COMS 4721 Spring 2022 Homework 2

Due at 11:59 PM on March 4, 2022

Grade for a *late submission* is scaled by factor of $(0.75)^{(\#\text{ days late, rounded up to an integer})}$

About this assignment. This homework assignment is a mix of some problems that involve programming and data analysis using Python, as well as some problems that do not involve programming but may require a bit of mathematical reasoning, simple derivations, and/or "calculations" (usually done by hand).

Data files needed for the programming/data analysis parts are available on Courseworks (under "Files").

Working in pairs is explicitly permitted on this assignment (subject to the conditions detailed in the syllabus).

Submission. For the programming and data analysis portions, please create and use a Jupyter Notebook to solve these problems. Include all of the code you write for this assignment in the notebook, and add appropriate comments so that the code is easy to understand. Make sure your answers to the various questions are clearly stated and labeled in separate Markdown cells (even if the answer also appears as an output of a code cell). You will need to submit the Jupyter Notebook on Courseworks as part of your homework submission.

You will also need to create a PDF document containing your solutions to *all problems* (including those involving programming and data analysis). **This PDF document should also contain the code and results from the Jupyeter Notebook.** Please read the homework policy (in the syllabus) for information about how to write-up your solutions, and please see the instructions for properly converting a Jupyter Notebook to PDF. You will need to submit this document on Gradescope.

- For redundancy purposes, please make sure your name and UNI appear prominently on the first page of the PDF. If you are working in pairs, both students' names and UNIs should appear in this manner.
- On Gradescope, for each problem, you need to **select all of the pages that contain your solutions** for that problem. If part of your solution is on an unselected page, we will consider it missing. See the instructions for submitting a PDF on Gradescope.
- On Gradescope, if you are working in pairs, then only one student in the pair should make the submission. That student must make sure to **add both students in the pair to the submission**. This is something that has to be done at the time of submission within the Gradescope submission system. See the instructions for adding group members on Gradescope. Do not forget to do this, or else one student will not receive credit for the homework submission!
- On Courseworks, submit only the Jupyter Notebook (a `.ipynb` file). Do not include any of the data files—we already have those `:-)`
- On Courseworks, if you are working in pairs, then only one student in the pair should make the submission of the Jupyter Notebook. (We are only using this to collect your Jupyter Notebooks; the grading will happen on Gradescope.) If both students submit, we'll only look at one of the submissions (and we'll pick one arbitrarily and ignore the other).
- Please submit both your write-up and Jupyter Notebook by the deadline. Do not wait until the last minute to make your submissions!

Please consult the Gradescope Help Center if you need further help with submitting on Gradescope, and the Canvas Guides if you need further help with submitting on Courseworks.

1 Linear regression

In this part of the assignment, you'll practice analyzing a simple data set concerning prostate cancer using linear regression. The data set `prostate-train.csv` is available on Courseworks, together with the Jupyter notebook `prostate-data.ipynb`, which can help you get started. Regard this data set as "training data" in which the goal is to predict the variable `lpsa` (the logarithm of the prostate specific antigen level) using the remaining variables (`lcavol`, `lweight`, `age`, `lbph`, `svi`, `lcp`, `gleason`, `pgg45`) as features.[1]

You are welcome to use any code library for this part of the assignment as long as give an appropriate citation. For example, if you use `sklearn`, state precisely which library functions you use in your write-up.

Problem 1.1 (1 point). For each of the eight features, find the best fit affine function of that variable to the label `lpsa`. (By "best", we mean of minimum SSE.) Report the slope and intercept in each case.

Problem 1.2 (1 point). For each of the eight features, make a separate plot showing the training data (as points) and the affine function learned from Problem 1.1. To make it possible to compare the different plots, make sure both horizontal and vertical axes are exactly the same for all plots. Also make sure the axes of each plot are appropriately labeled, and that the plot itself comes with a suitable caption/title. This plot should be included in your write-up.

Problem 1.3 (1 point). In Problem 1.2, you'll notice that because you use the same horizontal axis for all plots, some of the plots may seem a bit unclear because they have a very different "natural" scale. So, repeat Problem 1.2 but this time, change the units of the horizontal axis (for a given feature x_i) to $(x_i - \mu_i)/\sigma_i$, where μ_i is the mean of feature x_i on the training data, and σ_i is the standard deviation of feature x_i on the training data. This plot should be included in your write-up.

