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

Assignment 1 Solution

Note: The **blue** words represent the scoring points

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

(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?

Solution:

(a) A hypothesis set is **a set of functions (i.e. a function space)** from which the hypothesis function $f(\mathbf{x})$ is chosen to best approximate the target function. [1 points]

(b) The hypothesis set of a linear model is **all functions of the form $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + \mathbf{b}$**; in other words, this is the set of linear functions of $\mathbf{x}$ with all possible values of the weights and biases. [1 points]

(c) Overfitting occurs when a model has a much lower **in-sample error** than **out-of-sample error**. This usually happens when the model is fitted too closely to the in-sample error and loses **robustness**. [1 points]

(d) Any of the following answers are acceptable:

- **Regularization**, which adds a penalty to the model complexity as complex models are more likely to overfit.
- **Cross-validation**, which monitors training error by comparing the validation error of each partition.

– **Using fewer features or more data**, which prevents overfitting by increasing model stability. [1 points]

(e) Training data is the data used to **generate models** and is occasionally further partitioned to use for validation. Test data is the data used to **evaluate a model to get an accurate representation** of how the model would perform on additional data. It is important to never use information from the test data to modify the model to avoid the risk of contaminating the model. [1 points]

(f) We assume that the training data is sampled from the **same distribution** as the target data, and that each data point is sampled independently from the rest. We call such data "independently and identically distributed" or **"i.i.d."**. [1 points]

(g) A potential input space may be the **bag-of-words representation** of the emails. The output space is **a binary decision** of whether an email is spam. [1 points]

(h) The dataset is divided into k subsets, and a subset is chosen to be used as a holdout validation set with the other $k-1$ subsets used to train a model. This procedure is repeated k times, once for each subset. The training and validation errors are then averaged across the k models. This reduces the variance from having just a single holdout set, but increases the computation required. [1 points]

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.

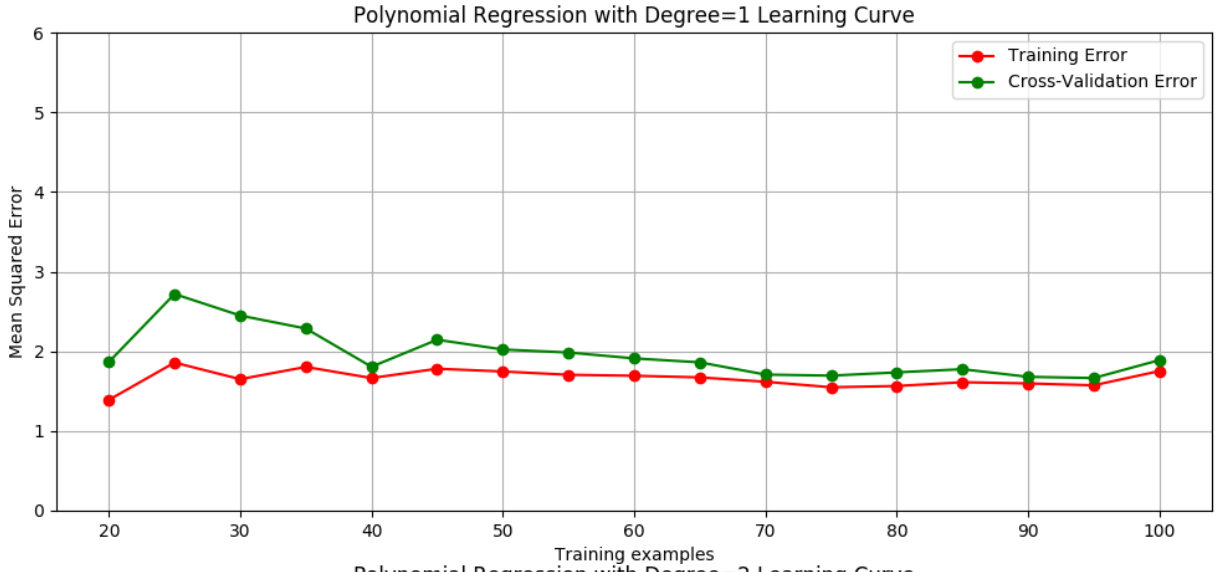
(2) Create a learning curve by plotting both the average training and validation error as functions of N. [10 points]

