Homework 2, Machine Learning, Fall 2024

***IMPORTANT* Homework Submission Instructions**

1. For all coding components, complete the solutions with a Jupyter notebook/Google Colab, and export the notebook (including both code and outputs) into a PDF file. Concatenate the theory solutions with the coding solutions into **a single** PDF file which you will submit to Gradescope.
2. All solutions should be typed.
3. Gradescope will prompt you to label the pages for each question. Please do this carefully.
4. You must show work or give an explanation for every question. "Yes" and "No" are not sufficient answers. Always show the steps of your calculations. Only partial credit can be given if you do not explain your answer.
5. Failure to adhere to the above submission format may result in penalties.

As a reminder, don't wait until the last minute to ask for help, because the person you need to speak with might be dealing with many other students close to the deadline. Also, the teaching staff has been instructed that it is not mandatory for them to answer questions on the day the homework is due. So get your questions in early!

There is no collaboration on homework. You must do your own work; we feel this is the best way to learn. If you get stuck, you are welcome to discuss conceptual issues with your classmates or the TAs but you may not show your classmates any part of your code or solutions or proof steps on paper. Do not ask another classmate to perform coding for you or to show you the answer to the problem. You can look at reference material on the Internet such as ChatGPT, but don't look at that unless you are really stuck, and make sure your answer is complete - don't write "Because I found a theorem that says the answer is X". You must show your complete work for each question. You are required to state what references you used (if you used an outside source). **Please copy the following statement at the top of your assignment file**:

This assignment represents my own work. I did not work on this assignment with others. All coding was done by myself.

1 Optimization, Regularization, and Constraints (Theory) (Hayden) - 24 pts

The following problem concerns binary classification optimized over n training samples using a 0-1 loss. We will add regularization and finally add a hard constraint on complexity. For simplicity in this question, we focus on decision trees with regularization on the number of leaves and a hard constraint on depth. **Please show your work for these questions.**

(a) No regularization

Define our 0-1 loss for tree f on training samples $\mathbf{X}$ with training labels $\mathbf{y}$ as $L(f, \mathbf{X}, \mathbf{y})$:

$$L(f, \mathbf{X}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{f(\mathbf{x}_i) \neq y_i} \tag{1}$$

where $f(\mathbf{x}_i)$ is the prediction of tree f for the i^{th} training sample, and n is the number of training samples.

i. Why would it be a bad idea to optimize Objective 1 as is, without adding sparsity? Give a one sentence answer.

ii. For a dataset with k boolean features, what is the deepest valid[1] tree you can make, as a function of k? You could consider this an implicit constraint on the depth of the tree. (Define the depth of the tree as the number of nodes in the longest path from root to leaf - so a tree with just 1 leaf is depth 1).

iii. For Objective 1, how many valid optimal trees are there for the dataset in Table 1? What are these trees (please describe or draw them)?

$\mathbf{x}_{\cdot 1}$	$\mathbf{x}_{\cdot 2}$	$\mathbf{y}$
1	0	1
0	1	0

Table 1: A simple dataset for decision tree optimization

(b) Soft Regularization

Define $s(f)$ as a measure of the sparsity of the tree f (specifically the number of leaves), and adding the regularization penalty λ, we get the following objective:

$$R(f, \mathbf{X}, \mathbf{y}|\lambda) = L(f, \mathbf{X}, \mathbf{y}) + \lambda s(f) \tag{2}$$

Let $L(f, \mathbf{X}, \mathbf{y})$ remain the 0-1 loss, as in the previous subsection.

i Suppose an optimal tree for some dataset happens to have only one leaf. For the worst possible dataset, what is the largest possible value of $R(f, \mathbf{X}, \mathbf{y}|\lambda)$ for this optimal tree? Note that a tree of depth 1 is considered to have 1 leaf.

ii Find the smallest depth d (as a function of λ) such that the following claim is true:
For any binary classification dataset $(\mathbf{X}, \mathbf{y})$*, any tree* f *of depth* d *or greater is guaranteed to be suboptimal with respect to* $R(f, \mathbf{X}, \mathbf{y}|\lambda)$.
Define the depth of the tree as the number of nodes in the longest path from root to leaf (so a tree with just 1 leaf is depth 1). Assume $\lambda > 0$.

iii For the dataset in Table 1, how many optimal trees exist for Objective 2 with any $0 < \lambda < \frac{1}{2}$?

