CS 189/289A Introduction to Machine Learning
Spring 2025 Jonathan Shewchuk HW5

Due: Wednesday, April 2 at 11:59 pm

- Homework 5 consists of coding assignments and math problems.
- We prefer that you typeset your answers using LaTeX or other word processing software. If you haven't yet learned LaTeX, one of the crown jewels of computer science, now is a good time! Neatly handwritten and scanned solutions will also be accepted.
- In all of the questions, **show your work**, not just the final answer.
- **We will not provide points back with respect to homework submission errors.** This includes, but is not limited to: 1) not assigning pages to problems; 2) not including code in the write-up appendix; 3) not including code in the designated code Gradescope assignment; 4) not including Kaggle scores; 5) submitting code that only partially works; 6) submitting late regrade requests.
- **Start early; you can submit models to Kaggle only twice a day!**

Deliverables:

Submit your predictions for the test sets to Kaggle as early as possible. Include your Kaggle scores in your write-up (see below). The Kaggle competition for this assignment can be found at

- Spam: `https://www.kaggle.com/t/91eab33eda2c4a9ca7b09455a98159bc`
- Titanic: `https://www.kaggle.com/t/85e83287ff854fc286e79f4f3953c8f8`

Write-up: Submit your solution in **PDF** format to "Homework 5 Write-Up" on Gradescope.

- State your name, and if you have discussed this homework with anyone (other than GSIs), list the names *of them all*.
- Please start each main question Q2, Q3, etc. on a new page. (You don't have to start Q2.2, Q2.3, etc. on a new page.) You may use multiple pages for a question.
- If you include figures, graphs or tables for a question, any explanations should accompany them in *the same page*. Do NOT put these in an appendix!
- **Only PDF uploads to Gradescope will be accepted**. You may use LaTeX or Word to typeset your solution or scan a neatly handwritten solution to produce the PDF.
- **Replicate all your code in an appendix**. Begin code for each coding question in a fresh page. Do not put code from multiple questions in the same page. When you upload this PDF on Gradescope, *make sure* that you assign the relevant pages of your code from appendix to correct questions.

Code: Additionally, submit all your code as a .zip file to "Homework 5 Code" on Gradescope.

- **Set a seed for all pseudo-random numbers generated in your code.** This ensures your results are replicated when readers run your code.

- Include a README with your name, student ID, the values of the random seed (above) you used, and instructions for running (and compiling, if appropriate) your code.
- Do NOT provide any data files, but supply instructions on how to add data to your code.
- Code that the readers can't run because it requires exorbitant memory or execution time might not receive marks.
- Code submitted here must match that in the PDF Write-up, and produce the *exact* output submitted to Kaggle. Inconsistent or incomplete code might not receive marks.

1 Honor Code

Declare and sign the following statement:

"I certify that all solutions in this document are entirely my own and that I have not looked at anyone else's solution. I have given credit to all external sources I consulted."

Signature : _____

While discussions are encouraged, *everything* in your solution must be your (and only your) creation. Furthermore, all external material (i.e., *anything* outside lectures and assigned readings, including figures and pictures) should be cited properly. We wish to remind you that consequences of academic misconduct are *particularly severe*!

2 Random Forest Motivation

Ensemble learning is a general technique to combat overfitting, by combining the predictions of many varied models into a single prediction based on their average or majority vote.

1. **The motivation of averaging.** Consider a set of uncorrelated random variables $\{Y_i\}_{i=1}^n$ with mean μ and variance σ^2. Calculate the expectation and variance of their average. (In the context of ensemble methods, these Y_i's are analogous to the prediction made by classifier i.)

2. In part (a), we see that averaging reduces variance for uncorrelated classifiers. Real-world prediction will of course not be completely uncorrelated, but reducing correlation among decision trees will generally reduce the final variance. Reconsider a set of correlated random variables $\{Z_i\}_{i=1}^n$ with mean μ and variance σ^2, where each $Z_i \in \mathbb{R}$ is a scalar. Suppose $\forall i \neq j$, $\text{Corr}(Z_i, Z_j) = \rho$. (If you don't remember the relationship between correlation and covariance from your prerequisite classes, please look it up.) Calculate the variance of the average of the random variables Z_i, written in terms of σ, ρ, and n.

   What happens when n gets very large, and what does that tell us about the potential effectiveness of averaging? ...if ρ is large ($|\rho| \approx 1$)? ...if ρ is very very small ($|\rho| \approx 0$)? ...if ρ is middling ($|\rho| \approx 0.5$)? We're not looking for anything too rigorous—qualitative reasoning using your derived variance is sufficient.