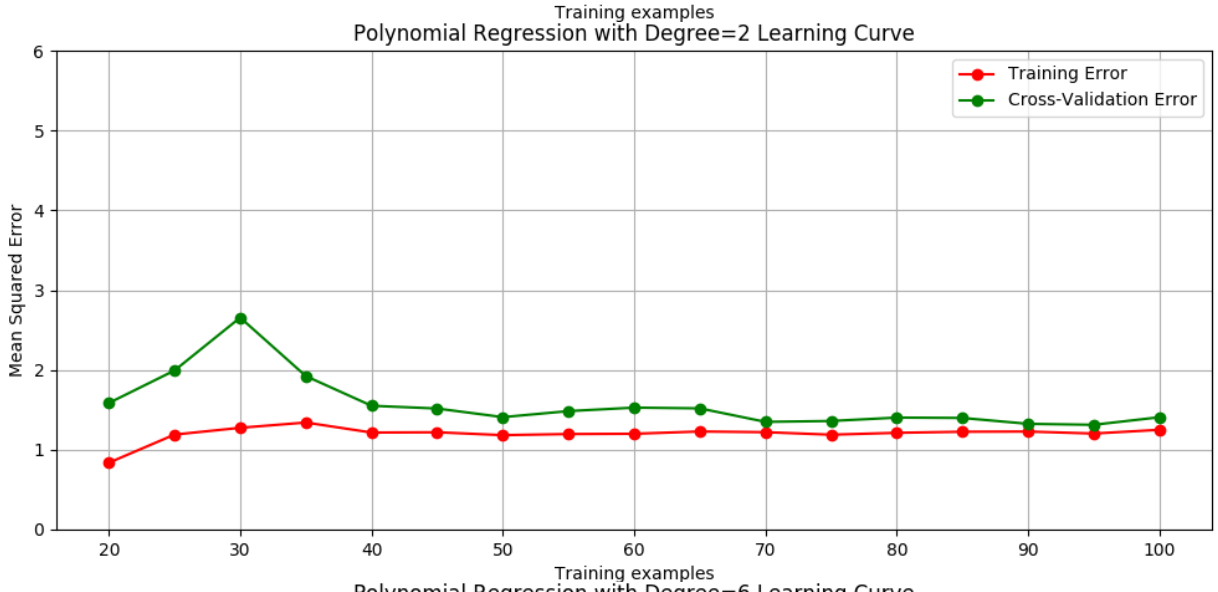
Solution:

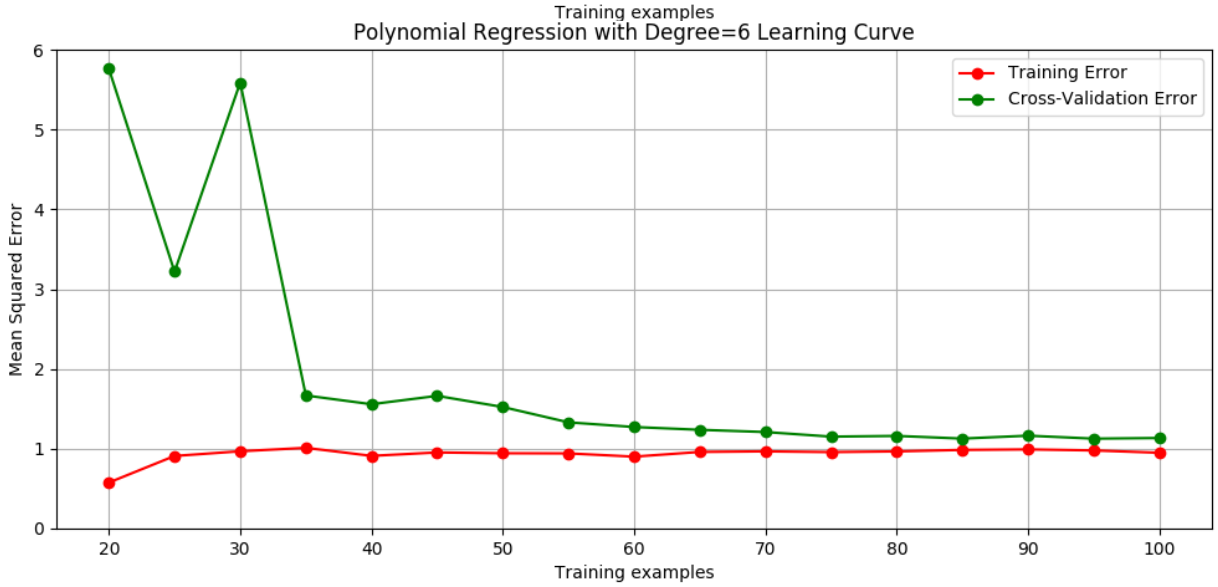
(a) Scoring key point: Using $F(x) = \mathbb{E}_S[f_S(x)]$ twice:

