Introduction

For this assignment, we will perform binary classifications of facial expressions using a logistic regressor, a 1-layer neural network, and the SVM classifier. The main goal is to successfully write the predict and train functions for both classifiers and compare these with values obtained from other classifiers. The dataset we are using is quite noisy and was obtained from the very 2013 Kaggle facial expression classification competition[1]. Although the original dataset contained 7 emotional classes (six of which are shown in Figure 1), we will only use two of them (happy and sad) for this assignment.
Download the homework3.zip file from myCourses `Contents`→`Programming_assignments`→`Homework3`; this contains the instructions, starter code and image data needed for the assignment.

Figure 1: The top row shows three examples of correctly labeled faces from the Kaggle challenge; left to right - angry, disgust and fear. The bottom row shows three incorrectly labeled faces; left to right - happy, sad and surprise. Neutral face is not shown.

Requirements

You should perform this assignment using Python along with any image library of your choice, and it is due on **Sunday November 3rd by 11:59pm**. You are required to submit your code in a Jupyter notebook along with a brief report containing short write-ups based on the question(s) in the assignment. Your solutions should be zipped and uploaded to myCourses via Assignments (formerly known as Dropbox) before the due date.

Your submitted zipped file for this assignment should be named **LastnameFirstname_hw3.zip** and should contain at least two files - *LastnameFirstname_hw3.pdf* and *LastnameFirstname_hw3.ipynb*. Feel free to submit any other auxiliary files required to run your code. We should be able to execute your code for the assignment from your submitted Jupyter notebbok. Include a `Readme` file if necessary (especially if using external libraries other than those used in the starter code).

[1] `https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-\ \challenge/data`

The Data Files

You are provided with two data files, where the first file `fer3and4train.csv` contains the training data with 12,066 data samples and the second file `fer3and4test.csv` contains the data that you will be testing your classifier on. Results should be reported on the test dataset which contains 2000 data samples. The files were created using `fer2013.csv` from Kaggle but have been shuffled and augmented to avoid the class imbalance problem.
All the files are stored as comma separated files with three columns each. The first column contains the label of the emotional expression, where 3=happy and 4=sad; the second column contains 2304 integer values (between 0 and 255) obtained by vectorizing 48×48 grayscale images of faces; and the last column states in which part of the development process the data should be used (i.e training or testing).

Problem 1. The logistic regression classifier (Total 30 points)

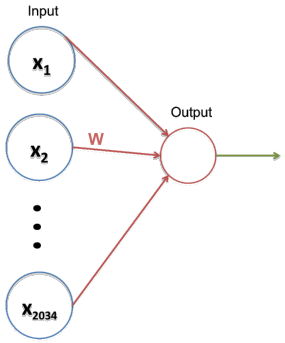

Figure 2: The logistic regression classifier

The file `LRClass.py` contains the skeleton code for your logistic regression module. This is accompanied by an auxiliary file called `helper.py` containing the names of some helper functions needed for your module to run. In the provided Jupyter notebook, different parts of the code have been marked for your implementation. First load the the training data samples $\mathbf{X}$ and their corresponding class labels Y using the helper function `getBinaryfer13Data`. Then use the class object to train the data. Call the `train` function to learn the weights and bias of the unit. The following occur within the `train` function:

i Initialize the weights W to small random numbers (variance - zero); also initialize the bias b to zero. The function `init_weight_and_bias` is provided to aid in this.

ii Create a loop over the number of epochs specified. Within the loop, the following occur:

iii Call a `forward` function to calculate the predictions $P(Y|X)$ also known as pY. The `forward` function implements $\sigma(\mathbf{X} \cdot \mathbf{W} + b)$. The argument of this equation can be implemented in *numpy* as $\sigma(np.dot(X, W) + b)$ where σ is the sigmoid activation function; this is provided also as a helper function.

iv Next, learn the weights via back-propagation, by performing gradient descent using the equations below:

$$W = W - \eta \frac{\partial J}{\partial W}; \qquad W = W - \eta \cdot (\mathbf{X}^\top \cdot (pY - Y)) \tag{1}$$

