Machine Learning & Data Mining
Set 1

Caltech CS/CNS/EE 155
January 6^{th}, 2023
Manuel Rodriguez

Policies

- Due 9 PM PST, January 13th 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.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code K3RPGE), 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: *The set of all possible hypotheses that a given learning model can yield.*

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

Solution B: *The set of all results to $\mathbf{w}^T\mathbf{x} + b$ where $b \in \mathbb{R}$, and $\mathbf{w}$ has the same dimensionality as the input data, $\mathbf{x}$.*

Problem C [2 points]: What is overfitting?

Solution C: *Overfitting occurs when a learning model performs very well on a training sample but poorly out of sample. This means that the model is learning information that is either randomly determined or unlearnable.*

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

Solution D: *Two ways to prevent overfitting are by increasing the size of the training data (to decrease variance) or by decreasing the complexity of the model you are using.*

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 used to choose a hypothesis that is hopefully representative of the real target function that is desired. The test data is used to get an estimate for how good this final hypothesis is at approximating the target function. You can never change your model based on test data information because this makes it so that there is no guarantee on how well this changed hypothesis approximates the target function.*

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

Solution F: *We assume that the dataset's samples are drawn randomly and that they are representative of the target population's distribution.*

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 could be all possible tuples with each entry corresponding to either True or False depending on whether some word is contained in an email. The output space could be a True or False value (0 or 1) which corresponds to whether a given tuple is identified as spam or not.*

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

Solution H: *The k-fold cross-validation procedure involves dividing a training set into training and validation data. The data is divided into N/k groups and for each group division, train on the remaining $N - N/k$ data points and test on the N/ k points. Average the test error across all the groups, and if k is small, then this test error is a good estimate for the out of sample error when training the learning model on the entire data set.*

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:

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

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.

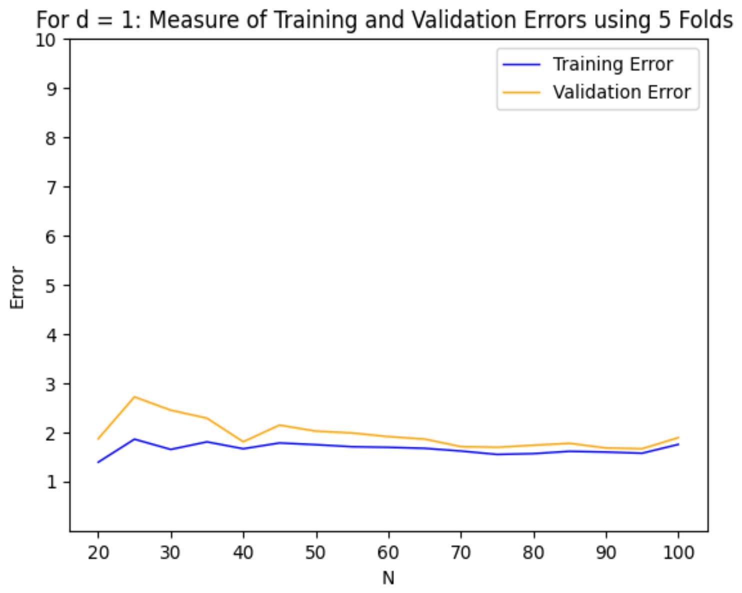
Problem B [14 points]: Use the provided `2_notebook.ipynb` Jupyter notebook to enter your code for this question. This notebook contains examples of using NumPy's polyfit and polyval methods, and scikit-learn's KFold method; you may find it helpful to read through and run this example code prior to continuing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