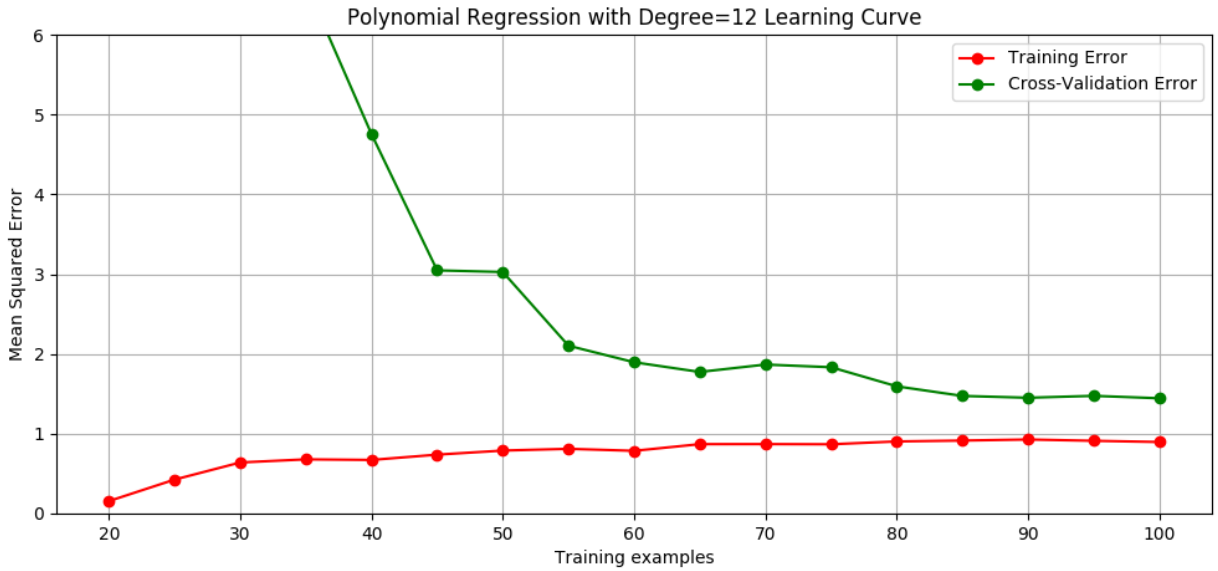
$$\begin{aligned}
\mathbb{E}_S\left[E_{\text{out}}(f_S)\right] &= \mathbb{E}_S\left[\mathbb{E}_x\left[(f_S(x) - y(x))^2\right]\right] \\
&= \mathbb{E}_S\left[\mathbb{E}_x\left[f_S(x)^2 + y(x)^2 - 2f_S(x)y(x)\right]\right] \\
&= \mathbb{E}_x\left[\mathbb{E}_S\left[f_S(x)^2 - 2f_S(x)y(x)\right] + y(x)^2\right]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

(b) See the relevant file for the solution code. The graphs are as follows:


Polynomial Regression with Degree=1 Learning Curve
Training Error
Cross-Validation Error
Mean Squared Error
Training examples
0
1
2
3
4
5
6
20
30
40
50
60
70
80
90
100


Polynomial Regression with Degree=2 Learning Curve
Training Error
Cross-Validation Error
Mean Squared Error
Training examples
0
1
2
3
4
5
6
20
30
40
50
60
70
80
90
100


Polynomial Regression with Degree=6 Learning Curve
Training Error
Cross-Validation Error
Mean Squared Error
Training examples
0
1
2
3
4
5
6
20
30
40
50
60
70
80
90
100



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]

Solution:

(a) You need to be familiar with some matrix and norm computations.

$$\begin{aligned}\sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{W}\boldsymbol{x}_i - \boldsymbol{b}\|^2 &= \sum_{i=1}^{N} \left\| \begin{pmatrix} y_{i,1} \\ \vdots \\ y_{i,K} \end{pmatrix} - \begin{pmatrix} W_{1,1} & \cdots & W_{1,D} \\ \vdots & \ddots & \vdots \\ W_{K,1} & \cdots & W_{K,D} \end{pmatrix} \begin{pmatrix} x_{i,1} \\ \vdots \\ x_{i,D} \end{pmatrix} - \begin{pmatrix} b_1 \\ \vdots \\ b_K \end{pmatrix} \right\|^2 \\ &= \left\| \begin{pmatrix} \boldsymbol{y}_1^T \\ \vdots \\ \boldsymbol{y}_N^T \end{pmatrix} - \begin{pmatrix} 1 & \boldsymbol{x}_1^T \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_N^T \end{pmatrix} \begin{pmatrix} b_1 & \cdots & b_K \\ W_{1,1} & \cdots & W_{1,K} \\ \vdots & \ddots & \vdots \\ W_{D,1} & \cdots & W_{D,K} \end{pmatrix} \right\|_F^2 \\ &= \|\boldsymbol{Y} - \bar{\boldsymbol{X}}\bar{\boldsymbol{W}}\|_F^2,\end{aligned}$$

where $\boldsymbol{Y} = \begin{pmatrix} \boldsymbol{y}_1^T \\ \vdots \\ \boldsymbol{y}_N^T \end{pmatrix} \in \mathbb{R}^{N\times K}$, $\bar{\boldsymbol{X}} = \begin{pmatrix} 1 & \boldsymbol{x}_1^T \\ \vdots & \vdots \\ 1 & \boldsymbol{x}_N^T \end{pmatrix} \in \mathbb{R}^{N\times(D+1)}$, and $\bar{\boldsymbol{W}} = \begin{pmatrix} b_1 & \cdots & b_K \\ W_{1,1} & \cdots & W_{1,K} \\ \vdots & \ddots & \vdots \\ W_{D,1} & \cdots & W_{D,K} \end{pmatrix} \in \mathbb{R}^{(D+1)\times K}$.

