Machine Learning & Data Mining
Set 1

Caltech CS/CNS/EE 155
January 5^{th}, 2024

Policies

- Due 5 PM PST, January 12^{th} on Gradescope.
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- If you have trouble with this homework, it may be an indication that you should drop the class.
- In this course, we will be using Google Colab for code submissions. You will need a Google account.
- You are allowed to use up to a total of 48 late hours throughout the term. Late hours must be used in units of whole hours. Specify the total number of hours you have ever used when turning in the assignment.
- **No use of large language models is allowed.** Students are expected to complete homework assignments based on their understanding of the course material.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code 2P8P28), under "Set 1 Report".
- In the report, **include any images generated by your code** along with your answers to the questions.
- Submit your code by **sharing a link in your report** to your Google Colab notebook for each problem (see naming instructions below). Make sure to set sharing permissions to at least "Anyone with the link can view". **Links that can not be run by TAs will not be counted as turned in.** Check your links in an incognito window before submitting to be sure.
- For instructions specifically pertaining to the Gradescope submission process, see https://www.gradescope.com/get_started#student-submission.

Google Colab Instructions

For each notebook, you need to save a copy to your drive.

1. Open the github preview of the notebook, and click the icon to open the colab preview.
2. On the colab preview, go to File → Save a copy in Drive.
3. Edit your file name to "lastname_firstname_originaltitle", e.g."yue_yisong_3_notebook_part1.ipynb"

1 Basics [16 Points]

Relevant materials: lecture 1

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

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

Solution A:

A hypothesis set is the set of all models that approximate the target function which maps all inputs to outputs. Basically, it is the set of all possible functions over the input and output space that can be candidates for the true target function over that same input and output space.

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

Solution B:

The hypothesis set of a linear model is the set of all possible weights w that create a function/ model of the form $w^T x + b$ to map input x to some output y through a linear combination of the weights w.

Problem C [2 points]: What is overfitting?

Solution C:

Overfitting is the event that the test error $>>$ training error and is implied by high variance in the expected test error. In general terms, this means that the model fit too exact to this training data that it does not accurately generalize conclusions for the entire population/ held out test data.

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

Solution D:

One way to prevent overfitting is to use a validation set/ cross validation to improve the training/ structure of the model by lowering variance of expected test error during validation in order to avoid overfitting on the test data set later on. Alternatively, to lower variance of expected test error, one can also increase the amount of training data used as fitting a model to larger training sets decreases variance and lessens the probability of overfitting the model.

Problem 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 is the data set used to fit the model and develop the model's weights/ parameters while test data is the data set used to evaluate the model's performance after training. You should never change your model based on information/ results from test data because that inherently makes that data training data since you are using that data/ information from that data to develop the model's weights/ parameters as you would with training data, potentially falsely displaying the model as well fit to non-training data since fitting well to the test set by examining its information could create a bias towards that data and increase error on other data in the population.

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

Solution F:

Two assumptions we make about how our dataset is sampled is that it is sampled at random from the entire data population, meaning that we expect no bias present in the dataset itself to deviate from the population data it represents (sampling is random and independent of population distribution). We also assume that our data is sufficiently spread across the distribution of all possible data values (e.g. we do not just sample multiple examples of one singular data value, we see many different data values, so that the model can differentiate data and thus learn).

Problem 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 a vocabulary vector representing the presence of a word in the email by indicating 1 at the index of each word in the vocabulary present and 0 if not. We can have the output space Y be a binary indicator of -1 for not spam and 1 for spam.

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

Solution H:

The k-fold cross-validation procedure is a process used to evaluate model performance given limited data. It splits the data set given into k-groups and trains the model on $k-1$-groups and evaluates on the single group held out computing the models error on that singly held out group. It repeats this procedure a total of k-times, each iteration holding out a new group out of the k-groups that were split and computes the test error on that group

as the test set, the average of these errors is then computed as the test error estimate of the model having trained and tested on many (k) different splits of train and test data to develop a more accurate error estimate.

