Machine Learning & Data Mining
Set 1

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: *A hypothesis set is a set of functions that are candidates to represent the target function. The ML algorithm picks an optimal function (closest to target function) from the hypothesis set. For example, the algorithm gradient descent can be used to choose a fit g from the hypothesis set H of all linear formulas.*

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

Solution B: *The hypothesis set of a linear model is $H = \{w|w \in \mathbb{R}^{D+1}\}$ where D is the dimension of each input or each data point (x_1, x_2) in the training set. For instance, in 2D linear models, where the inputs are represented as $(1, x_1, x_2)$, the hypothesis set is of the form $H = \{w|w = (a, b, c), a, b, c \in R\}$, and the model is of the form $f(x|w) = w^t x$*

Problem C [2 points]: What is overfitting?

Solution C: *Overfitting is when the algorithm picks a hypothesis that fits the training data very well (low in sample error) but is no longer a good approximation for the target function (high out of sample error).*

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

Solution D: *Reducing model complexity - such as by choosing models with fewer degrees of freedom. For example, degree 7 polynomials have more degrees of freedom than polynomials of degree 2. Or, you could increase the amount of training data. But, finding data is expensive.*

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 the data set that the algorithm uses to learn (i.e. approximate the target function). The test data is used to estimate the out-of-sample error (to measure how well the model will perform on new data). Even making a slight change in the model because of the test data is a form of learning. In order to get an unbiased estimate of the out of sample error, it's important that we never use the test data to change the model.*

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

Solution F: *We make the assumptions that each data point is independent and identically distributed (i.i.d.)*

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 would have datapoints with values for a set of features (occurrence of the word 'lottery' or other spam related words). This dataset X should be labelled (we know whether or not the email is spam) in supervised learning models. The output space Y would be 0 or 1 based on whether or not the email is spam.*

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

Solution H: *In k-fold cross-validation, we split the dataset into k parts. We use k-1 of the parts to train the model, then we use the last part (kth part) as a validation set. Then, we calculate this error for each of the other parts (training on the remaining k-1 parts in every case). This way, we can have an estimate of the out of sample error based on the whole dataset.*

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:

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

Now consider the middle term:

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

Expand the quadratic to get the following:

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

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

Taking expected value w.r.t x again,

$$\mathbb{E}_x\left[\mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2\right] + \left(F(x) - y(x)\right)^2\right] = \mathbb{E}_x\left[bias(x) + var(x)\right] = bias + var$$

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.

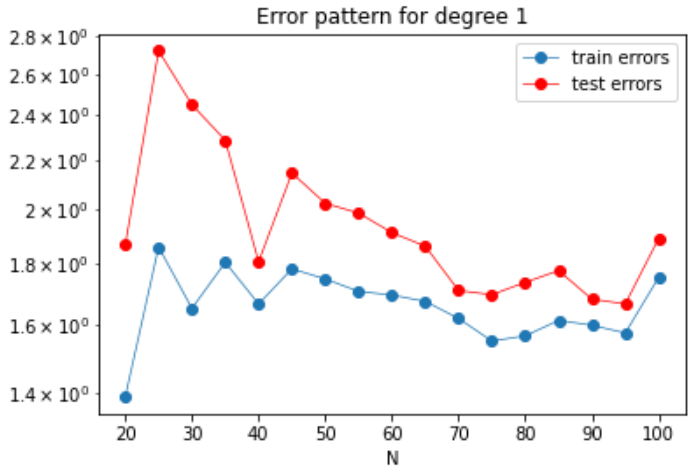
Polynomial regression is a type of regression that models the target y as a degree–d polynomial function of the input x. (The modeler chooses d.) You don't need to know how it works for this problem, just know that it produces a polynomial that attempts to fit the data.

