HOMEWORK 2

Lucas Poon
llpoon

Instructions: Use this latex file as a template to develop your homework. Submit your homework on time as a single pdf file to Canvas. Please wrap your code and upload to a public GitHub repo, then attach the link below the instructions so that we can access it. You can choose any programming language (i.e. python, R, or MATLAB), as long as you implement the algorithm from scratch (e.g. do not use sklearn on questions 1 to 7 in section 2). Please check Piazza for updates about the homework.

1 A Simplified Decision Tree

You are to implement a decision-tree learner for classification. To simplify your work, this will not be a general purpose decision tree. Instead, your program can assume that

- each item has two continuous features $\mathbf{x} \in \mathbb{R}^2$
- the class label is binary and encoded as $y \in \{0, 1\}$
- data files are in plaintext with one labeled item per line, separated by whitespace:

$$x_{11} \quad x_{12} \quad y_1$$
$$\ldots$$
$$x_{n1} \quad x_{n2} \quad y_n$$

Your program should implement a decision tree learner according to the following guidelines:

- Candidate splits (j, c) for numeric features should use a threshold c in feature dimension j in the form of $x_j \geq c$.
- c should be on values of that dimension present in the training data; i.e. the threshold is on training points, not in between training points. You may enumerate all features, and for each feature, use all possible values for that dimension.
- You may skip those candidate splits with zero split information (i.e. the entropy of the split), and continue the enumeration.
- The left branch of such a split is the "then" branch, and the right branch is "else".
- Splits should be chosen using information gain ratio. If there is a tie you may break it arbitrarily.
- The stopping criteria (for making a node into a leaf) are that
  - the node is empty, or
  - all splits have zero gain ratio (if the entropy of the split is non-zero), or
  - the entropy of any candidates split is zero
- To simplify, whenever there is no majority class in a leaf, let it predict $y = 1$.

GitHub Repository for source code (main.py).

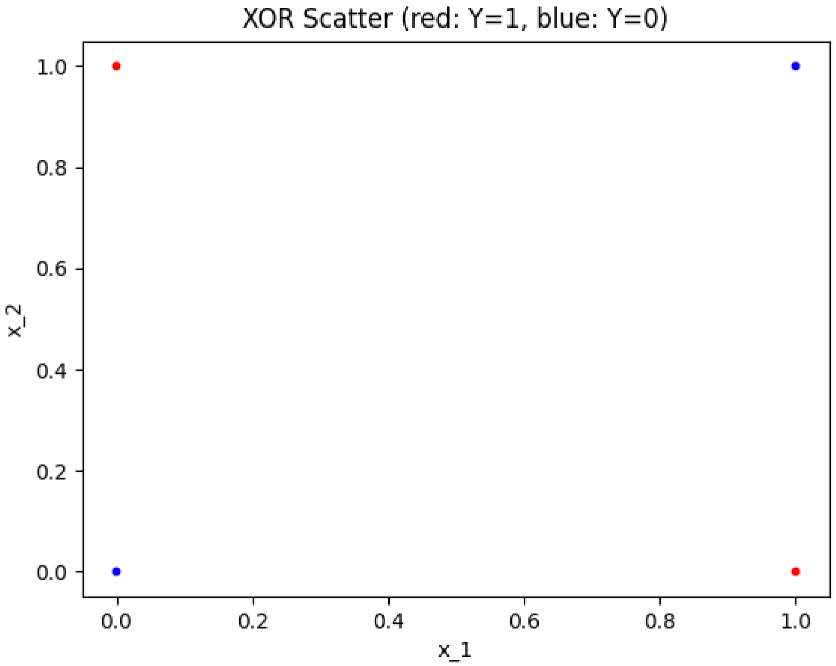
2 Questions

1. (Our algorithm stops at pure labels) [10 pts] If a node is not empty but contains training items with the same label, why is it guaranteed to become a leaf? Explain. You may assume that the feature values of these items are not all the same.
When a node is not empty, we determine the candidate splits and find the greatest info-gain.
Let the Y be the data set, n denote the number of data points in Y, P_0, $P(Y=0)$ and P_1 denote $P(Y=1)$.
Then the entropy of the whole set is:
$H_D(Y) = -P_1 log_2(P_1) - P_0 log_2(P_0) = -1log_2(1) - 0log_2(0) = 0$
The entropy of any candidate split S:
$H_D(Y|S) = -P_1 log_2(P_1) - P_0 log_2(P_0) = -1log_2(1) - 0log_2(0) = 0$
Hence, the gain ratio for all candidate splits would be: $\frac{0-0}{n} = 0$
By the given guidelines, it suffices one of the stopping criteria, as all splits to have zero gain ratio, therefore it is guaranteed to become a leaf.

