IIT BOMBAY

Assignment #1
Points: 50

Course: *CS 725* – Instructor: *Preethi Jyothi*
Due date: *11:55 pm, September 6, 2024*

General Instructions

- Download the file `assgmt1.tgz` from Moodle and extract the file to get a directory named `assgmt1` with all the necessary files within.
- For your final submission, create a final submission directory named `assgmt1` with the following internal directory structure:

```
assgmt1/
  |
  +-README
  +- part1A.py
  +- histograms/
  +- part1B.py
  +- closedForm.py
  +- batchGradientDescent.py
  +- smilingJoker.py
  +- part3.py
  +- kaggle.csv
```

  Compress your submission directory using the command: `tar -cvzf assgmt1.tgz assgmt1` and upload assgmt1.tgz to Moodle. This submission is due on or before **11:55 pm on Sept 6, 2024**. No extensions will be entertained.
- `README` should contain the names and roll numbers of all your team members.
- A quick and short introduction to NumPy is here. Please make sure you go through this carefully; `numpy` operations beyond what is mentioned in this document can also be used for the assignment.

Part I: Numpy Basics (15 points)

(A) Coin Tossing. Say you toss 100 fair coins simultaneously and want to programmatically record the total number of heads. One simultaneous toss of the 100 coins counts as a single trial. `part1A.py` contains a list `num_trials_list` of varying numbers of trials. For every value in this list, we want to count the number of heads and plot a histogram. The histogram should take the shape of a binomial distribution and get more accurate with larger numbers of trials.

`part1A.py` has two functions:

1. `toss(num_trials)`: Takes the number of times the experiment is to be performed as an argument and returns a numpy[a] array of the same size with the number of heads obtained in each trial. **[2 pts]**

2. `plot_hist(trial)`: Takes an array of the number of heads obtained for k trials from `toss(`k`)` and plots a histogram based on these counts. Generate plots for each value in `trials_list` and save them in a directory. **[3 pts]**

Complete the following tasks:

1. Complete the function definitions in `part1A.py` and submit it as `assgmt1/part1A.py`.

2. Create a directory named `histograms` inside the directory `assgmt1/`.

3. Save the histograms obtained by `plot_hist()` in `histograms` with names formatted as `hist_<num_trials>.png`. For example, for `num_trials` = 1000, the filename would be `hist_1000.png`.

Notes:

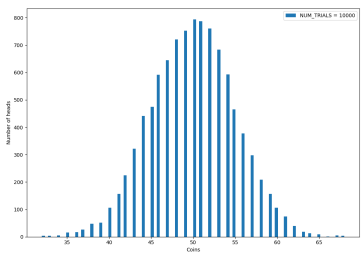
1. Use `plt.savefig()` function to save the histograms.

2. Use the given template only. **DO NOT** change the names of the given functions as it will cause the autograder to fail.

3. Use `for` loops to simulate the 100 coin tosses for a given `num_trials` value. Do not use predefined functions to calculate the numpy array in *''toss()"* function else marks will be deducted.

Here is an example histogram for **num_trials** = 10000.

[a] `https://numpy.org/doc/stable/user/index.html`



(B) Tensor Manipulations and Softmax. Consider N column vectors $\{\mathbf{u}_1, \ldots, \mathbf{u}_N\}, \mathbf{u}_i \in \mathbb{R}^d_{\geq 0}$, and two non-negative real-valued $d \times d$ matrices $\mathbf{M}_1, \mathbf{M}_2$.

