Machine Learning & Data Mining — Caltech CS/CNS/EE 155
Set 1 — January 4th, 2022

Marta Gonzalvo

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"

Marta Gonzalvo

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: *It's the set of all possible candidate models that can approximate the objective function for mapping inputs to outputs.*

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

Solution B: *It's all the functions of the form $f(x) = w^t x + b$.*

Problem C [2 points]: What is overfitting?

Solution C: *It's when a model is so aligned to the training set that does not extrapolate well to the test set or the underlying true probability distribution of the data.*

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

Solution D:

- *Reducing the number of parameters in the model.*
- *Adding more data points to 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: *The training data is used for the model to learn and the test data, sampled independently, is used to evaluate the trained model.*

You should never change your model based on information from the test data because to evaluate the model fairly (i.e. how would it do on future unseen data, how well it models the underlying true probability distribution of the data) you need an independent set of data that the model has never seen.

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

Solution F: *We assume that the model is sampled i.i.d., independent and identically distributed.*

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 X would be the vector of encoded words in the subject of the email, and the Y would be the boolean indicating whether they are spam or not.*

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

Solution H: *It's when you split the original dataset into k equal parts, train on $k-1$ parts and test on the remaining one, and repeat for every choice of the $k-1$ parts. Afterwards, the validation error is averaged.*

Marta Gonzalvo

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:

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

Solution A:

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

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.

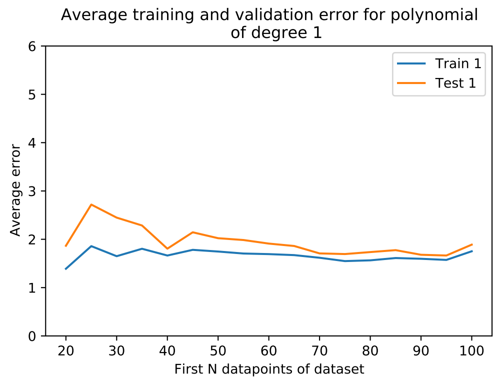
Problem B [14 points]: Use the provided `2_notebook.ipynb` Jupyter notebook to enter your code for this question. This notebook contains examples of using NumPy's polyfit and polyval methods, and scikit-learn's KFold method; you may find it helpful to read through and run this example code prior to continuing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

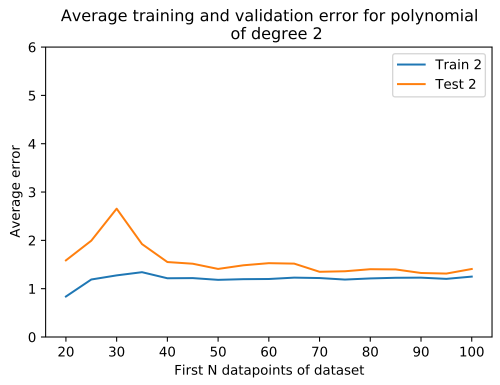
The dataset `bv_data.csv` is provided and has a header denoting which columns correspond to which values. Using this dataset, plot learning curves for 1st–, 2nd–, 6th–, and 12th–degree polynomial regression (4 separate plots) by following these steps for each degree $d \in \{1, 2, 6, 12\}$:

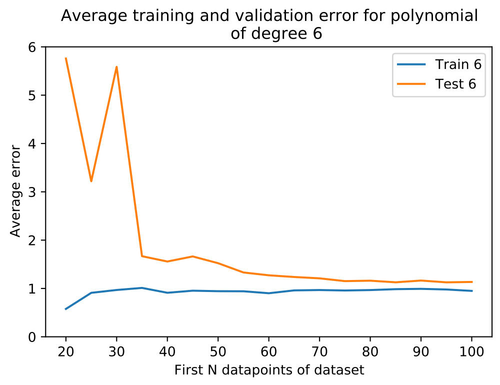
1. For each $N \in \{20, 25, 30, 35, \cdots, 100\}$:
   i. Perform 5-fold cross-validation on the first N points in the dataset (setting aside the other points), computing the both the training and validation error for each fold.
      - Use the mean squared error loss as the error function.
      -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

