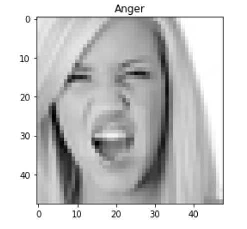
Logistic Regression for Image Classification

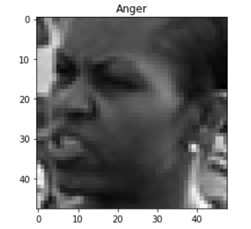
Question 1 – Logistic Regression Models for Binary Image Classification

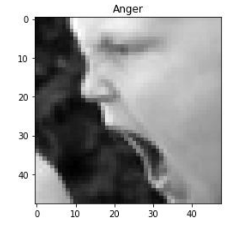
In the practical lab folder you will find a train and validation csv file that has been adapted from a Kaggle competition that focuses on learning facial expressions from a library of images. The original dataset contains 48*48 grayscale pixels which compose an image (**2304 features**) and the true class of the image. In the original dataset there are 7 possible classes (angry, disgust, fear, happy, sad, surprise and neutral).

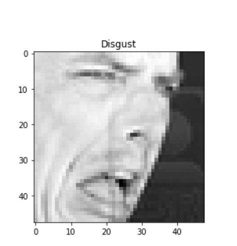
The objective of this exercise is to create a logistic regression model using vectorization and contrast the standard and vectorized implementations. I have created a modified training and validation set of the above dataset so that it only contains images that are either classified as angry or disgust. The following is a sample of some of the images. The images in the top row are example of anger and those on the bottom row are examples of disgust. You will find the training and validation csv files on Canvas. Please note that the final column in each file is the class we are trying to predict (0=Anger and 1 = Disgust). There are **7919** images in the training set and **110** images in the validation set.

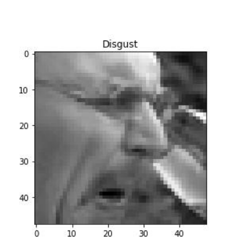
Please note that basic logistic regression will not perform well with on task. We don't expect the model to achieve high levels of accuracy but instead you should see the impact of using a vectorized over a non-vectorized approach when dealing with a non-trivial dataset.

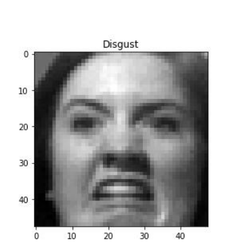


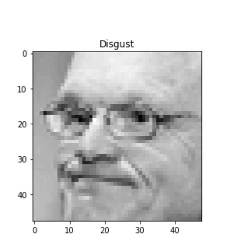










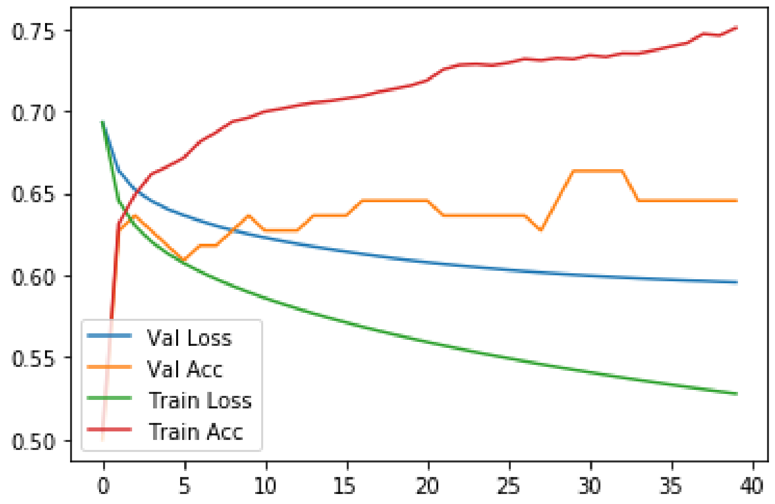


<u>Instructions</u>:

1) In the Canvas folder you will find a fully completed program that implements a logistic regression model for solving this problem (without any vectorization). Gradient descent is set to iterate 300 times. This may take a little time to complete fully.

2) When the code finishes running you will see the following it will output a graph similar to what is shown below.

   What can you interpret from the behaviour depicted on this graph?



3) While the code which you have run in step 1 is relatively efficient (in that we are making significant use of NumPy) we can improve it further by making it completely vectorised. On Canvas you will find a code template for a vectorised solution.

   Your first task is to change the shape of the training and validation feature data. Originally the shape of the training feature data was (7919 * 2304) corresponding to (number of training examples * number of features).

   However, for the vectorised implementation we will transpose the training feature data so that it's shape is now (2304 * 7919) corresponding to (number of features * number of training examples). The validation feature data should undergo the same process. (See TODO: 1 comment)

$$X = \begin{bmatrix} x_1^1 & \cdots & x_1^m \\ \vdots & \ddots & \vdots \\ x_n^1 & \cdots & x_n^m \end{bmatrix}$$

4) You should initialize your coefficients (tuneable weights) as a column vector with one weight for each feature in our dataset as shown below (see TODO: 2).

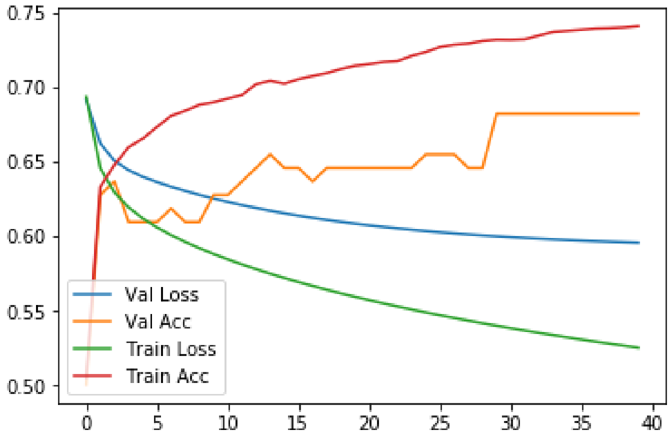
$$coefficients = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_n \end{bmatrix}$$

5) Your next task is to complete the hypothesis function. Please refer to the vectorized logistic regression lecture notes (see TODO: 3).

6) Next you should write the code to calculate the gradients for the bias and the coefficients (see TODO: 4).

7) Finally complete the gradient descent update rules for updating the bias and each of the coefficients (see TODO: 5).

8) When you have parts 4 -6 completed you should now be able to run your vectorised code. You should notice a very significant acceleration in speed and the outputted results should be similar to what you originally observed with the basic implementation. The following is the result obtained from the vectorized logistic regression.


Val Loss
Val Acc
Train Loss
Train Acc