2. (Our algorithm is greedy) [10 pts] Handcraft a small training set where both classes are present but the algorithm refuses to split; instead it makes the root a leaf and stop; Importantly, if we were to manually force a split, the algorithm will happily continue splitting the data set further and produce a deeper tree with zero training error. You should (1) plot your training set, (2) explain why. Hint: you don't need more than a handful of items.
Consider the XOR dataset:

x_1	x_2	y
0.0	0.0	0
1.0	0.0	1
0.0	1.0	1
1.0	1.0	0

Scatter plot of XOR:



The entropy of the whole data set is: $H_D(Y) = -\frac{2}{4}log_2(\frac{2}{4}) - \frac{2}{4}log_2(\frac{2}{4}) = 0.5$
Then we choose a split:

Considering $x_1 \geq 0.0$: InfoGain $= 0.5 - (-\frac{2}{4}log_2(\frac{2}{4})) = 0$, then Gain Ratio = 0

Considering $x_1 \geq 1.0$: InfoGain $= 0.5 - (-\frac{2}{4} * \frac{1}{2}log_2(\frac{1}{2}) + (-\frac{2}{4} * \frac{1}{2}log_2(\frac{1}{2})) = 0$, then Gain Ratio = 0

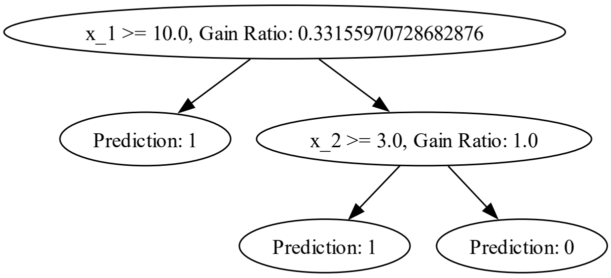
Similarly, considering $x_2 \geq 0.0$: InfoGain $= 0.5 - (-\frac{2}{4}log_2(\frac{2}{4})) = 0$, then Gain Ratio = 0

Considering $x_2 \geq 1.0$: InfoGain $= 0.5 - (-\frac{2}{4} * \frac{1}{2}log_2(\frac{1}{2}) + (-\frac{2}{4} * \frac{1}{2}log_2(\frac{1}{2})) = 0$, then Gain Ratio = 0
Since all splits have zero gain ratio, the algorithm refuses to split.

3. (Information gain ratio exercise) [10 pts] Use the training set Druns.txt. For the root node, list all candidate cuts and their information gain ratio. If the entropy of the candidate split is zero, please list its mutual information (i.e. information gain). Hint: to get $\log_2(x)$ when your programming language may be using a different base, use `log(x)/log(2)`. Also, please follow the split rule in the first section.

Feature	Threshold	Gain Ratio
x_1	$\geq$0.0	0.0
x_1	$\geq$0.1	0.10051807676021852
x_2	$\geq$-2.0	0.0
x_2	$\geq$-1.0	0.10051807676021852
x_2	$\geq$0.0	0.055953759631263686
x_2	$\geq$1.0	0.005780042205152451
x_2	$\geq$2.0	0.001144349517276868
x_2	$\geq$3.0	0.016411136842102245
x_2	$\geq$4.0	0.049749064181778 66
x_2	$\geq$5.0	0.1112402958633981
x_2	$\geq$6.0	0.2360996061436081
x_2	$\geq$7.0	0.055953759631263686
x_2	$\geq$8.0	0.4301569161309809 5

4. (The king of interpretability) [10 pts] Decision tree is not the most accurate classifier in general. However, it persists. This is largely due to its rumored interpretability: a data scientist can easily explain a tree to a non-data scientist. Build a tree from D3leaves.txt. Then manually convert your tree to a set of logic rules. Show the tree[1] and the rules.

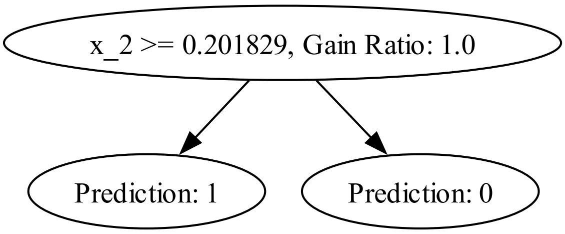

x_1 >= 10.0, Gain Ratio: 0.33155970728682876
Prediction: 1
x_2 >= 3.0, Gain Ratio: 1.0
Prediction: 1
Prediction: 0

This tree can be written as the following logic rule:
$(x_1 \geq 10.0) \vee (x_2 \geq 3.0)$

[1] When we say show the tree, we mean either the standard computer science tree view, or some crude plaintext representation of the tree – as long as you explain the format. When we say visualize the tree, we mean a plot in the 2D $\mathbf{x}$ space that shows how the tree will classify any points.