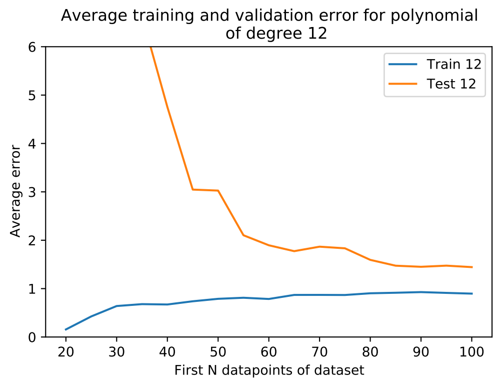
   ii. Compute the average of the training and validation errors from the 5 folds.
2. Create a learning curve by plotting both the average training and validation error as functions of N.
   Hint: Have same y-axis scale for all degrees d.

Solution B: *https://colab.research.google.com/drive/1QMzJm9CJizyUtCmujx6D_x9xwZAzXL7T?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 polynomial regression model of degree 1 has the highest bias, because it has the highest training error. Also, as we increase the number of points, the error increases, the model is not complex enough to describe the data accurately.*

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

Solution D: *The polynomial regression model of degree 12 has the highest variance, because the test error is the highest while the training error is the lowest: it is overfitted. As we increase the number of data points, the test error decreases.*

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: *The learning curve tells me that it's probably not going to improve too much, because the training and test errors have converged to close values.*

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

Solution F: *The training error is lowest than the validation error because the model has been directly optimized on the training set 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: *I expect the polynomial regression model of degree 6 to perform best, because it has a low bias and variance, the training and test errors are pretty similar. The test error is lowest when compared to the other polynomial models, when using more than approx. 60 data points.*

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: *By adding an additional element to* $\mathbf{x}$ *and* $\mathbf{w}$ *as* $x_0 = 1$ *and* $w_0 = b$*, we can condense* b *into the sum:*

$$\begin{aligned}
f(x_1, x_2, \cdots, x_d) &= \left(\sum_{i=1}^{d} w_i x_i\right) + b \\
f(x_1, x_2, \cdots, x_d) &= b \cdot 1 + w_1x_1 + w_2x_2 + \cdots w_dx_d \\
f(x_1, x_2, \cdots, x_d) &= w_0x_0 + w_1x_1 + w_2x_2 + \cdots w_dx_d \\
f(x_1, x_2, \cdots, x_d) &= \mathbf{w}^T\mathbf{x}
\end{aligned}$$

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:

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

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

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

The following few problems ask you to work with the first of two provided Jupyter notebooks for this problem, `3_notebook_part1.ipynb`, which includes tools for gradient descent visualization. This notebook utilizes the files `sgd_helper.py` and `multiopt.mp4`, but you should not need to modify either of these files.

For your implementation of problems C-E, **do not** consider the bias term.

Problem C [8 points]: Implement the `loss`, `gradient`, and `SGD` functions, defined in the notebook, to perform SGD, using the guidelines below:

- Use a squared loss function.
- Terminate the SGD process after a specified number of epochs, where each epoch performs one SGD iteration for each point in the dataset.
- It is recommended, but not required, that you shuffle the order of the points before each epoch such that you go through the points in a random order. You can use `numpy.random.permutation`.
- Measure the loss after each epoch. Your `SGD` function should output a vector with the loss after each epoch, and a matrix of the weights after each epoch (one row per epoch). Note that the weights from all epochs are stored in order to run subsequent visualization code to illustrate SGD.