Problem 1.4 (1 point). Now find the best fit affine function of all eight features (together as a vector in $\mathbb{R}^8$) to the label `lpsa`. Report the coefficients in the weight vector and the intercept term.

You should find that some of the variables have a negative coefficient in the weight vector from Problem 1.4, even though its corresponding affine function from Problem 1.1 has a positive slope. This might seem like a paradox: for such a feature, your answers from Problem 1.1 might lead you to think that increasing the feature's value should, on average, increase the (predicted) value of `lpsa`; whereas your answer from Problem 1.4 might lead you to think that increasing the feature's value should, on average, decrease the (predicted) value of `lpsa`.

Of course, there is no paradox. The following problem shows how this can happen.

Problem 1.5 (3 points). Suppose $X_1 \sim \mathrm{N}(0,1)$ and $X_2 \sim \mathrm{N}(0,1)$, and $\mathbb{E}[X_1 X_2] = \frac{2}{3}$. This means that the random vector $\vec{X} := (X_1, X_2)$ follows a bivariate normal distribution, with mean zero and covariance matrix $\begin{pmatrix} 1 & 2/3 \\ 2/3 & 1 \end{pmatrix}$. Furthermore, suppose $Y = \frac{3}{2}X_1 - \frac{3}{4}X_2$.

(a) What is the linear function of X_1 that has smallest mean squared error for predicting Y? Give the slope and show a short derivation.

(b) What is the linear function of X_2 that has smallest mean squared error for predicting Y? Again, just give the slope and show a short derivation.

(c) And finally, what is the linear function of (X_1, X_2) that has smallest mean squared error for predicting Y? Give the weight vector and show a short derivation.

Note that in this problem, we are considering *linear* functions as opposed to *affine* functions. But considering affine functions would not change any of your answers to the questions above.

You should find that even though each of X_1 and X_2 is positively correlated with Y (analogous to the situation in Problem 1.1), the best linear predictor of Y that considers both X_1 and X_2 has a positive coefficient for one variable and a negative coefficient for the other variable (analogous to the situation in Problem 1.4).

[1] A description of the variables appears on page 4 of *Elements of Statistical Learning* by Hastie, Tibshirani, and Friedman, available here: https://web.stanford.edu/~hastie/ElemStatLearn/

This example shows that one must be careful when interpreting the sign of the regression coefficient.

There are 30 additional labeled examples available in the data set `prostate-test.csv`, also on Courseworks. Regard these data as "test data".

Problem 1.6 (1 point). Report the mean squared errors of all affine functions from Problems 1.1 and Problem 1.4 on the training data (from `prostate-train.csv`), and then also on the test data (from `prostate-test.csv`).

2 Model averaging

In this part of the assignment, you'll work out why $f_{\text{avg}} := \frac{1}{M}\sum_{t=1}^{M} f_t$ satisfies

$$\mathbb{E}[(f_{\text{avg}}(\vec{X}) - Y)^2] = \frac{1}{M}\sum_{t=1}^{M}\mathbb{E}[(f_t(\vec{X}) - Y)^2] - \frac{1}{2M^2}\sum_{s=1}^{M}\sum_{t=1}^{M}\mathbb{E}[(f_s(\vec{X}) - f_t(\vec{X}))^2] \tag{1}$$

for any random example $(\vec{X}, Y)$ and any real-valued functions $f_1, \ldots, f_M$.

Problem 2.1 (1 point). Suppose A and B are independent and identically distributed random variables, each with variance σ^2. Determine the relationship between $\mathbb{E}[(A-B)^2]$ and σ^2. Briefly (but precisely) explain your answer. *Hint: The bias-variance decomposition can be useful here. Or, just expand the square.*

It turns out the original claim in Equation (1) is true even if we remove the expectations and replace $(\vec{X}, Y)$ with an arbitrary (non-random) example $(\vec{x}, y)$:

$$(f_{\text{avg}}(\vec{x}) - y)^2 = \frac{1}{M}\sum_{t=1}^{M}(f_t(\vec{x}) - y)^2 - \frac{1}{2M^2}\sum_{s=1}^{M}\sum_{t=1}^{M}(f_s(\vec{x}) - f_t(\vec{x}))^2. \tag{2}$$

The original claim in Equation (1) follows from Equation (2) simply by replacing $(\vec{x}, y)$ with $(\vec{X}, Y)$ and taking expectations. So let's just focus on understanding why Equation (2) is true.

