Machine Learning & Data Mining
Set 1

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

Policies

- Due 9 PM PST, January 13^{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 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 thelink 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: Is the process of draw inferences about a population from a sample.

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

Solution B: Is when we use a linear regression model to make inferences of a population from a sample or training data. Is the set of all formulas in the form $f(x \mid w, b) = wtx - b$

Problem C [2 points]: What is overfitting?

Solution C: Is when the model gives good results for the training data but not for new data. The test error is much larger than the training error.

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

Solution D: Two ways to prevent overfitting are early stop and pruning (identifying the most important features within the training set)

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 set is the data used to feed or train the algorithm meanwhile test data is the data used once the model is built to test the performance of the model. Is not a good practice because you are using the same data both for training your algorithm and testing it, and the data used for these two process should be different.

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

Solution F: The sample is identically and independent distributed from the real distribution. The distribution of the sample is approximate the same as the true population

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: X (input space) would be a vector indicating the email contains specific words and Y (output space) would be if the email is a spam or not

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

Solution H: You make a partition of your data in kth partition, then you train your algorithm in k-1 partitions and evaluate in the remaining set. Then you have to repeat the procedure rotating the test partition

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,

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

given the following definitions:

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

Solution A:

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

($\mathbb{E}_x[F(x) - y(x)] = 0$, $\mathbb{E}_x[(F(x) - y(x))^2] = \text{Bias}(x)$)

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

($\mathbb{E}_x[\text{Bias}(x)]$ is constant)

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

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

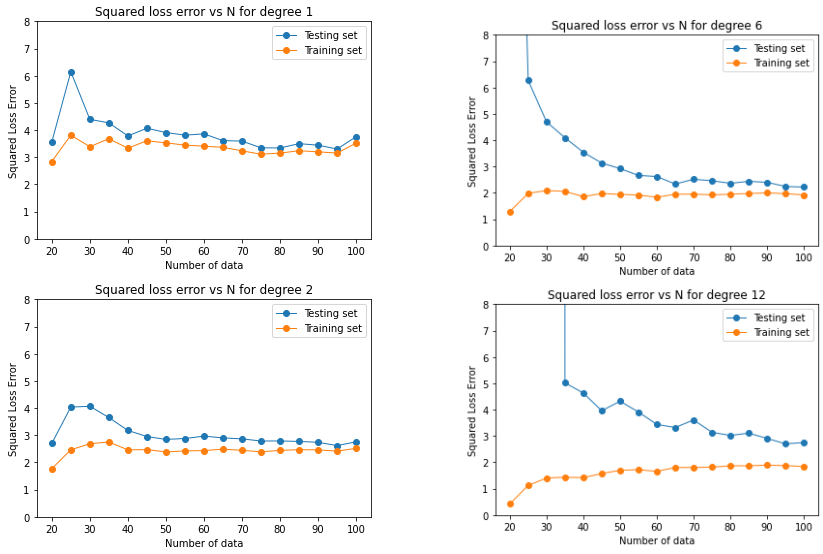
1. For each $N \in \{20, 25, 30, 35, \cdots, 100\}$:
    i. Perform 5-fold cross-validation on the first N points in the dataset (setting aside the other points), computing the both the training and validation error for each fold.

- 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/1iPO4o1_4b-d02JYpY7JaZof3-h8dWePQ?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: Degree 1. Because the estimated value by the model produces higher squared mean errors (respect to the real value). It has the highest error for training data and this is because the bias decreases with the complexity of the models. Since degree 1 is the model with lower complexity, the model tends to be underfitted

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

Solution D: degree 12. This model depends very much of the training data and don't generalize in the data that has not seen before. This model tends to be overfitted. This means the model considering too much noises in the data. It agree with the theory that a model with higher complexity tends to be overfitted

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: That with more data the model will not improve very much.

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

Solution F: Because the model is fit on the training data.

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: Degree 6 seems to be the models with the best trade off between bias and variance

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:

Let x be a local minimum of F. Then, by definition, there exists an open neighborhood N of x' such that for all x in N, F(x) ≥ F(x').