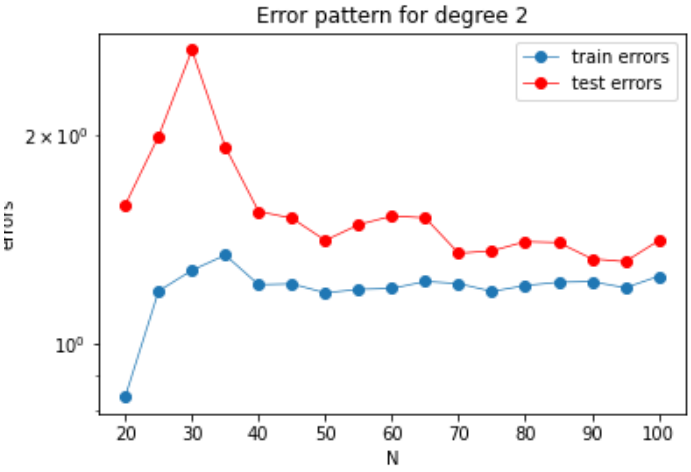
Problem B [14 points]: Use the provided `2_notebook.ipynb` Jupyter notebook to enter your code for this question. This notebook contains examples of using NumPy's polyfit and polyval methods, and scikit-learn's KFold method; you may find it helpful to read through and run this example code prior to continuing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

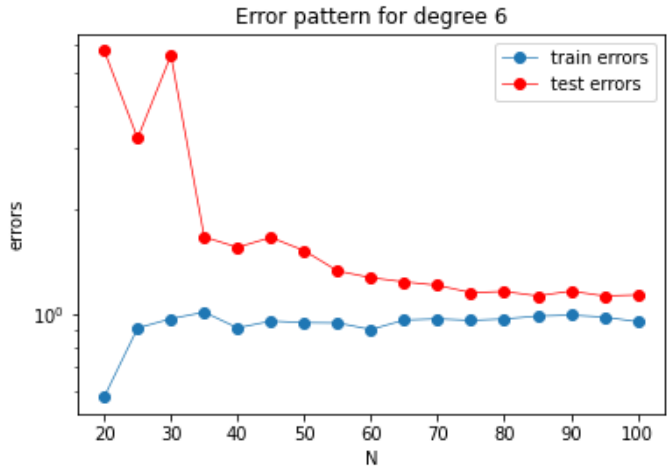
The dataset `bv_data.csv` is provided and has a header denoting which columns correspond to which values. Using this dataset, plot learning curves for 1st–, 2nd–, 6th–, and 12th–degree polynomial regression (4 separate plots) by following these steps for each degree $d \in \{1, 2, 6, 12\}$:

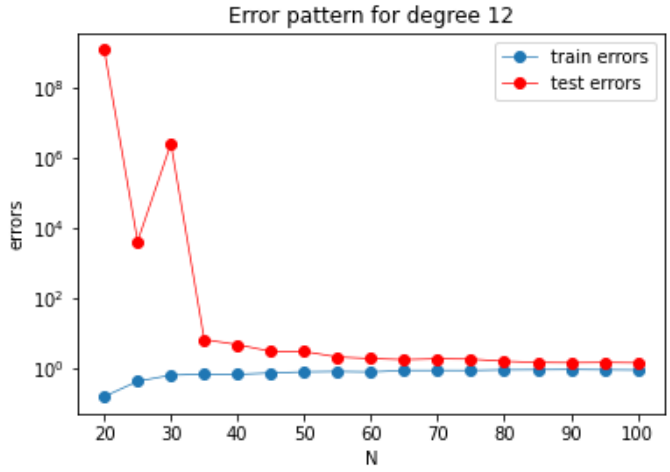
1. For each $N \in \{20, 25, 30, 35, \cdots, 100\}$:
   i. Perform 5-fold cross-validation on the first N points in the dataset (setting aside the other points), computing the both the training and validation error for each fold.
      - Use the mean squared error loss as the error function.
      -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

