Machine Learning & Data Mining
Set 1 (1/48 hr late)

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

Policies

- Due 9 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.

Submission Instructions

- Submit your report as a single .pdf file to Gradescope (entry code 7426YK), 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 $\rightarrow$ 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 hypothesis set is essentially the set of functions produced when the model that we are using to map our input points to some output assumes all possible parameters.*

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

Solution B: *The hypothesis set of a linear model would be all possible functions $f(x|w) = w^t * x$ where w is any vector such that $w \in \mathbb{R}^D$ where D is the dimension of our input vector x.*

Problem C [2 points]: What is overfitting?

Solution C: *Overfitting occurs when you try to fit your model to the training data such that the resulting model has a somewhat small error with the training data but a very large amount of error with the test data.*

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

Solution D: *The best way to prevent overfitting is to reduce your variance. There are two ways to do this: we can reduce our variance by using a simpler model, or we can reduce our variance by using more training data.*

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 data that we use to determine the best parameters for our model such that we minimize the error between outputs given by our model by taking in the training input and the actual training output. Test data is the data that we use to evaluate the model, and we do so by taking the error between the outputs given by our model when it takes in the test inputs and the actual test outputs. Test data is essentially data that our model has not seen, and evaluating in our model with the test data allows us to see if our model performs well in new scenarios. The issue with changing our model based off of information from our test data is that doing so is essentially turning our test data into more training data, and while the error for the training and the stolen test data may be low, we can't conclude that the model will extrapolate to new inputs or situations.*

Problem 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 independent and identically distributed from our probability distribution $P(x, y)$ where $P(x, y)$ is all the possible data given to us*

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 d-dimensional vector $< x_1, x_2, x_3..x_d >$ where d is the number of words in our dictionary and each element in x is either 0 or 1. Element x_i would refer to whether the ith dictionary word is present in our input text. Our output space Y would be 0/1 where 0 would classify our input as not spam, and 1 would classify our output as spam.*

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

Solution H: *The k-fold cross-validation procedure involves splitting our original data into k parts, and we train our model on k-1 parts and leave the last part for test data. We do this for every choice of k-1 parts and we take the average of our test/validation errors and evaluate our model on this.*

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: *We start with:*

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

We can take the $\mathbb{E}_s$ with respect to both sides and switch the order of expectation to obtain:

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

We can then focus on $\mathbb{E}_s\left[\left(f_S(x) - y(x)\right)^2\right]$. We know that the average hypothesis $F(x) = \mathbb{E}_S\left[f_S(x)\right]$. We can then rewrite our expectation as:

$$\mathbb{E}_s\left[\left(f_S(x) + F(x) - F(x) - y(x)\right)^2\right]$$

Which expands out to:

$$\mathbb{E}_s\left[\left(f_S(x) - F(x)\right)^2 + \left(F(x) - y(x)\right)^2 + 2(f_S(x) - F(x))(F(x) - y(x))\right].$$

We can rewrite the expression above to to obtain:

$$\mathbb{E}_s\left[\left(f_S(x) - F(x)\right)^2\right] + \mathbb{E}_s\left[\left(F(x) - y(x)\right)^2 + 2(f_S(x) - F(x))(F(x) - y(x))\right].$$

We can split the expectations to get:

$$\mathbb{E}_s\left[\left(f_S(x) - F(x)\right)^2\right] + \mathbb{E}_s\left[\left(F(x) - y(x)\right)^2\right] + \mathbb{E}_s\left[2(f_S(x) - F(x))(F(x) - y(x))\right].$$

$\mathbb{E}_s\left[(F(x)-y(x))^2\right] = (F(x)-y(x))^2$ *since* $F(x)$ *is already an average function over* S *and since* $y(x)$ *is independent of* S*. Thus, we can substitute to obtain:*

$$\mathbb{E}_s\left[(f_S(x)-F(x))^2\right] + (F(x)-y(x))^2 + \mathbb{E}_s\left[2(f_S(x)-F(x))(F(x)-y(x))\right].$$