Problem 2.2 (2 points). Rewrite Equation (2) using the relationship identified in Problem 2.1 (in particular to change the final term on the right-hand side), so that the rewritten equation can be interpreted as an instance of the bias-variance decomposition. Briefly (but precisely) explain the interpretation. *Hint: Define a random variable T whose distribution is uniform over $\{1, 2, \ldots, M\}$, and consider the random variable $f_T(\vec{x})$.*

3 Bagging regression and classification trees

In this part of the assignment, you'll fit regression and classification trees to some data sets.

Start with the "Dartmouth" data set that we discussed in lecture. The data set `dartmouth-train.csv` is available on Courseworks, together with the Jupyter notebook `dartmouth-data.ipynb`, which can help you get started. Regard this data set as "training data" in which the goal is to predict the variable `College GPA` using the remaining variables (`SAT Score`, `HS GPA`) as features. Even though two features are available, please only use `HS GPA`; so the problem is to predict `College GPA` from `HS GPA`. Test data `dartmouth-test.csv` is also available on Courseworks.

Problem 3.1 (1 point). Learn a regression tree with (up to) 300 leaf nodes to Dartmouth data using `sklearn.tree.DecisionTreeRegressor`. You'll want to use the `max_leaf_nodes` hyperparameter. Please also use `random_state=42` for reproducibility. Use the defaults for all other hyperparameters. What is the mean squared error on the test data?

Problem 3.2 (1 point). Now use Bagging to learn an ensemble of 100 regression trees, where each tree has (up to) 3 leaf nodes. Here you should use `sklearn.ensemble.BaggingRegressor` together with `sklearn.tree.DecisionTreeRegressor`. The hyperparameters for the `BaggingRegressor` you'll need are `n_estimators` and `random_state=42`. Use the defaults for all other hyperparameters. What is the mean squared error on the test data?

Problem 3.3 (1 point). Make a plot that depicts the training data as points, and also the two predictors learned in Problem 3.1 and Problem 3.2. Make sure you use different line styles or colors for the two predictors, and include a legend so it's clear which is which. As usual, make sure axes are labeled and the plot is given an appropriate caption/title. What qualitative differences do you observe between the (large) regression tree from Problem 3.1 and the ensemble of 100 (small) regression trees from Problem 3.2?

To depict a predictor f in a plot, you can evaluate f on a fine grid of points from the relevant domain, such as $1.5, 1.51, 1.52, \ldots, 4.98, 4.99, 5$, and then plot the resulting points, $(1.5, f(x)), (1.51, f(1.51)), (1.52, f(1.52)), \ldots$. Here is some example code for doing this:

```
import numpy as np
from matplotlib import pyplot as plt

# Suppose at this point, you have learned a regression tree 'my_tree'
x = np.arange(1.5, 5, 0.01)[:, np.newaxis] # grid of points from 1.5 to 5
my_tree_y = my_tree.predict(x) # predictions at each grid point
plt.figure(figsize=(8,5))
plt.plot(x, my_tree_y, label='Predictions of regression tree')
plt.xlabel('HS GPA')
plt.ylabel('College GPA')
```

In lecture, we didn't discuss the use of Bagging with classification trees. It doesn't make sense to form the "average" predictor $f_{\text{avg}} = \frac{1}{M}\sum_{t=1}^{M} f_t$ because one cannot average the (non-numeric) categorical predictions of a classification tree. Instead (in the case of binary classification), we use the majority vote predictor f_{maj}, where $f_{\text{maj}}(\vec{x}) = \text{majority}\{f_1(\vec{x}), f_2(\vec{x}), \ldots, f_M(\vec{x})\}$, breaking ties arbitrarily.

Now move on to the "Spambase" data set `spam.pickle`, also available on Courseworks, together with the Jupyter notebook `spam-data.ipynb`, which can help you get started. The data provided is for a binary classification problem for predicting whether an email is spam or not, based on various features computed using the text of the email.[2]

[2]The features are described here: https://web.stanford.edu/~hastie/ElemStatLearn/datasets/spam.info.txt

Problem 3.4 (1 point). Use `sklearn.tree.DecisionTreeClassifier` with default parameters (except `random_state=42`) to learn a single classification tree. Report the training and test error rates. Then use `sklearn.ensemble.BaggingClassifier` to learn an ensemble of 10 classification trees (using `sklearn.tree.DecisionTreeClassifier`). Use default hyperparameters in all cases (except `random_state=42`). Report the training and test error rates. You should see an improvement with the ensemble of classification trees, compared to the single classification tree.