2 Bias-Variance Tradeoff [34 Points]

Relevant materials: lecture 1

Problem 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:

$$\mathbb{E}_S\left[E_{out}\left(f_S\right)\right] = \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x) - y(x)\right)^2\right]\right]$$

$$\mathbb{E}_S\left[E_{out}\left(f_S\right)\right] = \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x) - F(x) + F(x) - y(x)\right)^2\right]\right]$$

$$\mathbb{E}_S\left[E_{out}\left(f_S\right)\right] = \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2 + 2\left(\left(f_s(x) - F(x)\right)\left(F(x) - y(x)\right)\right)\right]\right]$$

$$\mathbb{E}_S\left[E_{out}\left(f_S\right)\right] = \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2\right]\right] + \mathbb{E}_S\left[\mathbb{E}_x\left[2\left(\left(f_s(x) - F(x)\right)\left(F(x) - y(x)\right)\right)\right]\right]$$

$$\mathbb{E}_S\left[E_{out}\left(f_S\right)\right] = \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2\right]\right] + 0\left(\mathbb{E}_S\left(f_s(x) - F(x)\right) = 0\right)$$

$$\mathbb{E}_S\left[E_{out}\left(f_S\right)\right] = \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2\right] + \left(F(x) - y(x)\right)^2\right]$$

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

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.

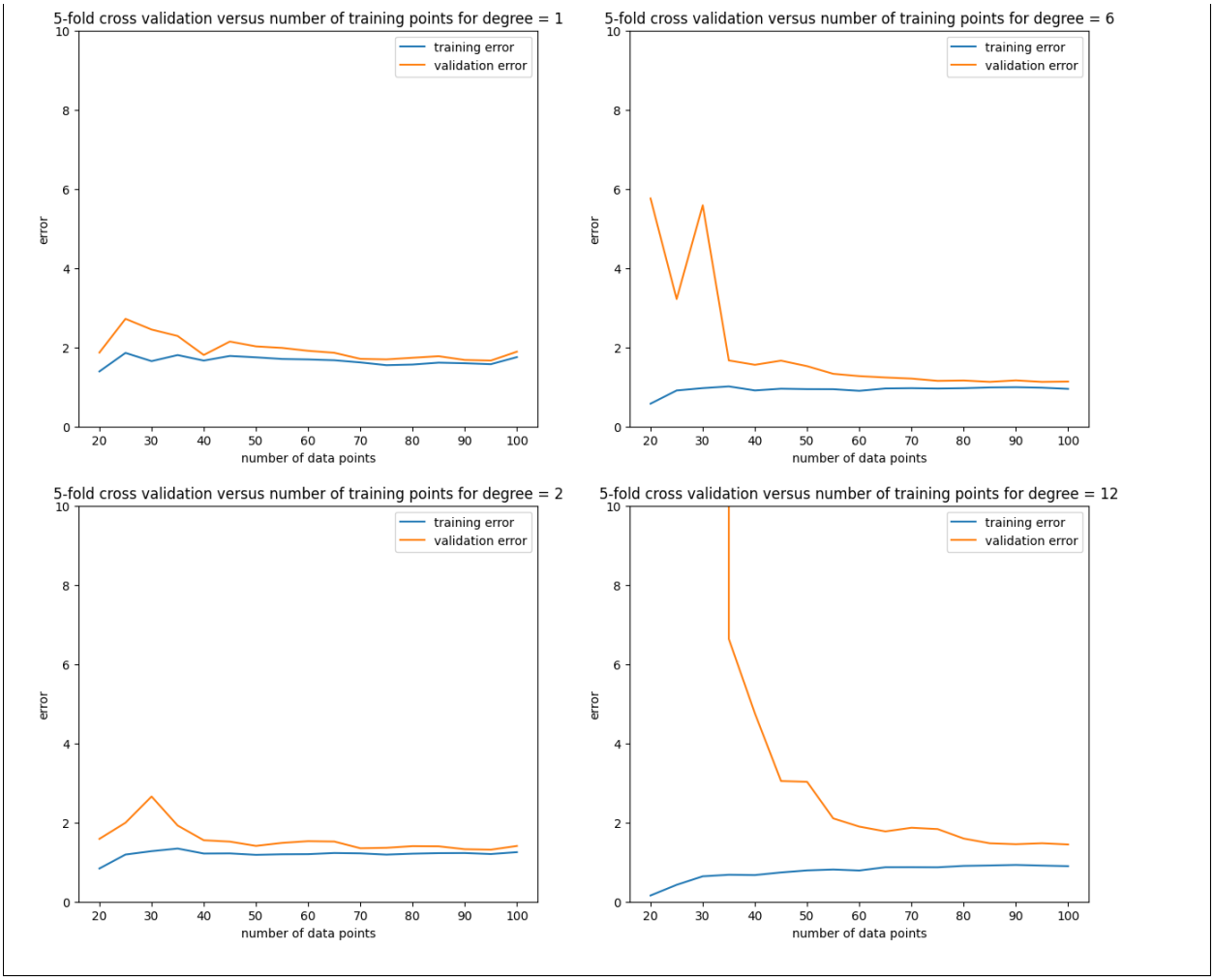
Problem B [14 points]: Use the provided `2_notebook.ipynb` Jupyter notebook to enter your code for this question. This notebook contains examples of using NumPy's polyfit and polyval methods, and scikit-learn's KFold method; you may find it helpful to read through and run this example code prior to continuing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