. If we expand the last expectation, we get:

$$2 * E_s\left[f_s(x)F(x) - f_s(x)y(x) - F(x)^2 - F(x)y(x)\right] = 0.$$

Thus, we are left with:

$$\mathbb{E}_s\left[(f_S(x)-F(x))^2\right] + (F(x)-y(x))^2$$

If we sub in $\mathrm{Bias}(x) = (F(x)-y(x))^2$ *and* $\mathrm{Var}(x) = \mathbb{E}_S\left[(f_S(x)-F(x))^2\right]$*, we obtain:*

$$\mathbb{E}_s\left[(f_S(x)-y(x))^2\right] = \mathrm{Bias}(x) + \mathrm{Var}(x)$$

Thus:

$$\mathbb{E}_{out}(f_S) = \mathbb{E}_x\left[\mathbb{E}_s\left[(f_S(x)-y(x))^2\right]\right] = \mathbb{E}_x\left[\mathrm{Bias}(x) + \mathrm{Var}(x)\right].$$

And so we are done.

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 $d \in \{1, 2, 6, 12\}$:

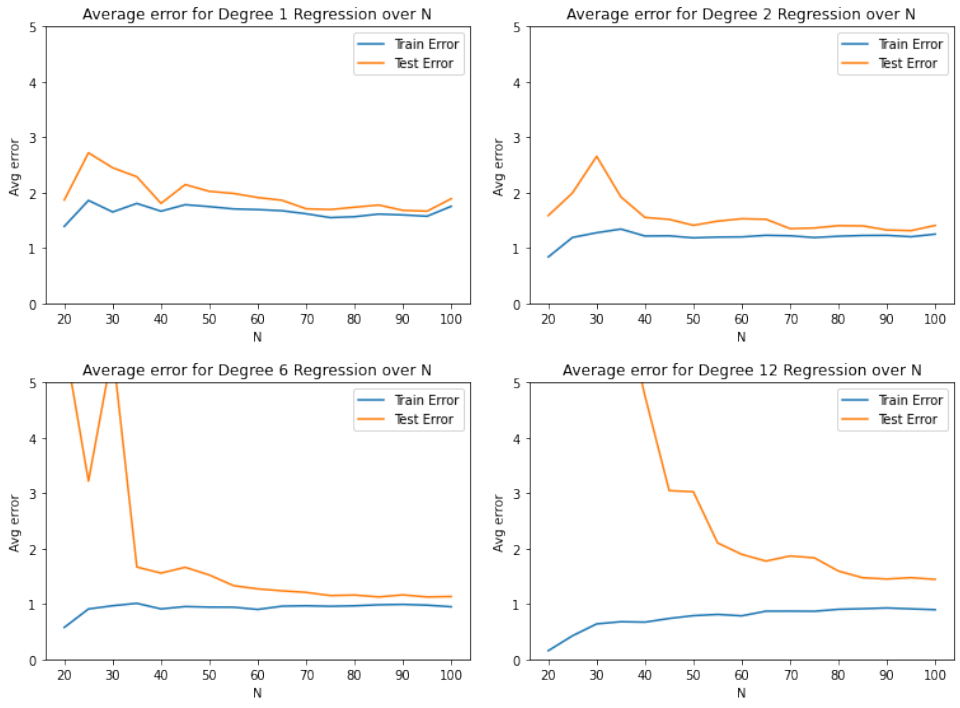
1. For each $N \in \{20, 25, 30, 35, \cdots, 100\}$:

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:



Link to code: https://colab.research.google.com/drive/1koyK3xP9G4Rkg6w966l7ev23wBcoN4bT?usp=sharing

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 first degree polynomial would have the highest bias because there is definitely a bit of underfitting as the training error is higher in the plot of the first degree polynomial than the other plots*

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

Solution D: *The 12th degree polynomial has the highest variance because there is overfitting since the training error in the plot seems to be very low, but the actual test error seems to be very high, thus implying that the degree 12 polynomial fit the training points well, but did not extrapolate well to test data.*