Solution C: *See code.*

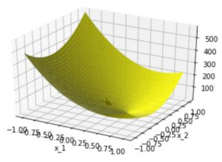
https://colab.research.google.com/drive/1vsXg-nROuWdBnZec6FqxtHIL7i-XBo-h?usp=sharing

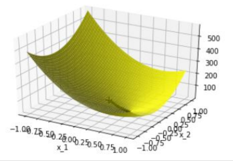
Problem D [2 points]: Run the visualization code in the notebook corresponding to problem D. How does the convergence behavior of SGD change as the starting point varies? How does this differ between datasets 1 and 2? Please answer in 2-3 sentences.

Solution D: *The path to the minimum is longer or shorter depending on the starting point distance to it. However, the behavior of SGD does not change in any appreciable way as the starting point and the dataset vary.*

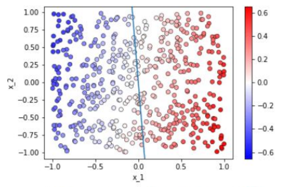
All the plots generated in this section are included below:

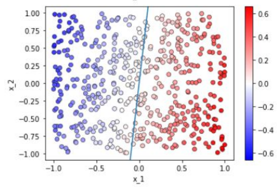
SGD from single starting point:



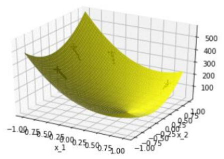


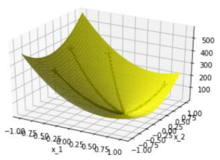
Weights change as the algorithm converges:



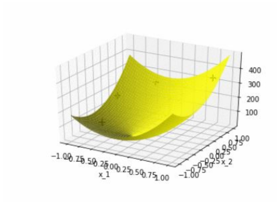


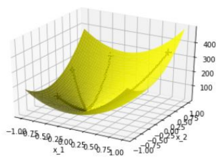
SGD from multiple arbitrary starting points, dataset 1:





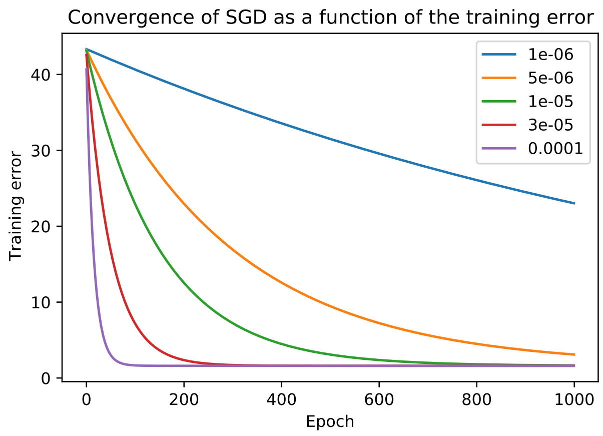
SGD from multiple arbitrary starting points, dataset 2:





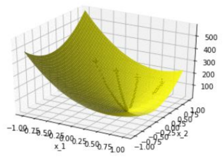
Problem E [6 points]: Run the visualization code in the notebook corresponding to problem E. One of the cells—titled "Plotting SGD Convergence"—must be filled in as follows. Perform SGD on dataset 1 for each of the learning rates $\eta \in \{1e\text{-}6, 5e\text{-}6, 1e\text{-}5, 3e\text{-}5, 1e\text{-}4\}$. 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?

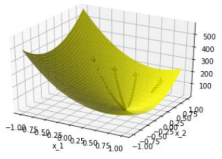
Solution E: *As η increases, the loss decreases much less per epoch and so the SGD converges much slower. The value that the training error converges to is the same for all η.*



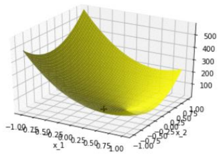
All the other plots generated in this section are included below:

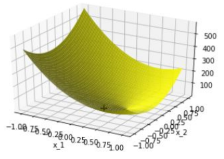
SGD with different step sizes (η):





η=1:





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:

https://colab.research.google.com/drive/1mpvOHkoXWf-qLY9D2CT2xPvfGsAfkU_3?usp=sharing

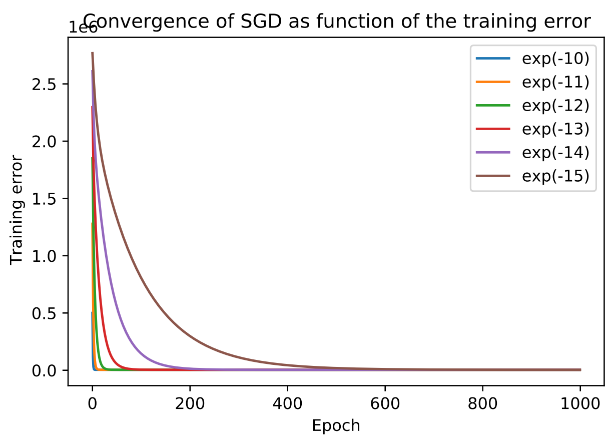
After 1000 epochs, the weights are: (-0.22791757), -5.9787249, 3.98816869, -11.85719653, 8.91110894, where the first weight is w_0, the bias, and the other four are w_1, w_2, w_3 and w_4.

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: *The largest learning rate of e^{-10} is small enough that the losses don't oscillate up and down, but still converges very fast. As the learning rate decreases, the convergence is slower.*



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: *The result from the analytical solution is [-6.00714404, 4.00035947, -11.94691938, 8.97655242]. This result matches up pretty well with the result I got from SGD, having a difference of less than 0.1 for each of the values between SGD and the analytical solution.*

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: *Depending on the size of the dataset, the closed-form solution involving the inverse could take a long time to calculate, and using SGD would be faster.*

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: *A stopping condition would be when the training error does not change between epochs, as the SGD is already converged.*

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

Solution K: *In the SGD algorithm, the convergence is very smooth towards a horizontal asymptote. However, in the perceptron, the convergence is step-wise, and it is not guaranteed to converge.*

Marta Gonzalvo

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

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

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.

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.

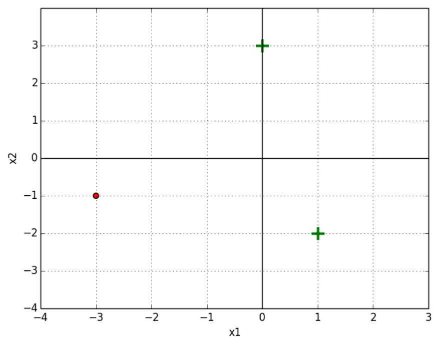


Figure 1: The green $+$ are positive and the red $\circ$ is negative

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/1ltx6sRQlk_WdxuKlqLnwFMtkTcqhCa4p?usp=sharing*

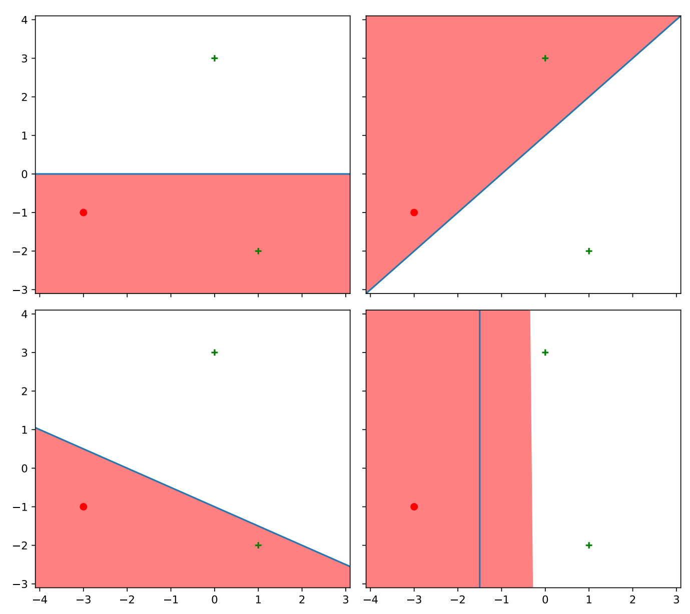
The outputs from my code are:

```
Iteration 0, bias 0.0, weights [0.  1.], X [ 1 -2], Y 1
Iteration 1, bias 1.0, weights [ 1.  -1.], X [0 3], Y 1
Iteration 2, bias 2.0, weights [1.  2.], X [ 1 -2], Y 1
Iteration 3, bias 3.0, weights [2.  0.],
```

