HOMEWORK 2

>>Thong Nguyen <<
>>9084850198<<

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. **This is my Github Link**

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 $x_{\cdot j} \geq c$.
- 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 $y = 1$.

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.
   Decision tree is an algorithm splitting the data into subsets until they are pure (Therefore, there is no further splitting necessary and the tree should predict the value in this stage). Since the training items with the same label (the same class), the entropy of one of the splitting candidates is equal to zero (which means the subsets are pure). Therefore, our decision tree algorithm would stops at pure labels.

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.

x_1	x_2	y
1	1	1
1	0	1
0	1	1
0	0	0

   The value of the node would be the combination of features and values of 4 instances. In this dataset, if the tree tries to split the data on either features, we will end up with two subsets containing both classes since these two features are not enough to split perfectly into 2 classes at the node. Then, the prediction for any instance falling within this data would be the majority class, 1. Therefore, the tree would refuse to split and set the value y = 1. However, if we manually force it to split, it would consider whether x1 is 1 or not. If it is, then we consider whether x2 is 1 or not. If it is true, then y = 1; otherwise, it would predict y = 0. So if the tree splits in this way, the algorithm would reach 0 training error.

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.

Table 1: Candidate splits of the root node of 'Druns.txt'

	x1	x2
Candidate split	0.1	8
Entropy	0.439497	0.439497
Information Gain Ratio	0.100518	0.430157

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.

   - The tree in plaintext representation: {'x2 $\geq$ 0.201829': [1, 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.

- Explain: The '{ }' represents a subtree (the original tree is also considered a subtree). ' ' means the node and condition of splitting. For example, 'x1 $\geq$ 10.0' is one node, and in this case, it is the root node. Followed by a ': [a, b]' is the two results with a is the left subtree (or a leaf) and b is the right subtree (or a leaf). If a or b are continued subtrees, this will lead to the next '{}'
- Set of logic rules : If x2 is larger than 201829, then the result y is equal to 1. Otherwise, y = 0.

5. (Or is it?) [20 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.

     – {'x1 $\geq$ 10.0': [1, {'x2 $\geq$ 3': [1, 0]}]}
     – Explain notation: The '{ }' represents a subtree (the original tree is also considered a subtree). ' ' means the node and condition of splitting. For example, 'x1 $\geq$ 10.0' is one node, and in this case, it is the root node. Followed by a ': [a, b]' is the two results with a is the left subtree (or a leaf) and b is the right subtree (or a leaf). If a or b are continued subtrees, this will lead to the next '{}' (1)

   - 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. **Set of logic rules:** If x1 is larger than 10, then the result y is equal to 1. Otherwise, we continue to compare x2 with 3. If it is larger or equal to 3, then y = 1; otherwise, y = 0.
   - Build a decision tree on D2.txt. Show it to us.


```
{'x1 >=  0.533076': [{'x2 >=  0.228007': [{'x2 >=  0.424906': [1,
     {'x1 >=  0.708127': [1,
       {'x2 >=  0.32625': [{'x1 >=  0.595471': [{'x1 >=  0.646007': [1,
            {'x2 >=  0.403494': [1, 0]}]},
          0]},
        0]}]}]},
   {'x1 >=  0.887224': [{'x2 >=  0.037708': [{'x2 >=  0.082895': [1,
         {'x1 >=  0.960783': [1, 0]}]},
       0]},
     {'x1 >=  0.850316': [{'x2 >=  0.169053': [1, 0]}, 0]}]}]},
 {'x2 >=  0.88635': [{'x1 >=  0.041245': [{'x1 >=  0.104043': [1,
       {'x1 >=  0.07642': [0, 1]}]},
     0]},
   {'x2 >=  0.691474': [{'x1 >=  0.254049': [1,
       {'x1 >=  0.191915': [{'x2 >=  0.792752': [1, 0]},
         {'x2 >=  0.864128': [{'x1 >=  0.144781': [1, 0]}, 0]}]}]},
     {'x2 >=  0.534979': [{'x1 >=  0.426073': [1,
         {'x1 >=  0.409972': [{'x1 >=  0.417579': [0, 1]},
           {'x1 >=  0.393227': [{'x1 >=  0.39583': [0, 1]}, 0]}]}]},
       0]}]}]}]}
```


   - Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
     Explain: This plain text uses the same notation as the above (1). It starts with the rode node testing if x1 $\geq$ 0.533076. If x1 $\geq$ 0.533076, then we continue with x2 $\geq$ 0.228007. If the condition is satisfied, we would dig deeper into each depth of the tree (to the left-handed side of the tree). And when the condition is not satisfied, we would go to the right of the "," to see what the condition of other side is (right-handed size). For example the third condition, x2 $\geq$ 0.425, if it is true, then y is equal to 1; otherwise we consider the next condition (this is the node on the right of the node x2 $\geq$ 0.425), which is x1 $\geq$ 0.708127