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: *It seems like adding additional training points past around N=60 to 70 would not improve the model very much as the decrease in the test error seems to level off and decrease at a very slow rate past that.*

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

Solution F: *Training error is typically lower than validation error because we train our models and set our parameters with the intention of minimizing the error between the model's output given the training inputs and the actual training output. The validation error on the other hand would be expected to be higher because it is the error calculated when the model evaluates outputs for inputs it has not seen before.*

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 model of degree 6 to perform the best because it seems to have the lowest training and lowest validation error for high N.*

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 define* x *as* $(1, x_1, x_2, \cdots, x_d)$ *and* w *as* $(w_0, w_1, w_2, \cdots, w_d)$*. That way,* $\mathbf{w}^T \mathbf{x} = w_0 + x_1 w_1 + x_2 w_2 + \cdots + x_d w_d$*. In this case, our* w_0 *would be our bias term.*

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: *We can take the derivative of the loss function with respect to the weights. Doing so gives:*

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

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: *Link: https://colab.research.google.com/drive/1MZ15tqVFjR3xL8t2GTASMCyoZ4XlWvlp?usp=sharing*

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:

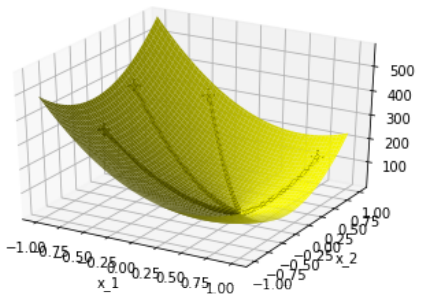


Figure 1: This is for the first dataset

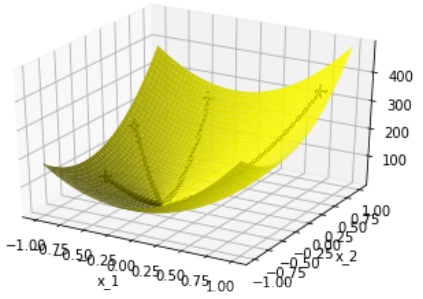
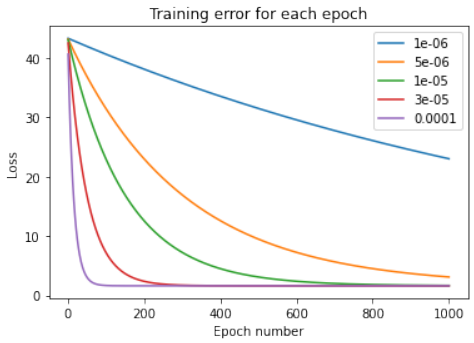


Figure 2: This is for the second dataset

The convergence behavior does change as the starting point varies as the the weight vector takes different paths to reach the same global minimum. This observation seems to be consistent with the second dataset as well.

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:



As η increases, the training error converges to 0 sooner. We can see that the for low step size (blue in the graph), it takes over a 1000 epochs to converge, while for a higher step size (purple), it gets very close to 0 in less than 200 epochs.

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: *Running the SGD code gave: [-0.22712376 -5.94204908 3.94398075 -11.72380433 8.78574124]*

Thus our bias is -0.22712376, and our weight vector is [-5.94204908 3.94398075 -11.72380433 8.78574124].

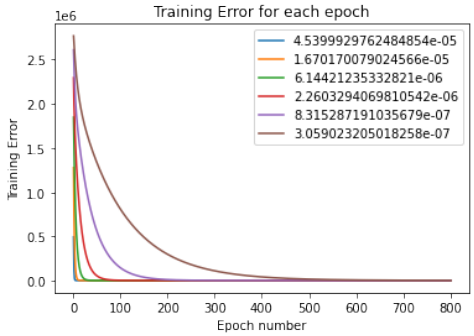
Link: https://colab.research.google.com/drive/17bzWlhacfKHnolu3zE8BhZNN550472OA?usp=sharing

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:



We can see that the larger the learning rates, the faster the training error converged to 0. We can observe that the largest two learning rates e^{-15} and e^{-14} in particular converged to 0 within the first few epochs.

