Introduction

For this assignment, we will perform binary classifications of facial expressions using a logistic regressor, the SVM classifier with multiple kernels and a freeform CNN. The main goal is to successfully write the `predict` and `train` functions for all three classifiers and compare these with values with each other. The dataset we are using is quite noisy and was obtained from the 2013 Kaggle facial expression classification competition. 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 since we are performing binary classification.
Download the homework2.zip file from Brightspace in the Homework 2 sub-module under Assignments; this contains the instructions, starter code and 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 **Friday November 14, 2024 by 11:59pm**. You are required to submit your code in the Jupyter notebook provided along with a brief report containing short write-ups based on the question(s) in the assignment. Your solutions should be zipped and submitted via Brightspace, by the due date.

Your submitted zipped file for this assignment should be named **LastnameFirstname_hw2.zip** and should contain at least two files - *LastnameFirstname_hw2.pdf* and *LastnameFirstname_hw2.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 zipped file on our data. Include a `ReadMe` file if necessary (especially if using external libraries other than those used in the starter code). **Please do not submit the data**, we have a different set to test your code on.

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. Out of your training set, select a small fraction to use as your validation dataset. Results should be reported on the test dataset which contains 2000 data samples.
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 35 points)

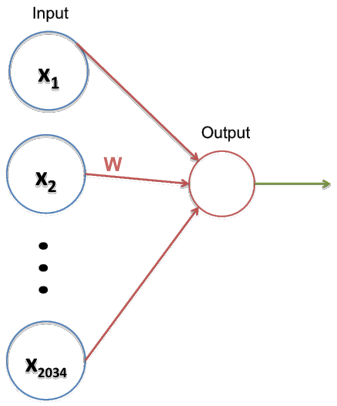

Figure 2: The logistic regression classifier

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 should 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 σ(`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 3. Display 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.

For a bonus of 5 points, add a regularizer to your implementation and indicate how this improves/downgrades your performance.

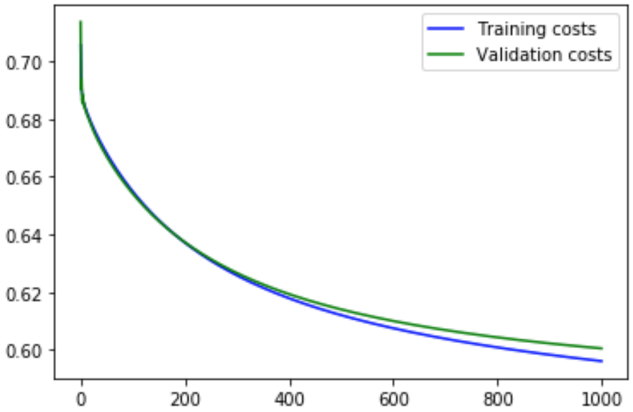

Figure 3: An example of the loss curves from the logistic regressor after 1000 epochs

See problems 2 and 3 on the next page...

Problem 2. Training with SVM (Total 15 points)

Now we will train a new model using SVM. Scitkit-learns, a machine learning library with an easy-to-use SVM interface. Similar to the previous problem, 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 accuracy 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.

Problem 3. Training with a freeform CNN (Total 10 points)

Lastly, you can download any freeform CNN from the Internet or build one of your own and use it to train a new classifier. You can use any of the standard deep learning libraries, `PyTorch` or `TensorFlow`, as long as your `.ipynb` file is self-contained and has all the necessary imports.
First reshape the vectors given into 48×48 image matrices. Then similar to the two prior problems, train a CNN on a portion of the training data. Next, test out your ongoing performance on a validation set and finally when you are satisfied with the train/validatino results, run your predictions on the test dataset given.
Report your test results in form of a confusion matrix and highlight the accuracy. Briefly discuss how this model compares with the other 2, both in computational speed and accuracy.

You should turn in both your code (`.ipynb` file) and report (`.pdf` file) in the zipped file to get full credit.