$$J_1(\bar{\boldsymbol{W}}) = J_1(\boldsymbol{W}, \boldsymbol{b}) = \sum_{i=1}^{N} \|\boldsymbol{y}_i - \boldsymbol{W}\boldsymbol{x}_i - \boldsymbol{b}\|^2 = \text{Tr}\left((\boldsymbol{Y} - \bar{\boldsymbol{X}}\bar{\boldsymbol{W}})^T(\boldsymbol{Y} - \bar{\boldsymbol{X}}\bar{\boldsymbol{W}})\right).$$

Let $\frac{\partial J_1}{\partial \bar{\boldsymbol{W}}} = -2\bar{\boldsymbol{X}}^T(\boldsymbol{Y} - \bar{\boldsymbol{X}}\bar{\boldsymbol{W}}) = 0$, we have $\bar{W}^* = \begin{pmatrix} \boldsymbol{b}^{*T} \\ \boldsymbol{W}^{*T} \end{pmatrix} = \left(\bar{\boldsymbol{X}}^T\bar{\boldsymbol{X}}\right)^{-1}\bar{\boldsymbol{X}}^T\boldsymbol{Y}$.

(b) $J_2(\bar{\boldsymbol{W}}) = J_2(\boldsymbol{W}, \boldsymbol{b}) = \frac{1}{2}\mathrm{Tr}\left[(\boldsymbol{Y} - \bar{\boldsymbol{X}}\bar{\boldsymbol{W}})^T(\boldsymbol{Y} - \bar{\boldsymbol{X}}\bar{\boldsymbol{W}})\right] - \frac{1}{2}\mathrm{Tr}\left(\bar{\boldsymbol{W}}^T\boldsymbol{\Lambda}\bar{\boldsymbol{W}}\right)$, where $\boldsymbol{\Lambda} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & \lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda \end{pmatrix} = \begin{pmatrix} 0 & \mathbf{0} \\ \mathbf{0} & \lambda \boldsymbol{I}_D \end{pmatrix} \in \mathbb{R}^{(D+1)\times(D+1)}$.

Let $\frac{\partial J_2}{\partial \bar{\boldsymbol{W}}} = -\bar{\boldsymbol{X}}^T(\boldsymbol{Y} - \bar{\boldsymbol{X}}\bar{\boldsymbol{W}}) + \boldsymbol{\Lambda}\bar{\boldsymbol{W}} = 0$, we have $\bar{W}^* = \begin{pmatrix} \boldsymbol{b}^{*T} \\ \boldsymbol{W}^{*T} \end{pmatrix} = \left(\bar{\boldsymbol{X}}^T\bar{\boldsymbol{X}} + \boldsymbol{\Lambda}\right)^{-1}\bar{\boldsymbol{X}}^T\boldsymbol{Y}$.

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]

Solution:

Denote $J(\boldsymbol{W}) := \frac{1}{2}\|\boldsymbol{W}\boldsymbol{\Phi} - \boldsymbol{Y}\|_F^2 + \frac{\lambda}{2}\|\boldsymbol{W}\|_F^2$, let $\frac{\partial J}{\partial \boldsymbol{W}} = (\boldsymbol{W}\boldsymbol{\Phi} - \boldsymbol{Y})\boldsymbol{\Phi}^T + \lambda\boldsymbol{W} = 0$, we have $\boldsymbol{W}^* = \boldsymbol{Y}\boldsymbol{\Phi}^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1}$.
Claim: $\boldsymbol{\Phi}^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1}\boldsymbol{\Phi}^T$.
Proof of the claim:

$$\begin{aligned}(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)\boldsymbol{\Phi}^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1} &= (\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T + \lambda\boldsymbol{\Phi}^T)(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1} \\ &= \boldsymbol{\Phi}^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1} \\ &= \boldsymbol{\Phi}^T\end{aligned}$$

Therefore,

$$(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1}(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)\boldsymbol{\Phi}^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1} = \boldsymbol{\Phi}^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1}\boldsymbol{\Phi}^T$$

We can make prediction as:

$$\begin{aligned}\mathbf{y} &= \mathbf{W}\phi(\mathbf{x}) \\ &= \boldsymbol{Y}\boldsymbol{\Phi}^T(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1}\phi(\boldsymbol{x}) \\ &= \boldsymbol{Y}(\boldsymbol{\Phi}^T\boldsymbol{\Phi} + \lambda I)^{-1}\boldsymbol{\Phi}^T\phi(\boldsymbol{x}) \\ &= \mathbf{Y}(\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{k}(\mathbf{x}) \ \left(\text{since } \mathbf{k}(\mathbf{x}) = [k(\mathbf{x}_1, \mathbf{x}), \ldots, k(\mathbf{x}_N, \mathbf{x})]^\top = \left[\phi(\mathbf{x}_1)^T\phi(\mathbf{x}), \ldots, \phi(\mathbf{x}_N)^T\phi(\mathbf{x})\right]^\top\right)\end{aligned}$$

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 3(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 $O(n^3)$.]

