Policies

- Due 9 PM PST, January 12th 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 → 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]

lecture 1

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

[2] What is a hypothesis set?

Solution : *A hypothesis set is the set of ways to map inputs to outputs. This set of mappings will depend on the model being used.*

[2] What is the hypothesis set of a linear model?

Solution : *For a linear model, $f(x|w,b) = w^t x + b$, the hypothesis set is the set of all possible w and b values. This is equivalent to all possible lines on a graph.*

[2] What is overfitting?

Solution : *Overfitting is when you model the training data too well so that it doesn't generalize to to testing data well. This could from an overly complex model and is an issue because the model is fitting noise in the data rather than the underlying trend.*

[2] What are two ways to prevent overfitting?

Solution : *One way of preventing overfitting is to use split your training data into training and validation sets and use test error from the validation sets select a model that isn't overfitting. Another way of preventing overfitting is to do cross-validation to improve the estimate of the test error and select a model that isn't overfitting.*

[2] 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 : *Training data and test data are both generated from the same data that we are trying to model, but where training data is used to fit our model, test data is used to understand how well our model works. We never change your model based on information from test data, because it will bias the model to the test data, and the test data will no longer be able to tell us how well our model will work in reality.*

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

Solution : *Two assumptions that we make about our dataset is that there is a "true" probability distribution over all possible data and that all of our training datapoints are sampled i.i.d., independent and identically distributed, from this probability distribution.*

[2] 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 : *The input space, X, could be a bag of the keywords used in the subject line or in the email itself, such as "homework" (probably not spam) or "Nigerian"/"Prince" (probably spam). The output space, Y, could be the classification of the email as "spam" or "not spam".*

[2] What is the k-fold cross-validation procedure?

Solution : *To perform k-fold cross-validation, we first split original dataset into k equal subsets. We then want to train a model on $k-1$ of the subsets of data and test on the remaining subset. We will then repeat for every choice of the $k-1$ parts and average the validation errors, improve the estimate of the test error.*

2 Bias-Variance Tradeoff [34 Points]

lecture 1

[5] 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:

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

Solution :

$$\begin{aligned}
\mathbb{E}_S\left[E_{out}\left(f_S\right)\right] &= \mathbb{E}_S\left[\mathbb{E}_x\left[\left(f_S(x)-y(x)\right)^2\right]\right] && (1)\\
&= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x)-y(x)\right)^2\right]\right] && (2)\\
&= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x)-F(x)+F(x)-y(x)\right)^2\right]\right] && (3)\\
&= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x)-F(x)\right)^2+\left(F(x)-y(x)\right)^2+2\left(f_S(x)-F(x)\right)\left(F(x)-y(x)\right)\right]\right] && (4)\\
&= \mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x)-F(x)\right)^2\right]+\left(F(x)-y(x)\right)^2\right] && (5)\\
&= \mathbb{E}_x\left[\text{Var}(x)+\text{Bias}(x)\right] && (6)\\
&= \mathbb{E}_x[\text{Bias}(x)+\text{Var}(x)] && (7)
\end{aligned}$$

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.

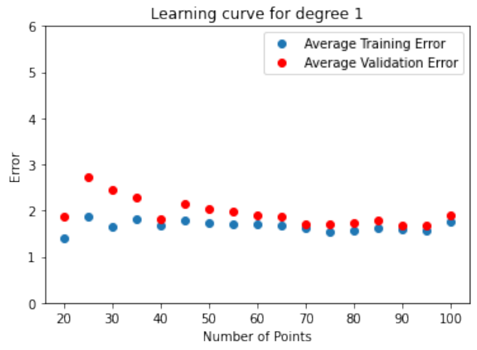
[14] Use the provided `2_notebook.ipynb` Jupyter notebook to enter your code for this question. This notebook contains examples of using NumPy's polyfit and polyval methods, and scikit-learn's KFold method; you may find it helpful to read through and run this example code prior to continuing with this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