which results in the same table as above:

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			

Marta Gonzalvo



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.

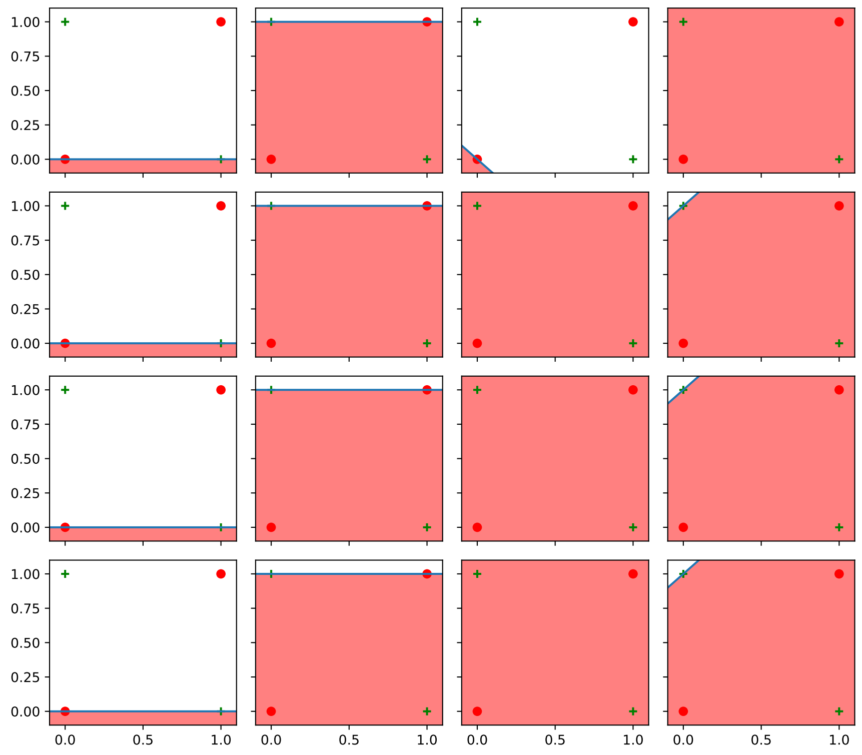
Solution B: *In 2D, the smallest possible dataset not linearly separable has 4 points. As seen in Problem C, this could happen when the four points are placed in the corners of a square, for example, and the points in either diagonal belong to different classes. Anything smaller than that is linearly separable, as 3 non collinear points can always be split in any 2-1 combination by a straight line.*

For 3D, the smallest possible non linearly separable dataset would be 5 points. If no 4 points are coplanar, 4 points can be divided by a hyperplane in any combination. However, if 3 points in the same class are coplanar, and 2 in another class are form a perpendicular line to the triangle formed by the 3 points, the system is not linearly separable.

Generalizing, this means that for a N-dimensional dataset, the smallest non-linearly separable dataset contains N+2 points.

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: *I do not think the Perceptron Learning Algorithm will ever converge, because a solution with no misclassified points does not exist, so it will always continue to the next iteration. We can see that in the figure below, where the perceptron weights and bias at every step seem go through 4 interation cycles.*


1.00
0.75
0.50
0.25
0.00
0.0
0.5
1.0