Note: When doing matrix computations, the product of the vectors $X^\top Y$ can be written as $np.dot(\mathbf{X}.\mathrm{T}, \mathbf{Y})$; Also,

$$b = b - \eta \frac{\partial J}{\partial b}; \qquad b = b - \eta \cdot (pY - Y) \tag{2}$$

v Apply the forward algorithm to predict the new labels for both the training and validation data. Compute the training cost and validation cost at each epoch and append to a growing array (one training, the other validation). Keep note of the best error value on the validation data.

vi Print out the best error value on the validation data. Provide this value in your final report.

vii Plot your training and validation costs to show how the errors are changing over time. See Figure 4. Dispaly this in your report.

viii Lastly load a new dataset, your test data from the file `fer3and4test.csv`. Compute and print the accuracy (or classification rate) for predicting this new dataset from your trained network. Provide this information in your report.

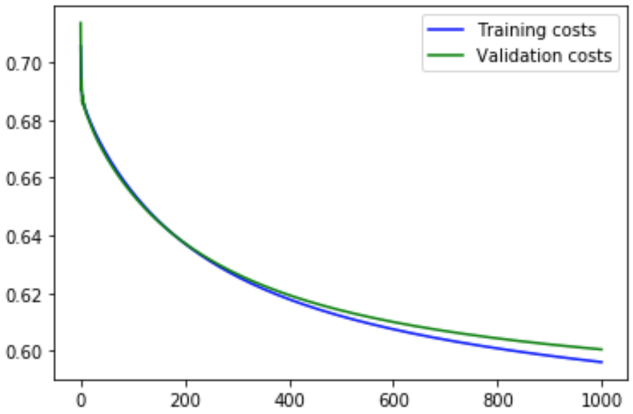

Figure 3: An example of the loss curves from the logistic regressor after 1000 epochs

Problem 2. The neural network classifier (Total 45 points)

Similar to the previous exercise, the class `NNclass` in the Jupyter notebook contains the skeleton code for your neural network module. Again there are some helper functions in `helper.py`. Load the training data samples $\mathbf{X}$ and their corresponding class labels Y using the helper function `getBinaryfer13Data`. Call the `train` function to learn the weights and bias of the unit. The following occur within the `train` function.

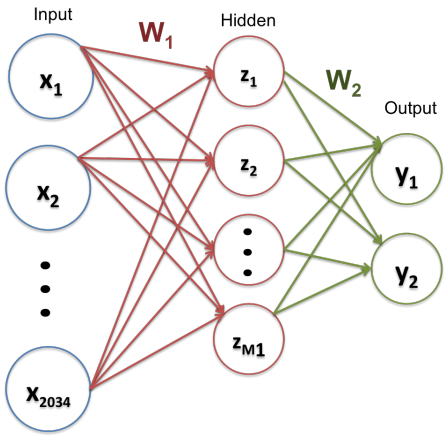

Figure 4: The single hidden layer neural network with two softmax output nodes

i Initialize the weights W_1, W_2 to small random numbers (variance - zero); also initialize the bias b to zero. The function `init_weight_and_bias_NN` is provided to aid in this. Remember, this time you need to set the number of hidden units in layer 1. This is your design choice.
Note: The dimensions of these parameters here are different from those in the LRClass. W_1 is $D \times M1$, where $M1$ is the number of hidden nodes and the dimension of W_2 is $M1 \times K$, where K is the number of output classes.

ii Create a loop over the number of epochs specified. Within the loop, the following occur:

iii Call a `forward` function twice to calculate $P(Ytrain|X)$ also known as pY and $Ztrain$ (activations at hidden layer); and the other to calculate $P(Yvalid|Xvalid)$ and $Zvalid$ on the validation data. This implies that your `forward` function needs to return two values (i) pY, the output of the softmax classifier and (ii) the hidden activations Z, based on which activation function you choose (*tanh, sigmoid, or ReLU*).

