Machine Learning & Data Mining
Set 1

Caltech CS/CNS/EE 155
January 4^{th}, 2022

Dallas Taylor

Colab Links

All colab links are under their relevant problems. Here is a master list as well.

Problem 2:

https://colab.research.google.com/drive/1JbV7jzfVw5j0sdGwo79n6Te3LfCwkIR?usp=sharing

Problem 3:

https://colab.research.google.com/drive/1apfjM-yNzaUowqr7fgno83Jnhmdo3Alx?usp=sharing

https://colab.research.google.com/drive/1OiuDM1-gVzjvyEZ0OGfjhmT1uKzOJ0H9?usp=sharing

Problem 4:

https://colab.research.google.com/drive/1fahYw7ZmPFPzdV9XxzcGyPenjJg2XrX?usp=sharing

1 Basics [16 Points]

Answer each of the following problems with 1-2 short sentences.

Question A [2 points]: What is a hypothesis set?

Solution A: *A hypothesis set is the set of all possible candidate hypotheses that could be returned by our machine as the final approximation function.*

Question B [2 points]: What is the hypothesis set of a linear model?

Solution B: *The hypothesis set of a linear model is a set of scalars for which a set of observed data impacts a final prediction. These scalars represent all possible* $\mathbf{w}, b \; s.t. f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$.

Question C [2 points]: What is overfitting?

Solution C: *Overfitting is when we find that the error from the testing set is higher than the error of the training set, often result from trying to find too accurate of an approximation during training.*

Question D [2 points]: What are two ways to prevent overfitting?

Solution D: *Overfitting can be prevented by (1) utilizing a high volume of training data in order to produce the most accurate correlations and (2) minimizing the complexity of the model (i.e. lower degree polynomial approximations).*

Question E [2 points]: What are training data and test data, and how are they used differently? Why should you never change your model based on information from test data?

Solution E: *Training data and testing data are of the same format and distribution, but training data is used to directly develop the model while testing data is used to quantify the error observed in the model's predictions. Models should not be changed based on testing information data as it introduces bias specific to the testing data and may produce overfitting or incorrect classifications.*

Question F [2 points]: What are the two assumptions we make about how our dataset is sampled?

Solution F: *We assume that our dataset is sampled (1) independently (2) and identically distributed.* **

Question G [2 points]: Consider the machine learning problem of deciding whether or not an email is spam. What could X, the input space, be? What could Y, the output space, be?

Solution G: *The input space X could be the words within the email, which can be assigned integer values to form an overall "word array" for ease of computation. The output space Y could be a float value between 0 and 1 for the "percentage of likelihood" that the email is spam.*

Question H [2 points]: What is the k-fold cross-validation procedure?

Solution H: *The k-fold cross-validation procedure involves breaking the complete dataset into training and validation data in different ways (contiguous blocks or random sampling). Cross-validation occurs when a different subset of data is used for training and validation over each iteration.*

2 Bias-Variance Tradeoff [34 Points]

Question A [5 points]: Derive the bias-variance decomposition for the squared error loss function. That is, show that for a model f_S trained on a dataset S to predict a target $y(x)$ for each x,

$$\mathbb{E}_S\left[E_{\text{out}}\left(f_S\right)\right] = \mathbb{E}_x[\text{Bias}(x) + \text{Var}(x)]$$

given the following definitions:

$$\begin{aligned} F(x) &= \mathbb{E}_S\left[f_S(x)\right] \\ E_{\text{out}}(f_S) &= \mathbb{E}_x\left[\left(f_S(x) - y(x)\right)^2\right] \\ \text{Bias}(x) &= \left(F(x) - y(x)\right)^2 \\ \text{Var}(x) &= \mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2\right] \end{aligned}$$

Solution A:

$$\begin{aligned} \mathbb{E}_S\left[E_{out}(f_S)\right] &= \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x) - y(x)\right)^2\right]\right] \\ &= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) + F(x) - F(x) - y(x)\right)^2\right]\right] \\ &= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(F(x) - y(x)\right)^2 + \left(f_S(x) - F(x)\right)^2 + 2(f_S(x) - F(x))(F(x) - y(x))\right]\right] \\ &= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(F(x) - y(x)\right)^2 + \left(f_S(x) - F(x)\right)^2\right] + 2(F(x) - F(x))(F(x) - y(x))\right] \\ &= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(F(x) - y(x)\right)^2\right] + \mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2\right] + 2(0)\right] \\ &= \mathbb{E}_x\left[\left(F(x) - y(x)\right)^2 + \mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2\right]\right] \\ &= \mathbb{E}_x\left[\textit{Bias}(x) + \textit{Var}(x)\right], \textit{ as desired.} \end{aligned}$$

In the following problems you will explore the bias-variance tradeoff by producing learning curves for polynomial regression models.