Problem 2B Code



Problem 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:

Problem 2C Code

Based on the learning curves, the polynomial regression with degree = 1 has the highest bias. This can be identified by the average error that degree model of roughly 2 for both training and validation being larger than the other models even as the number of data points increases suggesting that the model is underfitting the data and thus displaying highest bias.

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

Solution D:

Problem 2D Code

The polynomial fit with degree = 12 has the highest variance which can be seen from the validation and training errors for that model having the highest difference. High variance implies overfitting and we can see in the figure for polynomial with degree 12 that the training data has very low error and the validation has extremely high error (before convergence) which suggests that the data in training is being most overfit and thus should also have the highest variance.

Problem 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:

Problem 2E Code

As the number of data points increases, the validation error seems to slightly decrease towards the training error suggesting that if we have additional training points that the model will generalize better but not to a great extent as beyond roughly 50 points the change in errors for training and validation as quantity of data points increases remains roughly in the same significance level.

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

Solution F: *training error is generally lower than validation error because the models weights/ parameters are being optimized to perform on the training set. Thus there is some bias in the models training towards performing better on the training set since the model has seen that data and is trying to optimize/learn to perform well against that data while it might not do as well against new data (validation) that it hasn't seen yet.*

Problem 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:

Problem 2G Code

I would expect the polynomial fit with degree = 6 to have the best performance on some unseen data drawn from the same distribution as the training data. This is because the average validation error beyond around 60 datapoints is lowest for degree = 6 amongst all models and the validation error curve is decreasing shape suggests

that the model has been fit well and generalizes the distribution of the entire traing data population best.

3 Stochastic Gradient Descent [36 Points]

Relevant materials: lecture 2

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:

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

Problem 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 include an additional x_0 term to x which is set to 1 and have its corresponding element w_0 added to vector w, so that w_0 essentially replicates the use of b in the linear regression model formula as a weight/value for the bias term $x_0 = 1$.*

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:

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

Problem 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:

Since derivatives of vectors can be done the same as with variables, we can differentiate $L(f)$ as if w were a variable:

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

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

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.