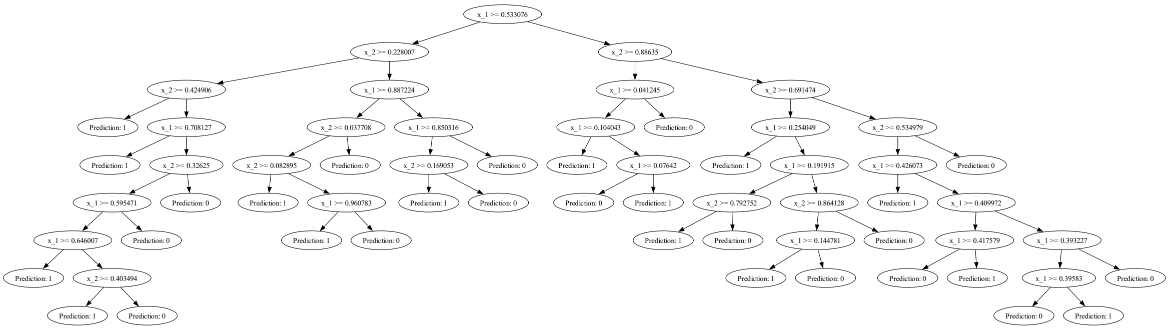
5. (Or is it?) [10 pts] For this question only, make sure you DO NOT VISUALIZE the data sets or plot your tree's decision boundary in the 2D $\mathbf{x}$ space. If your code does that, turn it off before proceeding. This is because you want to see your own reaction when trying to interpret a tree. You will get points no matter what your interpretation is. And we will ask you to visualize them in the next question anyway.

   - Build a decision tree on D1.txt. Show it to us in any format (e.g. could be a standard binary tree with nodes and arrows, and denote the rule at each leaf node; or as simple as plaintext output where each line represents a node with appropriate line number pointers to child nodes; whatever is convenient for you). Again, do not visualize the data set or the tree in the $\mathbf{x}$ input space. In real tasks you will not be able to visualize the whole high dimensional input space anyway, so we don't want you to "cheat" here.
     The decision tree looks like this:



   - Look at your tree in the above format (remember, you should not visualize the 2D dataset or your tree's decision boundary) and try to interpret the decision boundary in human understandable English. The decision tree suggests that there is a decision boundary at $x_2 \geq 0.201829$. In simpler terms, for any data point with feature $x_2 \geq 0.201829$, it will predict the label 1 and for any data point with feature $x_2 < 0.201829$, it will predict the label 0.

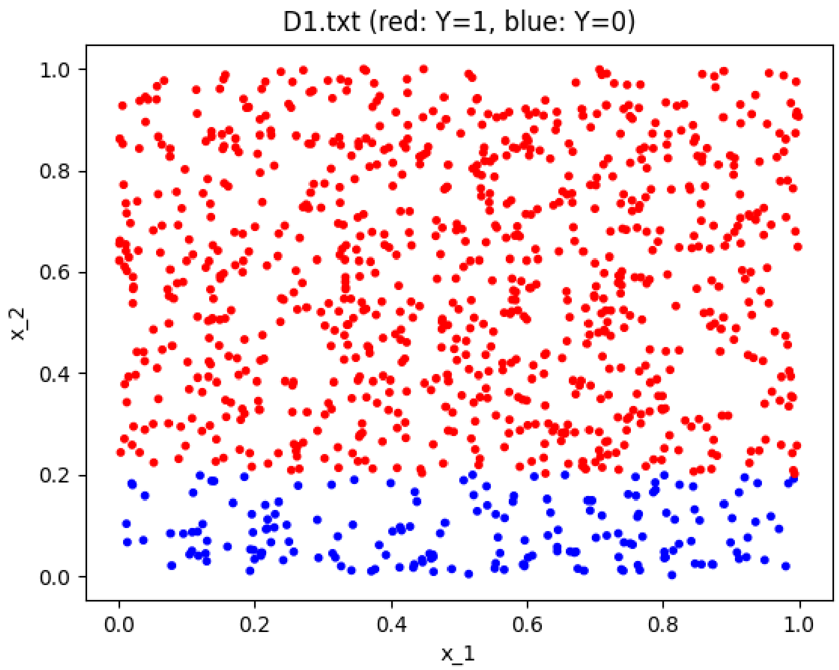
   - Build a decision tree on D2.txt. Show it to us.
     The decision tree looks like this (Gain Ratio ommited due to size):