Problem 3.5 (1 point). Let's see how varying the number of classification trees in the ensemble (i.e., the `n_estimators` hyperparameter) affects the test error rate. Repeat the ensemble learning part of Problem 3.4 eight times, using the following values for `n_estimators`: $2, 4, 8, 16, 32, 64, 128, 256$. (Again, please also use `random_state=42`.) Make a plot of the test error rates as a function of this hyperparameter. Again, make sure axes are labeled and the plot is given an appropriate caption/title.

4 Optimal classifiers

Let's adopt the statistical model in which 100 training examples $(X_1, Y_1), (X_2, Y_2), \ldots, (X_{100}, Y_{100})$, as well as the "future" example (X, Y), are IID random examples. Here, each example only has a single numerical feature, and the labels take values in $\{0, 1\}$ (binary classification). Furthermore, let's assume that the marginal distribution of X is the uniform probability density on the interval $[0, 1] = \{x \in \mathbb{R} : 0 \leq x \leq 1\}$. Finally, let's assume that for any $x \in [0, 1]$, the conditional probability that $Y = 1$ given $X = x$ is

$$\Pr(Y = 1 \mid X = x) = \begin{cases} 0.2 & \text{if } x \leq 0.2, \\ 0.6 & \text{if } 0.2 < x < 0.8, \\ 0.3 & \text{if } x \geq 0.8. \end{cases}$$

For any classifier $f \colon [0, 1] \to \{0, 1\}$, let $\mathrm{err}(f) = \Pr(f(X) \neq Y)$ denote its *error rate*. Let $f^\star$ denote an optimal classifier (i.e., a classifier with minimum error rate).

Problem 4.1 (1 point). Compute the value of $\mathrm{err}(f^\star)$. Explain your answer.

Problem 4.2 (2 points). Write a Python procedure that, given (the parameters of) an arbitrary decision stump T, returns its error rate $\mathrm{err}(f_T)$. (Here, f_T refers to the function implemented by the decision stump T.) Use your code to determine the error rates of the following three decision stumps:

$T_1 =$ "if $x \leq 0.4$, then return 0, else return 1";
$T_2 =$ "if $x \leq 0.6$, then return 0, else return 1";
$T_3 =$ "if $x \leq 0.6$, then return 1, else return 0".

(Your procedure should compute the error rates *exactly*; in particular, it should not simply estimate it using a finite number of random examples.)

Problem 4.3 (1 point). Which decision stump T has the smallest error rate $\mathrm{err}(f_T)$ among all decision stumps? And what is its error rate? Explain your answers. *Note: the smallest error rate achievable by decision stumps could be larger than the answer you got in Problem 4.1.*

Problem 4.4 (1 point). For each $\theta \in [0, 1]$, let $e(\theta)$ denote the smallest error rate achievable by a decision stump that uses the predicate "$x \leq \theta$?" with its non-leaf node. Plot $e(\theta)$ as a function of θ. Make sure the axes of your plot are appropriately labeled.

Problem 4.5 (2 points). Let $\hat{T}$ be a decision stump with the smallest number of classification mistakes on the training examples $(X_1, Y_1), (X_2, Y_2), \ldots, (X_{100}, Y_{100})$. Let $\hat{\theta}$ denote the threshold in the predicate used by the decision stump $\hat{T}$. Let $\mathrm{err}(f_{\hat{T}})$ denote the error rate of $\hat{T}$. Write Python code to simulate the random generation of the training examples (using the pseudo-random number generation facilities in Python), the construction of $\hat{T}$, and the computation of $\mathrm{err}(f_{\hat{T}})$. (Of course, for error rate computation, you can re-use your Python procedure from Problem 4.2.) Run your simulation code 1000 times (using different random seeds each time, so that the training data are different each time), recording the values of $\hat{\theta}$ and $\mathrm{err}(f_{\hat{T}})$. Plot two histograms: one for $\hat{\theta}$, and another for $\mathrm{err}(f_{\hat{T}})$. Make sure the axes of your histograms are appropriately labeled.

Note that `sklearn.tree.DecisionTreeClassifier` does not optimize the "mistakes" objective function.