Solution:

(a) $\bar{W}^* = \left(\bar{\boldsymbol{X}}^T \bar{\boldsymbol{X}} + \boldsymbol{\Lambda}\right)^{-1} \bar{\boldsymbol{X}}^T \boldsymbol{Y}$.

$\bar{\mathbf{X}} \in \mathbb{R}^{N \times (D+1)}, \boldsymbol{\Lambda} \in \mathbb{R}^{(D+1) \times (D+1)}$ and $\mathbf{Y} \in \mathbb{R}^{N \times K}$,

Space complexity:

The space complexity of computing $\bar{\mathbf{W}}^*$ can also be broken down into three main steps: storing $\bar{\boldsymbol{X}}^T \bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}^T \mathbf{Y}$, and storing the inverse matrix.

- Storing $\bar{\boldsymbol{X}}^T \bar{\boldsymbol{X}}$ requires a matrix of size $(D+1) \times (D+1)$, which has space complexity $\mathcal{O}(D^2)$.
- Storing $\bar{\boldsymbol{X}}^T \mathbf{Y}$ requires a matrix of size $(D+1) \times K$, which has space complexity $\mathcal{O}(DK)$.
- If we use the formula $(\bar{\boldsymbol{X}}^T \bar{\boldsymbol{X}} + \boldsymbol{\Lambda})^{-1}$ to compute the inverse, we need to store a matrix of size $(D+1) \times (D+1)$, which has space complexity $\mathcal{O}(D^2)$.

Therefore, the overall space complexity of computing $\bar{\mathbf{W}}^*$ is dominated by the storage of the two intermediate matrices, which have space complexity $\mathcal{O}(D^2 + DK)$. The space complexity of storing the inverse matrix depends on the method used, but is typically also of order $\mathcal{O}(D^2)$.

Thus, the overall space complexity is $\mathcal{O}(D^2 + DK)$.

Time complexity:

computing $\bar{\boldsymbol{X}}^T \bar{\boldsymbol{X}}, \bar{\boldsymbol{X}}^T \mathbf{Y}$, and matrix inversion.

- The matrix multiplication $\bar{\boldsymbol{X}}^T \bar{\boldsymbol{X}}$ has time complexity $\mathcal{O}(ND^2)$.
- The multiplication $\bar{\boldsymbol{X}}^T \mathbf{Y}$ has time complexity $\mathcal{O}(NKD)$.
- The matrix inversion using the formula $(\bar{\boldsymbol{X}}^T \bar{\boldsymbol{X}} + \boldsymbol{\Lambda})^{-1}$ have time complexity $\mathcal{O}(D^3)$.

Therefore, the overall time complexity of computing $\bar{\mathbf{W}}^*$ is dominated by the matrix inversion step, which has time complexity $\mathcal{O}(D^3)$.

Hence, the overall time complexity of computing $\bar{\mathbf{W}}^*$ is $\mathcal{O}(ND^2 + NKD + D^3)$.

(b) **Space complexity:**

- The input data, which consists of N data points of D dimensions, requires $\mathcal{O}(ND)$ space.
- The covariance matrix, which is a $D \times D$ matrix that needs to be computed from the input data, requires $\mathcal{O}(D^2)$ space.

- The eigenvectors and eigenvalues of the covariance matrix. Since we only need to compute d principal components, we only need to store the d eigenvectors with the largest eigenvalues. Each eigenvector has D elements, so the total space required for storing the eigenvectors is $\mathcal{O}(Dd)$.
- The projected data, which is the input data transformed into the new coordinate system defined by the principal components. This requires $\mathcal{O}(ND)$ space.

Therefore, the overall space complexity is dominated by the space required for storing the input data and the projected data, which is $\mathcal{O}(ND)$.

Time complexity:

- Computing the mean of the input data, which is a D-dimensional vector, requires a single pass over all N data points, and therefore has a time complexity of $\mathcal{O}(ND)$.
- Computing the covariance matrix from the centered input data. This requires a matrix multiplication between an $N \times D$ matrix and its transpose, which has a time complexity of $\mathcal{O}(ND^2)$.
- Computing the d eigenvectors with the largest eigenvalues of the covariance matrix, if we use eigenvalue decomposition, a time complexity of $\mathcal{O}(D^3)$ is required.
- Projecting the input data onto the new coordinate system defined by the principal components. This requires a matrix multiplication between the centered input data, which is an $N \times D$ matrix, and the $D \times d$ matrix of eigenvectors. Therefore, it has a time complexity of $\mathcal{O}(NDd)$.