Complete each of the following steps using numpy operations and do not use any `for` loops:

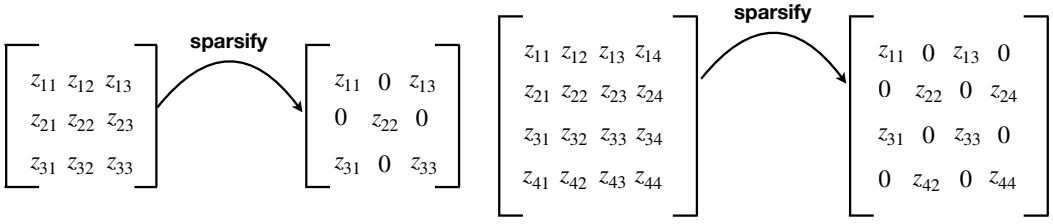
1. Compute $\mathbf{x}_i = \mathbf{u}_i^T \mathbf{M}_1, \forall i \in \{1, \ldots, N\}$. Stack the N row vectors $\mathbf{x}_i$ to form a new matrix $\mathbf{X} \in \mathbb{R}_{N \times d}$. Similarly, construct a matrix $\mathbf{Y} \in \mathbb{R}_{N \times d}$ using row vectors computed from $\mathbf{u}_i^T \mathbf{M}_2, \forall i \in \{1, \ldots, N\}$. **[1 pts]**

2. Modify $\mathbf{X}$ to add the integer i to all elements of its i^{th} row, where $i \in \{1, \ldots, N\}$. Let this offset-modified matrix now be $\hat{\mathbf{X}}$. **[2 pts]**

3. Compute $\mathbf{Z} = \hat{\mathbf{X}}\hat{\mathbf{Y}}^T, \mathbf{Z} \in \mathbb{R}_{N \times N}$. Let

$$\mathbf{Z} = \begin{bmatrix} \leftarrow \mathbf{z}_1 \rightarrow \\ \leftarrow \mathbf{z}_2 \rightarrow \\ \vdots \\ \leftarrow \mathbf{z}_N \rightarrow \end{bmatrix}_{N \times N} = \begin{bmatrix} z_{11} & z_{12} & \ldots & z_{1N} \\ z_{21} & z_{22} & \ldots & z_{2N} \\ \vdots & & & \\ z_{N1} & z_{N2} & \ldots & z_{NN} \end{bmatrix}_{N \times N}$$

   Apply the following "sparsify" operation on $\mathbf{Z}$ (examples shown below for $N = 3$ and $N = 4$):



   You should use numpy broadcasting in this step. **[2 pts]**

4. Compute $\hat{\mathbf{Z}}$ such that each $\mathbf{z}_i$ in $\mathbf{Z}$ is replaced by $\hat{\mathbf{z}}_i$, where $\hat{z}_{ij} = \frac{\exp(z_{ij})}{\sum_j \exp(z_{ij})}$. Note that each row in $\hat{\mathbf{Z}}$ will now be a probability distribution. **[2 pts]**

5. Print the index of the maximum probability in each row of $\hat{\mathbf{Z}}$. **[1 pts]**

Add your solution to `part1B.py` that already contains an `initialise_input` function and submit it as `assgmt1/part1B.py`. We will add new test cases during grading; successful completion will get you 2 points. **[2 pts]**

Part II: Linear Regression (30 points)

(A) Closed-form Solution of Linear Regression (10 points). For a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^{(d+1)}$ represented by a feature matrix $\mathbf{X}$ and a label vector $\mathbf{y}$, the least squares solution $\mathbf{w}^*$ can be computed by $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$ where

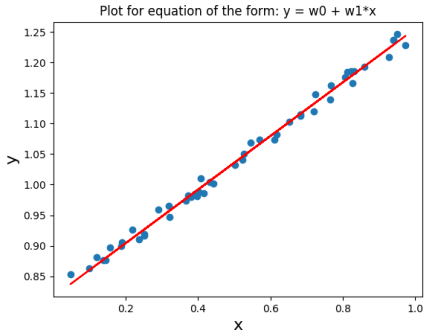
$$\mathbf{X} = \begin{bmatrix} \longleftarrow & \mathbf{x}_1{}^\top & \longrightarrow \\ \longleftarrow & \mathbf{x}_2{}^\top & \longrightarrow \\ & \vdots & \\ \longleftarrow & \mathbf{x}_n{}^\top & \longrightarrow \end{bmatrix}_{n \times (d+1)}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}_{n \times 1}, \quad \mathbf{w} = \begin{bmatrix} w_0 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}_{(d+1) \times 1}$$

Make the following changes to `closedForm.py`:

1. `fit()`: This function should calculate the `weights` using the closed-form equation as defined on the given values of X and y passed as input. **[3 pts]**

2. `predict()`: This function should return the predicted values of the model on an input set of feature values. **[2 pts]**

3. `plot_learned_equation()`: This function should generate a plot (to be saved as `closed_form.png`) that shows all the data points along with the best fit. The generated plot should look something like: **[2 pts]**



Note:

1. `generate_toy_dataset()` can be used to generate the data. On completing your tasks, run:

   ```
   $ python closedForm.py
   ```

2. To make your life easy, we have a test suite that checks your code on a few test cases. You will need PyTest to run these test cases. We might add some additional cases during the final grading. **[3 pts]**

   To run the test suite, execute:

   ```
   $ pytest test_closedForm.py
   ```

3. Stick to the given template code. **DO NOT** change the names of the functions given as it will cause the autograder to fail.

(B) Gradient Descent for Linear Regression (12 points). Find the least squares solution using *Batch Gradient Descent*. `batch_size` indicates the number of data points in a batch. If `batch_size` is `None`, this function implements full gradient descent and computes the gradient over all training examples. The code for generating and creating batches on a toy dataset is in `batchGradientDescent.py`.

Your task is to complete the following functions:

1. `fit()`: There are two loops inside this function. You have to complete the inner `for` loop and use `compute_gradient()` to calculate the gradient of the loss w.r.t. the weights. Next, you must calculate the training loss using `compute_rmse_loss()` and store these losses across training epochs in `error_list`. **[2 pts]**

2. `compute_gradient()`: This function should return the gradient of the loss w.r.t weights of the model. (Normalize the values before returning, or it may cause gradients to explode.) **[2 pts]**

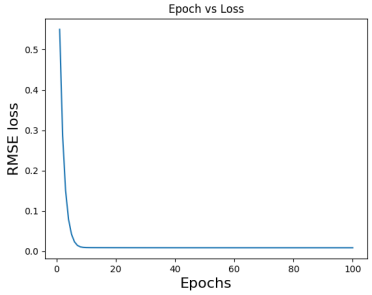
3. `compute_rmse_loss()`: This function should return the *Root Mean Square Error* loss ($\sqrt{\frac{1}{n}||\mathbf{y} - \mathbf{Xw}||_2^2}$) between the target labels and predicted labels. **[1 pts]**

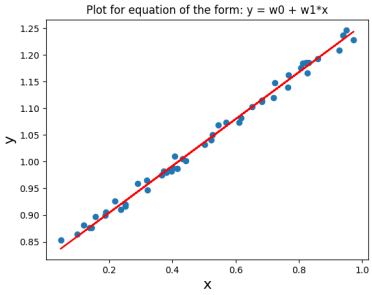
4. `predict()`: This function should return the predicted values of the model on the given set of feature values passed as an argument to it. **[1 pts]**

5. `plot_loss()`: This function is used to plot the losses stored in `error_list` of the model. Save the image as *plot_loss.png*. **[1 pts]**

6. `plot_learned_equation()`: This function generates a plot (save the image as *gradient_descent.png*) on a dataset with 1 feature (i.e $d = 1$). **[1 pts]**

Finally, your code should generate two plots that look like this:





Note:

1. Similar to the previous question, to generate these plots, simply execute:

   ```
   $ python batchGradientDescent.py
   ```

2. You can run test cases using: **[4 pts]**

   ```
   $ pytest test_batchGradientDescent.py
   ```

3. Stick to the given template code. **DO NOT** change the names of the functions given as it will cause the autograder to fail.