A *learning curve* for a model is a plot showing both the training error and the cross-validation error as a function of the number of points in the training set. These plots provide valuable information regarding the bias and variance of a model and can help determine whether a model is over– or under–fitting.

Polynomial regression is a type of regression that models the target y as a degree–d polynomial function of the input x. (The modeler chooses d.) You don't need to know how it works for this problem, just know that it produces a polynomial that attempts to fit the data.

Question B [14 points]: Use the provided `2_notebook.ipynb` Jupyter notebook to enter your code for this question. This notebook contains examples of using NumPy's polyfit and polyval methods, and scikit-learn's KFold method; you may find it helpful to read through and run this example code prior to continuing with this problem. Additionally, you may find it helpful to look at the documentation for scikit-learn's learning_curve method for some guidance.

The dataset `bv_data.csv` is provided and has a header denoting which columns correspond to which values. Using this dataset, plot learning curves for 1st–, 2nd–, 6th–, and 12th–degree polynomial regression (4 separate plots) by following these steps for each degree $d \in \{1, 2, 6, 12\}$:

1. For each $N \in \{20, 25, 30, 35, \cdots, 100\}$:
   i. Perform 5-fold cross-validation on the first N points in the dataset (setting aside the other points), computing the both the training and validation error for each fold.
      - Use the mean squared error loss as the error function.
      - Use NumPy's polyfit method to perform the degree–d polynomial regression and NumPy's polyval method to help compute the errors. (See the example code and NumPy documentation for details.)
      - When partitioning your data into folds, although in practice you should randomize your partitions, for the purposes of this set, simply divide the data into K contiguous blocks.
   ii. Compute the average of the training and validation errors from the 5 folds.
2. Create a learning curve by plotting both the average training and validation error as functions of N. *Hint: Have same y-axis scale for all degrees d.*

Solution B:

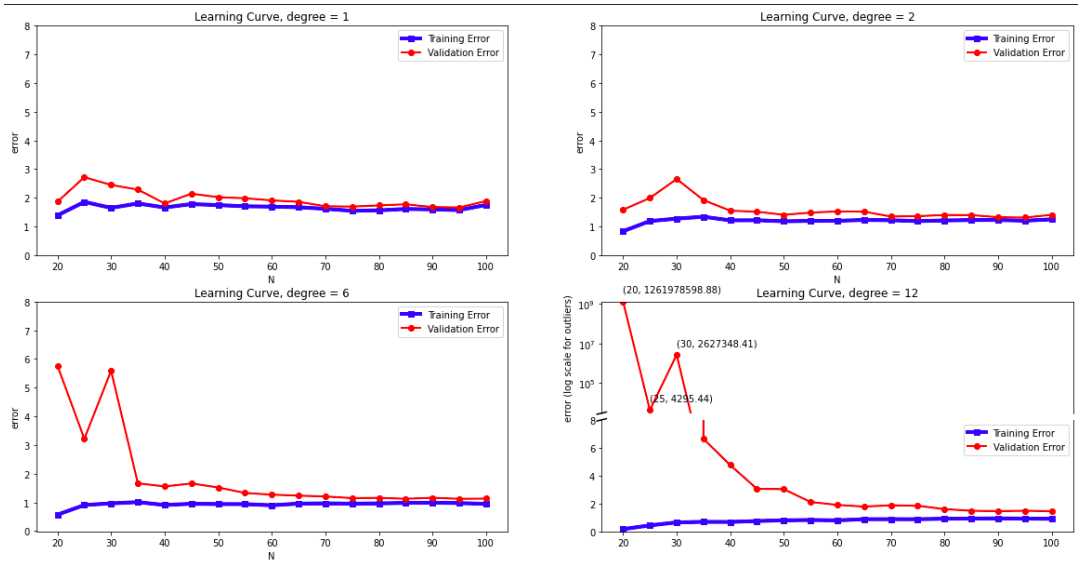

Figure 1: The learning curves across various $n \in N$. The bottom right figure displays all error for when $n \geq 35$ on the same linear scale as the other graphs and all error for when $n \leq 30$ on a logarithmic scale for ease of display.

Colab Link : https://colab.research.google.com/drive/1JbV7jzfVw5j0sdGwo79n

6Te3LfCwkIR?usp=sharing

Question C [3 points]: Based on the learning curves, which polynomial regression model (i.e. which degree polynomial) has the highest bias? How can you tell?

Solution C: *Degree $d = 1$ has the highest bias. This is clear since the validation and training error both converge to a value around 2, which is the highest among all the other hypotheses, and thus the highest bias.*

Question D [3 points]: Which model has the highest variance? How can you tell?