3. **Ensemble Learning – Bagging.** In lecture, we covered bagging (Bootstrap AGGregatING). Bagging is a randomized method for creating many different learners from the same data set.

   Given a training set of size n, generate T random subsamples, each of size n', by sampling with replacement. Some points may be chosen multiple times, while some may not be chosen at all. If $n' = n$, around 63% are chosen, and the remaining 37% are called out-of-bag (OOB) sample points.

   (i) Why 63%?

   *Hint: when n is very large, what is the probability that a sample point won't be selected? Please only consider the probability of a point not being selected in any **one** of the subsamples (not all of the T subsamples).*

   (ii) The number of decision trees T in the ensemble is usually chosen to trade off running time against reduced variance. (Typically, a dozen to several thousand trees are used.) The sample size n' has a smaller effect on running time, so our choice of n' is mainly governed by getting the best predictions. Although it's common practice to set $n' = n$, that isn't necessarily the best choice. How do you recommend we choose the hyperparameter n'?

3 The Bias and Variance of Ridge Regression

Recall the statistical model for ridge regression from lecture. We have a set of labeled sample points $\{X_i, y_i\}_{i=1}^n$ and a vector of Gaussian noise $e_i \in \mathbb{R}^n$. Our model follows, where the rows of X are $X_i^\top$.

$$\begin{aligned} Y &= Xv + e, \\ e &\sim N(0, \sigma^2 I). \end{aligned}$$

Throughout this problem, assume $X^\top X$ is invertible. Recall the estimators from ordinary least-squares regression and ridge regression.

$$\begin{aligned} w_{\text{ols}} &= \arg\min_{w \in \mathbb{R}^d} \|Xw - y\|_2^2. \\ w_{\text{ridge}} &= \arg\min_{w \in \mathbb{R}^d} \|Xw - y\|_2^2 + \lambda \|w\|_2^2. \end{aligned}$$

For simplicity, we either are penalizing the bias term α, or there is no bias term. (Thus the ℓ_2-regularization term is $\lambda\|w\|_2^2$ rather than $\lambda\|w'\|_2^2$.)

1. Write the solutions for w_{ols} and w_{ridge}. No need to derive them.

2. Let $\widehat{w} \in \mathbb{R}^d$ denote any estimator of v. In the context of this problem, an estimator $\widehat{w} = \widehat{w}(X, Y)$ is any function which takes the data X and a realization of Y, and computes a guess of v.

   Define the MSE (mean squared error) of the estimator $\widehat{w}$ to be

   $$\text{MSE}(\widehat{w}) = \mathbb{E}\left[\left\|\widehat{w} - v\right\|_2^2\right].$$

   The expectation is taken with respect to the randomness inherent in e. Define $\mu = \mathbb{E}\left[\widehat{w}\right]$. Show that the MSE decomposes as

   $$\text{MSE}(\widehat{w}) = \left\|\mu - v\right\|_2^2 + \text{Tr}(\text{Cov}(\widehat{w})) \,.$$

   Hint: Expectation and trace commute, so $\mathbb{E}\left[\text{Tr}(A)\right] = \text{Tr}(\mathbb{E}\left[A\right])$ for any square matrix A.

3. Show that

   $$\begin{aligned} \mathbb{E}\left[w_{\text{ols}}\right] &= v \quad \text{and} \\ \mathbb{E}\left[w_{\text{ridge}}\right] &= (X^\top X + \lambda I)^{-1} X^\top X v. \end{aligned}$$

   That is, w_{ols} is an *unbiased* estimator of v, whereas w_{ridge} is a *biased* estimator of v.

4. Let $\gamma_1 \le \gamma_2 \le \ldots \le \gamma_d$ be the d eigenvalues of the matrix $X^\top X$ arranged in nondecreasing order. First, show that

   $$\begin{aligned} \text{Tr}(\text{Cov}(w_{\text{ols}})) &= \sigma^2 \sum_{i=1}^d \frac{1}{\gamma_i} \quad \text{and} \\ \text{Tr}(\text{Cov}(w_{\text{ridge}})) &= \sigma^2 \sum_{i=1}^d \frac{\gamma_i}{(\gamma_i + \lambda)^2}. \end{aligned}$$

Second, use these formulas to conclude that if $\lambda > 0$, then

$$\mathrm{Tr}(\mathrm{Cov}(w_{\mathrm{ridge}})) < \mathrm{Tr}(\mathrm{Cov}(w_{\mathrm{ols}})).$$

Hint: For the ridge variance, consider writing $X^\top X$ in terms of its eigendecomposition $U\Sigma U^\top$. Also note that it's not hard to figure out the value of $\mathbb{E}\left[ee^\top\right]$.