Problem 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.*

Problem 3C Code

Problem 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:

Problem 3D Code

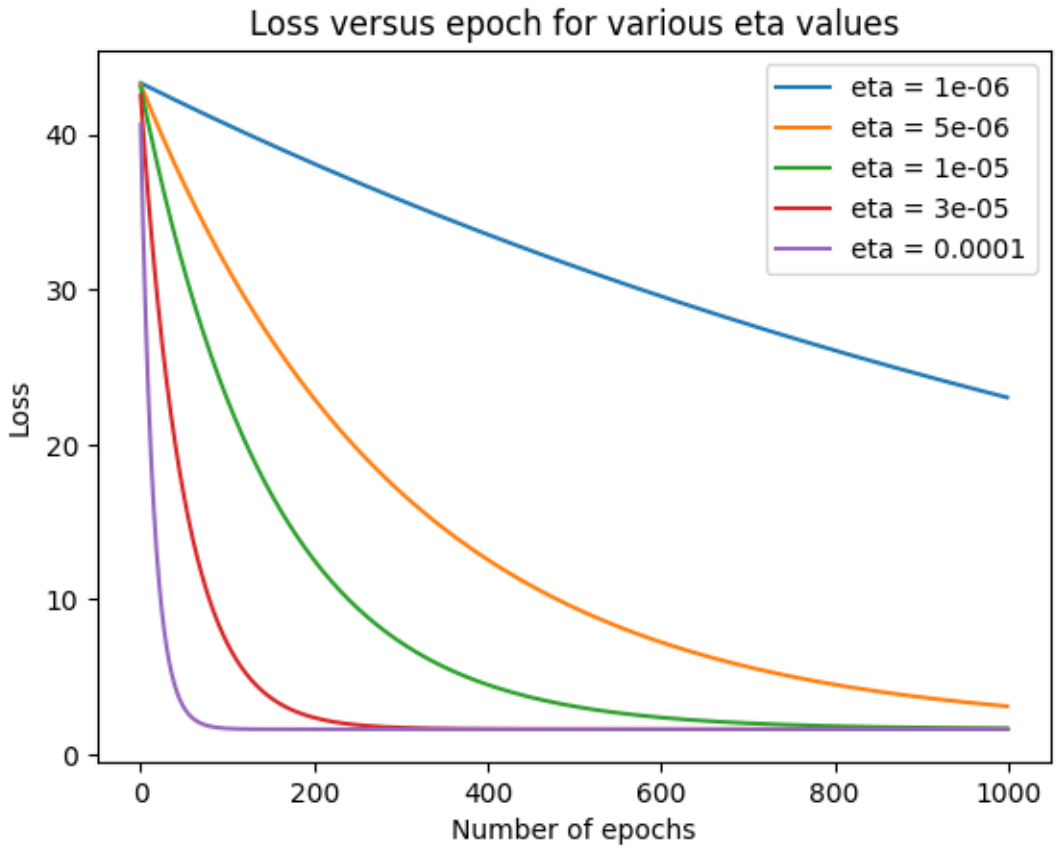
The convergence behavior of the SGD changes with the starting point's variation in that the rate of convergence initially and in early steps is larger for points that are farther away from the optimal minimum will take steeper steps towards convergence than points that initially start out closer. This is further noticeable in the second dataset where on (-.8, -.3) is much closer to the minimum versus (.8, .8) and we can see from the animation that (.8, .8) initially makes much steeper and faster steps in the direction of convergence compared to the (-.8,

-.3) start. All points when approaching very close to minimum do not take very steep steps as the slope towards convergence is much lower but the initialy starting points can be very far away resulting in different initial SGD behavior as farther away points have a steeper initial descent.

Problem 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:

Problem 3E Code



We see from the plot that as η becomes smaller that the convergence of the loss function to a minimum becomes slower. This is likely because the η step size becomes too small for signficant progress to be made in convergence to a minimum loss given the specified number for epochs (1000).

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.