Solution D: *Degree $d = 12$ has the highest variance as the differences between the predicted and actual values for the validation data are orders of magnitude larger than the other models.*

Question E [3 points]: What does the learning curve of the quadratic model tell you about how much the model will improve if we had additional training points?

Solution E: *The learning curve of the quadratic model tells us that if we had additional training points, the model would not improve by much more as the validation and training error converge around $N = 50$ and the bias levels out around $N = 35$.*

Question F [3 points]: Why is training error generally lower than validation error?

Solution F: *Training error is generally lower than validation error since the training data is directly utilized in model fitting, where error between expected and actual values is specifically minimized. However, the training data does not perfectly represent the entire data distribution. The validation data is not directly utilized in the fitting process, and thus can produce more error.*

Question G [3 points]: Based on the learning curves, which model would you expect to perform best on some unseen data drawn from the same distribution as the training data, and why?

Solution G: *Based on the learning curves, the model that I would expect to perform best on unseen data drawn from the same distribution as the training data is the model with $d = 6$ since the bias and variance are the smallest compared to the other models when considering a high number of training points ($N > 80$).*

3 Stochastic Gradient Descent [36 Points]

Stochastic gradient descent (SGD) is an important optimization method in machine learning, used everywhere from logistic regression to training neural networks. In this problem, you will be asked to first implement SGD for linear regression using the squared loss function. Then, you will analyze how several parameters affect the learning process.

Linear regression learns a model of the form:

$$f(x_1, x_2, \cdots, x_d) = \left(\sum_{i=1}^{d} w_i x_i \right) + b$$

Question A [2 points]: We can make our algebra and coding simpler by writing $f(x_1, x_2, \cdots, x_d) = \mathbf{w}^T \mathbf{x}$ for vectors $\mathbf{w}$ and $\mathbf{x}$. But at first glance, this formulation seems to be missing the bias term b from the equation above. How should we define $\mathbf{x}$ and $\mathbf{w}$ such that the model includes the bias term?

Hint: *Include an additional element in* $\mathbf{w}$ *and* $\mathbf{x}$*.*

Solution A: *We can define* $\mathbf{x}$ *such that we introduce a 'dummy' term:* $x^{(0)} = 1$*. We can additionally define* $\mathbf{w}$ *such that we have* $w^{(0)}$ *which represents our bias.*

Linear regression learns a model by minimizing the squared loss function L, which is the sum across all training data $\{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\}$ of the squared difference between actual and predicted output values:

$$L(f) = \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i)^2$$

Question B [2 points]: SGD uses the gradient of the loss function to make incremental adjustments to the weight vector $\mathbf{w}$. Derive the gradient of the squared loss function with respect to $\mathbf{w}$ for linear regression.

Solution B:

$$\begin{aligned} \frac{\partial L(f)}{\partial \mathbf{w}} &= \frac{\partial \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i)^2}{\partial \mathbf{w}} \\ &= \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i) \cdot (2) \cdot (-1) \cdot (\mathbf{x}_i) \\ \frac{\partial L(f)}{\partial \mathbf{w}} &= -2 \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x}_i) \cdot (\mathbf{x}_i) \end{aligned}$$

The following few problems ask you to work with the first of two provided Jupyter notebooks for this problem, `3_notebook_part1.ipynb`, which includes tools for gradient descent visualization. This notebook utilizes the files `sgd_helper.py` and `multiopt.mp4`, but you should not need to modify either of these files.

For your implementation of problems C-E, **do not** consider the bias term.

Question C [8 points]: Implement the `loss`, `gradient`, and `SGD` functions, defined in the notebook, to perform SGD, using the guidelines below:

- Use a squared loss function.
- Terminate the SGD process after a specified number of epochs, where each epoch performs one SGD iteration for each point in the dataset.
- It is recommended, but not required, that you shuffle the order of the points before each epoch such that you go through the points in a random order. You can use `numpy.random.permutation`.
- Measure the loss after each epoch. Your `SGD` function should output a vector with the loss after each epoch, and a matrix of the weights after each epoch (one row per epoch). Note that the weights from all epochs are stored in order to run subsequent visualization code to illustrate SGD.

Solution C: *See code.*
Colab Link : https://colab.research.google.com/drive/1apfjM-yNzaUowqr7fgno83Jnhmdo3Alx?usp=sharing

Question D [2 points]: Run the visualization code in the notebook corresponding to problem D. How does the convergence behavior of SGD change as the starting point varies? How does this differ between datasets 1 and 2? Please answer in 2-3 sentences.

