Machine Learning in Physics
Homework 1

UCSD PHYS 139/239
Draft version due: Friday, January 20, 2023, 5:00pm
Final version due: Wednesday, January 25, 2023, 5:00pm

Policies

- Draft version due 5:00pm, Friday, January 20 on Gradescope (report and code).
- Final version due 5:00pm, Wednesday, January 25 on Gradescope (report and code).
- You are free to collaborate on all of the problems, subject to the collaboration policy stated in the syllabus.
- You should submit all code used in the homework. Please use Python 3 and sklearn version $\geq$0.18 for your code, and that you comment your code such that the TA can follow along and run it without any issues.

Submission Instructions

PLEASE NOTE that there are two steps to submitting your Homework. Both must be submitted by the deadline.

- Please submit your report as a single .pdf file to Gradescope under "Homework 1 Report Draft" or "Homework 1 Report Final". **In the report, include any images generated by your code along with your answers to the questions.** For instructions specifically pertaining to the Gradescope submission process, see https://www.gradescope.com/get_started#student-submission.
- Please submit your code as a .zip archive to Gradescope under "Homework 1 Code Draft" or "Homework 1 Code Final". The .zip file should contain your code files. Submit your code either as Jupyter notebook .ipynb files or .py files.

---

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?

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

Problem C [2 points]: What is overfitting?

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

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?

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

Problem G [2 points]: Consider the machine learning problem of classifying neutrino interaction events as in slide 14 of lecture 1. What could X, the input space, be? What could Y, the output space, be?

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

2 Bias-Variance Tradeoff [39 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 in the input space $\mathcal{X}$,

$$\mathbb{E}_S\left[E_{\text{out}}\left(f_S\right)\right] = \mathbb{E}_x[\text{Bias}(x) + \text{Var}(x)] \tag{1}$$

given that $F(x)$ is the "average function" over all possible datasets S

$$F(x) = \mathbb{E}_S\left[f_S(x)\right] \tag{2}$$

the out-of-sample error for a particular trained model is

$$E_{\text{out}}(f_S) = \mathbb{E}_x\left[\left(f_S(x) - y(x)\right)^2\right] \tag{3}$$

and we define the bias for a given data sample $\text{Bias}(x)$ by how much the average function deviates from the target function

$$\text{Bias}(x) = \left(F(x) - y(x)\right)^2 \tag{4}$$

and the variance for a given data sample $\text{Var}(x)$ measures

$$\text{Var}(x) = \mathbb{E}_S\left[\left(f_S(x) - F(x)\right)^2\right] \tag{5}$$

Problem B [5 points]: When there is noise in the data, the out-of-sample error is

$$E_{\text{out}}(f_S) = \mathbb{E}_{x,z}[\left(f_S(x) - z(x)\right)^2] \tag{6}$$

where $z(x) = y(x) + \epsilon$. If ϵ is a Gaussian-distributed random variable with mean of zero and variance σ^2, show that the bias-variance decomposition becomes

$$\mathbb{E}_S\left[E_{\text{out}}\left(f_S\right)\right] = \mathbb{E}_x[\text{Bias}(x) + \text{Var}(x)] + \sigma^2 \tag{7}$$

Hint: *Given the mean of ϵ is zero,*

$$\mathbb{E}_{x,z}[\epsilon] = \mathbb{E}_{x,\epsilon}[\epsilon] = \mathbb{E}_{\epsilon}[\epsilon] = 0 \tag{8}$$

Likewise,

$$\mathbb{E}_{x,z}[\epsilon^2] = \mathbb{E}_{\epsilon}[\epsilon^2] = \mathbb{E}_{\epsilon}[(\epsilon - \mathbb{E}_{\epsilon}[\epsilon])^2] = \sigma^2 \tag{9}$$

by the definition of variance.

Problem C [14 points]: 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.

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 $N \in \{20, 25, 30, 35, \ldots, 100\}$:
   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.

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

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

Problem F [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?

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

Problem H [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?

3 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 perceptron learning algorithm (PLA) 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 `3_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.

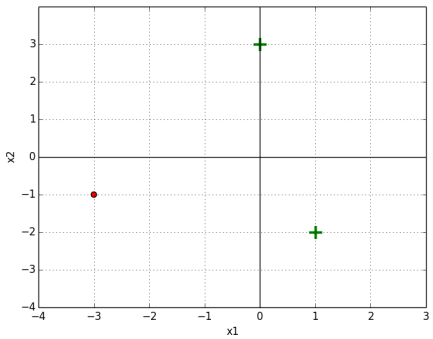

Figure 1: The green $+$ are positive and the red $\circ$ is negative

---

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 time step 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).

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.

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 PLA ever converge? Why or why not?

---

4 TensorFlow Playground [14 Points]

The purpose of this problem is to build some intuition around linear models and neural networks.

Problem A [6 points]: Navigate your web browser to the TensorFlow Playground (https://playground.tensorflow.org/#networkShape=&dataset=xor&discretize=true). Select the checkerboard pattern for the data, which is known as the XOR dataset. The data is sampled from a 2D probability density distribution represented by (x_1, x_2). The regions $x_1, x_2 > 0$ and $x_1, x_2 < 0$ have a target value $y = +1$ and are shown as blue data points, while the regions $x_1 > 0, x_2 < 0$ and $x_1 < 0, x_2 > 0$ have a target value of $y = -1$ and are shown as orange data points.

First, select a linear model with no hidden layers. For the features, select the two independent variables x_1 and x_2. Can you fit the data with this linear model? Why or why not? What happens if you add the feature $x_1 x_2$?

Now, return the features to just (x_1, x_2) and start adding hidden layers. What's the smallest neural network (least number of layers and least number of neurons per layer) you can create that fits the training data "perfectly" (i.e. a training loss <0.001)? What is the corresponding test loss? Detail your hyperparameter choices by providing a screenshot and the URL to your solution (the URL contains all your settings choices).

Problem B [8 points]: Navigate your web browser to the TensorFlow Playground https://playground.tensorflow.org/#dataset=spiral&discretize=true. Select the spiral pattern for the data.

Using all the features available, find a solution that fits the training data. What training and test error do you achieve? Is this a low bias/high variance or high bias/low variance model? How do you know?

Using only (x_1, x_2), find a solution that fits the training data. What training and test error do you achieve? Is this a low bias/high variance or high bias/low variance model? How do you know?

For both solutions, detail your hyperparameter choices by providing a screenshot and the URL to your solution (the URL contains all your settings choices).

Bonus: Can you find two engineered features that would allow you to find a solution with a linear model?