ISE - 364/464: Introduction to Machine Learning
Homework Assignment 5

This assignment requires the accompanying Python file "HW_5_Code.py" to complete problem 4 (found in the .zip file).

The goal of this assignment is to provide a series of problems that strengthen knowledge in regularization, multi-class classification, and tree-based models.

Grading: This assignment is due on Coursesite by E.O.D. 11/15/2024. All problems are worth the same number of points. If a problem has multiple parts, each of those parts will be worth equal amounts and will sum to the total number of points of the original problem (Example: If each problem is worth a single point, and problem 1 has 4 parts, each part will be worth 1/4th of a point). ISE - 364 students are only required to answer problems 1 through 4; however, you are allowed to answer the 5th graduate-level question (if done so correctly, you will receive extra credit in the amount that the 5th problem will be worth for the ISE - 464 students). ISE - 464 students are required to answer all 5 problems.

Submitting: Only electronic submissions on Coursesite are accepted. Students should submit a .zip file for this assignment, with two files inside: one file for the homework write-up and another for the code the students will write.

1 Problems

1. (**Optimal Parameters for the ℓ_2-Regularized Ridge Regression Model**) Recall that the Mean Squared Error (MSE) loss function for a linear regression model is defined as $J(\theta) := \frac{1}{2m}\|X\theta - y\|_2^2$, where $X \in \mathbb{R}^{m\times n}$ is the design matrix, $y \in \mathbb{R}^m$ is the target vector, and $\theta \in \mathbb{R}^n$ is the vector of parameters. Now consider the optimization problem of learning the optimal vector of parameters $\theta^\star$ for a ridge regression model; this problem is stated formally as
$$\min_{\theta\in\mathbb{R}^n} f(\theta;\lambda) := J(\theta) + \frac{1}{2}\lambda\|\theta\|_2^2,$$
where $\lambda > 0$ is the regularization parameter. Derive the optimal solution $\theta^\star$ to this problem. To do this, you may assume that X is full-column rank.
(Hint: The Hessian matrix of f with respect to θ will be positive definite when $\lambda > 0$ and X is full-column rank.)

2. (**Partial Derivatives of the Softmax Function**) Recall that, for a vector $z \in \mathbb{R}^k$, the softmax function is defined as

$$\text{softmax}\left(\begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_k \end{bmatrix}\right) = \begin{bmatrix} \frac{e^{z_1}}{\sum_j e^{z_j}} \\ \frac{e^{z_2}}{\sum_j e^{z_j}} \\ \vdots \\ \frac{e^{z_k}}{\sum_j e^{z_j}} \end{bmatrix} = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_k \end{bmatrix},$$

where ϕ_j are the probabilities corresponding to each of the $j \in \{1, 2, ..., k\}$ target classes.

a) Derive the partial derivative of the first entry of the output of the softmax function with respect to z_1. That is, compute $\frac{\partial \phi_1}{\partial z_1}$. Your final answer should be written in terms of the appropriate ϕ_j terms.

b) Derive the partial derivative of the first entry of the output of the softmax function with respect to z_2. That is, compute $\frac{\partial \phi_1}{\partial z_2}$. Your final answer should be written in terms of the appropriate ϕ_j terms.

3. (**Decision Trees by Hand**) Let $\mathcal{D}$ be a dataset consisting of two features $x_1 \in \mathbb{R}$ and $x_2 \in \mathbb{R}$ along with a target variable $y \in \{0, 1\}$. Specifically, suppose the dataset consists of the following 5 datapoints

$$\mathcal{D} := \{(10, 5, 1), (5, 8, 1), (8, 6, 0), (9, 10, 1), (5, 4, 0)\}.$$

a) Suppose that you wish to build a 1-layer deep decision tree (you will only utilize a single decision rule) that will have a minimal Gini Impurity. Determine the decision rule (of the form $x_j \leq \theta$) that returns the smallest Gini index and report the corresponding Gini index value.
(Hint: You should compute a total of 7 Gini indices to determine the optimal split.)

b) Based on the decision rule that you chose in (3a), write the regions S_1 and S_2. That is, determine which datapoints fall on the side of the hyperplane that satisfy your decision rule (this will be the set S_1) and which datapoints fall on the side that violate the rule (this will be the set S_2).

c) Draw (or plot in Python) the 5 datapoints in this dataset, draw the hyperplane corresponding to the decision rule you chose, and then shade the side of the hyperplane that satisfies the decision rule (this is called the feasible region).