Solution D: *Resulting Plots:*

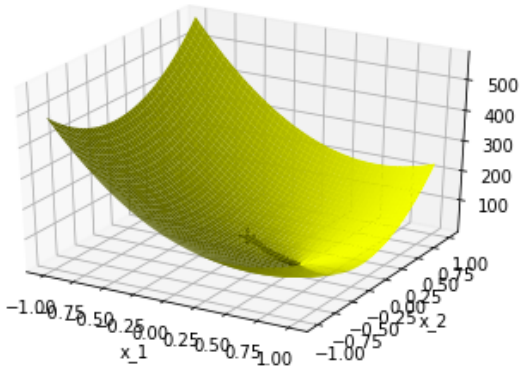

Figure 2: SGD from a single starting point - Dataset 1.

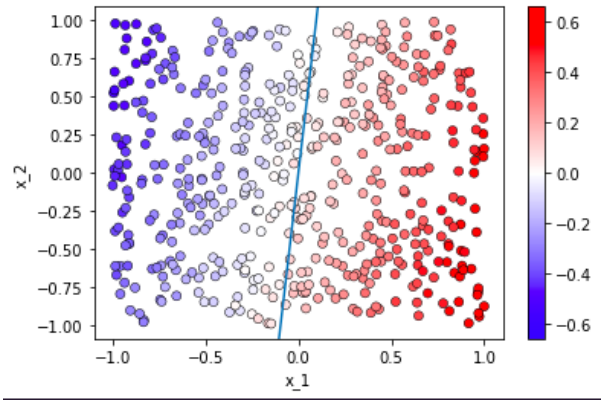

Figure 3: Weight Convergence - Dataset 1.

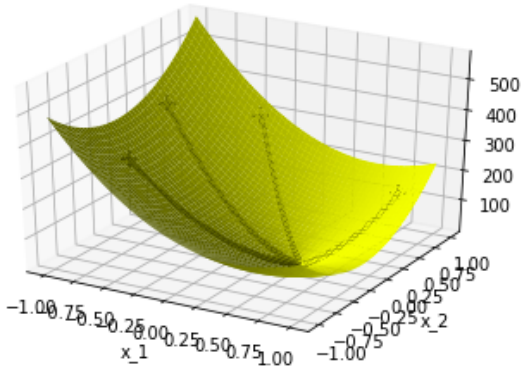

Figure 4: SGD from various starting points - Dataset 1.

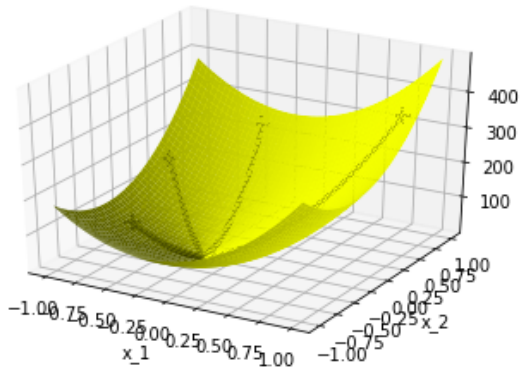

Figure 5: SGD from various starting points - Dataset 2.

We see that the SGD converges to the minimum of the graph no matter where the initial starting point is, at around the same speed as other starting points. This behavior was consistent between both datasets 1 and 2. However, dataset 2 displayed a starting point that was much closer to the local minimum, and this trajectory converged slightly faster than the others.

Question E [6 points]: Run the visualization code in the notebook corresponding to problem E. One of the cells—titled "Plotting SGD Convergence"—must be filled in as follows. Perform SGD on dataset 1 for each of the learning rates $\eta \in \{$1e-6, 5e-6, 1e-5, 3e-5, 1e-4$\}$. On a single plot, show the training error vs. number of epochs trained for each of these values of η. What happens as η changes?

Solution E:

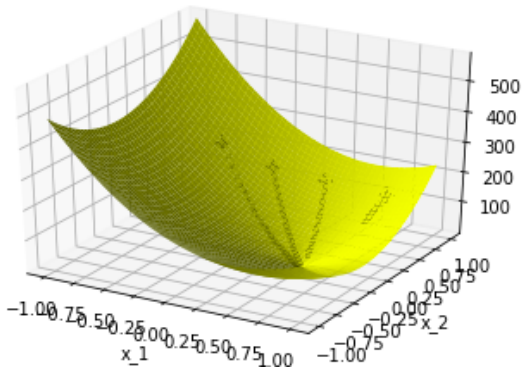

Figure 6: SGD with various step sizes - Dataset 1.