Problem 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:

Problem 3F Code

Final Weights: [-0.22789089213695748, -5.978534423276596, 3.9883932701298366, -11.857003285049679, 8.911300807073154]

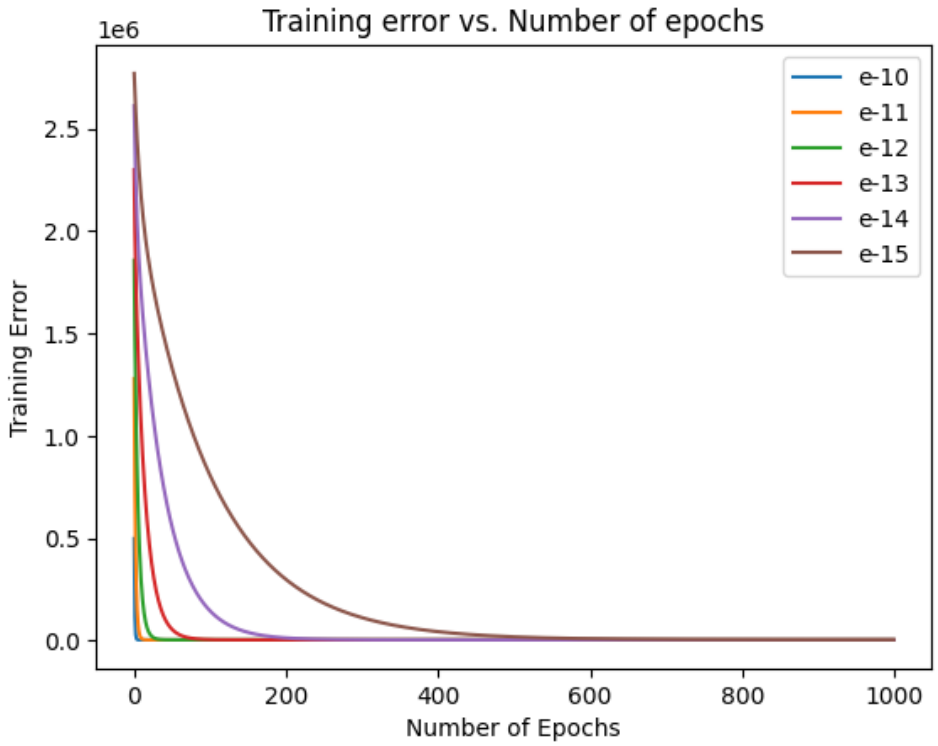
Problem G [2 points]: Perform SGD as in the previous problem for each learning rate η in

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

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:

Problem 3G Code



We notice that the training error over the number of epochs declines at a smaller rate as the η gets smaller. This happens due to small eta values making less significant progress towards a minimum at each epoch because the overall step it is taking in the minimal direction is small.

Problem 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:

Problem 3H Code

Analytical solution weights: [-0.31644251 -5.99157048 4.01509955 -11.93325972 8.99061096]. These weights are pretty similar to the SGD solution, so it matches up fairly well.

Answer the remaining questions in 1-2 short sentences.

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

Solution I:

Yes, it might be better to use SGD versus closed form solution if we are willing to be less accurate for the tradeoff of better computational complexity. The closed form linear regression solution relies on a matrix inversion which with many features can become very computationally expensive where as the gradient descent computational complexity is much cheaper and thus favorable in cases where we want to be computationally more efficient and willing to be a little less accurate.

Problem 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:

From the SGD convergence plots we see the behavior of the convergence to be asymptotic over the number of epochs increasing. Thus, instead of a predefined number of epochs, we can proceed to a new epoch only if the decrease in the loss function from the last epoch to the current epoch was above a certain threshold (essentially testing if the optimization of the loss function hasn't flattened out yet).

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

Solution K:

The perceptron learning algorithm relies on the data being linearly separable which is not always the case, and which will lead the algorithm to never converge as it will continuously make changes point to point but never achieve the full classification for the algorithm to terminate/ converge to a solution (it can sometimes actually get thrown very far off by non-separable data). On the other hand, the SGD algorithm performs a gradient descent which given the convexity of a loss function will definitely converge.