[1] An invalid tree would be one which splits on the same feature multiple times in a single path - so you shouldn't have a tree which splits on feature 1 at the root, then splits on feature 1 again after that. You could still have a tree which splits on feature 1 at the root, then splits on feature 2 in both the left and right children.

(c) Hard Constraints

We now add a hard depth constraint d to the problem above. We again define the depth of the tree as the number of nodes in the longest path from root to leaf (so a tree with just 1 leaf is depth 1). One way to represent the full training objective in this optimization problem is with the piecewise function R below:

$$R(f, \mathbf{X}, \mathbf{y}|\lambda, d) = \begin{cases} L(f, \mathbf{X}, \mathbf{y}) + \lambda s(f), \text{if depth}(f) \leq d \\ \infty, \text{if depth}(f) > d \end{cases} \tag{3}$$

Let $L(f, \mathbf{X}, \mathbf{y})$ remain the 0-1 loss, as in the previous subsection.

Suppose for some dataset X there is a unique optimal tree f^* with respect to this objective, out of all decision trees of depth limit 5. Suppose this tree has depth 4, and has $R(f^*, \mathbf{X}, \mathbf{y}|\lambda, 5) = c$ for some constant c.

i Leaving $\mathbf{X}, \mathbf{y}$ and λ unchanged, and considering all possible trees f, what is the strongest claim we can make about the optimal objective with depth limit 4, $\min_f R(f, \mathbf{X}, \mathbf{y}|\lambda, 4)$. Give your answer as a function of c (inequality symbols are allowed, if needed)?

ii Leaving $\mathbf{X}, \mathbf{y}$ and λ unchanged, and considering all possible trees f, what is the strongest claim we can make about the optimal objective with depth limit 6, $\min_f R(f, \mathbf{X}, \mathbf{y}|\lambda, 6)$ as a function of c (inequality symbols are allowed, if needed)?

iii As a function of λ, what is the minimum possible improvement in accuracy needed, relative to f^*, for there to exist a tree of depth 6 that is better than f^*?

2 Exploring Gradient Based Optimization Methods (Theory + Applied) Varun - 22 pts

In this question, we are going to explore a few gradient based optimization methods and see why some perform better than others. First, consider the function $f(x) = x^2$. We will evaluate the performance of two iterative methods:

- Gradient descent, which has the following form:

$$x_t = x_{t-1} - \eta \nabla f(x)\Big|_{x_{t-1}} \tag{4}$$

- Gradient descent with momentum, which has the following form:

$$v_t = \beta v_{t-1} + \eta \nabla f(x)\Big|_{x_{t-1}} \tag{5}$$

$$x_t = x_{t-1} - v_t \tag{6}$$

  where $\beta \in [0, 1]$

For all the sub-questions below, you need to show all your work in order to gain full points. Initialize all methods with $x_0 = 1$ and $v_0 = 0$.

(a)

i Perform three iterations of gradient descent manually, starting from x_0, with a learning rate $\eta = 0.1$. Compute the optimality gap $|x_3 - x^*|$.

ii Perform three iterations of gradient descent with momentum with $\eta = 0.1$ and $\beta = 0.5$, again starting from x_0. Compute the new optimality gap $|x_3 - x^*|$ and compare it with the value in a. After the 3rd iteration, which method has a lower optimality gap (i.e. has converged faster)?

(b) Now let's see how both methods perform under a more challenging scenario. Consider the function $f(x) = \sin(5x) + \frac{1}{5}x^2$. For this part, you can use Python to simulate both methods and plot your results.

i. Plot this function in order to understand its behaviour. Why is this challenging for gradient descent to optimize?

ii. Implement gradient descent for this function with learning rate $\eta = 0.1$, $x_0 = 20$, and 20 iterations. Plot the trajectory of the resulting x values (i.e., plot x over iterations).

iii. Implement gradient descent with momentum for this function with learning rate $\eta = 0.1$, $\beta = 0.5$, $x_0 = 20$, and 20 iterations. Plot the trajectory of the resulting x values. Which method achieves the optimum value?

iv. Let's see how momentum affects trajectory behaviour near a minima. Set $x_0 = 1$ and keep all parameters for both methods fixed. Plot the resulting trajectories for both methods. What do you see now?