Using the first-order Taylor expansion, we have that for some x, h ∈ ℝⁿ, there exists c ∈ (0,1) such that F(x+h) = F(x) + ∇F(x+ch)ᵀh

Since x' is a local minimum, for all x in N, F(x) ≥ F(x'). Therefore, for all h in N, F(x+h) ≥ F(x')

Substituting the Taylor expansion, we have that for all h in N, F(x) + ∇F(x+ch)ᵀh ≥ F(x')

Since this holds for all c ∈ (0,1), we can take the limit as c approaches 0 to get that ∇F(x')ᵀh ≥ 0

Finally, since this holds for all h in N, we can take the limit as h approaches 0 to get

∇F(x') = 0

Linear regression learns a model of the form:

$$f(x_1, x_2, \cdots, x_d) = \sum_{i=1}^{d}$$

$w_i x_i + b$

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:
Adding one additional element to vectors w and x. This is, instead of $w_1, w_2 \ldots$ we will have $w_0, w_1, w_2 \ldots$ and instead of $x_1, x_2 \ldots$ we will have $x_0, x_1, x_2 \ldots$. If we set $X_0=1$, the element $w_0 * x_0$ will be the b element.

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:

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.

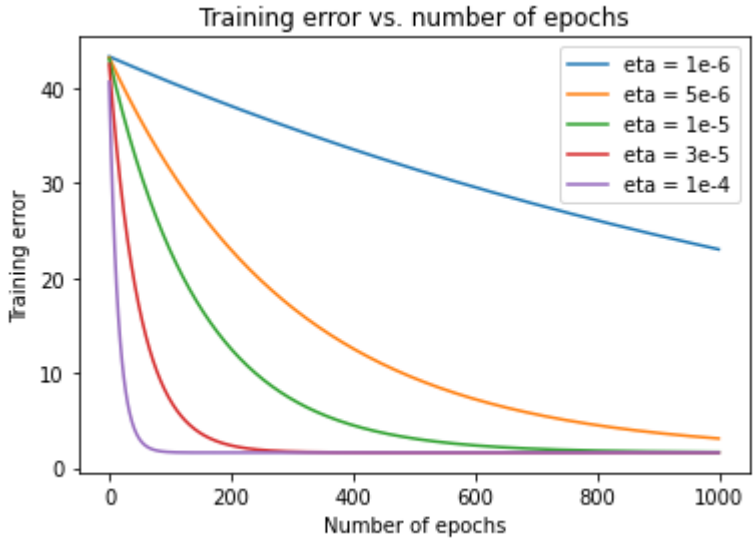
Solution D: *See code.*
https://colab.research.google.com/drive/19KwcT5OyOx-PvQ7jsqtxfO7Dgt93ylTt?usp=sharing

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: *Seems that no matter which the starting point, the SGD algorithm tends to converge to the same point (global minimum). The behavior is similar in datasets 1 and 2*

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:



SGD show to converge faster as eta increases

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.

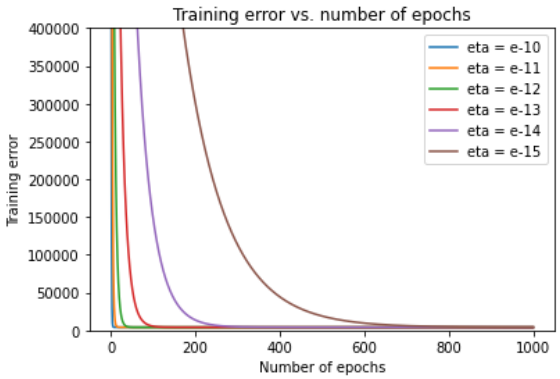
Solution G:
https://colab.research.google.com/drive/1NcgTQdma5wdyEubyLhbftiIHfnTVPQJD?usp=sharing

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:



As eta increases, SGD takes less iteration to converge. This is, with smaller learning rates, it takes the algorithm more epochs to get the training error down.

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

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

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

Solution I:

The analytical solution is:

```
[ -0.31644251  -5.99157048   4.01509955 -11.93325972   8.99061096]
This is pretty close to which the SGD
```

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: If the size of the data is big, it can be very computationally expensive to compute the closed form solution. Thus, SGD can be more convenient for cases like this

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 stopping condition is stopping when the changes in errors are sufficient small (threshold defined by the programmer) so doing more iterations doesn't change very much the output

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

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

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 ∘ point is in the -1 class.

