train the model on the whole training set, then overfitting problem may reoccur.

d) This feature of 1's is there to include w_0 as bias. So, when we are multiplying the matrices like $w^T x$

these 1 will eventually multiply w_0 to create biases.

c), e),f) code implemented in the code submission.

Result:



Analysis:

We can clearly see the class is mainly depending on the symmetry. It is clear that class 1 is higher symmetry than class 2. Also, there are difference in the symmetry as class 2 has higher intensity than class 1.

2)

a) The Cross Entropy Loss Function: $E(w) = ln\left(1 + e^{-y*w^T x}\right)$

b) Gradient of Cross Entropy: $E(w) = ln\left(1 + e^{-y*w^T x}\right) \rightarrow \nabla E(w) = \frac{1}{1+e^{-y*w^T x}} * \nabla\left(1 + e^{-y_n*w^T x}\right)$

$= \frac{\nabla -y*w^T x}{1+e^{-y*w^T x}} * e^{-y*w^T x}$, We know that $\nabla y * w^T x$ is in fact $-yx$. So the final gradient is

$$\frac{-yx}{1+e^{-y*w^T x}} * e^{-yw^T x} = \frac{-\boldsymbol{yx}}{\boldsymbol{e^{yw^T x}+1}}$$

c) Using the sigmoid may not be the most efficient. It is because sigmoid is a monotonically increasing function which makes its decision boundary linear. In fact, decision boundary can be simply obtained by the equation line $\boldsymbol{w^T x = 0.}$

We will need to necessarily use the sigmoid function when we are trying to get the probability of the event. As it can provide probability of the event.

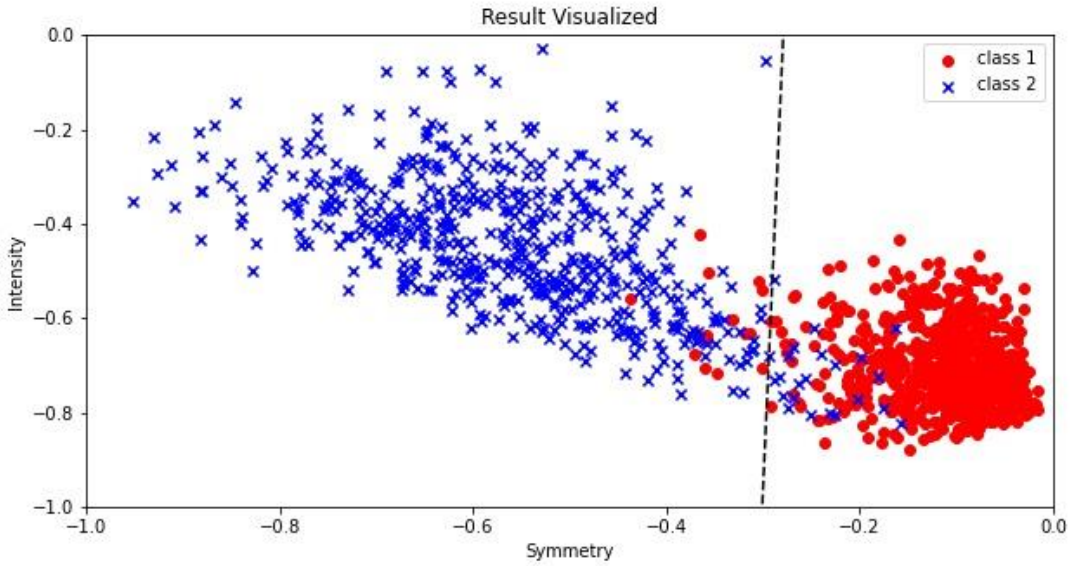
d) Yes, the model is still linear. In fact it has been moved from $\theta(w^t x) = 0.5$ to $\theta(w^t x) = 0.9$ Which is $\frac{1}{1+e^{-w^t x}} = 0.9$ -> $e^{-w^t x} = \frac{1}{9}$ -> $\boldsymbol{w^t x = ln\ (9)}$

e) The essential property is that logistic regression is linear because it is monotonically increasing function. I.e. for any given threshold in the function, there would be a identical linear decision boundary that could be represented into $w^t x = k$

3)

a),b),c) code implemented in code submission

d) Below is the result for train_results_sigmoid.jpg:



e)