Therefore, the overall time complexity is dominated by the time required for computing the covariance matrix , which is $\mathcal{O}(ND^2)$.

(c) **Space complexity:**

For neural networks with H_1 neurons in the first hidden layer, H_2 neurons in the second hidden layer, and K output neurons, the number of parameters in the network is:

$$(D+1)H_1 + (H_1+1)H_2 + (H_2+1)K,$$

where the $+1$ term in each layer corresponds to the bias term.

The space required to store the input data and the intermediate values of the network during the forward pass is proportional to the batch size B.

Therefore, the space complexity is $O(BD + BH_1 + BH_2 + BK)$;

Time complexity:

Assuming that the activation functions are computationally cheap and the forward pass involves only matrix multiplication and addition operations, the number of operations required to process one input example in the network is proportional to the number of weights in the network.

- The number of weights in the network is:$(D+1)H_1 + (H_1+1)H_2 + (H_2+1)K$, where the $+1$ term in each layer corresponds to the bias term. Therefore, the time complexity of processing one input example in the network is: $O(DH_1 + H_1H_2 + H_2K)$.

– To process a mini-batch of size B, we need to perform the forward pass on each input example in the batch. Therefore, the time complexity on a mini-batch of size B is: $O(BDH_1 + BH_1H_2 + BH_2K)$.

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 $\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle \geq 0$ 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]

Solution:

Since $\mathcal{L}$ is Lipschitz continuous, we have

$$\begin{aligned}
\mathcal{L}(H_t) - \mathcal{L}(H_{t+1}) &= \mathcal{L}(H_t) - \mathcal{L}(H_t + \alpha_{t+1}h_{t+1}) \\
&\geq -\alpha_{t+1}\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle - \frac{L\alpha_{t+1}^2}{2}\|h_{t+1}\|^2, \forall\alpha_{t+1} > 0 \text{ (by hint)} \\
&= \frac{(\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle)^2}{L\|h_{t+1}\|^2} - \frac{(\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle)^2}{2L\|h_{t+1}\|^2} \\
&= \frac{(\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle)^2}{2L\|h_{t+1}\|^2} \geq 0.
\end{aligned}$$

$\{\mathcal{L}(H_t)\}_{t=1}^{\infty}$ generated by the algorithm will monotonically decrease if $\alpha > 0$.

Therefore, AnyBoost will terminate when stepsize $\alpha \leq 0$ or $\lim_{t\to\infty}(\mathcal{L}(H_t) - \mathcal{L}(H_{t+1})) = 0$, i.e. terminate when:

$$-\frac{\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle}{L\|h_{t+1}\|} \leq 0 \Longrightarrow \langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle \geq 0$$

Or $\lim_{t\to\infty}\langle\nabla\mathcal{L}(H_t), h_{t+1}\rangle = 0$ and $\lim_{t\to\infty}(H_t - H_{t+1}) = 0$ (Taylor approximation), i.e. $\mathcal{L}(H_t)$ converges to a finite value since $\{H_t\}_{t+1}^{\infty}$ converges.

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:

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

(a) 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?

[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]

Solution:

(a) Hint: Include an additional element in $\mathbf{w}$ and $\mathbf{x}$.

We can simply add a 1 to the $\mathbf{x}$ vector and include the bias term $\mathbf{b}$ in the weight vector $\mathbf{w}$. Thus we write

$$\mathbf{w} = [b, w_1, w_2, \cdots, w_d], \mathbf{x} = [1, x_1, x_2, \cdots, x_d]$$

Note that the formula then evaluates to $\mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b$, which is exactly how linear regression is defined.

(b) The gradient with respect to the point (x_i, y_i) is:

$$\begin{aligned}\nabla_{\mathbf{w}} L &= \sum_{i=1}^{N} 2(y_i - \mathbf{w}^T\mathbf{x}_i)\nabla_{\mathbf{w}}(y_i - \mathbf{w}^T\mathbf{x}_i) \\ &= \sum_{i=1}^{N}[-2\mathbf{x}_i(y_i - \mathbf{w}^T\mathbf{x}_i)]\end{aligned}$$

(c) Scoring key point: The code needs to conform to the loss function, the termination condition, and the outputs proposed in the question. And it would be good to give a brief explanation here.

(d) These different starting points converge to the same global optimum, but SGD converges more quickly when the starting point is closer to the global minimum, and converges more slowly when the starting point lies in a region of the parameter space where the loss function has a flatter slope. Different datasets result in different loss functions over which to optimize, and different loss functions converge at different rates.