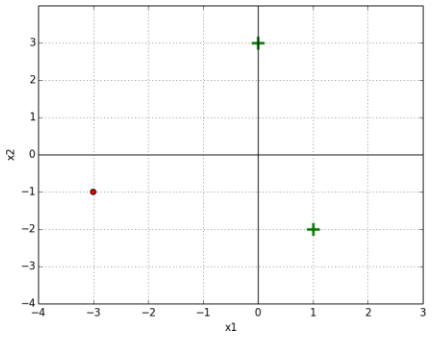


Figure 1: The green + are positive and the red ◦ is negative

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

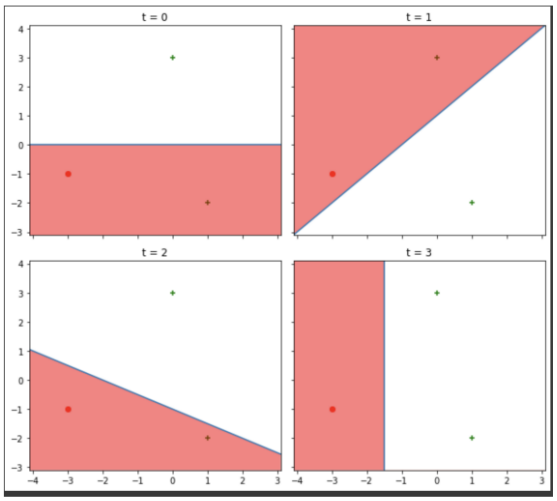
Give your solution in the form a table showing the weights and bias at each timestep and the misclassified point ($[x_1, x_2]$, y) that is chosen for the next iteration's update. You can iterate through the three points in any order. Your code should output the values in the table below; cross-check 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/18fVJDHsN0RFD-9xm-L8zPEraYGWZLAQO?usp=sharing



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?

Problem B [4 points]: A dataset $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\} \subset \mathrm{R}^d \times \mathrm{R}$ 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.

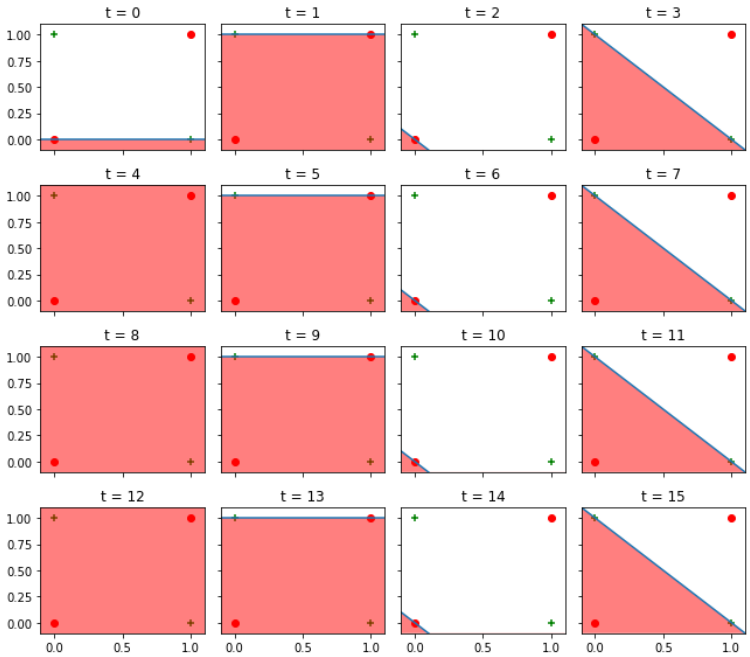
In 2D space, what is the minimum size of a dataset that is not linearly separable, such that no three points are collinear? How about the minimum size of a dataset in 3D that is not linearly separable, such that no four points are coplanar? Please limit your explanation to a few lines - you should justify 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 points are on the same hyperplane? For the N-dimensional case, you may state your answer without proof or justification.

Solution B: in 2D, the minimum size will be 4 points. Because for any two random (same sign) points there is a line that pass through them, thus a third point always will be linearly separably, but a fourth point can be no separable. We can do a similar analogy in 3D (a plane that pass though three points of the same sign), so four points is the minimum size.

In general, N data points are linearly separable in N−1 dimensions.

Solution C:



No. Because if there are no hyperplane that can separate the points correctly, the algorithm never will find one and will be iterating in a infinite loop.

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: In the perceptron algorithm, the weight vector fluctuates before converging (if it converges), meanwhile the SGD will converge smoothly without diverging (the step size should be selected correctly)