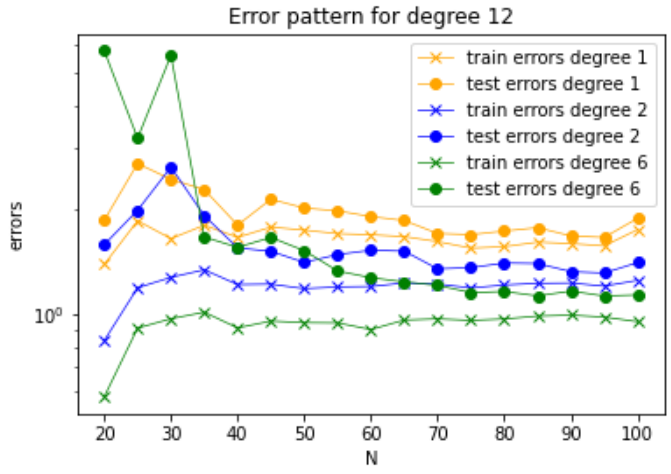
Solution B: *Link to Google Colab:* https://colab.research.google.com/drive/1-wUUFX40LgO_RHh_XlAdTM_yHq41gV-H?usp=sharing


Error pattern for degree 1
train errors
test errors
2.8×10^0
2.6×10^0
2.4×10^0
2.2×10^0
2×10^0
1.8×10^0
1.6×10^0
1.4×10^0
20 30 40 50 60 70 80 90 100
N


Error pattern for degree 2
train errors
test errors
2×10^0
10^0
20 30 40 50 60 70 80 90 100
N


Error pattern for degree 6
train errors
test errors
errors
10^0
20
30
40
50
60
70
80
90
100
N


Error pattern for degree 12
train errors
test errors
errors
10^8
10^6
10^4
10^2
10^0
20
30
40
50
60
70
80
90
100
N



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

Solution C: *The highest bias is in the degree 1 polynomials. This is because, though the difference in train and test errors isn't that much, the errors are high. This implies underfitting and a simple model class with low complexity.*
On the diagram with 3 models plotted together we find that the linear model is generally higher in error than the degree-2 or degree-6 even as N increases. So, even the best of the linear models are not performing as well as other models. Thus, linear models are too simplistic and the bias is high.

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

Solution D: *The highest variance is in the degree 12 polynomial. We know this because in the diagram for degree 12, we find that the test error of the models is several orders of magnitude away from the train error. As N increases this phenomenon reduces are overfitting reduces. But generally, the highest variance is in the degree 12 model. The variance between different models of degree 12 is very high.*

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 training and test error of the quadratic model seem to have reached an equilibrium or stable value. This means, we might have found the closest quadratic model to the training data. So, adding training points may not necessarily increase the performance of the model or improve any errors. More likely, the errors will stay at a similar amount.*

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

Solution F: *While training, the learning algorithm picks suitable coefficients for the model on the basis of the training set. The training set has been "seen" by the model before. The model has learned the patterns in the training set. The test set on the other hand has not been seen before and is an indication of how the model would perform out of sample. This is why, the training error is generally lower than the validation error.*

Problem G [3 points]: Based on the learning curves, which model would you expect to perform best on some unseen data drawn from the same distribution as the training data, and why?

Solution G: *The test errors for model 6 are the lowest, I would expect it to perform the best out of sample.*

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: *Include an additional 1 in the beginning of each datapoint x (so that x is (1, x1, x2, ...) and include an additional w_0 in w which will act as the 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:

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

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.