4 The Perceptron [14 Points]

Relevant materials: lecture 2

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.

Problem 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 1: 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.

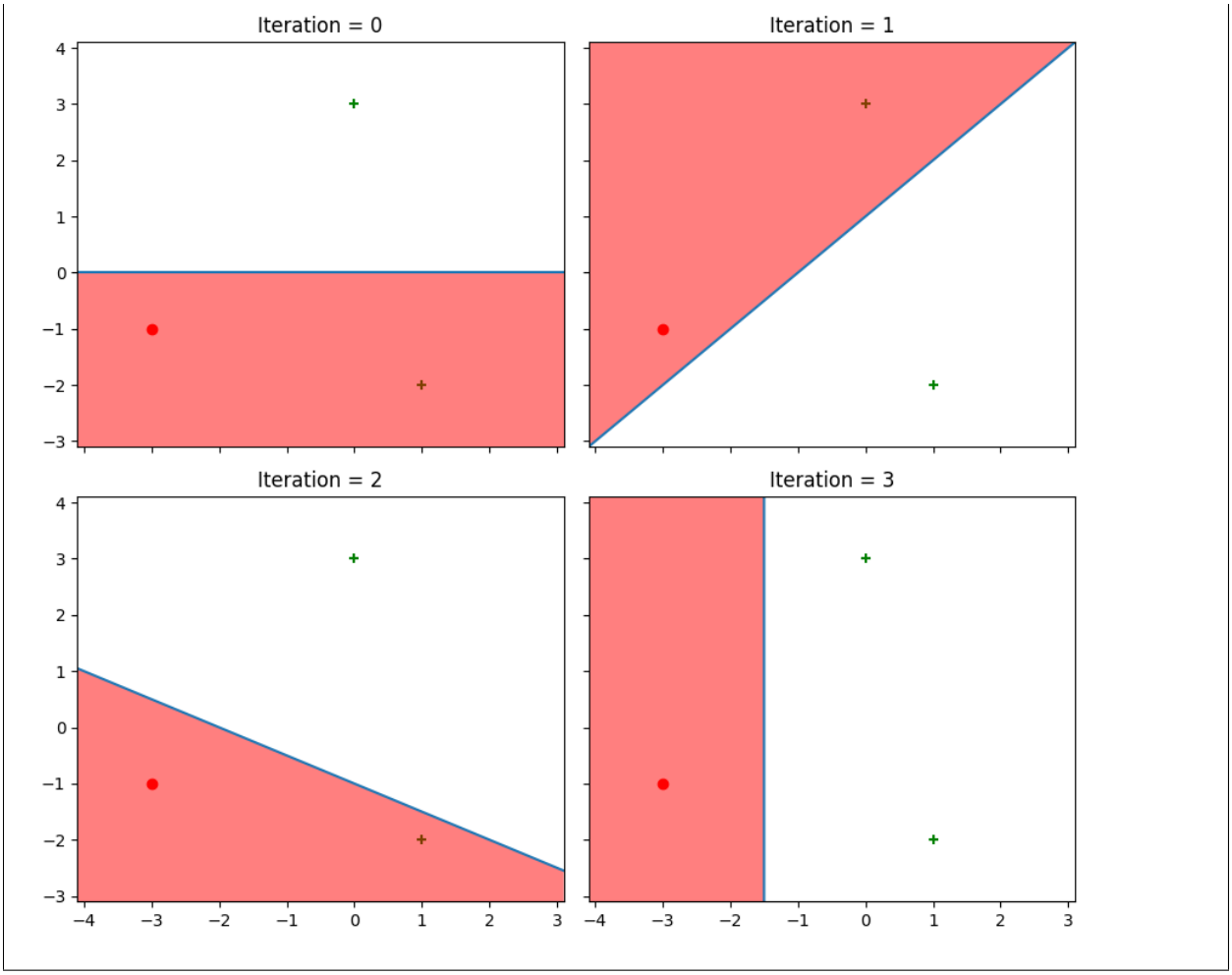
t	b	w_1	w_2	x_1	x_2	y
0	0	0	1	1	-2	+1
1	1	1	-1	0	3	+1
2	2	1	2	1	-2	+1
3	3	2	0			

