DDA4210 - 24 Spring **Assignment Guru: Yifei Wu**

Assignment 1: Basics, Bias-Variance, LS Opt., and Alg. Analysis

Assigned: Jan./1/9 Due: Jan./1/31 at 10:00 p.m.

Rules

- **English:** Answer the questions in English. Otherwise, you'll lose half of the points.
- **Electronic submission:** Turn in solutions electronically via Blackboard. Be sure to submit your homework as a single readable file.
- **Collaboration policy:** Collaboration is allowed for all problems, but please list all the people with whom you discussed. Crediting help from other classmates will not take away any credit from you.

  Notably, only insightful discussions are allowed. Directly sharing the solutions is prohibited. The details of the collaboration policy for this course are available in the Resources tab on Piazza.
- **Time issue:** Note that late submissions will result in discounted scores: 0-24 hours $\rightarrow$ 80%, 24-72 hours $\rightarrow$ 50%, 72 or more hours$\rightarrow$ 0%.
- **Score and weight:**

  DDA4210: 100 points, the weight of this assignment in the final grade is 10%.
  AIR6002: 150 points, the weight of this assignment in the final grade is 15%.
- **Important Note:** Problems/Sub-problems with a label [**AIR6002**] are only required for master students in **AIR6002**.
- **Questions on assignment:** Start early and come to TA office hours with your questions on the assignments.

1. Basics (you are assumed to know the answers before taking this course)
Answer each of the following problems with 1-2 short sentences. [8 points]

(a) What is a hypothesis set?

(b) What is the hypothesis set of a linear model?

(c) What is overfitting?

(d) What are two ways to prevent overfitting?

(e) 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?

(f) What are the two assumptions we make about how our dataset is sampled?

(g) 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?

(h) What is the k-fold cross-validation procedure?

2. Bias-Variance Tradeoff

(a) 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: [10 points]

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

(b) [**AIR6002**] 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 `notebook_2.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 scikitlearn'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. [10 points]

3. Find the closed-form solutions of the following optimization problems ($\mathbf{W} \in \mathbb{R}^{K \times D}, N \gg D > K$):

(a) $\text{minimize}_{\mathbf{W},\mathbf{b}} \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{W}\mathbf{x}_i - \mathbf{b}\|^2$ [5 points]

(b) $\text{minimize}_{\mathbf{W},\mathbf{b}} \frac{1}{2} \sum_{i=1}^{N} \|\mathbf{y}_i - \mathbf{W}\mathbf{x}_i - \mathbf{b}\|^2 + \frac{\lambda}{2}\|\mathbf{W}\|_F^2$ [7 points]

4. Consider the following problem

$$\underset{\mathbf{W}}{\text{minimize}} \ \frac{1}{2}\|\mathbf{W}\Phi - \mathbf{Y}\|_F^2 + \frac{\lambda}{2}\|\mathbf{W}\|_F^2$$

where $\|\cdot\|_F$ denotes the Frobenius norm; $\mathbf{Y} \in \mathbb{R}^{K \times N}$, $\Phi = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \ldots, \phi(\mathbf{x}_N)]$, $\mathbf{x}_i \in \mathbb{R}^D$, $i = 1, 2, \ldots, N$, and ϕ is the feature map induced by a kernel function $k(\cdot, \cdot)$. Prove that for any $\mathbf{x} \in \mathbb{R}^D$, we can make prediction as

$$\mathbf{y} = \mathbf{W}\phi(\mathbf{x}) = \mathbf{Y}\left(\mathbf{K} + \lambda\mathbf{I}\right)^{-1}\mathbf{k}(\mathbf{x}),$$

where $\mathbf{K} = \Phi^\top\Phi$ and $\mathbf{k}(\mathbf{x}) = [k(\mathbf{x}_1, \mathbf{x}), k(\mathbf{x}_2, \mathbf{x}), \ldots, k(\mathbf{x}_N, \mathbf{x})]^\top$. [14 points]

5. Compute the space and time complexities (in the form of big O, consider only the training stage) of the following algorithms: [6 points]

(a) Ridge regression (Question 2(b)) with the closed-form solution

(b) PCA (N data points of D-dimension, choose d principal components)

(c) Neural network with architecture $D - H_1 - H_2 - K$ on a mini-batch of size B (consider only the forward process and neglect the computational costs of activation functions)

* [Hint: the time complexity of $A \in \mathbb{R}^{m\times n} \times B \in \mathbb{R}^{n\times l}$ is $O(mnl)$; the time complexities of eigenvalue decomposition and inverse of an $n \times n$ matrix are both $O(n^3)$.]