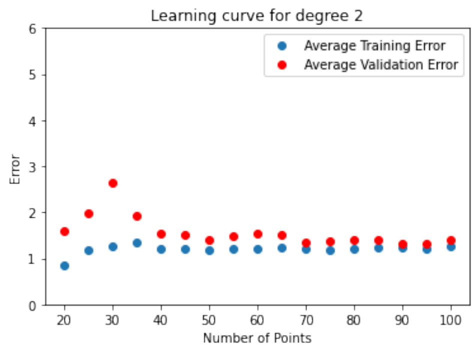
The dataset `bv_data.csv` is provided and has a header denoting which columns correspond to which values. Using this dataset, plot learning curves for 1st–, 2nd–, 6th–, and 12th–degree polynomial regression (4 separate plots) by following these steps for each degree $d \in \{1, 2, 6, 12\}$:

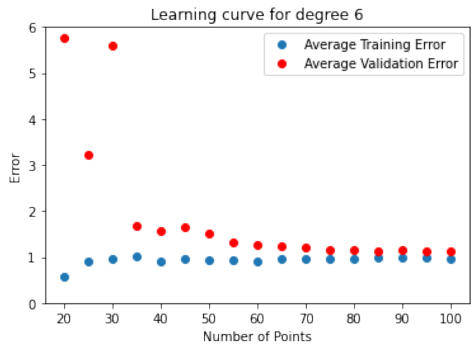
1. For each $N \in \{20, 25, 30, 35, \cdots, 100\}$:
   i. Perform 5-fold cross-validation on the first N points in the dataset (setting aside the other points), computing the both the training and validation error for each fold.
      - Use the mean squared error loss as the error function.
      -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

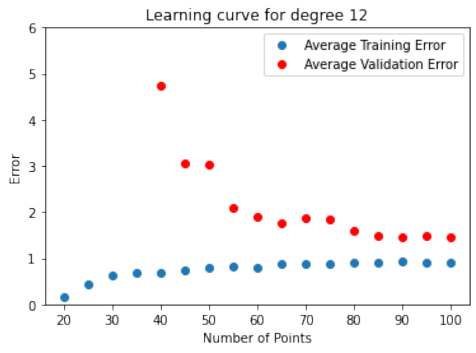
   ii. Compute the average of the training and validation errors from the 5 folds.
2. Create a learning curve by plotting both the average training and validation error as functions of N.
   Hint: Have same y-axis scale for all degrees d.

Solution :
https://colab.research.google.com/drive/1QU65k4s9lplEySEXryYrPkHetgkRPIE7?usp=sharing









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

Solution : *The Degree 1 polynomial model has the highest bias because it is underfitting the data. We can see this through the high error in both the training and validation data.*

[3] Which model has the highest variance? How can you tell?

Solution : *The Degree 12 polynomial model has the highest variance because it is overfitting the data. We can see this through the lowest error on training data but high error on validation data (largest difference in training and validation error).*

[3] 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 : *The learning curve of the quadratic model tells us that the model does not seem like it will improve if we had additional training points. Both the training and validation error seem to have stopped improving with the number of training points that we are using.*

[3] Why is training error generally lower than validation error?

Solution : *The training error generally lower than validation error because the model is being fit on the training data while it does not see the validation data at all. While both sets of data are generated from the same distribution, the model will be biased towards the data it is training on (the model will try to fit the underlying trend of the training data, but the training data will not perfectly model the underlying distribution).*

[3] 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 : *I would expect the degree 6 model to perform best on some unseen data drawn from the same distribution as the training data because it has the lowest validation error. Since it does the best among the models on data it has not seen during training, I would expect it to do the best on more unseen data from the same distribution.*

3 Stochastic Gradient Descent [36 Points]

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:

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

[2] 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 : *We will define* $\mathbf{x}$ *and* $\mathbf{w}$ *such that the model includes the bias term by adding an additional term to each of* $\mathbf{x}$ *and* $\mathbf{w}$, $\mathbf{x}^{(0)}$ *and* $\mathbf{w}^{(0)}$ *respectively. We will define* $\mathbf{x}^{(0)} = 1$ *so that* $\mathbf{w}^{(0)}$ *will 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:

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

[2] 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 :

$$\begin{aligned} \nabla L(f) &= \nabla \sum_{i=1}^{N} (y_i - \mathbf{w}^T\mathbf{x}_i)^2 & (8) \\ &= \sum_{i=1}^{N} -2\mathbf{x}_i(y_i - \mathbf{w}^T\mathbf{x}_i) & (9) \\ & & (10) \end{aligned}$$

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.