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:

Problem 4A Code

```
iteration = 1   w1, w2, b = [0. 1.], 0.0        x1, x2 = [ 1 -2]        Y = 1
iteration = 2   w1, w2, b = [ 1. -1.], 1.0      x1, x2 = [0 3]  Y = 1
iteration = 3   w1, w2, b = [1. 2.], 2.0        x1, x2 = [ 1 -2]        Y = 1
iteration = 4   w1, w2, b = [2. 0.], 3.0
final w = [2. 0.], final b = 3.0
```



Problem 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.

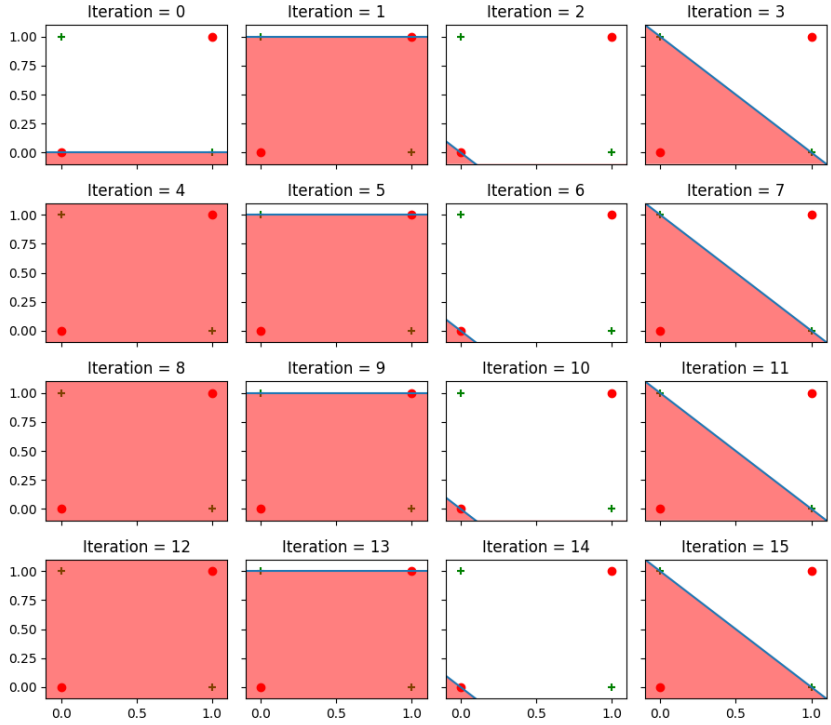
Solution B:

For a 2D dataset 4 points is the smallest non-linearly separable dataset since given no 3 points are collinear we can divide the set into two points each where the two line segments of both sets intersect, and thus if we classify those two sets differently no 2D-linear decision boundary can be made that doesn't cross those line segments and therefore cannot perfectly classify the points. For the 3D case, the smallest dataset is 5 points since we can divide 5 points into 2 sets such as the triangular plane created by 3 of the points (classify as 1 set) and the line segment connecting the other 2 points intersects with that triangular plane. As a result no boundary plane can be made in this configuration that does not cross the plane or the line segment and thus cannot perfectly classify the two points. From here, we see a general case that for an N-dimensional set there is no $<N+2$ dimensional hyperplane that contains a non-linearly-seperable subset.

Problem 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:

Problem 4C Code



No the algorithm will not converge because it will continue to iterate and make changes point by point but each change will still result in another point being misclassified since the data is not linearly separable. As seen by the

plots, this results in the algorithm repeatedly making the same changes in an infinite loop and will thus never converge and will only terminate once it hits the maximum iterations allowed.