5. What happens to the bias and variance of the ridge regression estimator w_{ridge} as λ increases? Consider the MSE decomposition from part (b), what does that tell us about how we should choose λ if we wish to minimize the MSE?

4 Decision Trees for Classification

In this problem, you will implement decision trees and random forests for classification on two datasets: 1) the spam dataset and 2) a Titanic dataset to predict survivors of the infamous disaster. The data is with the assignment. See the Appendix for more information on its contents and some suggestions on data structure design.

In lectures, you were given a basic introduction to decision trees and how such trees are trained. You were also introduced to random forests. Feel free to research additional decision tree techniques online (AdaBoost and XGBoost are particularly interesting!)

For your convenience, we provide starter code which includes preprocessing and some decision tree functionality already implemented. Feel free to use (or not to use) this code in your implementation.

4.1 Implement Decision Trees

We expect you to implement the tree data structure yourself; you are not allowed to use a pre-existing decision tree implementation. The Titanic dataset is not "cleaned"—that is, there are missing values—so you can use external libraries for data preprocessing and tree visualization (in fact, we recommend it). Removing examples with missing features is not a good option; there is not enough data to justify throwing some of it away. Be aware that some of the later questions might require special functionality that you need to implement (e.g., maximum depth stopping criterion, visualizing the tree, tracing the path of a sample point through the tree). You can use any programming language you wish as long as we can read and run your code with minimal effort. If you choose to use our starter code, a skeleton structure of the decision tree implementation is provided, and you will decide how to fill it in. After you are done, **attach your code in the appendix and select the appropriate pages when submitting to Gradescope.**

4.2 Implement a Random Forest

You are not allowed to use any off-the-shelf random forest implementation. However, you are allowed to now use library implementations for individual decision trees (we use sklearn in the starter code). If you use the starter code, you will mainly need to implement the superclass the random forest implementation inherits from, an implementation of bagged trees, which creates decision trees trained on different samples of the data. After you are done, **attach your code in the appendix and select the appropriate pages when submitting to Gradescope.**

4.3 Describe implementation details

We aren't looking for an essay; 1–2 sentences per question is enough.

1. How did you deal with categorical features and missing values?
2. What was your stopping criterion?
3. How did you implement random forests?
4. Did you do anything special to speed up training? ("No" is an acceptable response.)
5. Anything else cool you implemented? ("No" is an acceptable response.)

4.4 Performance Evaluation

For each of the 2 datasets, train both a decision tree and random forest and report your training and validation accuracies. You should be reporting 8 numbers (2 datasets $\times$ 2 classifiers $\times$ training/validation). In addition, for both datasets, train your best model and submit your predictions to Kaggle. Include your Kaggle display name and your public scores on each dataset. You should be reporting 2 Kaggle scores.

4.5 Writeup Requirements for the Spam Dataset

1. For your decision tree, and for a data point of your choosing from each class (spam and ham), state the splits (i.e., which feature and which value of that feature to split on) your decision tree made to classify it. An example of what this might look like:

   (a) ("hot") ≥ 2

   (b) ("thanks") < 1

   (c) ("nigeria") ≥ 3

   (d) Therefore this email was spam.

   (a) ("budget") ≥ 2

   (b) ("spreadsheet") ≥ 1

   (c) Therefore this email was ham.

2. Generate a random 80/20 training/validation split. Train decision trees with varying maximum depths (try going from depth = 1 to depth = 40) with all other hyperparameters fixed. Plot your validation accuracies as a function of the depth. Which depth had the highest validation accuracy? Write 1–2 sentences explaining the behavior you observe in your plot. If you find that you need to plot more depths, feel free to do so.

A Appendix

Titanic Dataset Details

Here's a brief overview of the fields in the Titanic dataset.