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: *Computing the closed form gave: [-0.31644251, -5.99157048, 4.01509955, -11.93325972, 8.99061096]*

Thus our bias is -0.31644251, and our weight vector is [-5.99157048, 4.01509955, -11.93325972, 8.99061096].

While the result isn't exactly the same as what we got from SGD, it is very close.

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, SGD would be much faster than the closed form solution for very large data sets. The reason for this is because the closed form involves matrix inversion which can be very costly and time expensive for large data sets. On the other hand, SGD makes it's updates as it iterates through each point in the data set, so it can get close to the global minimum much more quickly.*

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: *Our stopping condition would involve stopping when the weight vector doesn't have any improvements to make. This occurs when the error stops changing, which we can compute by taking a ratio of the error of the current iteration and the previous iteration and seeing if it is close to 1 for a consistent period of time.*

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

Solution K: *The convergence of the SGD algorithm is smoother than the perceptron model. The reason for this is because the perceptron model can have either no loss or some loss at each iteration, while the SGD algorithm's loss decreases in a continuous manner. The SGD algorithm could go on infinitely, while the perceptron model stops of the data is linearly separable and if all the points are classified correctly. If the data is not linearly separable, then the perceptron algorithm fails and runs indefinitely.*

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.

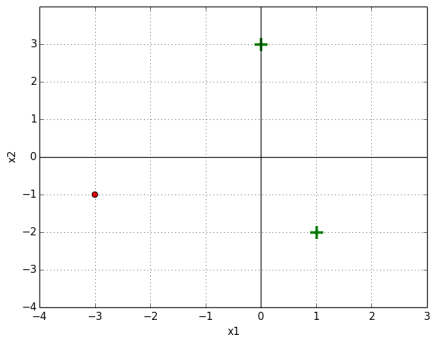


Figure 3: The green $+$ are positive and the red $\circ$ is negative

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:  0
Weights:  [ 1. -1.]
Bias:  1.0
Misclassified x:  [ 1 -2]
Misclassified y:  1
t:  1
Weights:  [1. 2.]
Bias:  2.0
Misclassified x:  [0 3]
Misclassified y:  1
t:  2
Weights:  [2. 0.]
Bias:  3.0
Misclassified x:  [ 1 -2]
Misclassified y:  1
t:  3
Weights:  [2. 0.]
Bias:  3.0

final w = [2. 0.], final b = 3.0
```

Figure 4: These are the table values outputted by the code

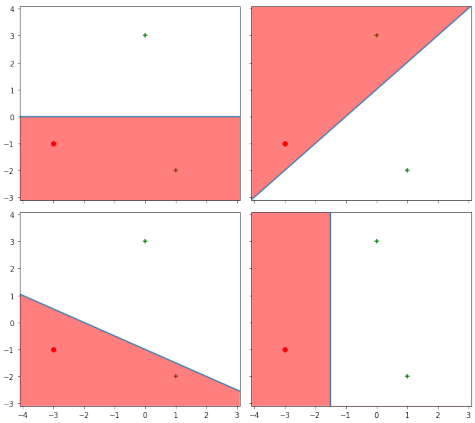


Figure 5: These are the plots showing the perceptron classifier at each step

Link to code: https://colab.research.google.com/drive/1wirDlgWZDfzPvIsgpdHRBhEPXFYV66kb?usp=sharing

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: *In the 2D data set, you can have 4 data points in the smallest data set that is not linearly separable. We can do this by imagining a line and having two points of the same classification be on the same line, and two point of the second classification on separate sides of the line*

In a 3D data set, you can imagine keeping 3 points of the same classification on a plane, and two point of the second classification on opposite sides of the plane. We see that we cannot have a plane separate out the two points that are on either side from the 3 points that are already coplanar.

In the ND data set, you can see that there are N+2 points in the smallest data set that is not linearly separable.

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 preceptron learning algorithm will never converge because there will always be at least one misclassified point. You can see this in the plots below for 16 iterations of the perceptron algorithm:*