4. (**Decision Trees, Random Forests, and Boosted Trees in Python**) For this problem, you will use Python to train three different machine learning models using SciKit-Learn: a decision tree, a random forest, and a gradient boosting model. Use the "HW_5_Code.py" file accompanying this document in the .zip file. This is the Python file that you will use to write your code. This file already has pre-written functions to generate the dataset that you will use to train your models on, perform an 80%-20% train-validation split of the data, and visualize the data as well as the learned decision boundaries. Follow the comments in the .py file; they should guide you to where to write your code. The dataset that you will be working with has two features and a target variable with three target classes (a classification problem), and is displayed in Figure 1. Also, Figure 2 displays

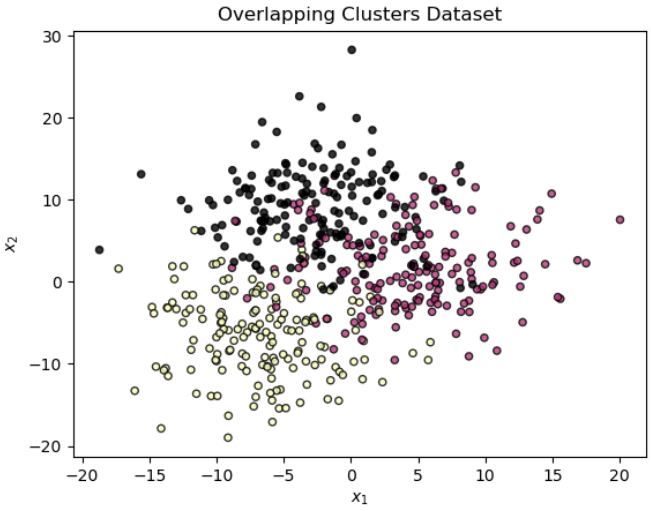

Figure 1: The three-cluster dataset that is used in problem 4.

the baseline performance plots that you should obtain for each of the three "off-the-shelf" tree-based models.

(**Requirements**): You will use SciKit-Learn to train (on the training dataset "X_train" and "y_train") a "DecisionTreeClassifier", a "RandomForestClassifier", and a "GradientBoostingClassifier". This should be a relatively simple task. Once you have, you can use the "Plot_Train_Val_w_Decision" function to plot the decision boundaries that each model generates on both the training and validation datasets; these plots will also compute and display the total classification accuracy of the model on each of the datasets. Report your findings when using an "off-the-shelf" version of these models as well as their initial performance on the training and validation sets. Then, using your knowledge and intuition that you gained from the lectures, hyperparameter-tune all three of these models by adjusting their input variables until the models are able to return better accuracies on the validation dataset than the original "off-the-shelf" models (I would suggest referring to the APIs for each of the models to see the list of possible hyperparameters you can adjust). Report your findings and include the plots of the best models that you were able to obtain for each of the three ML algorithms. Thus, your answer to this problem should include only three plots (one for each of the best version of these models, in terms of validation accuracy) as well as a thorough report of your findings. Which model were you able to get the best overall validation accuracy with? Lastly, submit your .py file along with your written (or typed) answers in a .zip file.