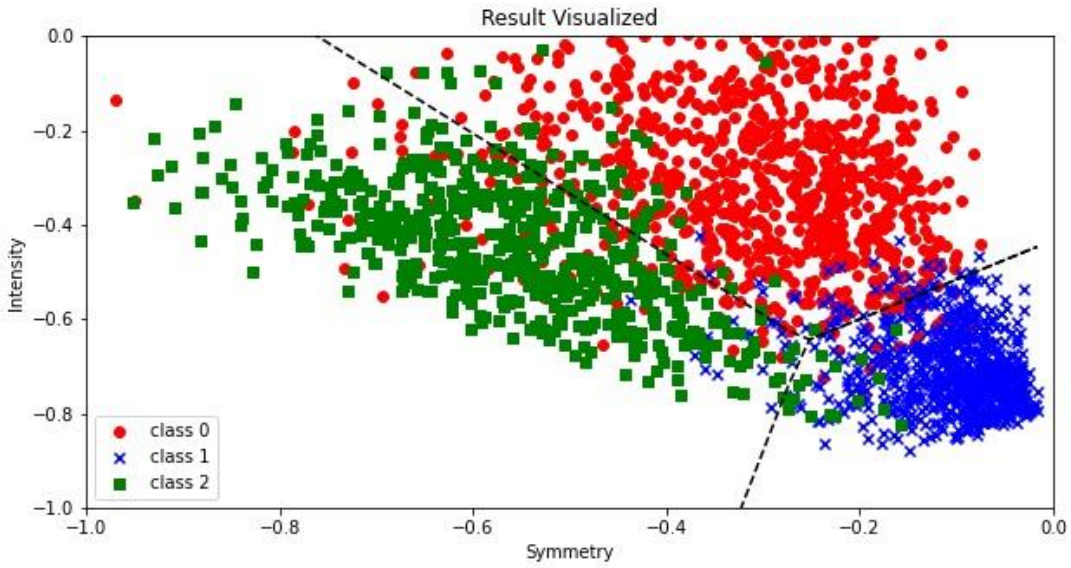
The best model that I got was Stochastic Gradient Descent with learning rate = 0.01 and max_iter = 1000.

The model gave me test accuracy: 93.29004329004329%

4)

a),b),c) code implemented in code submission

d) Below is the result for train_results_softmax.jpg:



e)

The best model that I got was batch of 10 with learning rate = 0.5 and max_iter = 1000.

The model gave me test accuracy: 86.47990255785626%

5)

a) With both model set as batch = 10, learning rate = 0.02, and max_iter = 10000

The test accuracy for the softmax was: 93.07359307359307%

The test accuracy for the sigmoid was: 58.87445887445888%

Potential reason for the low test accuracy for the sigmoid part was caused by the batch number as when I did the modeling in question 3, the best batch size was just simply 1.

When we set the softmax model as k = 2, ideally softmax model should actually be equivalent to the sigmoid model.

b)

Intuitively speaking, if the learning rate of sigmoid is double of the softmax then w2- w1 = w because softmax updates w1,w2 in a single step while sigmoid updates solely w. So, I set the learning rate of sigmoid as 0.04 which is double of the learning rate that I set the learning rate for softmax which 0.02.

Although theoretically this would'be gave me same weight. However due to some error in my code, I could not get them.

PART 2 Results and Analysis of the Programming Part:

Number of Batches in miniBGD vs SGD:

In my model using learning rate of 0.01 and max iteration of 1000 gave me best accuracy. So, using these learning rate, I tried to find what size of batch could give me the best model. In fact, the result gave me that batch size for the model that I created was 1. So, another comparison that I made was actually miniBGD of size 1 vs SGD. The result showed me that the model gave me the same test accuracy of 93.29004329004329.

Using this fact that I was able to realize miniBGD of batch size 1 is equivalent to the SGD model.

Softmax Classification:

In the case of Softmax Classification, using learning rate of 0.5 and having max iteration of 1000 gave me best accuracy. One thing that stands out most was difference in learning rate compared to the sigmoid model. One hypothesis that I can make is because of addition of new class, class_2 learning rate should be higher to maintain a better accuracy.

The best batch size in my case was 10, which gave me test accuracy of 86.47990255785626.

Conclusion from both model:

The two model has one similarity in that they both turned to have max iteration of 1000. 1000 was the biggest number of iterations that I tried for the model, so which mean it that any iteration number above 1000 could give me a better accuracy. The reason behind this would be intuitive as we are iterating more on the training model the training accuracy score would generally get better. However, we need to take an account that these just simply concluding that more iteration would give us better accuracy may be fraud because of potential overfitting problem could happen if we have too big of iteration.