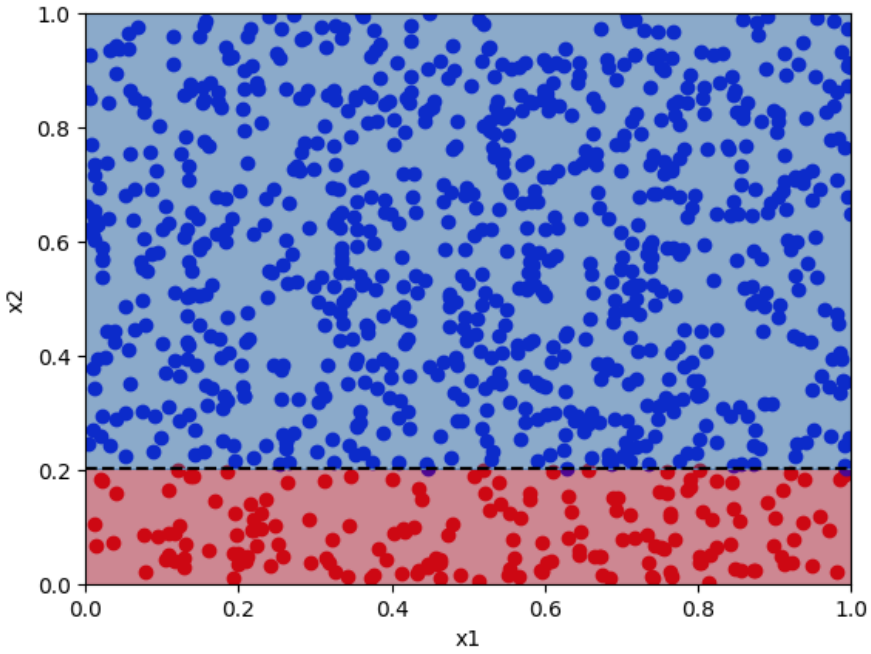
6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

   - Produce a scatter plot of the data set.

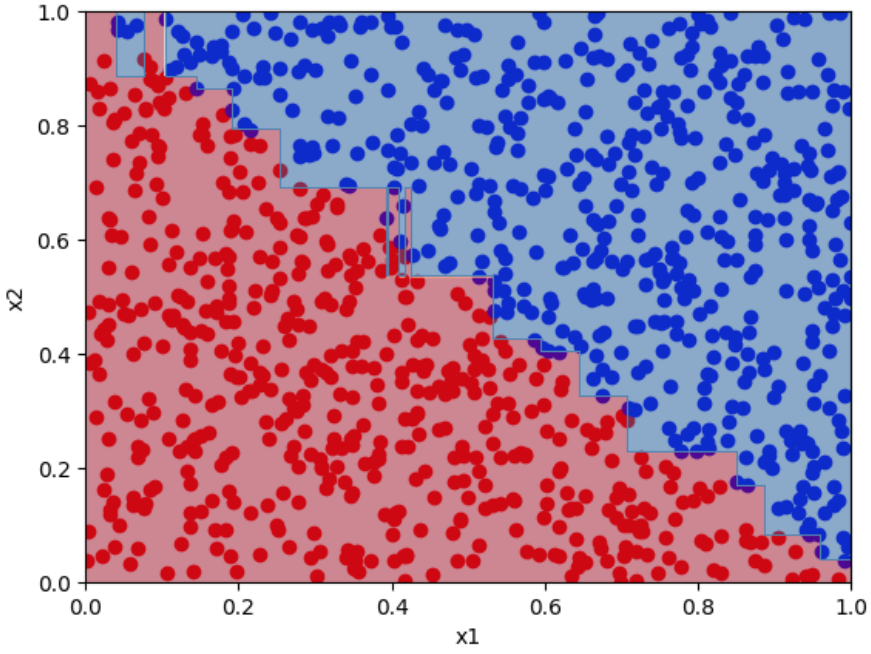
- Visualize your decision tree's decision boundary (or decision region, or some other ways to clearly visualize how your decision tree will make decisions in the feature space).

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.

- **Consider this graph for D1.txt** The red area is those points having x2 $<$ 0.201829, and the blue area is where the points having x2



- **Consider this graph for D2.txt** Each of the small line in the rectangular separation between two areas represents the set of conditions above. The blue area is where y is predicted to be 1, and otherwise. $\geq$ 0.201829