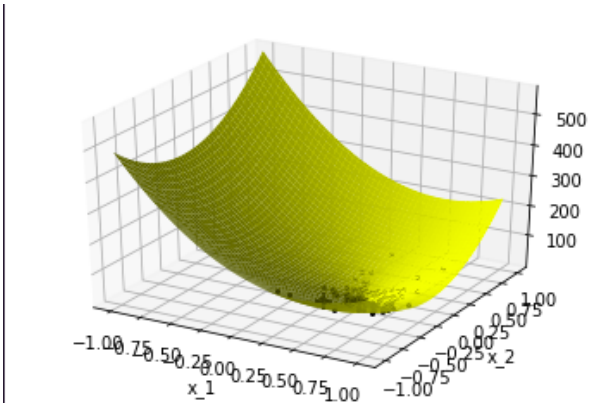

Figure 7: SGD with step size of 1 - Dataset 1.

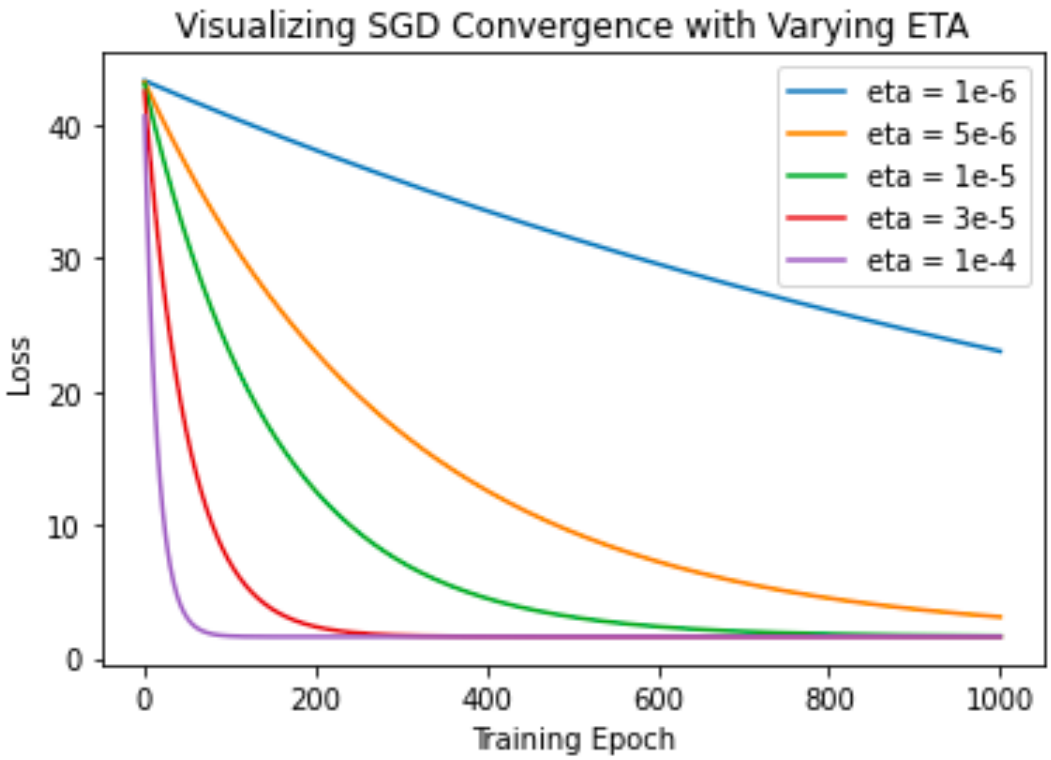

Figure 8: The training loss across various η.

The following problems consider SGD with the larger, higher-dimensional dataset, `sgd_data.csv`. The file has a header denoting which columns correspond to which values. For these problems, use the

---

Jupyter notebook `3_notebook_part2.ipynb`.

For your implementation of problems F-H, **do** consider the bias term using your answer to problem A.

Question F [6 points]: Use your SGD code with the given dataset, and report your final weights. Follow the guidelines below for your implementation:

- Use $\eta = e^{-15}$ as the step size.
- Use $\mathbf{w} = [0.001, 0.001, 0.001, 0.001]$ as the initial weight vector and $b = 0.001$ as the initial bias.
- Use at least 800 epochs.
- You should incorporate the bias term in your implementation of SGD and do so in the vector style of problem A.
- Note that for these problems, it is no longer necessary for the `SGD` function to store the weights after all epochs; you may change your code to only return the final weights.

Solution F: *Final Weights* = $[-0.22813456, -5.94204684, 3.94396165, -11.72378471, 8.78573876]$.

Colab Link : https://colab.research.google.com/drive/1OiuDM1-gVzjvyEZ0OGfjhmT1uKzOJ0H9?usp=sharing

Question G [2 points]: Perform SGD as in the previous problem for each learning rate η in

$$\{e^{-10}, e^{-11}, e^{-12}, e^{-13}, e^{-14}, e^{-15}\},$$

and calculate the training error at the beginning of each epoch during training. On a single plot, show training error vs. number of epochs trained for each of these values of η. Explain what is happening.

