24-787 Homework 5

Due date: 03/05/2022 @ 11:59 pm EST

Note: In case a problem requires programming, it should be programmed in `Python`. In Programming, you should use plain `Python` language, unless otherwise stated. For example, if the intention of a Problem is familiarity with numpy library, it will be clearly noted in that problem to use numpy. Please submit your homework through Gradescope.

Submissions: There are two steps to submitting your assignment on Gradescope:

1. **HW05_Writeup**: Submit a combined pdf file containing the answers to theoretical questions as well as the pdf form of the FILE.ipynb notebooks.

- To produce a pdf of your notebooks, you can first convert each of the .ipynb files to HTML.
- To do this, simply run: `ipython nbconvert -to html FILE.ipynb` for each of the notebooks, where FILE.ipynb is the notebook you want to convert. Then you can convert the HTML files to PDFs with your favorite web browser.
- If an assignment has theoretical and mathematical derivation, scan your handwritten solution and make a PDF file.
- Then concatenate them all together in your favorite PDF viewer/editor. The file name (FILE) for naming should be saved as HW-assignmentnumber-andrew-ID.pdf. For example for assignment 1, my FILE = HW-1-lkara.pdf
- Submit this final PDF on Gradescope, and **make sure to tag the questions correctly!**

2. **HW05_Code**: Submit a ZIP folder containing the FILE.ipynb notebooks for each of the programming questions. The ZIP folder containing your iPython notebook solutions should be named as HW-assignmentnumber-andrew-ID.zip

Q1: Detecting multicollinearity in the feature set (10 pts)

This question uses **Xtrain.csv**. Your goal is to detect any multicollinearity that may exist in this data. Each column is a feature. There is no output y you need to be concerned about. The csv file has 24 columns, however, only the first 13 columns have numerical entries. Please only use the visible 13 columns and the rest can be ignored.

(a) (5 pts) Identify the variance inflation factor (VIF) for each of the features. For this, implement a linear regression model that fits a linear function that attempts to predict one feature as a liner combination of all other features. Report a row vector that shows the calculated VIFs.

For this part of the problem, you may use inbuilt OLS regression functions and R^2 calculators (for instance, those from `sklearn` packages) for each feature. But you are not to use inbuilt VIF functions such as `variance_inflation_factor()` that comes with those package. However, you can use such functions to validate your calculations.

(b) (5 pts) If you were forced to remove a single feature, which one would it be? Explain your reasoning in 2-3 sentences.

Q2: Decision Trees (30 pts)

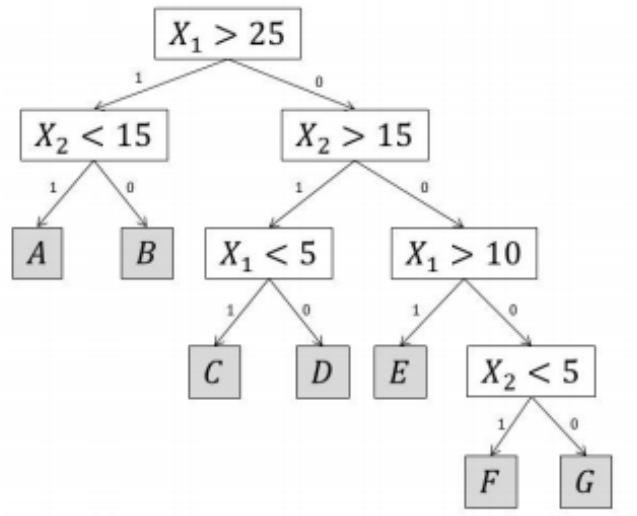
(a) (10 pts) Consider two binary variables x and y having the joint distribution given in the table below.

$x \downarrow, y \rightarrow$	0	1
0	1/3	1/3
1	0	1/3

Evaluate the following distributions and/or quantities

$$P(x),\ P(y),\ P(x|y),\ P(y|x),\ P(x,y),$$
$$H(x),\ H(y),\ H(x|y),\ H(y|x),\ IG(x|y).$$

(b) (20 pts) Consider the following decision tree. For the arrows: 1 = True, 0 = False.



(i) On a 2D plot whose axes are X_1 and X_2, draw the decision boundaries defined by this tree. Each leaf is labeled with a letter. Write this letter in the corresponding region of the instance space. Your plot should produce rectangular blocks.

(ii) Draw another decision tree that is syntactically different than the one shown above but defines the same decision boundaries.

Q3: Decision Trees (30 pts)

Suppose you are given a simple dataset listed in the table below and need to learn a decision tree from the data. These are 8 instances, each with 3 binary attributes (X_1, X_2, X_3) and a binary class label (Y). You can solve this problem by hand. Use the data to answer the following questions.

Instance	X_1	X_2	X_3	Y
1	0	0	0	0
2	0	0	1	0
3	0	1	0	1
4	0	1	1	1
5	1	0	1	1
6	1	0	1	1
7	1	1	0	0
8	1	1	0	0

(a) (3 pts) Which attribute should be selected for the root of the decision tree? Use the corresponding Gini impurity to justify your answer.

(b) (3 pts) After the root node is selected, the entire tree can be learned by recursively splitting the data into two subgroups, finding the next best attribute to split on, dividing the subgroup into smaller groups, and so forth. How do you know when to stop growing the tree? In other words, what are the stopping criteria? Describe in 1-2 sentences. You may consult the lecture slides and/or investigate this topic using online resources.