1. survived: the label we want to predict. 1 indicates the person survived, whereas 0 indicates the person died.
2. pclass: Measure of socioeconomic status. 1 is upper, 2 is middle, 3 is lower.
3. age: Fractional if less than 1.
4. sex: Male/female.
5. sibsp: Number of siblings/spouses aboard the Titanic.
6. parch: Number of parents/children aboard the Titanic.
7. ticket: Ticket number.
8. fare: Fare.
9. cabin: Cabin number.
10. embarked: Port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)

Suggested Architecture

This is a complicated coding project. You should put in some thought about how to structure your program so your decision trees don't end up as horrific forest fires of technical debt. Here is a rough, **optional** spec that only covers the barebones decision tree structure. This is only for your benefit—writing clean code will make your life easier, but we won't grade you on it. There are many different ways to implement this.

Your decision trees ideally should have a well-encapsulated interface like this:

```
classifier = DecisionTree(params)
classifier.fit(train_data, train_labels)
predictions = classifier.predict(test_data)
```

where `train_data` and `test_data` are 2D matrices (rows are data, columns are features).

A decision tree (or **DecisionTree**) is a binary tree. As you train your tree, your tree should create and configure subtrees to use for classification and store these internally. An instance of a **DecisionTree** class will be the root node of its resulting tree so you can directly reference its attributes and subtrees during inference time.

Each **DecisionTree** should have left and right pointers to its children, which are also trees, though some (like leaf nodes) won't have any children. Each node has a split rule that, during classification, tells you when you should continue traversing to the left or to the right child of the node. Leaf nodes, instead of containing a split rule, should simply contain a label of what class to classify a data point as. Leaf nodes can either be a special configuration of regular **DecisionTree** or an entirely different class.

DecisionTree fields:

- `split_idx, thresh`: Two fields that detail what feature to split on at a node, as well as the threshold value at which you should split. The former can be encoded as an integer index into your data point's feature vector.
- `left`: The left child of the current node.
- `right`: The right child of the current node.
- `pred`: If this field is set, the **DecisionTree** is a leaf node, and the field contains the label with which you should classify a data point as, assuming you reached this node during your classification tree traversal. Typically, the prediction is the mode of the labels of the training data points arriving at this node.

DecisionTree methods:

- `entropy(labels)`: A method that takes in the labels of data stored at a node and compute the entropy for the distribution of the labels.
- `information_gain(features, labels, threshold)`: A method that takes in some feature of the data, the labels and a threshold, and compute the information gain of a split using the threshold.
- `fit(data, labels)`: Grows a decision tree by constructing nodes. Using the entropy and information gain methods, it attempts to find a configuration of nodes that best splits the input data. This function figures out the split rules that each node should have and figures out when to stop growing the tree and insert a leaf node. There are many ways to implement this, but eventually your **DecisionTree** should store the root node of the resulting tree so you can use the tree for classification later on. Since the height of your **DecisionTree** shouldn't be astronomically large (you may want to cap the height—if you do, the max height would be a hyperparameter), this method is best implemented recursively.
- `predict(data)`: Given a data point, traverse the tree to find the best label to classify the data point as. Start at the root node you stored and evaluate split rules at each node as you traverse until you reach a leaf node, then choose that leaf node's label as your output label.

Random forests can be implemented without code duplication by storing groups of decision trees. You will have to train each tree on different subsets of the data (data bagging) and train nodes in each tree on different subsets of features (attribute bagging). Hopefully, the spec above gives you a good jumping-off point as you start to implement your decision trees. Again, it's highly recommended to think through design before coding.

Happy hacking!

Submission Checklist

Please ensure you have completed the following before your final submission.

At the beginning of your writeup...

1. Have you copied and hand-signed the honor code specified in Question 1?
2. Have you listed all students (Names and ID numbers) that you collaborated with?

In your writeup for Question 4...

1. Have you included your **Kaggle Score** and **Kaggle Username**?
2. Have you included your generated plots and visualizations?

At the end of the writeup...

1. Have you provided a code appendix including all code you wrote in solving the homework?

Executable Code Submission

1. Have you created an archive containing all ".py" files that you wrote or modified to generate your homework solutions?
2. Have you removed all data and extraneous files from the archive?
3. Have you included a `README` in your archive containing any special instructions to reproduce your results?

Submissions

1. Have you submitted your written solutions to the Gradescope assignment titled **HW5 Write-Up** and selected pages appropriately?
2. Have you submitted your executable code archive to the Gradescope assignment titled **HW5 Code**?
3. Have you submitted your test set predictions for **Spam** and **Titanic** dataset to the appropriate Kaggle challenges?
4. Is your Kaggle submission in integer format? Submissions in decimal format will receive a score of zero!

Congratulations! You have completed Homework 5.