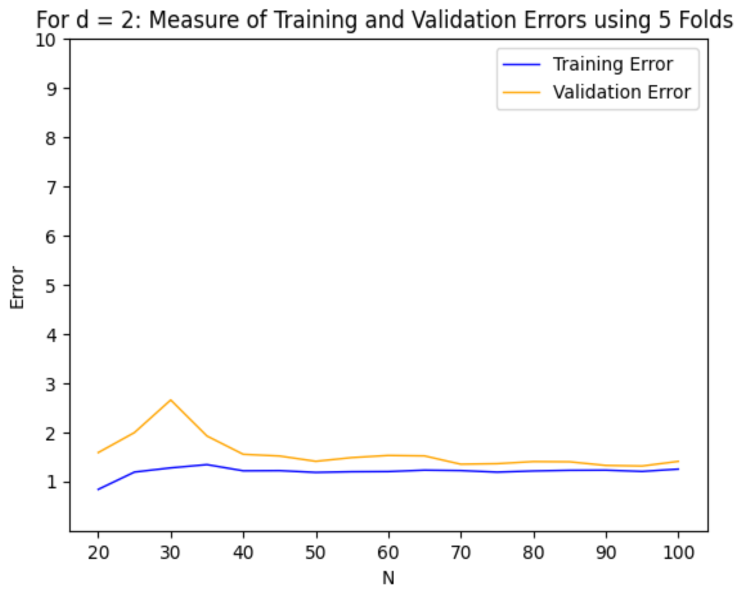
The dataset `bv_data.csv` is provided and has a header denoting which columns correspond to which values. Using this dataset, plot learning curves for 1st–, 2nd–, 6th–, and 12th–degree polynomial regression (4 separate plots) by following these steps for each degree $d \in \{1, 2, 6, 12\}$:

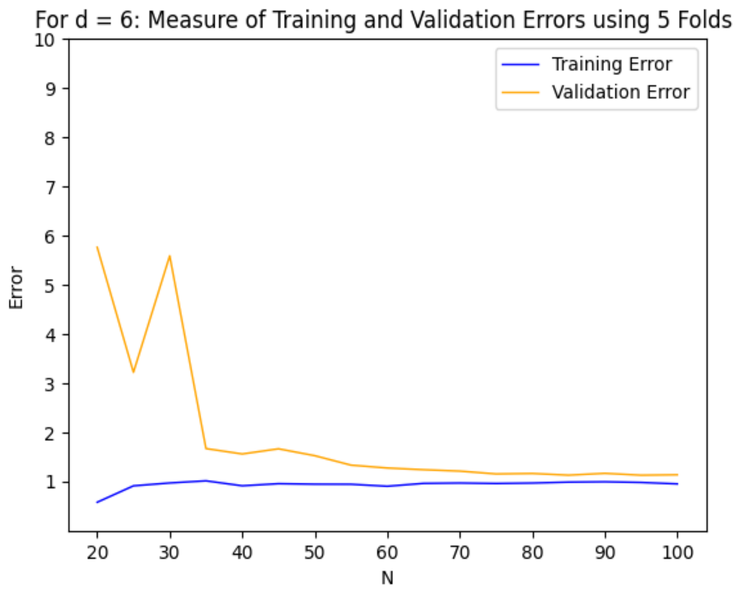
1. For each $N \in \{20, 25, 30, 35, \cdots, 100\}$:
   i. Perform 5-fold cross-validation on the first N points in the dataset (setting aside the other points), computing the both the training and validation error for each fold.
      - Use the mean squared error loss as the error function.
      - 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.

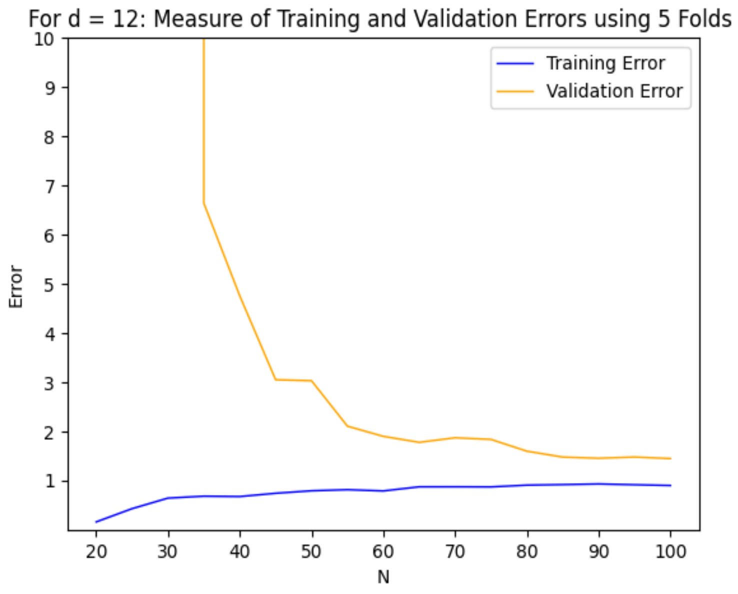
   ii. Compute the average of the training and validation errors from the 5 folds.
2. Create a learning curve by plotting both the average training and validation error as functions of N.
   Hint: Have same y-axis scale for all degrees d.