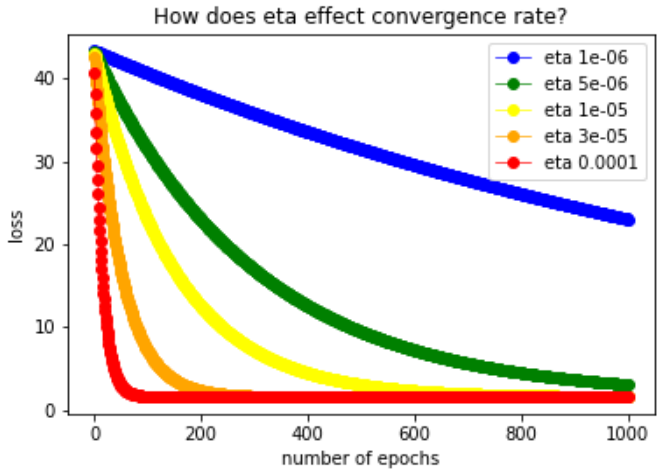
Solution C: *See code. Link to Google Colab:* https://colab.research.google.com/drive/16XURNiYJBCLqJJGgbh
usp=sharing

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 convergence behaviour/rate doesn't change much. Since the function is convex, it converges to the same minimum from each starting point. In a different dataset, the convergence point is different but the general behaviour is the same.*

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 eta increases, the convergence becomes faster. But, if eta increases too much, it could lead to chaotic behaviour that might not lead to convergence because the step size might be so large that it might overshoot and miss the target weights.

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.

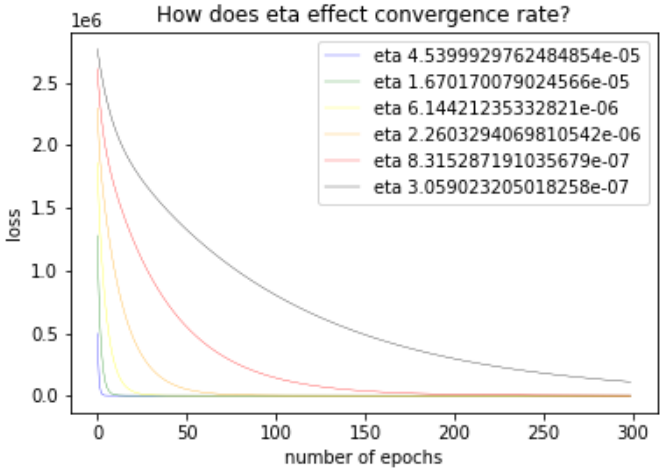
Solution F: *Link to google colab:* https://colab.research.google.com/drive/1JkFGswWXMIQLsvgVSMYbmiYPS usp=sharing $w_{final} = [-0.22716927, -5.94209596, 3.94391341, -11.72382737, 8.78569037]$

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: *Once again we see that as eta decreases, the number of epochs it takes to converge increases.*



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 solution from the closed form is*

$$-0.31644251, -5.99157048, 4.01509955, -11.93325972, 8.99061096$$

. This is very close to what was calculated with SGD.

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: *Generally, the closed form solution is preferred because it takes you directly to the optimum, there is no question of whether or not more epochs are needed. But, calculating the closed form can be expensive (uses the inverse function). Inverting big matrices is expensive. That is one case where gradient descent is more useful because it's 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: *One possible stopping condition might be to stop when the loss function is close to zero (or within a certain threshold of zero). Another might be to stop when the changes in loss from an additional epoch are below a certain threshold.*

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

Solution K: *In SGD, the weight vector slowly approaches the optimum, it is gradual with no big steps (provided a suitable eta has been chosen). The perceptron weight vector, on the other hand, changes it's values/position very quickly and by a lot before converging. Also, the perceptron weight vector may not always converge for eg. if the data is not linearly separable. But, the linear regression SGD vector will converge.*

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.