[8] 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.

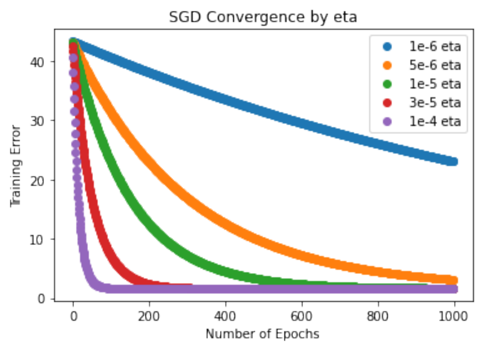
Solution : *See code.*
https://colab.research.google.com/drive/1mEq35M88VbVQacr88SPeg2i3PA5L5-Kv?usp=sharing

[2] 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 : *The convergence behavior of SGD changes in direction and magnitude as the starting point varies. The direction will always point down the slope towards the minimum and the speed/magnitude of the steps of convergence will be quicker/larger on steeper slopes (larger gradient of loss function). The two data sets will have similar general convergence behavior, but because of the difference in landscapes, will have different specific behaviors (the second and third points go faster for the first dataset because they are higher up the slope, while third and fourth points go faster for the second dataset because those are higher up the slope).*

[6] 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 :



As η increases, the SGD converges quicker and the training error decreases quicker.

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.

[6] 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.

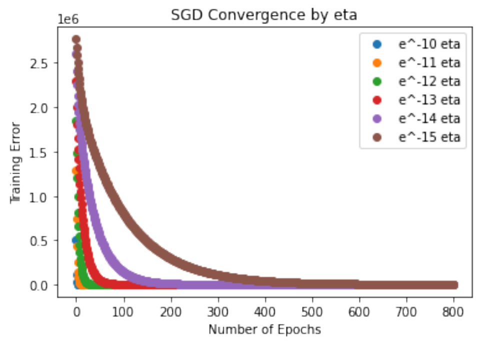
Solution :
https://colab.research.google.com/drive/1yhWlwwkMtmElODqOtEmXWFKe_Ywdd4Zx?usp=sharing
The end weights are $[-0.22718106, -5.94210372, 3.94390838, -11.72382957, 8.78568536]$.

[2] 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 :



As the values of the learning rate η get smaller, the training error decreases at a slower rate and it takes more epochs to start to converge.

[2] 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 : *Weights of* $[-0.31644251, -5.99157048, 4.01509955, -11.93325972, 8.99061096]$ *are obtained from the analytical solution; these are close to* $[-0.22718106, -5.94210372, 3.94390838, -11.72382957, 8.78568536]$ *given by SGD.*

Answer the remaining questions in 1-2 short sentences.
[2] Is there any reason to use SGD when a closed form solution exists?

Solution : *One might want to use SGD when a closed form solution exists, because it may be computationally cheaper to using SGD to get to the solution than using the closed form (for example the matrix operations can be computationally expensive in the linear regression closed form solution).*

[2] 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 : *Based on the SGD convergence plots that were generated earlier, we can describe a stopping condition that will stop SGD when the difference in loss/training error between consecutive epochs is below a certain threshold. Rather than stopping after a pre-defined number of epochs, this will stop once we get close to conver-*

gence. For example, we can make our stopping threshold be when the change in error over the initial error is below some value.

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

Solution : *Convergence behavior of the weight vector differ in the ability to converge between the perceptron and SGD algorithms because the perceptron algorithm will correct point by point can reach a stage where it will never converge, while the SGD algorithm will always move down the gradient towards convergence.*

4 The Perceptron [14 Points]

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

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

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.