Solution B:
https://colab.research.google.com/drive/1WkDZSAstOmldYx0A5vaf-Q8HqdtBSgCF?usp=sharing


For d = 1: Measure of Training and Validation Errors using 5 Folds
Training Error
Validation Error
Error
N


For d = 2: Measure of Training and Validation Errors using 5 Folds
Training Error
Validation Error
Error
N





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: *The 1 degree polynomial regression model has the highest bias since it has the highest average training error.*

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

Solution D: *The 12 degree polynomial regression model has the highest variance because it has very high error at small N and only comes near the training error when N is large.*

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: *If the number of points is too low, the training points have a higher probability of being clustered or biased and thus be unrepresentative of the population distribution we are trying to estimate. This is why we see that the bias and variance both go down as we increase the number of data points.*

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 algorithm we use to determine a hypothesis principally tries to minimize the training error of the data set and we hope that the validation error will also be low. Sometimes we can get "lucky" and have the validation error be lower than the training error.*

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: *I would expect the 6th degree polynomial model to perform the best because it seems to have the best bias-variance tradeoff of all four models, with the lowest overall out of sample error. The linear model has very high bias and the d = 12 model has very high variance while the d = 2 model has similar bias and variance, with enough training points, the 6th degree polynomial has a lower error.*

3 Stochastic Gradient Descent [34 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 analyze gradient descent and implement SGD for linear regression using the squared loss function. Then, you will analyze how several parameters affect the learning process.

Problem A [3 points]: To verify the convergence of our gradient descent algorithm, consider the task of minimizing a function f (assume that f is continuously differentiable). Using Taylor's theorem, show that if x' is a local minimum of f, then $\nabla f(x') = 0$.

Hint: *First-order Taylor expansion gives that for some $x, h \in \mathbb{R}^n$, there exists $c \in (0, 1)$ such that $f(x + h) = f(x) + \nabla f(x + c \cdot h)^T h$.*

Solution A:
We will use a particular h to show that $\nabla f(x')^T \nabla f(x') \leq 0 \implies \nabla f(x') = 0$ since the magnitude of a vector must be nonnegative.
Since we know that x' is a local minimum, then for small enough h, $f(x' + h) \geq f(x')$
Using Taylor's Theorem we know: $f(x' + h) = f(x') + \nabla f(x' + c \cdot h)^T h$
Plugging this into the inequality:
$f(x') + \nabla f(x' + c \cdot h)^T h \geq f(x') \implies \nabla f(x' + c \cdot h)^T h \geq 0$
Let $h = -\alpha \nabla f(x')$ s.t. $\alpha \geq 0$ and h is small enough that $f(x' + h) \geq f(x')$
Then, we know that:
$-\nabla f(x' + c \cdot -\alpha \nabla f(x'))^T \alpha \nabla f(x') \geq 0$ *(dividing by $-\alpha$)*
$\nabla f(x' + c \cdot -\alpha \nabla f(x'))^T \nabla f(x') \leq 0$
Taking $\alpha \to 0$:
$\lim_{\alpha \to 0} \nabla f(x' + c \cdot -\alpha \nabla f(x'))^T \nabla f(x') \leq 0$
$\nabla f(x')^T \nabla f(x') \leq 0 \implies \nabla f(x') = 0$ $\square$

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 B [1 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 B: *We should redefine* $\mathbf{x}$ *as having a zeroth term,* $x_0 = 1$, *and redefine* $\mathbf{w}$ *as having a zeroth term,* $w_0 = b$. *This reduces the above formuation to*

$$f(x_1, x_2, \cdots, x_d) = \sum_{i=0}^{d} w_i x_i$$

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 C [2 points]: Both GD and 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. Explain the difference in computational complexity in 1 update of the weight vector between GD and SGD.