(e) As the learning rate increases, SGD converges faster. If the learning rate is too high, then SGD oscillates rather than converging.

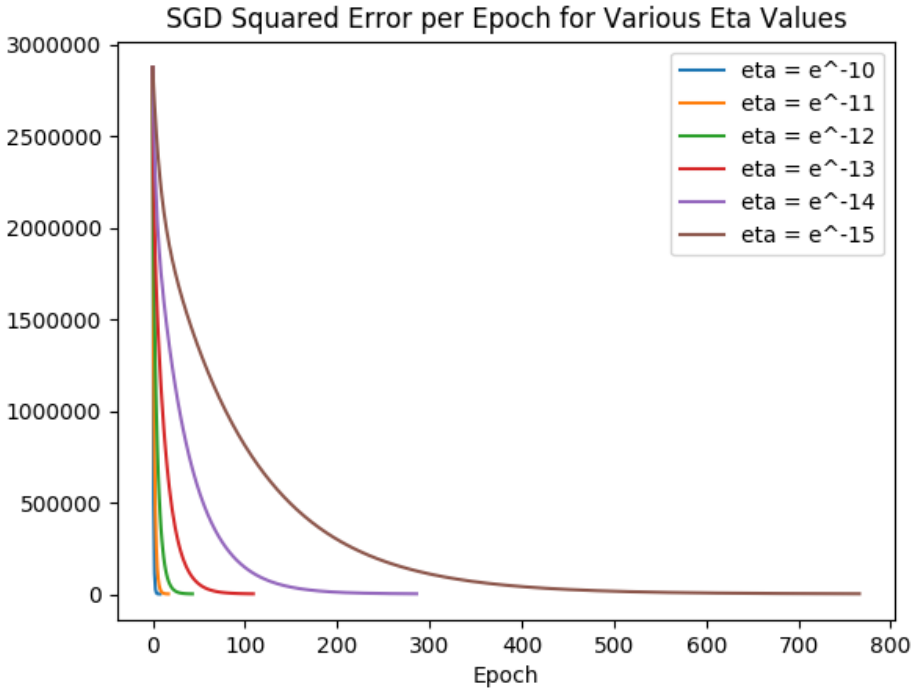
(f) Scoring key point: The code needs to conform to the step size, the initial weight and bias, and the outputs proposed in the question. And it would be good to give a brief explanation here.

The final weights should look similar to:

$$\mathbf{w} = [-0.2271, -5.9307, 3.9303, -11.6852, 8.7492]$$

where the first element of the weight vector is w_0 as defined above.

(g) The higher the learning rate, the faster SGD converges and the error decreases. Note that in some situations, if the learning rate is too high, SGD may oscillate around the optimal solution and never converge.



The total squared error is not important here, as it may be scaled by any constant. What matters is the shape of the plots.

(h) The two solutions should be reasonably close. The analytical solution should be similar to:

$$\mathbf{w} = [-0.3164, -5.9916, 4.0151, -11.9333, 8.9906]$$

using linear regression.

(i) Yes. In many cases, calculating the closed form solution is computationally intractable, in which case SGD can be used to get an arbitrarily good approximation. Also, SGD does not require operations on the entire dataset at once, and can return a valid hypothesis at any iteration.

(j) Answers may vary. For instance, one can keep track of the loss reduction from epoch to epoch, and stop when the relative loss reduction compared to the first epoch is less than some small value ϵ , such as 0.0001. That is, if $\triangle_{0,1}$ denotes the loss reduction from the initial model to end of the first epoch, and $\triangle_{i-1,i}$ denotes the loss reduction between the $(i-1)$th and ith epoch, then stop after epoch t if $\triangle_{t-1,t}/\triangle_{0,1} \leq \epsilon$.

(k) With SGD, the weights converge much more smoothly than with the perceptron algorithm, and the user can control the smoothness of the algorithm's convergence by modifying the learning rate. The perceptron algorithm is relatively jumpy compared to SGD, assuming that the SGD learning rate is not overly large.

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.

Solution:

(a) False. Since $G(\mathcal{F}, n) \leq 2^n$ holds for any model class $\mathcal{F}$, $G(\mathcal{F}, n)$ can be larger than n^2 when $n > 2$.

(b) True.

(c) False. The VC dimension of a cycle in a 2D space is 3.

(d) False. The VC dimension of 1-nearest neighbor classifier is infinite.

(e) False. The inequality $G(\mathcal{F}, n) \leq (\frac{en}{d})^d$ holds when $n \geq d$.

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.]