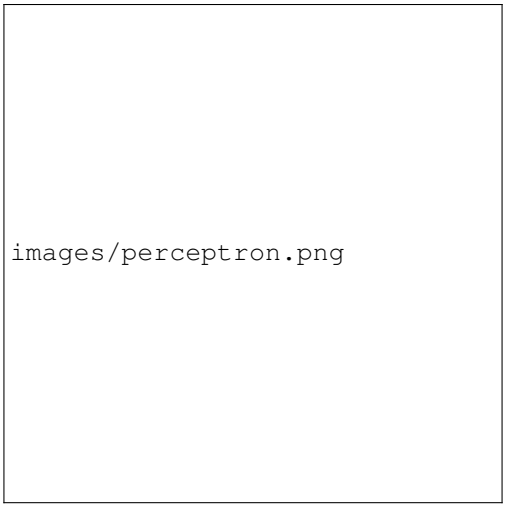


Figure 1: The green $+$ are positive and the red $\circ$ is negative

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.

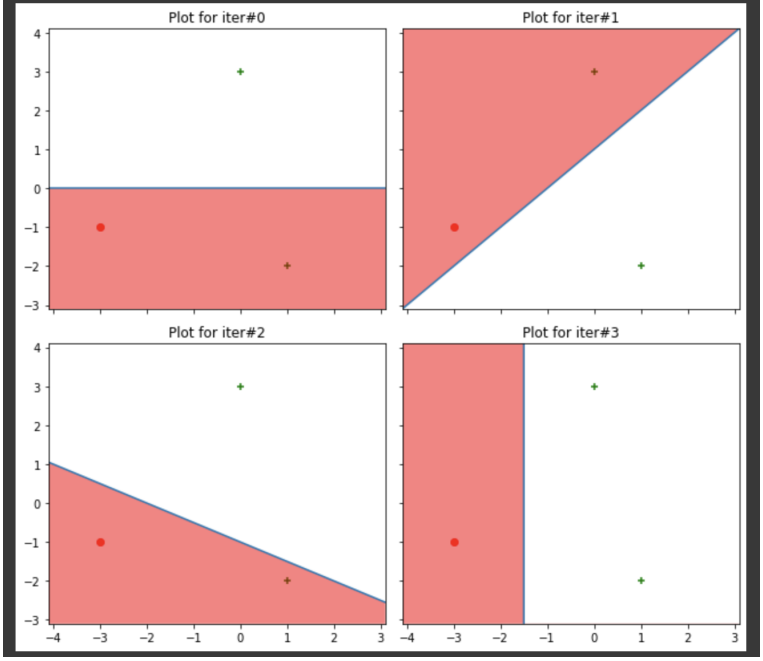
t	b	w_1	w_2	x_1	x_2	y
0	0	0	1	1	-2	+1
1	1	1	-1	0	3	+1
2	2	1	2	1	-2	+1
3	3	2	0			

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: *Link to google colab:* https://colab.research.google.com/drive/1YGucu6zvP3lbqlIxiMb1-BasE
usp=sharing

```
t    b   w1    w2    x1    x2    y

0   0.0   0.0   1.0   1   -2   1
1   1.0   1.0   -1.0   0   3   1
2   2.0   1.0   2.0   1   -2   1
3   3.0   2.0   0.0
```



Problem B [4 points]: A dataset $S = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)\} \subset \mathbb{R}^d \times \mathbb{R}$ is *linearly separable* if there exists a perceptron that correctly classifies all data points in the set. In other words, there exists a hyperplane that separates positive data points and negative data points.

In 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 a 2D dataset, the smallest dataset that is not linearly separable is of size 4 - consider the corners of a rectangle where diagonally opposite corners belong to the same class. This is not a linearly separable set in 2 dimensions. In 3D, the smallest dataset that is not separable by a plane is of size 5. This is because there is always a way to construct a tetrahedron such that one point is excluded from the interior of the tetrahedron. Let the base of the tetrahedron be one class, and let the second class be the top of the tetrahedron and the point in the exterior of the tetrahedron. There is no way to separate these two classes with a 2D plane. Generally, the number is N+2.*

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: *No, if the data is not linearly separable it will not converge. This is because there is no possible model to converge to, so there will always be misclassified points that the perceptron can use to update its current values.*