iv Now we do a first round of back propagation by first performing gradient descent using equations (3) and (4) below;

$$W_2 = W_2 - \eta \frac{\partial J}{\partial W_2}; \qquad W_2 = W_2 - \eta \cdot (\mathbf{Z}^\top \cdot (pY - Y)) \tag{3}$$

$$b_2 = b_2 - \eta \frac{\partial J}{\partial b_2}; \qquad b_2 = b_2 - \eta \cdot (pY - Y) \tag{4}$$

v Then we propagate the errors we got from the previous layer W_2 to update W_1 and b_1 via equations (5)-(7):

$$\frac{\partial J}{\partial Z} = (pY - Y) \cdot {W_2}^\top \cdot (1 - \mathbf{Z}^2) \tag{5}$$

$$W_1 = W_1 - \eta \cdot \mathbf{X} \cdot \frac{\partial J}{\partial Z} \tag{6}$$

$$b_1 = b_1 - \eta \cdot \frac{\partial J}{\partial Z} \tag{7}$$

Matrix multiplications in numpy will be sufficient to complete the processes.
Note: Only the training data is used for updating weights in gradient descent.

vi Apply the forward algorithm to predict the new labels for the training and validation data and also compute the sigmoid costs of predicting them compared with the true labels. Append the resulting cost to the growing arrays.

vii Keep note of the best error value on the validation data. Also compute and print out the training and validation classification rates. This should be given in your final report.

viii Display the graphs of both your training and validation errors (created from the above process) to show how the errors change with time. Display these in your report.

ix Lastly load a new dataset, your test data from the file `fer3and4test.csv`. Compute and display the accuracy (or classification rate) for predicting this new dataset from your trained network. This value should be shown in your final report.

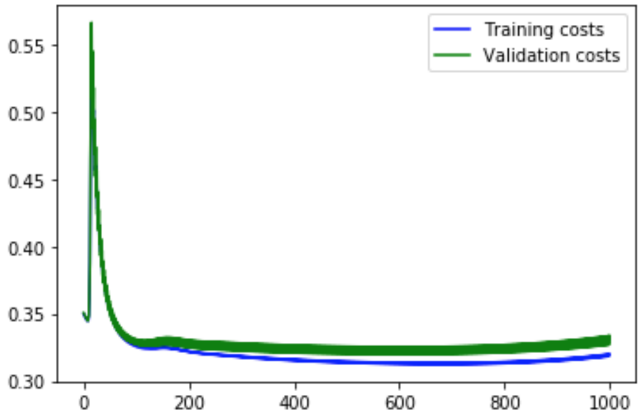

Figure 5: An example of the loss curves from the logistic regressor after 1000 epochs

Problem 3. Adding regularizers (Total 10 points)

After the initial training and testing, go back and add a regularizer to the cost function of the neural network and retrain. Now report your new error rates and accuracies.

(a) Add the L1, L2 or ElasticNet regularizer using the formula provided in the slides from class

(b) Print out your new classification rates, both for the training and validation datasets, as well as the test dataset.

(c) Discuss your observations in working with a regularizer (in your report).

Problem 4. Training with SVM (Total 15 points)

Now go back and we will train with SVM. Scitkit-learns is a good machine learning library with an easy-to-use SVM interface. Similar to the previous problems, train an SVM on a portion of the training data, test out your results on a validation set and finally run the true tests on the test data set. Report your results including the accuracies when using :

(a) A linear kernel

(b) A radial-basis-function (RBF) kernel

(c) A polynomial kernel. Play around with the orders of the polynomial to determine which one gives the best result.

(d) Discuss your results in working with the 3 different kernels in your report

Note: Training an SVM with over 11,000 data samples of dimension 2034 will take a very long time.

BONUS: (Total 10 points) Use a different activation function for the neural network to see if the results will improve. Don't forget to change your derivatives based on the new activation function you choose. There are some functions to help with this in `helper.py`. Add the code into your ipython notebook and discuss your findings in your report.

You should turn in both your code and report in the zipped file to get full credit.