Solution G:

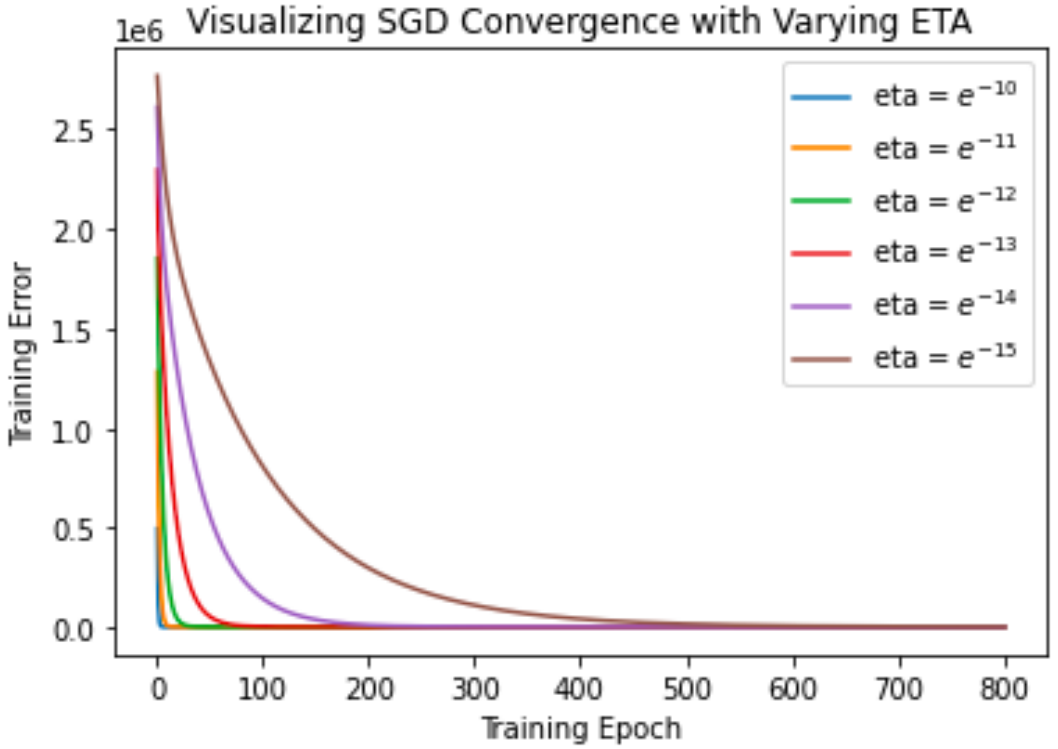

Figure 9: The training error vs. various η – higher-dimensional dataset.

Question H [2 points]: The closed form solution for linear regression with least squares is

$$\mathbf{w} = \left(\sum_{i=1}^{N} \mathbf{x_i} \mathbf{x_i}^T \right)^{-1} \left(\sum_{i=1}^{N} \mathbf{x_i} y_i \right).$$

Compute this analytical solution. Does the result match up with what you got from SGD?

Solution H: *The final weight array produced in linear regression with least squares is* $[-0.316, -5.992, 4.015, -11.933, 8.991]$. *This closely matches the result from SGD, with values varying by at most approximately* 0.2.

Answer the remaining questions in 1-2 short sentences.

Question I [2 points]: Is there any reason to use SGD when a closed form solution exists?

Solution I: *If a closed-form solution exists, it is still important to utilize SGD in order to provide the "best" or a "good" set of weights.*

Question J [2 points]: Based on the SGD convergence plots that you generated earlier, describe a stopping condition that is more sophisticated than a pre-defined number of epochs.

Solution J: *A stopping condition that could be more sophisticated than a pre-defined number of epochs could be once we see that the validation error has plateaued and is no longer decreasing.*

Question K [2 points]: How does the convergence behavior of the weight vector differ between the perceptron and SGD algorithms?

Solution K: *The convergence behavior of the weight vector for the perceptron differs from SGD as it converges in a finite number of steps (given the data is linearly separable), and that it will NOT converge if the data is not linearly separable (introducing a requirement for early stopping).*

4 The Perceptron [14 Points]

The perceptron is a simple linear model used for binary classification. For an input vector $\mathbf{x} \in \mathbb{R}^d$, weights $\mathbf{w} \in \mathbb{R}^d$, and bias $b \in \mathbb{R}$, a perceptron $f : \mathbb{R}^d \to \{-1, 1\}$ takes the form

$$f(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + b\right)$$

The weights and bias of a perceptron can be thought of as defining a hyperplane that divides $\mathbb{R}^d$ such that each side represents an output class. For example, for a two dimensional dataset, a perceptron could be drawn as a line that separates all points of class $+1$ from all points of class -1.