(c) Neural networks are powered by gradient descent, which uses simplistic linear approximations. However, one could also consider optimizing them using second order methods. Assume we have an objective function $L(\mathbf{w})$ (where $\mathbf{w}$ could correspond to the weights of a black box neural network) which we want to minimize. For this question, you can assume that $L(\mathbf{w})$ is at least twice differentiable and has a defined Hessian inverse.

i. Using a starting value of $\mathbf{w}_0$, write the second order Taylor series expansion of $L(\mathbf{w})$ around $\mathbf{w}_0$. Denote the resulting function $\bar{L}(\mathbf{w})$.

ii. Find the value of $\mathbf{w}$ in terms of $\mathbf{w}_0$ and other relevant parameters that minimizes $\bar{L}(\mathbf{w})$. Denote the resulting value $\mathbf{w}_1$. This is the update step of the second order method starting from point $\mathbf{w}_0$.

iii. Now consider the following objective function:

$$L(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{A}\mathbf{w} - \mathbf{b}^T\mathbf{w} \tag{7}$$

where $\mathbf{w}$, $\mathbf{b}$ are appropriately shaped vectors and $\mathbf{A}$ is a matrix (assumed to be invertible). Perform one step of this second order update starting from an arbitrary vector $\mathbf{w}_0$, showing your work. How does the answer to this question relate to the minimizer of $L(\mathbf{w})$?

iv. In many real world scenarios, $\mathbf{w}$ can be very high dimensional (e.g., a neural network with millions of parameters). Even though this method converges in fewer iterations (quadratically!) compared to gradient descent (linearly!), what are the two main bottlenecks in memory and computation that could make this method slow in practice?

3 Variance of Random Forest and Tree Bagging (Applied) (Harry) 20 pts

Recall the bootstrap aggregation (bagging) technique from class, which aims to reduce the variance of high variance, low bias predictors such as decision trees. To perform bagging, we repeatedly bootstrap a subsample of the dataset and train a predictor on this subsample. Our final model then combines the results of each of these predictors, usually by taking the majority vote in classification.

In this problem, we investigate in greater detail how bagging lowers the variance of a predictor both theoretically and experimentally.

(a) Suppose that we have M identically distributed random variables X_i with variance σ^2 and positive pairwise correlation ρ. Show that the variance of the mean of these random variables is

$$\text{Var}\left(\frac{1}{M}\sum_{i=1}^{M} X_i\right) = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2$$

For instance, each random variable X_i could be a predictor on a randomly drawn dataset where the correlation occurs since two predictors would be more similar on more similar sampled datasets.

(b) Let $\mathcal{D}$ be the true dataset distribution, and let (x, y) be any data-label pair. Suppose that we have a randomized training algorithm that takes in a point x and a training dataset $D \sim \mathcal{D}$ and outputs a prediction $f(x, D)$ for x after training on D. Then, we can decompose the mean squared error (MSE) as

$$\mathbb{E}_{D,f}\Big[\big(y - f(x,D)\big)^2\Big] = \Big(\text{Bias}_{D,f}\big[f(x,D)\big]\Big)^2 + \text{Var}_{D,f}\big[f(x,D)\big] \tag{8}$$

where

$$\begin{aligned}\text{Bias}_{D,f}\big[f(x,D)\big] &= \mathbb{E}_{D,f}\big[y - f(x,D)\big] \\ \text{Var}_{D,f}\big[f(x,D)\big] &= \mathbb{E}_{D,f}\left[\Big(f(x,D) - \mathbb{E}_{D,f}\big[f(x,D)\big]\Big)^2\right].\end{aligned}$$

Equation (8) is known classically as the *bias-variance decomposition*, and it shows that the MSE is the sum of the *bias*, how far the training algorithm is from the label in expectation, and the *variance*, how sensitive the training algorithm is to the training dataset. In particular, note that the expectations are taken with respect to the dataset draws and the randomization of the training algorithm, not with the data point evaluated on.