Solution:

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

$$\begin{aligned}
\mathcal{R}(\mathcal{F}) &= \mathbb{E}_\sigma \left[\sup_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} \sigma_i f(\mathbf{x}_i) \right] \\
&= \mathbb{E}_\sigma \left[\sup_{\|\mathbf{w}\|_1 \leq \alpha} \frac{1}{n} \sum_{i=1}^{n} \sigma_i \langle \mathbf{w}, \mathbf{x}_i \rangle \right] \\
&= \mathbb{E}_\sigma \left[\frac{1}{n} \sup_{\|\mathbf{w}\|_1 \leq \alpha} \left\langle \mathbf{w}, \sum_{i=1}^{n} \sigma_i \mathbf{x}_i \right\rangle \right] \\
&\leq \mathbb{E}_\sigma \left[\frac{1}{n} \sup_{\|\mathbf{w}\|_1 \leq \alpha} \|\mathbf{w}\|_1 \left\| \sum_{i=1}^{n} \sigma_i \mathbf{x}_i \right\|_\infty \right] \text{ (use } \langle \mathbf{a}, \mathbf{b} \rangle \leq \|\mathbf{a}\|_1 \|\mathbf{b}\|_\infty) \\
&\leq \mathbb{E}_\sigma \left[\frac{1}{n} \alpha \left\| \sum_{i=1}^{n} \sigma_i \mathbf{x}_i \right\|_\infty \right] \\
&= \frac{\alpha}{n} \mathbb{E}_\sigma \left[\left\| \sum_{i=1}^{n} \sigma_i \mathbf{x}_i \right\|_\infty \right]
\end{aligned}$$

For $j = 1, \ldots, d$, let $\mathbf{v}_j = \left[\mathbf{x}_1^{(j)}, \ldots, \mathbf{x}_n^{(j)} \right]^\top \in \mathbb{R}^n$ and $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_d, -\mathbf{v}_1, \ldots, -\mathbf{v}_d\}$. Due to $\left| \mathbf{x}_i^{(j)} \right| \leq \max_i \|\mathbf{x}_i\|_\infty$, we have $\max_j \|\mathbf{v}_j\|_2 \leq \sqrt{n} \max_i \|\mathbf{x}_i\|_\infty$.

$$\begin{aligned}
\mathcal{R}(\mathcal{F}) &\leq \frac{\alpha}{n} \mathbb{E}_\sigma \left[\left\| \sum_{i=1}^{n} \sigma_i \mathbf{x}_i \right\|_\infty \right] \\
&= \frac{\alpha}{n} \mathbb{E}_\sigma \left[\max_{\mathbf{v} \in V} \sum_{i=1}^{n} \sigma_i \mathbf{v}^{(i)} \right] \\
&\leq \frac{\alpha}{n} \max_j \|\mathbf{v}_j\|_2 \sqrt{2 \log |V|} \quad \text{(use Lemma 1 or Massart Lemma)} \\
&\leq \frac{\alpha}{n} \sqrt{n} \max_i \|\mathbf{x}_i\|_\infty \sqrt{2 \log(2d)} \\
&\leq \alpha \beta \sqrt{\frac{2 \log(2d)}{n}}
\end{aligned}$$

Note: see Lemma 26.8 for Massart Lemma, see Lemma 26.11 for Rademacher complexity of linear classes (ref. *Understanding Machine Learning From Theory to Algorithms*)

(b) Derive a generalization error bound for LASSO.

The inequality of Lecture 3: with the probability of at least $1-\delta$, we have

$$\sup_{f\in\mathcal{F}}\left\{\mathbb{E}[\ell(f(\mathbf{x}),y)]-\frac{1}{n}\sum_{i=1}^{n}\ell\left(f\left(\mathbf{x}_i\right),y_i\right)\right\}\leq 2\mathcal{R}_S(\mathcal{F})+3\sqrt{\frac{\log(2/\delta)}{2n}}$$

Since by Lemma 5 of Lecture 3, $\mathcal{R}(\ell\circ\mathcal{F})\leq\eta\mathcal{R}(\mathcal{F})$, for absolute loss, $|\ell(x')-\ell(x)|=||x'|-|x||\leq\|x-x'\|$, thus $\eta=1$. Then,

$$\begin{aligned}\sup_{f\in\mathcal{F}}\left\{\mathbb{E}[\ell(f(\mathbf{x}),y)]-\frac{1}{n}\sum_{i=1}^{n}\ell\left(f\left(\mathbf{x}_i\right),y_i\right)\right\}&\leq 2\mathcal{R}_S(\mathcal{F})+3\sqrt{\frac{\log(2/\delta)}{2n}}\\&\leq 2\eta\alpha\beta\sqrt{\frac{2\log(2d)}{n}}+3\sqrt{\frac{\log(2/\delta)}{2n}}\\&=2\alpha\beta\sqrt{\frac{2\log(2d)}{n}}+3\sqrt{\frac{\log(2/\delta)}{2n}}\end{aligned}$$

where $\ell(\cdot)$ is the absolute loss function and ℓ is η-Lipschitz.