The PLA (or the Perceptron Learning Algorithm) is a simple method of training a perceptron. First, an initial guess is made for the weight vector $\mathbf{w}$. Then, one misclassified point is chosen arbitrarily and the $\mathbf{w}$ vector is updated by

$$\mathbf{w}_{t+1} = \mathbf{w}_t + y(t)\mathbf{x}(t)$$
$$b_{t+1} = b_t + y(t),$$

where $\mathbf{x}(t)$ and $y(t)$ correspond to the misclassified point selected at the t^{th} iteration. This process continues until all points are classified correctly.

The following few problems ask you to work with the provided Jupyter notebook for this problem, titled `4_notebook.ipynb`. This notebook utilizes the file `perceptron_helper.py`, but you should not need to modify this file.

Question A [8 points]: The graph below shows an example 2D dataset. The $+$ points are in the $+1$ class and the $\circ$ point is in the -1 class.

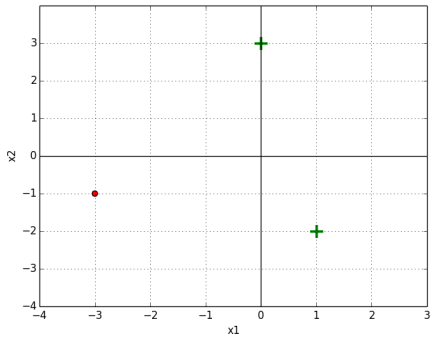

Figure 10: The green $+$ are positive and the red $\circ$ is negative

Implement the `update_perceptron` and `run_perceptron` methods in the notebook, and perform the perceptron algorithm with initial weights $w_1 = 0, w_2 = 1, b = 0$.

Give your solution in the form a table showing the weights and bias at each timestep and the misclassified point $([x_1, x_2], y)$ that is chosen for the next iteration's update. You can iterate through the three points

in any order. Your code should output the values in the table below; cross-check your answer with the table to confirm that your perceptron code is operating correctly.

t	b	w_1	w_2	x_1	x_2	y
0	0	0	1	1	-2	+1
1	1	1	-1	0	3	+1
2	2	1	2	1	-2	+1
3	3	2	0			

Include in your report both: the table that your code outputs, as well as the plots showing the perceptron's classifier at each step (see notebook for more detail).

Solution A:

t	w_1	w_2	b	x_1	x_2	y
0	0	1	0	1	-2	1
1	1	-1	1	0	3	1
2	1	2	2	1	-2	1
3	2	0	3	-	-	-

Figure 11: Weights, bias, and misclassified point at each time step.