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

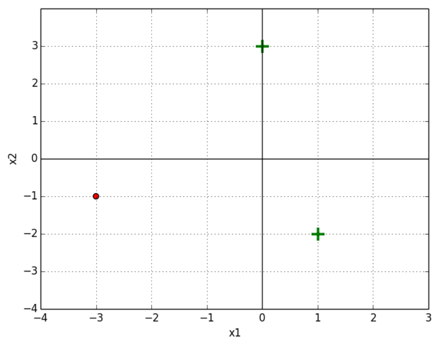


Figure 1: The green $+$ are positive and the red $\circ$ is negative

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

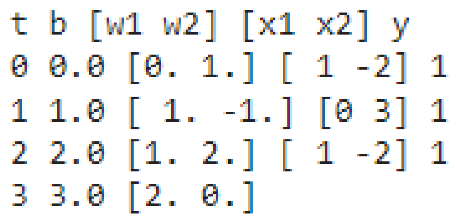
Give your solution in the form a table showing the weights and bias at each timestep and the misclassified point $([x_1, x_2], y)$ that is chosen for the next iteration's update. You can iterate through the three points in any order. Your code should output the values in the table below; cross-check 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			

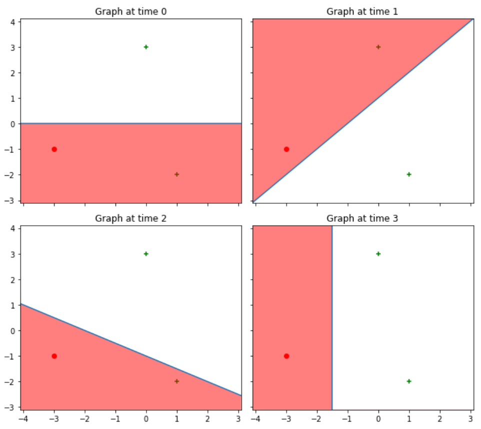
Include in your report both: the table that your code outputs, as well as the plots showing the perceptron's classifier at each step (see notebook for more detail).

Solution :

https://colab.research.google.com/drive/1lDyJ5iIWNx26mhdIMMugka5_q3XvAWz7?usp=sharing



```
t b [w1 w2] [x1 x2] y
0 0.0 [0. 1.] [ 1 -2] 1
1 1.0 [ 1. -1.] [0 3] 1
2 2.0 [1. 2.] [ 1 -2] 1
3 3.0 [2. 0.]
```



[4] 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.

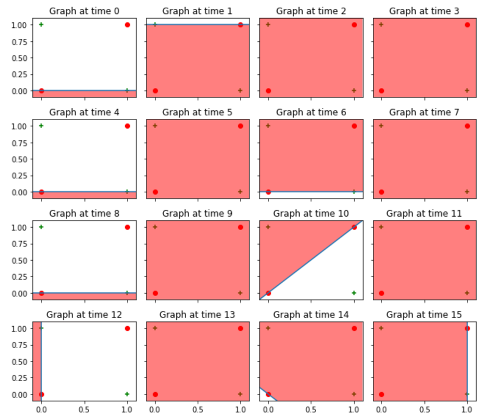
In a 2D dataset, how many data points are in the smallest dataset that is not linearly separable, such that no three points are collinear? How about for a 3D dataset such that no four points are coplanar? Please limit your solution to a few lines - you should justify but not prove your answer.

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 : *In a 2D dataset, there are 4 points in the smallest dataset that is not linearly separable, such that no three points are collinear; this is because for 3 points we can always separate any point from the other two points with a line, but with 4 datapoints we cannot (with no three points being collinear for both). For a 3D dataset such that no four points are coplanar, there are 5 points in the smallest dataset that is not linearly separable; this is because for 4 points we can always separate any point from the other three points with a plane as well can always separate any two points from the other two points with a plane, but with 5 datapoints we cannot (with no four points being coplanar for both). For the N-dimensional set, there are $N+2$ points in the smallest dataset in which* ***no*** *$< N$-dimensional hyperplane contains a non-linearly-separable subset.*

[2] 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 :



If a dataset is not linearly separable, the Perceptron Learning Algorithm will never converge because there will always be a point that is misclassified. This means that the algorithm will keep trying to update the weights for the misclassified points since all the points cannot be classified correctly at the same time.