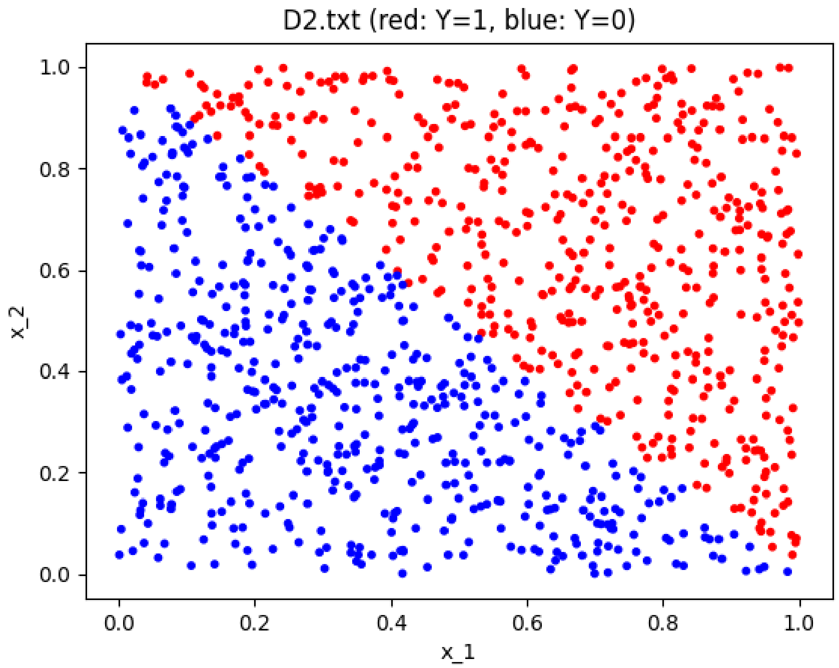


   - Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
     The decision tree for D2 is a lot more complex as there are a lot more branches and the tree is larger. It is very difficult to interpret this decision tree without visualization due to its complexity.

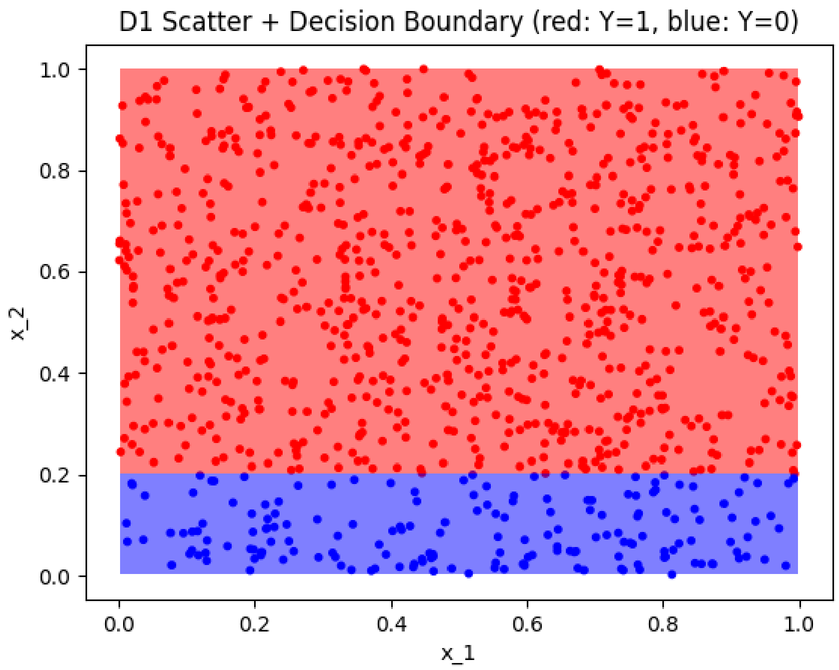
6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

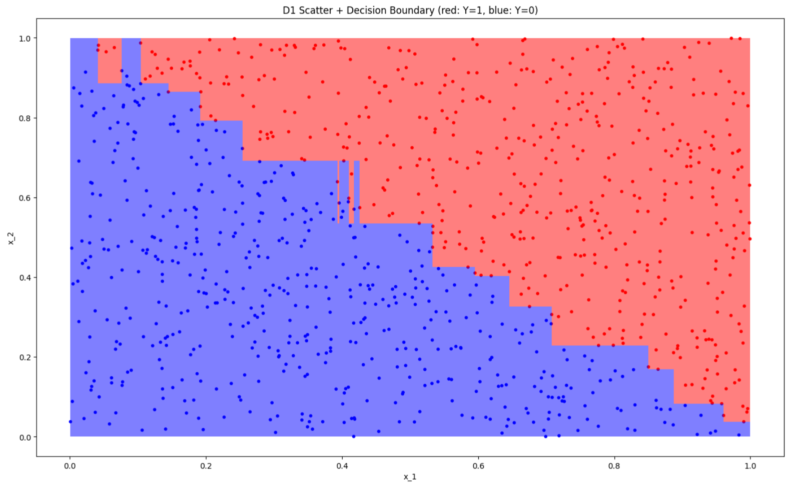
    - Produce a scatter plot of the data set.





- Visualize your decision tree's decision boundary (or decision region, or some other ways to clearly visualize how your decision tree will make decisions in the feature space).





Then discuss why the size of your decision trees on D1 and D2 differ. Relate this to the hypothesis space of our decision tree algorithm.

For D1, the the decision tree's decision boundary was very similar to the hypothesis space where we would have expected by looking at the scatter plot of D1, that is, a clear boundary at around $x_2 = 0.2$. For D2, the decision boundary from the decision tree is similar but not exactly what we would expect for the decision boundary to be. By looking at the scatter plot of D2, we would expect a decision boundary at $x_2 = x_1 + 1$. However, the decision tree's decision boundary gave a staircase shape around $x_2 = x_1 + 1$.