Solution C:

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

With GD, the N *in the above equation corresponds to the number of data points whereas with SGD, the* N *refers to some amount of points* $K << N$. *Therefore, GD is of the order of* $O(N)$ *and SGD is of the order of* $O(K)$ *where* N *is the number of data points and* K *is some number typically much smaller than* N.

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 D-F, **do not** consider the bias term.

Problem D [6 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.

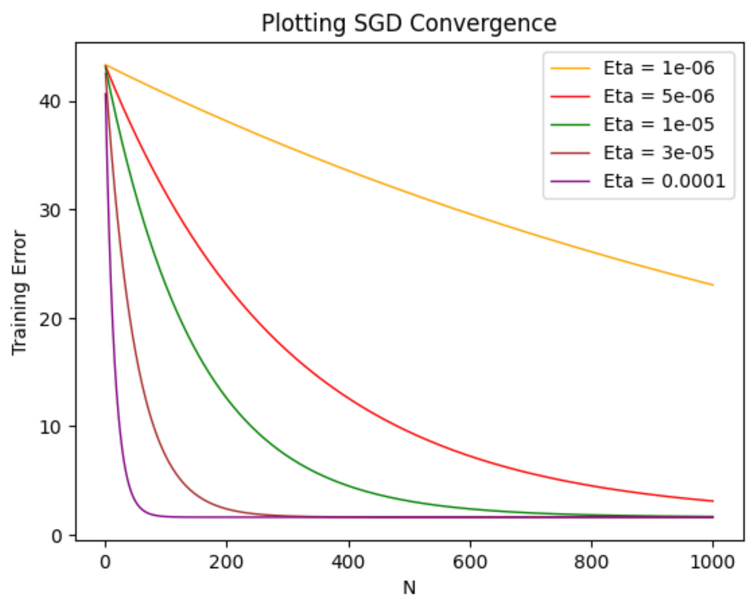
Solution D: *See code.*
https://colab.research.google.com/drive/1aXXdvJSYuKjImEn0hQplw-iI0vsuwUZ9?usp=sharing

Problem E [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 E: *Each starting point varies the path length and trajectory of the PLA convergence. Though it is the case in dataset 1 as well, with dataset 2 it is much clearer that from some starting points PLA converges very quickly to near its optimal value while with others it can take very long and get only slowly near the optimal value.*

Problem F [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 $\eta \in \{10^{-6}, 5 \cdot 10^{-6}, 10^{-5}, 3 \cdot 10^{-5}, 10^{-4}\}$. 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 F:



This shows us that as we increase η, the convergence rate of the SGD increases. However, if η is too big, then it could be the case that each step is overshooting the minimum and that could cause the SGD to diverge.

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 G-I, **do** consider the bias term using your answer to problem A.

Problem G [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.

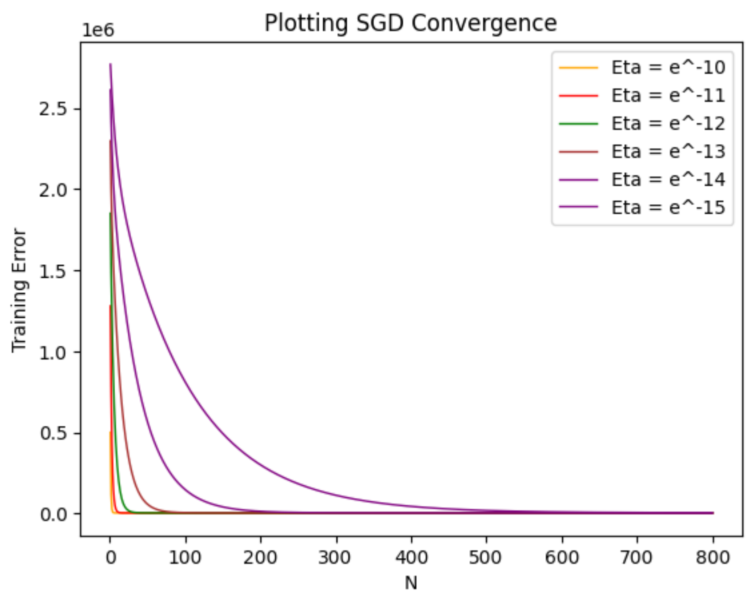
Solution G:
https://colab.research.google.com/drive/1YVETEMhLil3eSG7ax1USlk6xmBWQIL6E?usp=sharing
Using 800 epochs, $b_{final} = -0.22720591$ *and* $w_{final} = [-5.94229011, 3.94369494, -11.72402388, 8.78549375]$