To connect this with Part (a), let $f_i(x, D)$ be the ith identically distributed predictor, and let $f(x, D) = \frac{1}{M}\sum_{i=1}^{M} f_i(x, D)$ be the resulting aggregate training predictor. From Part (a), the variance of $f(x, D)$ approaches $\rho(x)\sigma^2(x)$ as $M \to \infty$, where $\rho(x)$ is the correlation between the predictors on x and $\sigma^2(x)$ is the variance of each predictor on x. This suggests that we can decrease the variance under bagging by decreasing either the variance of each predictor or the correlation between predictors. By bootstrapping the samples of the training dataset within the training algorithm, we reduce the correlation between predictors by introducing randomness into the decision trees making up the ensemble (if we fit decision trees without randomizing, we'll get perfectly correlated trees!).

When we use decision trees as the base predictor, we call this method *tree bagging*. However, we can further attempt to improve upon this method using *random forests*, where we additionally randomly select a subset of the features as options to split on at each possible split point. This further decreases the correlation between predictors, decreasing the overall variance. The remainder of this problem is dedicated towards verifying this claim.

It is highly recommended to use the `sklearn` library, particularly the `sklearn.ensemble.BaggingRegressor` and `sklearn.ensemble.RandomForestRegressor` tools. The data for this problem can be found in `Canvas -> Files -> HW2`.

i. We will model the true distribution of data $\mathcal{D}$ as a uniform random bootstrap sample drawn from the Wine dataset. Since it is computationally intractable to compute every possible bootstrap training dataset in $\mathcal{D}$, we will instead estimate $\mathbb{E}_{D\sim\mathcal{D}}$ by taking the average over many bootstrap datasets $D \sim \mathcal{D}$.

   Create a function that draws a bootstrapped training dataset $D \sim \mathcal{D}$ by uniformly sampling N samples with replacement from the `wine_true.csv` files. Furthermore, let X denote the test samples in the `wine_test.csv` file.

ii. Using the function you created in part (i), draw 3000 bootstrapped training datasets $D \sim \mathcal{D}$ of size $N = 1024$. For each dataset, train a tree bagging model with 2 trees, where each tree uses 500 bootstrapped samples (sampled with replacement from the size 1024 sample, which is itself a bootstrap from the true data). For each $x \in X$, estimate the correlation $\rho(x)$ between the predictions on x of the first and second trees of the model. (i.e., the correlation is between two vectors, one of size 3000 for the first trees, and the other of size 3000 for the second trees).

iii. For each of 4 values of a variable F that you choose (where F is an integer between 2 and the number of features minus 1), draw 3000 bootstrapped training datasets $D \sim \mathcal{D}$ of size $N = 1024$. For each dataset, train a random forest model with 2 trees, where each tree uses 500 bootstrapped samples and F bootstrapped features at each split. For each $x \in X$, estimate the correlation $\rho(x)$ between the predictions on x of the first and second trees of the model.

   For each F, compute the average value of the estimate for $\rho(x)$ over x.

   Hint: This may take a few minutes to run.

iv. Compare the results of Parts (ii) and (iii). Did the tree bagging method or random forest method produce a lower correlation? How does the correlation relate to F? Do these match what you expected?

4 Boosting (Applied) (Richard) - 10 pts

In this question, you will predict whether a customer will default on their payments next month, using Adaboost and XGBoost. We will use the `credit_card_default.csv` dataset in`Canvas->Files->HW2`, which includes 23 explanatory variables including the payers' demographic information and payment history, and 1 response variable named `Y`. More information on the dataset and the variables can be found here. You may use sklearn and XGBoost libraries for this question.

(a) Read the dataset and perform any data processing if necessary. Briefly summarize what you did and why.

(b) Split 80% of the data for training and 20% for testing using `train_test_split`. Fit an Adaboost classifier on the credit card default dataset using sklearn and use five-fold cross-validation to tune at least 1 hyperparameter. Repeat this with XGBoost. Use decision trees as the base estimators. Report the hyperparameter you tuned, and the value you found for that parameter.