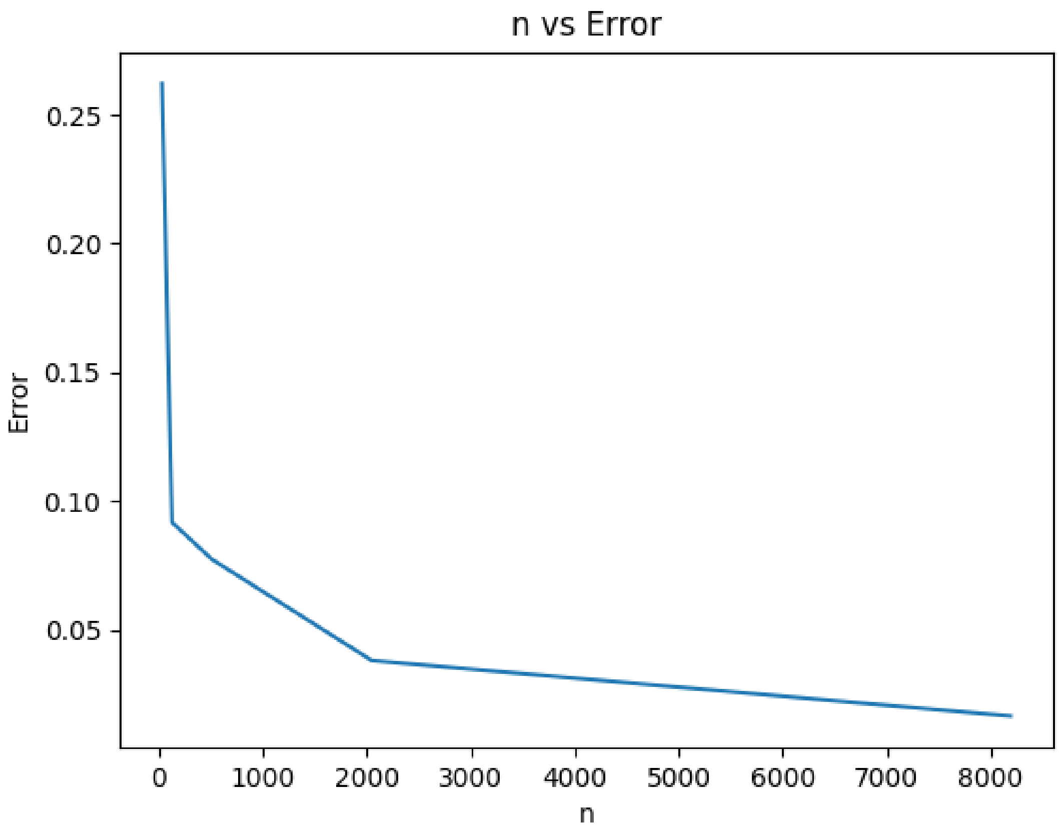
7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

- You will randomly split Dbig.txt into a candidate training set of 8192 items and a test set (the rest). Do this by generating a random permutation, and split at 8192.
- Generate a sequence of five nested training sets $D_{32} \subset D_{128} \subset D_{512} \subset D_{2048} \subset D_{8192}$ from the candidate training set. The subscript n in D_n denotes training set size. The easiest way is to take the first n items from the (same) permutation above. This sequence simulates the real world situation where you obtain more and more training data.
- For each D_n above, train a decision tree. Measure its test set error err_n. Show three things in your answer: (1) List n, number of nodes in that tree, err_n. (2) Plot n vs. err_n. This is known as a learning curve (a single plot). (3) Visualize your decision trees' decision boundary (five plots).

(1) number of nodes n and corresponding err_n

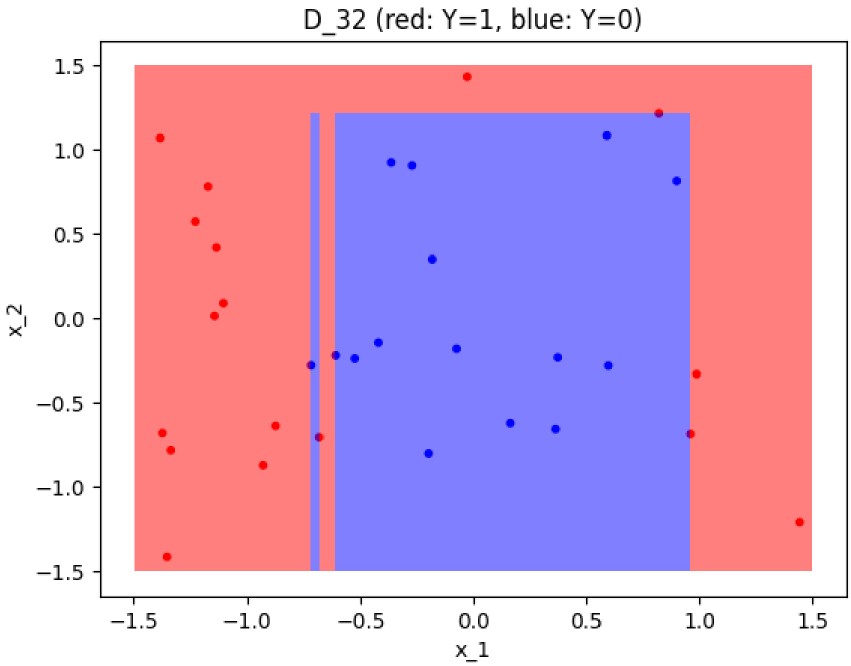
n	Number of Nodes	err_n
32	11	0.26216814159292035
128	19	0.0918141592920354
512	61	0.07743362831858407
2048	139	0.03816371681415929
8192	279	0.016592920353982302