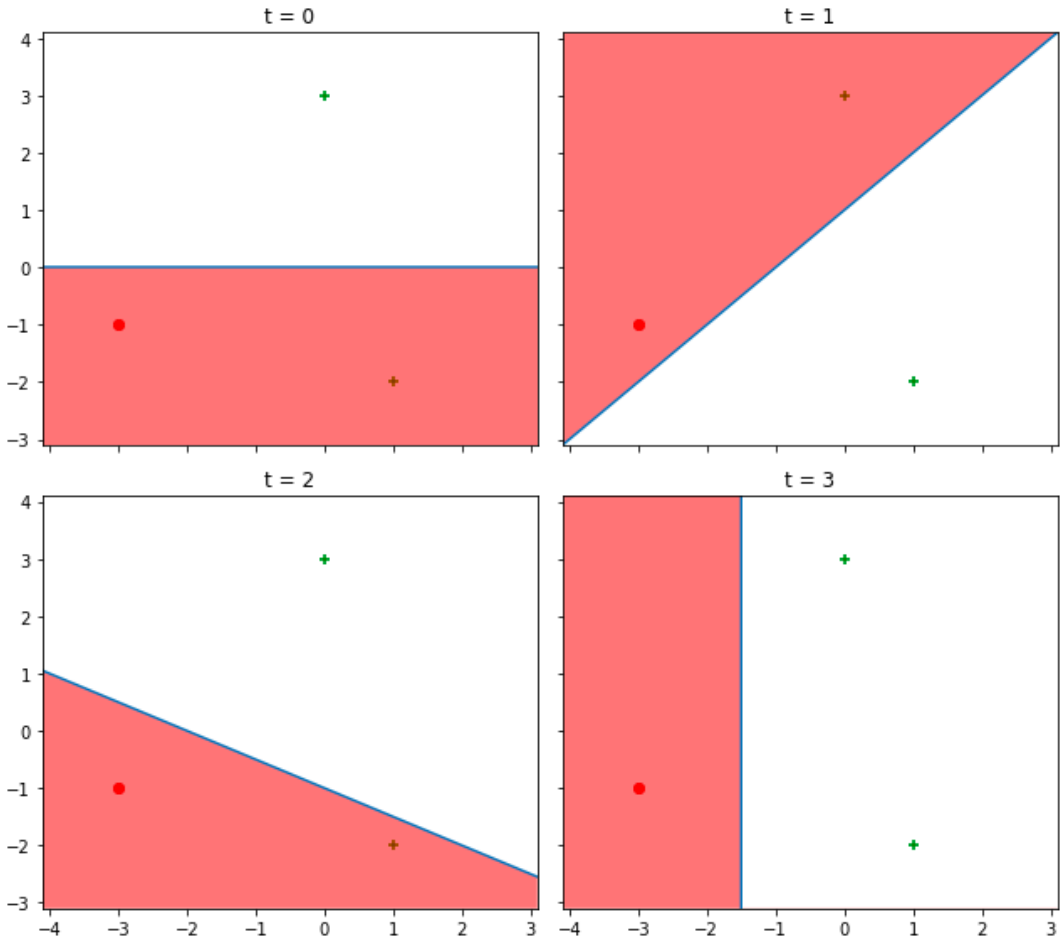

Figure 12: Perceptron classifiers at each step.

Colab Link : https://colab.research.google.com/drive/1fahYw7ZmPFPzdV9XxzcGyPenjJg2Xr_X?usp=sharing

Question B [4 points]: A dataset $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^d \times \mathbb{R}$ is *linearly separable* if there exists a perceptron that correctly classifies all data points in the set. In other words, there exists a hyperplane that separates positive data points and negative data points.

In a 2D dataset, how many data points are in the smallest dataset that is not linearly separable, such that no three points are collinear? How about for a 3D dataset such that no four points are coplanar? Please limit your solution to a few lines - you should justify but not prove your answer.

Finally, how does this generalize for an N-dimensional set, in which **no** $<N$-dimensional hyperplane contains a non-linearly-separable subset? For the N-dimensional case, you may state your answer without proof or justification.

Solution B: *The smallest dataset that is not linearly separable for a 2D dataset is four points (justification). The smallest dataset that is not linearly separable for a 3D dataset is 5 points. Thus, we find that for an N-dimensional set, the smallest dataset that is not linearly separable is $N + 2$ data points in size.*

Question C [2 points]: Run the visualization code in the Jupyter notebook section corresponding to question C (report your plots). Assume a dataset is *not* linearly separable. Will the Perceptron Learning Algorithm ever converge? Why or why not?

Solution C:

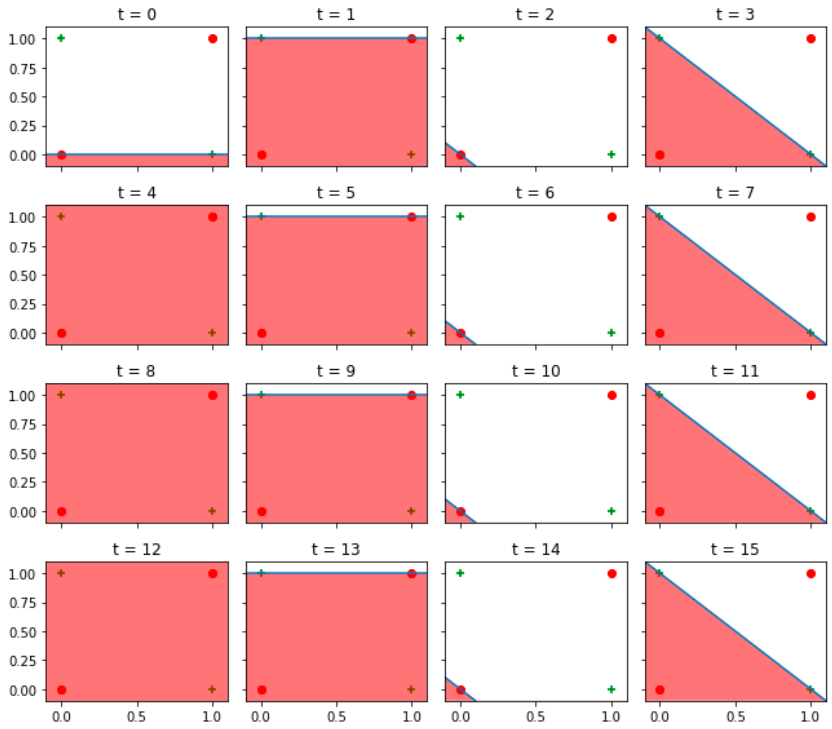

Figure 13: PLA with non-linearly-separable data.

If a dataset is not linearly separable, then the Perceptron Learning Algorithm will not ever converge because the algorithm will repeatedly visit each point in an attempt to classify them, and update the weights, however there will always be at least one point that is not classifiable, creating an infinite loop.