(c) (6 pts) Run the algorithm by hand, compute your decisions tree. Draw your tree. Consult the lecture slides if you are unsure about drawing conventions. Considering all 8 instances to be training samples, compute the training error on the training dataset.

(d) (3 pts) Now suppose you are presented with new instances for which the class Y is unknown. Use your decision tree to predict the label of each instance listed below:

Instance	X_1	X_2	X_3	Y
9	1	1	1	?
10	1	0	0	?
11	0	1	1	?

(e) (5 pts) For the decision tree you have learned so far, do you have any basis on which to evaluate if the tree is overfitting? Why or why not? How might you combat overfitting in a decision tree? (Answers will not be unique. Describe your favorite one in one or two sentences.)

(f) (5 pts) If the labels on Instances 6 and 7 were changed to 0 and 1, respectively, would the structure of the learned decision tree change? Why or why not? Also, would any of the leaf nodes change? Explain your reasoning.

(g) (5 pts) Now using `Python`, solve the original problem and show the constructed tree. From the `sklearn.tree` library, you can use `DecisionTreeClassifier` and `plot_tree` functions. Output the decisions of this tree for the data shown in part (e).

Q4: Classification and Regression Tree (CART) Implementation (30 pts)

This question provides a case where you can complete a coded CART model (a very common category of decision tree) to some real world data sets. The `Python` code uses **recursion** to implement the CART. You will not be asked to code the model from scratch, we will provide you with the base code and you just need to fill a small proportion of the code after you read and understand how the code works. After that, you will experiment your completed model as well as the `sklearn` library of CART on some datasets.

(a) (5 pts) Gini impurity: as has been covered in the class, Gini impurity describes how homogeneous a set of data is. A set of data is pure if all its members belong to the same class, while a set with many samples from many different classes will have a Gini close to 1.

Now that you have a set of samples of Pittsburgh daily weather in March like this:

{**snow**: 6 days, **rain**: 10 days, **cloudy**: 10 days, **sunny**: 2 days, **shower**: 3 days}.

Calculate the corresponding Gini impurity. Do you think Pittsburgh March weather is homogeneous?

(b) (20 pts) Code completion: In the `cart.py` for Q3. You will find an almost completed implementation of CART from scratch. You have to put `tree.py` and `wifi_localization.txt` under the same folder to run the code normally. In order to complete the code please do the following:

- **Read the code**: Read through the code forming the `class DecisionTreeClassifier`. You might want to pay attention to member functions:
  `def _gini(self,y)` and `def _grow_tree(self,X,y,depth=0)`
  where you will be asked to complete a few lines of code to make the CART working. You might look at the brief introduction below about Python class construction if you are not familiar with it. If you skip this step, you are very likely to complete your code incorrectly.

- **Complete the Gini impurity calculation**: In the place annotated with **#1.Your code goes here**, fill in small piece of code to calculate the Gini impurity. You will need the variable `self.n_classes_` which is the number of classes you need to calculate the gini impurity. The code could be done within 3 lines.

- **Complete the variables in the recursion function**: the way you actually construct the CART is by using the member function `def _grow_tree(self,X,y,depth=0)` recursively. This means that you will call this function inside itself. This might sounds tricky if you have not implemented this before, but it is just an efficient way to traverse a tree structure: when you are growing the tree from a certain node of the CART by calling `def _grow_tree(self,X,y,depth=0)`, you construct the branch of this node by calling this same function again inside itself for its children (two children for the decision tree case). But when you call this function inside itself, the variables {x,y,depth} have to be changed accordingly to values of the children. Once you complete this part correctly, the CART code is good to go! Based on this, fill in the variables for the function in the place annotated with **#2.Your code goes here**. When submitting homework, please include a screenshot of your added code.

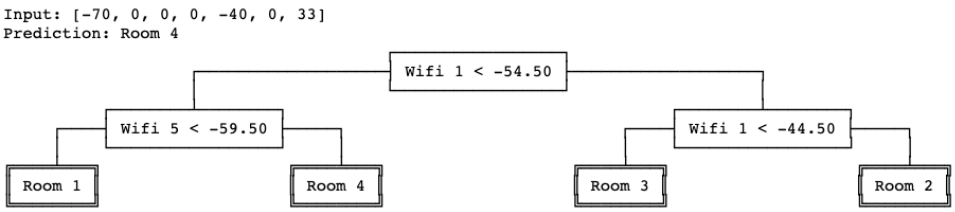


Figure 1: CART Visualized Trees Sample.

Hint: a brief introduction on how a `Python` class is constructed. Understanding this might help you complete the code easier. (`https://www.w3schools.com/python/python_classes.asp`)

(c) (5 pts) Experiment with the completed model: Now that you have a completed CART implementation. Please experiment it and compare against the `sklearn` results using two datasets. The corresponding scripts are on the Jupyter Notebook. Again, you have to put `tree.py`, `cart.py` and `wifi_localization.txt` under the same folder to run the code normally. You need to do the following four tests:

- `python3 cart.py --dataset="iris" --max_depth=2 --hide_details`
- `python3 cart.py --dataset="iris" --max_depth=2 --hide_details --use_sklearn`
- `python3 cart.py --dataset="wifi" --max_depth=2 --hide_details`
- `python3 cart.py --dataset="wifi" --max_depth=3 --use_sklearn`

You could run `python` instead of `python3` if your default `Python` version is 3.x.x. Ideally, for the `Python` implementation, you should get visualized results like Fig. 1. Display the generated results based on these experiments on the Jupyter Notebook for grading.