6. Prove the convergence of the generic gradient boosting algorithm (AnyBoost). Specifically, suppose in the algorithm of AnyBoost (page 14 of Lecture 02), the gradient of the objective function $\mathcal{L}$ is L-Lipschitz continuous, i.e., there exists $L > 0$ such that

$$\|\nabla\mathcal{L}(H) - \nabla\mathcal{L}(H')\| \leq L\|H - H'\|$$

holds for any H and H'. Suppose in the algorithm, α is computed as

$$\alpha_{t+1} = -\frac{\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle}{L\|h_{t+1}\|^2}.$$

Then the ensemble model is updated as $H_{t+1} = H_t + \alpha_{t+1}h_{t+1}$. Prove that the algorithm either terminates at round T with $\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle$ or $\mathcal{L}(H_t)$ converges to a finite value, in which case

$$\lim_{t\to\infty}\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle = 0.$$

* [Hint: Using the following result: Suppose $\mathcal{L} : \mathcal{H} \to \mathbb{R}$ and $\|\nabla\mathcal{L}(F) - \nabla\mathcal{L}(G)\| \leq L\|F - G\|$ holds for any F and G in $\mathcal{H}$, then $\mathcal{L}(F + wG) - \mathcal{L}(F) \leq w\langle\nabla\mathcal{L}(F), G\rangle + \frac{Lw^2}{2}\|G\|^2$ holds for any $w > 0$.]
[14 points]

7. Stochastic gradient descent (SGD) is an important optimization tool in machine learning, used every- where 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:

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

(a) We can make our algebra and coding simpler by writing $f\left(x_1, x_2, \cdots, x_d\right) = \mathbf{w}^T\mathbf{x}$ 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?
[3 points]

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:

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

(b) 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.

[3 points]

[**AIR6002**] The following few problems ask you to work with the first of two provided Jupyter notebooks for this problem, `notebook_7_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. In addition, to run the animation code provided in this notebook, you may need to install FFmpeg, which includes a library for handling multimedia data. For step-by-step instructions on installing FFmpeg, please refer to the file `installing_ffmpeg.pdf`.

For your implementation of problems (c)-(e), do not consider the bias term.

(c) [**AIR6002**] 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. [10 points]

(d) [**AIR6002**] 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. [3 points]

(e) [**AIR6002**] 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 $\eta \in \{1\text{e}-6, 5\text{e}-6, 1\text{e}-5, 3\text{e}-5, 1\text{e}-4\}$. 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? [6 points]

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 `notebook_7_part2.ipynb`.

For your implementation of problems (f)-(h), do consider the bias term using your answer to problem (a).

(f) [**AIR6002**] Use your SGD code with the given dataset, and report your final weights. Follow the guidelines below for your implementation: [6 points]

- 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 1000 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.

(g) [**AIR6002**] Perform SGD as in the previous problem for each learning rate η in

$$\left\{e^{-10}, e^{-11}, e^{-12}, e^{-13}, e^{-14}, e^{-15}\right\},$$

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. [3 points]

(h) [**AIR6002**] 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? [3 points]

Answer the remaining questions in 1-2 short sentences.

(i) [**AIR6002**] Is there any reason to use SGD when a closed form solution exists? [3 points]

(j) [**AIR6002**] 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. [3 points]

(k) [**AIR6002**] How does the convergence behavior of the weight vector differ between the perceptron and SGD algorithms? [3 points]

8. True or False? If False, then explain shortly. [10 points]

(a) The inequality $G(\mathcal{F}, n) \leq n^2$ holds for any model class $\mathcal{F}$.

(b) The VC dimension of an axis-aligned rectangle in a 2D space is 4.

(c) The VC dimension of a circle in a 2D space is 4.

(d) The VC dimension of 1-nearest neighbor classifier in d-dimensional space is $d + 1$.

(e) Let d be the VC dimension of $\mathcal{F}$. Then the inequality $G(\mathcal{F}, n) \leq \left(\frac{en}{d}\right)^d$ always holds.

9. In LASSO, the model class is defined as $\mathcal{F} = \{\mathbf{x} \mapsto \langle \mathbf{w}, \mathbf{x} \rangle : \|\mathbf{w}\|_1 \leq \alpha\}$. Suppose $\mathbf{x} \in \mathbb{R}^d$, $y \in \{-1, +1\}$, the training data are $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, and $\max_{1 \leq i \leq n} \|\mathbf{x}_i\|_\infty \leq \beta$, where $\|\cdot\|_\infty$ denotes the largest absolute element of a vector.

(a) Find an upper bound of the empirical Rademacher complexity

$$\mathcal{R}_S(\mathcal{F}) = \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \sigma_i f(\mathbf{x}_i) \right]$$

, where σ_i are the Rademacher variables. [15 points]

(b) Suppose the loss function is the absolute loss. Use the inequality (highlighted in blue) on page 30 and Lemma 5 on page 35 (i.e., $\mathcal{R}(\ell \circ \mathcal{F}) \leq \eta \mathcal{R}(\mathcal{F})$) of Lecture 03 to derive a generalization error bound for LASSO. [5 points]

* [Hint: For question (a), please use the inequality $\langle \mathbf{a}, \mathbf{b} \rangle \leq \|\mathbf{a}\|_1 \|\mathbf{b}\|_\infty$ and the following lemma:

Lemma 1. *Let $A \subseteq \mathbb{R}^n$ be a finite set of points with $r = \max_{\mathbf{x} \in A} \|\mathbf{x}\|_2$ and denote $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Then*

$$\mathbb{E}_{\boldsymbol{\sigma}} \left[\max_{\mathbf{x} \in A} \sum_{i=1}^n x_i \sigma_i \right] \leq r \sqrt{2 \log |A|},$$

where $|A|$ denotes the cardinality of set A and σ_i are the Rademacher variables.]