(2) n vs. err_n

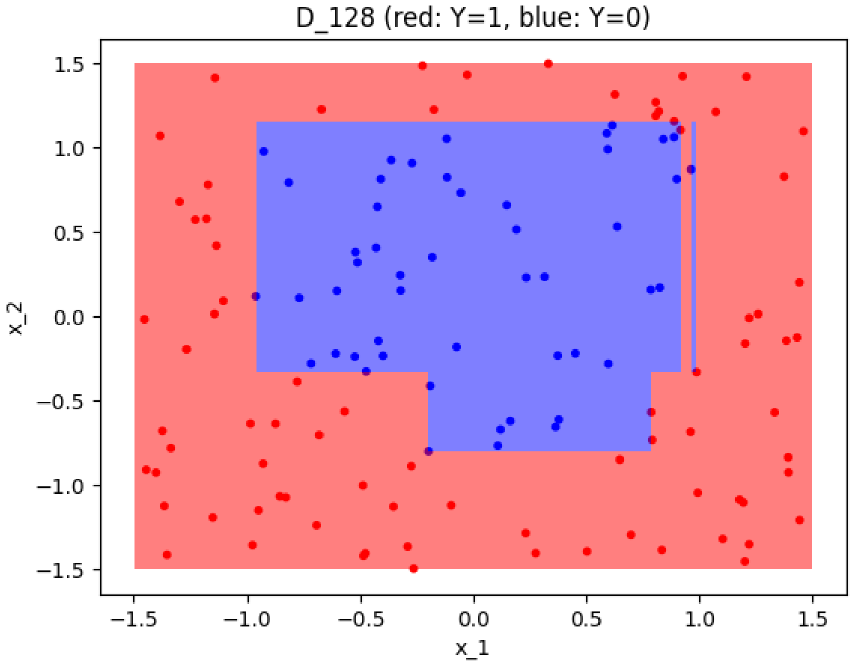


(3) Decision tree's decision boundaries:

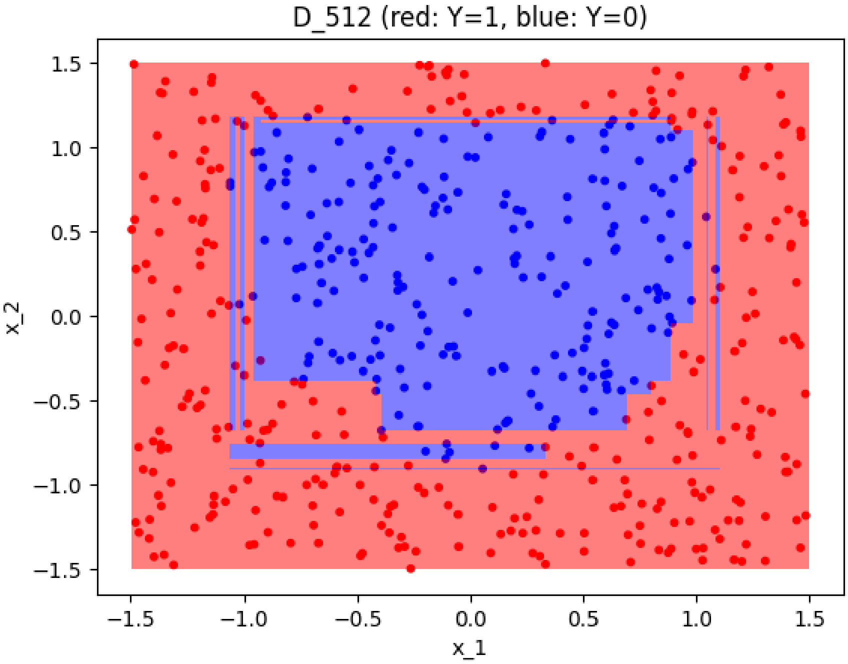
D_{32}:



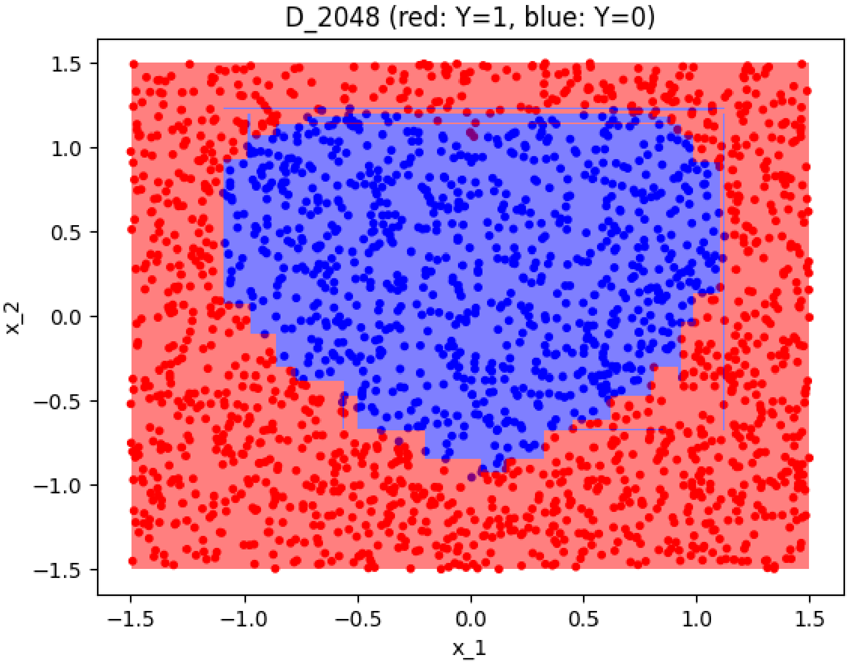
D_{128}:



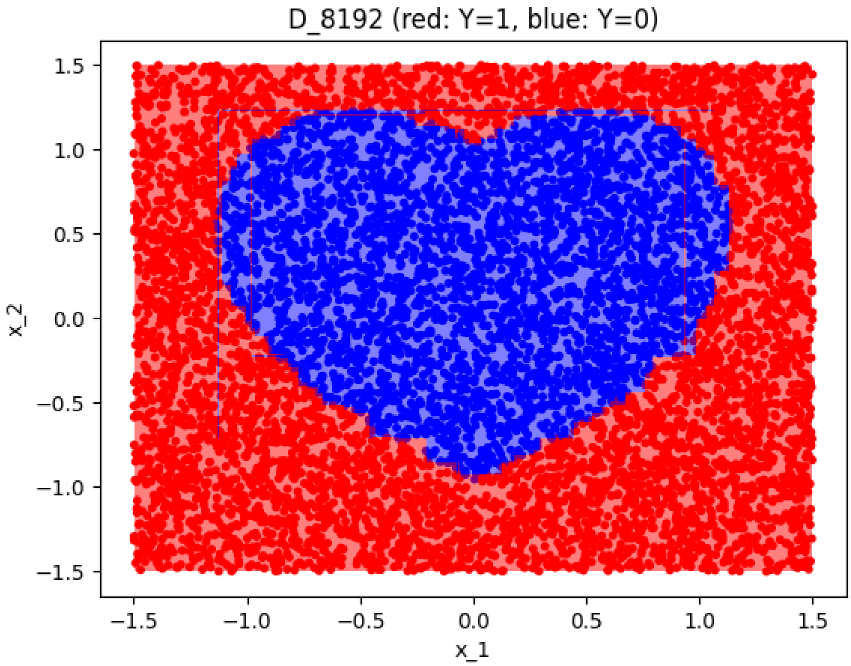
D_{512}:


D_512 (red: Y=1, blue: Y=0)
x_1
x_2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

D_{2048}:


D_2048 (red: Y=1, blue: Y=0)
x_1
x_2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

D_{8192}:

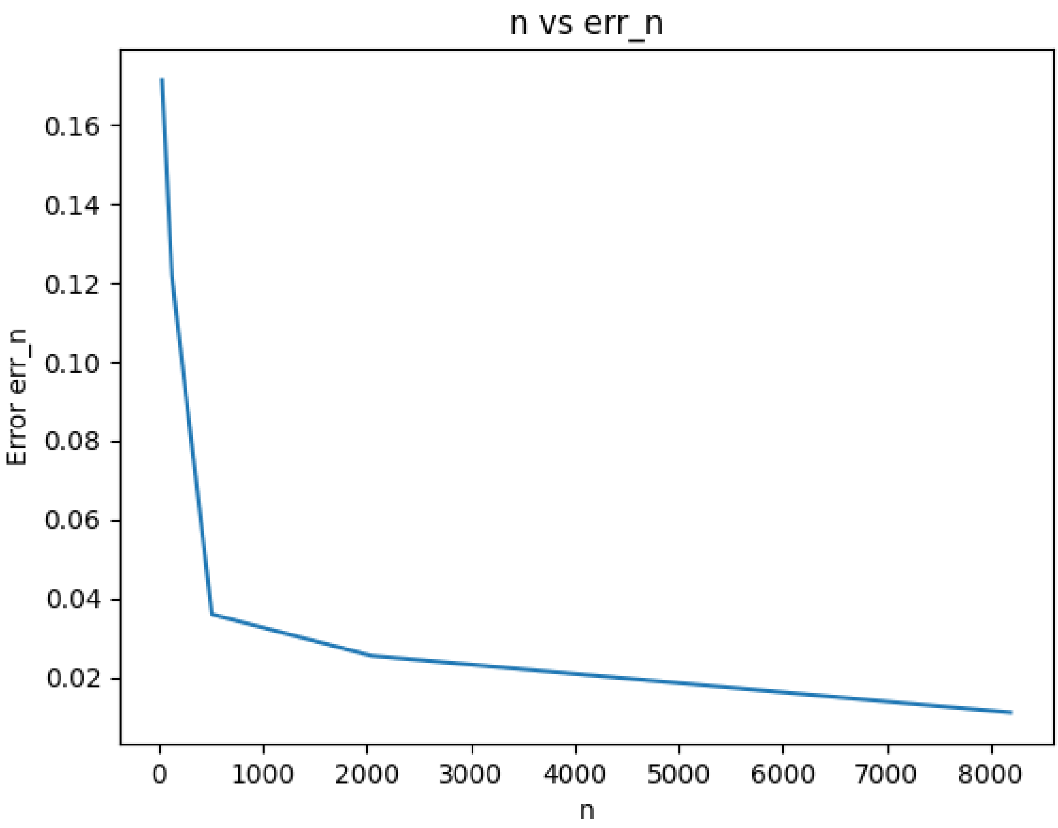


3 sklearn [10 pts]

Learn to use sklearn (https://scikit-learn.org/stable/). Use sklearn.tree.DecisionTreeClassifier to produce trees for datasets $D_{32}, D_{128}, D_{512}, D_{2048}, D_{8192}$. Show two things in your answer: (1) List n, number of nodes in that tree, err_n. (2) Plot n vs. err_n.
(1) number of nodes n and corresponding err_n

n	Number of Nodes	err_n
32	11	0.07743362831858407
128	31	0.08241150442477876
512	53	0.04424778761061947
2048	109	0.023230088495575223
8192	243	0.01327433628318584

(2) n vs. err_n



4 Lagrange Interpolation [10 pts]

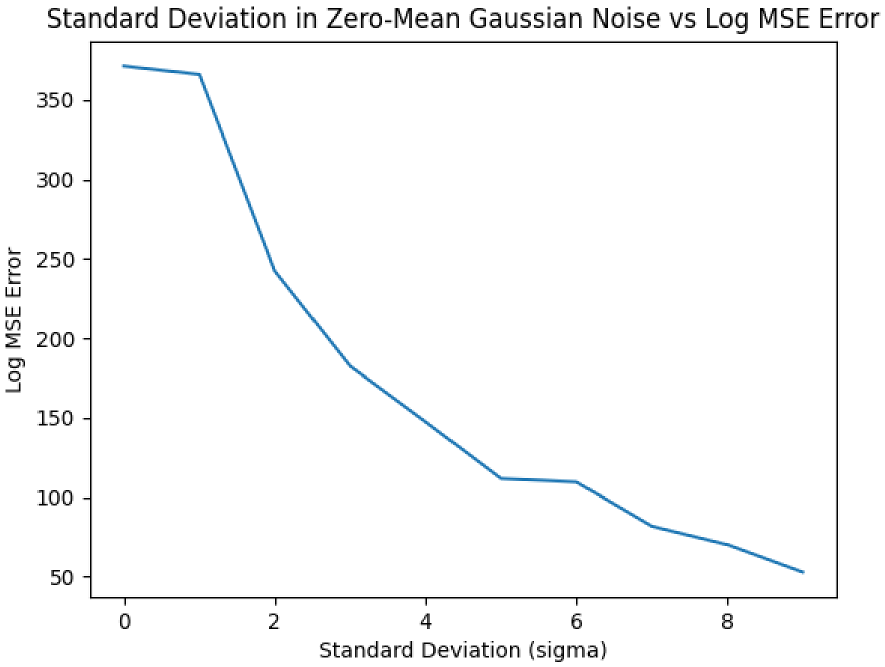
Fix some interval $[a, b]$ and sample $n = 100$ points x from this interval uniformly. Use these to build a training set consisting of n pairs (x, y) by setting function $y = sin(x)$.

Build a model f by using Lagrange interpolation, check more details in https://en.wikipedia.org/wiki/Lagrange_polynomial and https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.lagrange.html.

Generate a test set using the same distribution as your test set. Compute and report the resulting model's train and test error. What do you observe? Repeat the experiment with zero-mean Gaussian noise ϵ added to x. Vary the standard deviation for ϵ and report your findings.
The interval I chose was [0, 2π]
The results I got was:

Noise	$log_{10}MSE$	Standard Deviation (σ)
None	366.45576741318945	N/A
Yes	371.3808375446383	0
Yes	366.14317640682935	1
Yes	242.60394936227559	2
Yes	182.53917601501894	3
Yes	147.53281719892328	4
Yes	111.77762078246857	5
Yes	109.6468350555486	6
Yes	81.76901863719014	7
Yes	70.20895138776518	8
Yes	52.89304239643873	9

The original model f without zero-mean Gaussian noise ϵ was had a lower MSE than with the zero-mean Gaussian noise when there was no standard deviation. However, when there was standard deviation of ≥ 1, the MSE became less for the model with the zero-mean Gaussian noise.
Standard Deviation vs $log_{10}MSE$:



As the standard deviation increases, the MSE decreases as shown in the graph above.