Problem H [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 H:



The η *values are too small and are thus making it so that each epoch barely alters the hypothesis and our weights and bias. This makes it so that the test error remains relatively constant or takes a long time to change.*

Problem I [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 I:
The analytical solution yields: $b = -0.31644251$, *and* $w = [-5.99157048, 4.01509955, -11.93325972, 8.99061096]$. *This does not match what we got with the SGD and that makes because we saw in our plot that for* $\eta = 1e - 15$, *the step size is too small to make any changes to the weight vectors.*

Answer the remaining questions in 1-2 short sentences.

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

Solution J: *It could be better to use SGD because the closed form solution is expensive to compute when the number of examples becomes very large since we would have to compute the inverse of an N x N matrix where N is the number of examples.*

Problem K [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 K: *A more sophisticated method would be to stop the SGD when the difference in training error between the last epoch and the current epoch is less than some* ϵ *or if the average training error between the last, for example, 100 epochs and the most recent 100 epochs is less than some* ϵ.

4 The Perceptron [16 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.

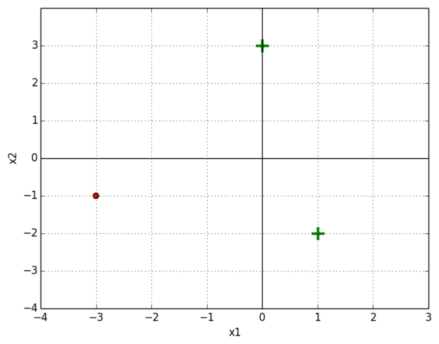


Figure 1: The green $+$ are positive and the red $\circ$ is negative

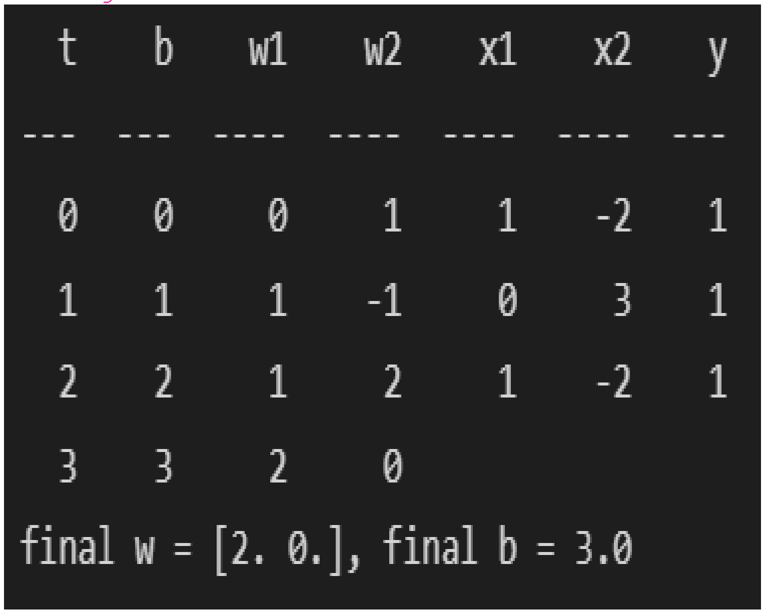
Implement the `update_perceptron` and `run_perceptron` methods in the notebook, and perform the perceptron algorithm with initial weights $w_1 = 0, w_2 = 1, b = 0$.