(C) Linear Regression with Basis Functions (8 points). (The Smiling Joker)

This question will require some basic reasoning about feature spaces. You are given a training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}$. The relation between $\mathbf{x}_i$ and y_i is defined by a weighted combination of transformations $\Phi = \{\phi_1, \phi_2, \ldots, \phi_K\}$ such that:

$$y_i = w_0 + \sum_{j=1}^{K} w_j \cdot \phi_j(\mathbf{x}_i) \qquad \forall i \in \{1, 2, \ldots, n\}$$

Given just the dataset, you are tasked with identifying the appropriate number and choice for these transformations ϕ_j and fitting a linear regression model.
More concretely, complete the following functions in `smilingJoker.py`.:

1. `read_dataset()`: This function should read the CSV file *dataset.csv* and generate `train` and `test` splits. The file contains 500 data points; you can use the first 90% of the data for training and the rest for testing your model. **[2 pts]**

2. `transform_input()`: This function should take $X \in \mathbb{R}^{n \times 1}$ as input and return the transformed $X \in \mathbb{R}^{n \times (K+1)}$ as output. (Note: This can be implemented as a matrix operation.) **[3 pts]**

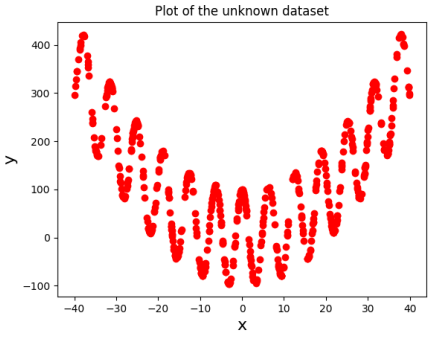
Note:

1. Similar to the previous questions, you should run the following and pass all the prediction checks: **[3 pts]**

```
$ python smilingJoker.py
```

2. Stick to the given template code. **DO NOT** change the names of the functions given as it will cause the autograder to fail.

Note:
When you plot the dataset, you will see a Smiling Joker [a] (hence the name!).



[a] `https://tinyurl.com/stock-smiley-joker`

Part III: Kaggle Competition (5 points)

Challenge: Predict Sentiments from Compressed High-dimensional Features.

Overview. In this Kaggle competition, you will solve a realistic linear regression task using code you have written in the previous parts. The task is to predict target scores (0 to 5 ranging from very negative to very positive) based on a set of features extracted from product reviews. The final model will be evaluated on a test dataset via Kaggle, and test performance will be measured using the Mean Squared Error (MSE) metric.

Competition Link: You can join the competition on Kaggle: IIT Bombay CS 725 Assignment 1 (Autumn 2024). Please sign up on Kaggle using your IITB LDAP email ID, with your Kaggle "Display Name" set to the roll number of anyone in your team. This is important for us to identify you on the leaderboard.

Dataset Description. You are given three CSV files:

- `train.csv`: This file contains the training data with 64 features and a corresponding target score for each entry.
- `test.csv`: This file contains the test data with 64 features but without the target scores.
- `sample.csv`: This file contains the submission format with predicted scores for the test data. You will have to submit such a file with your test predictions.

Each row in the data files represents an instance with the following columns: ID: A unique identifier for each data point, `feature_0, feature_1, ..., feature_63`: The 64 features extracted from the dataset. `score`: The target score for each data point (only in `train.csv`).

Task Description. Implement a linear regression model for the given problem. You can reuse any of the functions from Part II of this assignment. Tune the hyperparameters on a held-out set from `train.csv` to achieve best model performance on the test set. Predict the target scores on the test dataset. Round the predicted scores to the nearest integer before submission.

Evaluation. The performance of your model will be evaluated based on the Mean Squared Error (MSE) calculated on the test dataset. Your predicted scores must be rounded to the nearest integer. You should not implement the MSE metric; it will be automatically calculated via Kaggle. Your model will be evaluated on the provided test set, where a random 50% of the examples are marked as *private* and the remaining are *public*. The private/public distribution will be hidden. You can monitor your model's performance on the public part of the test set via the public leaderboard. The final evaluation will be based on the private part of the test set, which will be revealed via the private leaderboard after the competition concludes.

Submission. Submit your source file named `part3.py` and a CSV file `kaggle.csv` with your predicted scores (rounded to nearest integer) for the test dataset, following the format in `sample.csv`. If you match or outperform the baseline RMSE obtained with the solution from the closed form solution in part II, you will get all 5 points. Top-scoring performers on the "Private Leaderboard" (with a suitable threshold determined after the deadline passes) will be awarded up to 3 extra points.