(c) Fit using the best hyperparameters from cross-validation in (b), then evaluate both Adaboost and XGBoost on the test set and compare the models' performance using accuracy on the test set. Then, plot and interpret a confusion matrix for both models. Please interpret the confusion matrix in the context of credit lending.

(d) For the training set, plot ROC curves for all individual features and for the full model, on the same plot. We suggest coloring the full model's ROC curve in black and using lighter colors for the features.

(e) Calculate (Gini) variable importance for Adaboost using `feature_importances_`. In addition to using the built-in feature importance, write your own permutation importance on the training set without using any packages. Compare the values for the variable importance to see if they are similar. Visualize both results in a bar chart ordered by importance. Discuss the most important features and how they *might* influence credit card default in the future.

5 GAMs and the Rashomon Effect (Applied) (Zack) - 24 pts

In this question, you will fit a generalized additive model (GAM) to the Wine dataset with a few different approaches: first by using the `fastsparsegams` library to quickly find regularized GAMs, then by using `xgboost.XGBClassifier`, `sklearn.ensemble.AdaBoost`, and `sklearn.ensemble.RandomForest` to find a GAM with shape functions defined by ensembles of decision stumps (depth 1 decision trees). Then, you'll use `interpret.glassbox.GAMChanger` to explore the variety of viable GAMs, and pick your own!

Starter code (please only Google Colab):
https://colab.research.google.com/drive/1ZtKKh5RUt4aFmkPpF0ra--6ZI3K97fvx?usp=sharing

(a) Using `max_support_size=50`, the logistic loss, and `algorithm='CDPSI'`, fit a GAM using FastSparseGAMs. The output of this is a 'solution path', which is a sequence of models of increasing complexity up to your maximum support size. Find the model with the best validation accuracy and report its regularization parameter (`lambda_0`), its support size, and its validation accuracy. To access this solution path, use the `.characteristics()` method of the fit GAM.

(b) Using the provided utility function (`plot_shape_functions_fastsparse`), plot the shape functions of the model you found in part (a). Please interpret these shape functions in the context of the problem (i.e. what do these shape functions tell you about the predictive impact of each feature on wine quality)?

(c) Observe that the shape function is composed of weighted indicator variables for whether a feature is less than or equal to a specific value. Each indicator variable is a decision stump! So we should be able to find a GAM with an ensemble of decision stumps. The `AdaBoost` algorithm is based on this principle. You could also use an algorithm like `xgboost` or `Random Forests` to find an ensemble of decision stumps. With an `n_estimators=50` for all approaches, `algorithm='SAMME'` for `sklearn.ensemble.AdaBoost`, and `max_depth=1` for `xgboost.XGBClassifier` and `sklearn.ensemble.RandomForest`, fit GAMs with decision stumps and report the validation accuracy for each approach. Are any approaches significantly better or worse than the others? Please explain why you think this may be occurring.

(d) Plot the shape functions for the different tree ensembling methods using the provided utility function (`plot_shape_functions_tree_ensemble`). Compare and contrast their shape functions with those generated by `FastSparseGAM` (when they use the same feature). Do your intuitions about the predictive impact of the features hold across all four methods?

(e) You may have noticed in part (d) that there are several GAMs (found by different methods and with different shape functions) which perform about equally well. We will now use a tool called `GAMChanger` to explore this variety. Fit an Explainable Boosting Classifier (`interpret.glassbox.ExplainableBoostingClassifier`) with no interaction terms (`interactions=0`) on the data and use `gamchanger.visualize` to visualize its shape functions (pass in the validation data to see evaluation measures update in real time). You can use the GAMChanger tool to manually edit the GAM's shape functions! Use this tool to make the shape function monotonically increasing over '`alcohol`'. Make '`volatile_acidity`' monotonically decreasing. Now make any other changes you see fit (smooth out noisy sections, if a section looks monotonic enforce monotonicity, or anything else you want) without decreasing validation accuracy too much. Describe some of the changes you made. Were you able to make reasonable changes without affecting accuracy? What does this tell you about the size of the 'space' of good models for this problem?