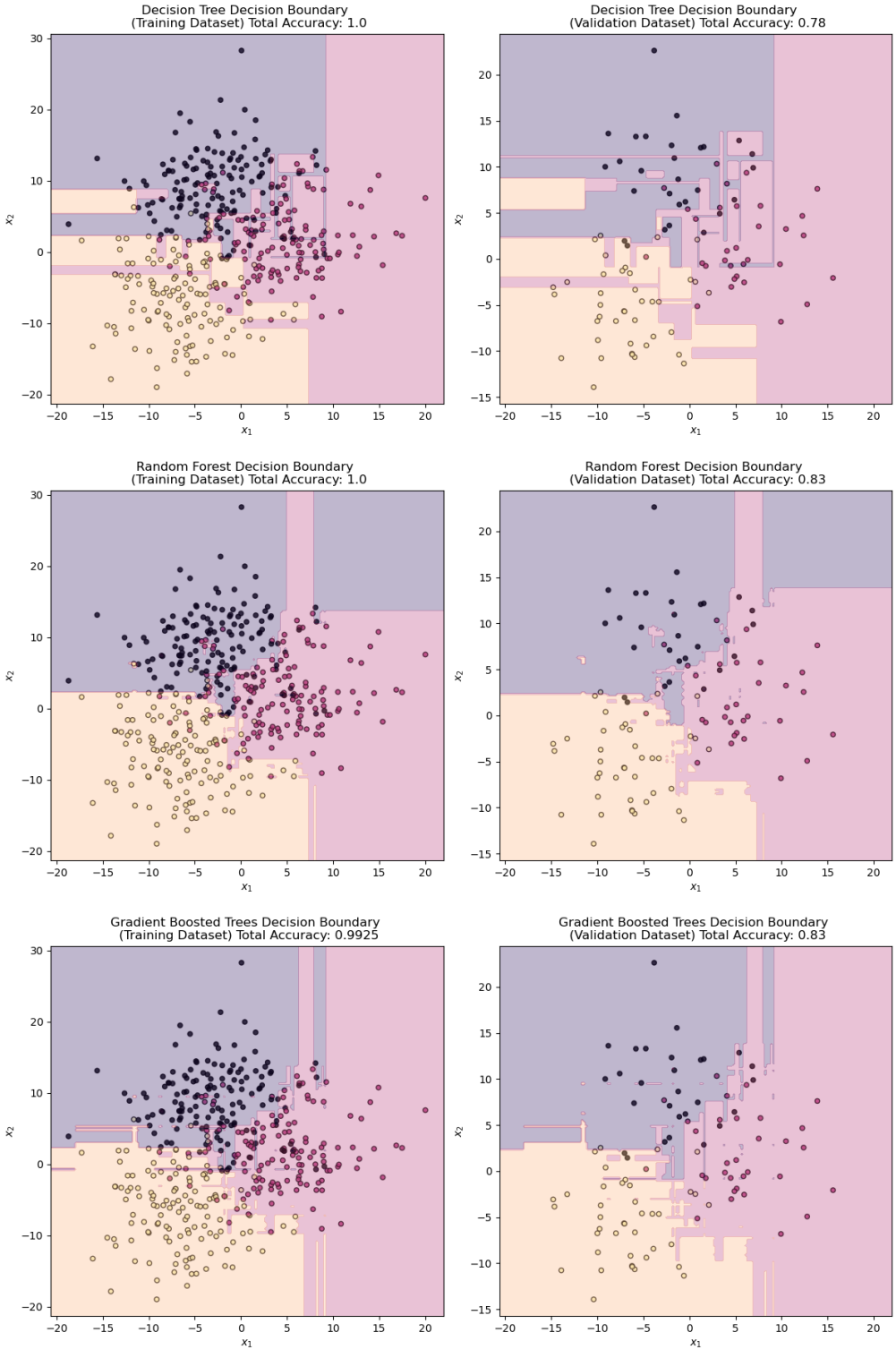

Figure 2: These figures display the "off-the-shelf" decision boundaries obtained (as well as the accuracy) from the "DecisionTreeClassifier", "RandomForestClassifier", and "GradientBoostingClassifier" models in SciKit-Learn on the training and validation datasets that were generated from the three-cluster dataset displayed in Figure 1.

5. **(ISE-464 Graduate Students)** (**Maximum-Likelihood Solution for Multi-Class Classification**) Recall from the lecture slides that the likelihood function for a multi-class classification problem (with the target variable $y \in \{1, 2, ..., K\}$ being modeled as a multinomial random variable with $K > 0$ different possible target classes) is defined as

$$L(\phi) = \prod_{i=1}^{m} \prod_{j=1}^{k} \phi_j^{\mathbb{I}\left[y^{(i)}=j\right]} = \prod_{j=1}^{k} \phi_j^{\sum_{i=1}^{m} \mathbb{I}\left[y^{(i)}=j\right]},$$

where $\mathbb{I}[\cdot]$ is the indicator function and ϕ_j denotes the probability of any datapoint having a target class of $j \in \{1, 2, ..., K\}$ (ϕ_j is also the j-th entry of the output of the softmax function on the affinity vector $z \in \mathbb{R}^k$).

a) Derive the corresponding log-likelihood function $\ell(\phi)$ of $L(\phi)$.

b) The optimization problem of maximizing the log-likelihood function ℓ over the parameters $\phi \in \mathbb{R}^k$ can be written as follows:

$$\begin{aligned} \max_{\phi \in \mathbb{R}^k} \quad & \ell(\phi) \\ \text{s.t.} \quad & \sum_{j=1}^{k} \phi_j = 1, \end{aligned}$$

where the constraint $\sum_{j=1}^{k} \phi_j = 1$ ensures that ϕ is a vector of probabilities. Write the Lagrangian function $\mathcal{L}(\phi; \lambda)$ for this optimization problem by introducing the Lagrange multiplier $\lambda \in \mathbb{R}$.

c) Derive the expression for the optimal solution $\phi_j^\star$, for each $j \in \{1, 2, ..., k\}$, that solves the optimization problem in part (5b). You may assume that the Hessian matrix of the Lagrangian $\mathcal{L}$ is negative definite with respect to the parameters ϕ (which is indeed true under mild assumptions on the dataset that one is working with). Notice that you must first determine the optimal value of λ.