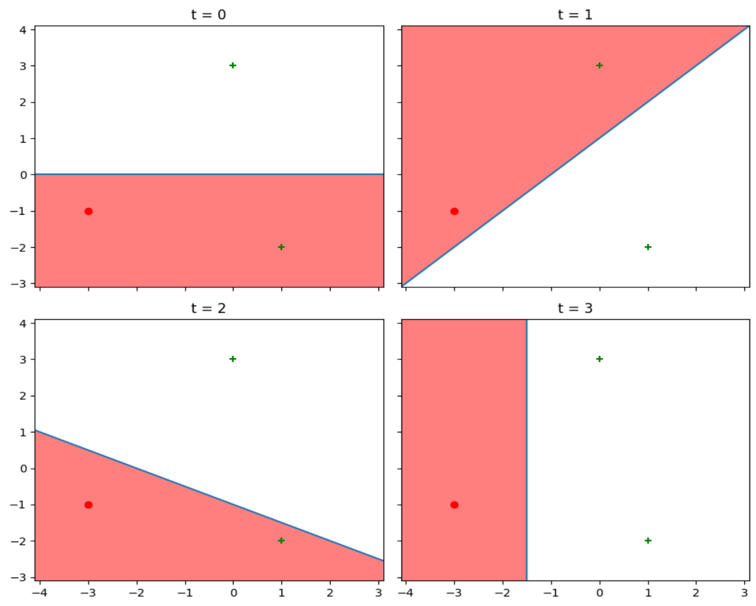
Give your solution in the form a table showing the weights and bias at each timestep and the misclassified point $([x_1, x_2], y)$ that is chosen for the next iteration's update. You can iterate through the three points in any order. Your code should output the values in the table below; cross-check 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:
https://colab.research.google.com/drive/1lyJWhHxZgQuCgGrPS-pXbIXzTZgswxHg?usp=sharing


```
  t    b    w1    w2    x1    x2    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
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 2D space, what is the minimum size of a dataset that is not linearly separable, such that no three points are collinear? How about the minimum size of a dataset in 3D that is not linearly separable, such that no four points are coplanar? Please limit your explanation to a few lines - you should justify but not prove your answer.

Finally, how does this generalize to N-dimension? More precisely, in N-dimensional space, what is the minimum size of a dataset that is not linearly separable, such that no $N + 1$ points are on the same hyperplane? For the N-dimensional case, you may state your answer without proof or justification.

Solution B: *In 2D space, the Perceptron breaks down at 4 points. The configuration of a triangle of points of one classification with one more point of opposing classification in the middle. In this configuration, there is no way to classify the middle point without also classifying one of the triangle points on either side of it incorrectly.*
In 3D space, the Perceptron breaks down at 5 points with the configuration of a triangle of coplanar points of one classification and two more points on opposite sides of the triangle of opposing classification, through the triangle's centroid. Similar to the 2D case, there is no plane that can divide both outer points correctly without misclassifying one of the triangle points on either side of the plane.
This generalizes to that the Perceptron breaks down at N + 2 points in an N-dimensional space, assuming that we are not allowed to have all the points lie on the same hyperplane.

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: *The Perceptron will never converge because it will continue to search for a perfect soluion where there isn't one. It will continue to run in a loop until the user tells it stop manually.*

Problem D [2 points]: How does the convergence behavior of the weight vector differ between the perceptron and SGD algorithms? Think of comparing, at a high level, their smoothness and whether they always converge (You don't need to implement any code for this problem.)

Solution D: *The Perceptron seems to have a rougher convergence behavior than the SGD. Though both methods try to move the weights based on the error of the current hypothesis, the Perceptron uses an all or nothing approach and can often overshoot because it is just changing based on a constant weight (+1 or -1) and a single point. On the other hand, the SGD tries to act based the average behavior. Naturally, this makes the SGD be a bit smoother in transition but also more dependent on the step size that we choose.*
The Perceptron does not always converge because not all data is linearly separable and SGD does not always converge because there could be no minimum to the target function but also because the step size might be too large and cause us to miss a local minimum.