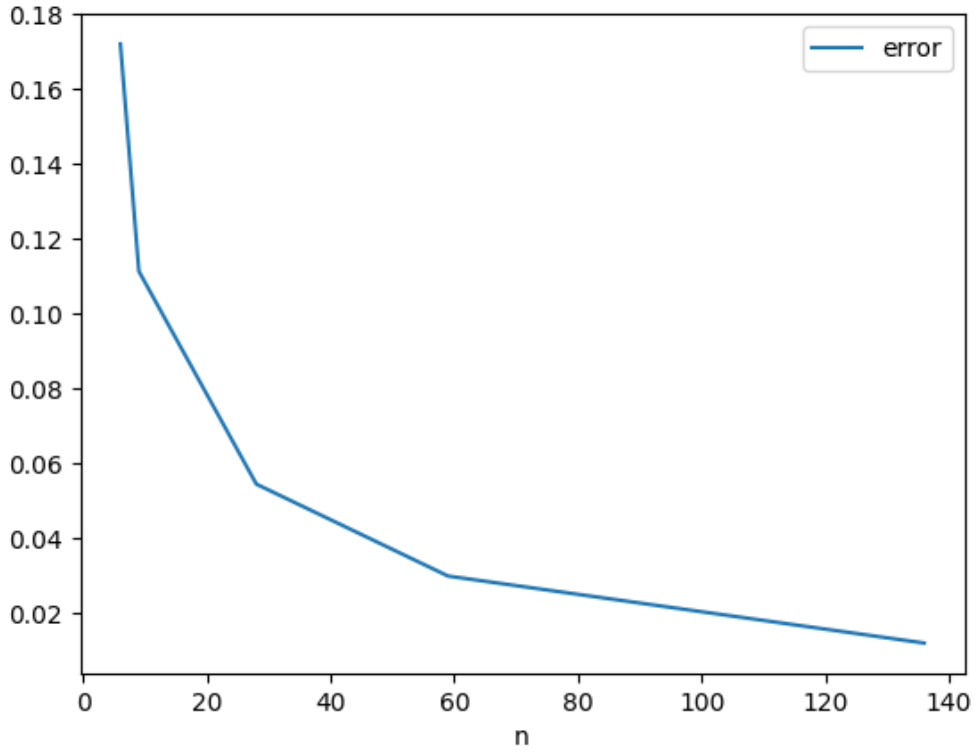
- There are many reasons why the decision trees' sizes of the two datasets are different; one can stem from the size, complexity, and distribution of the datasets. Even if the 2 datasets have the same size of 1000, D2 is shown to be more complex, and each candidate split has higher entropy, the higher predictability of the target variable. Therefore, we will need more split to reach the entropy of the set is 0, making D2 a larger tree. Furthermore, our hypothesis space is also based on the information gain ratio (the subtree stops when all splits have zero gain ratio), and this value is based on the impurity of the set associated with each feature. Therefore, complex datasets like D2 is more complicated to reach zero gain ratio and the pure subsets than D1.

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

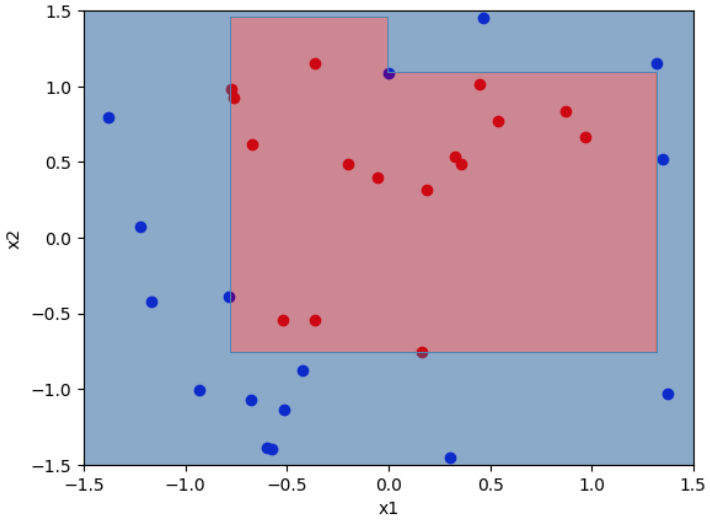
   - You will randomly split Dbig.txt into a candidate training set of 8192 items and a test set (the rest). Do this by generating a random permutation, and split at 8192.
   - Generate a sequence of five nested training sets $D_{32} \subset D_{128} \subset D_{512} \subset D_{2048} \subset D_{8192}$ from 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).

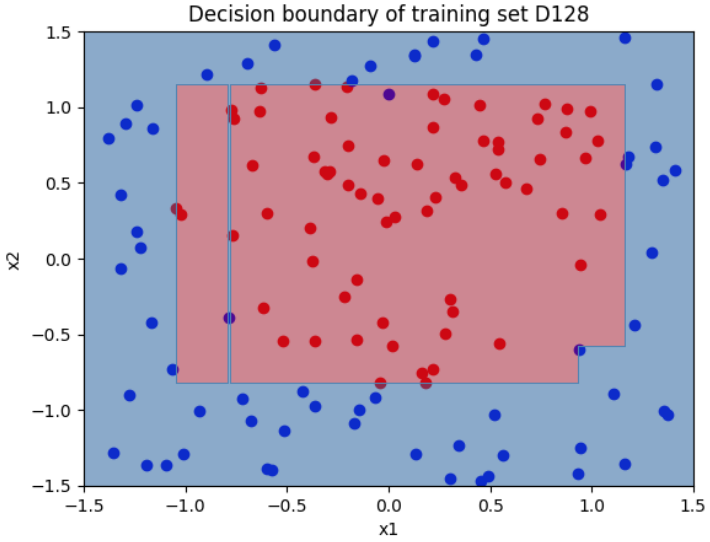
Table 2: The number of nodes vs their prediction errors

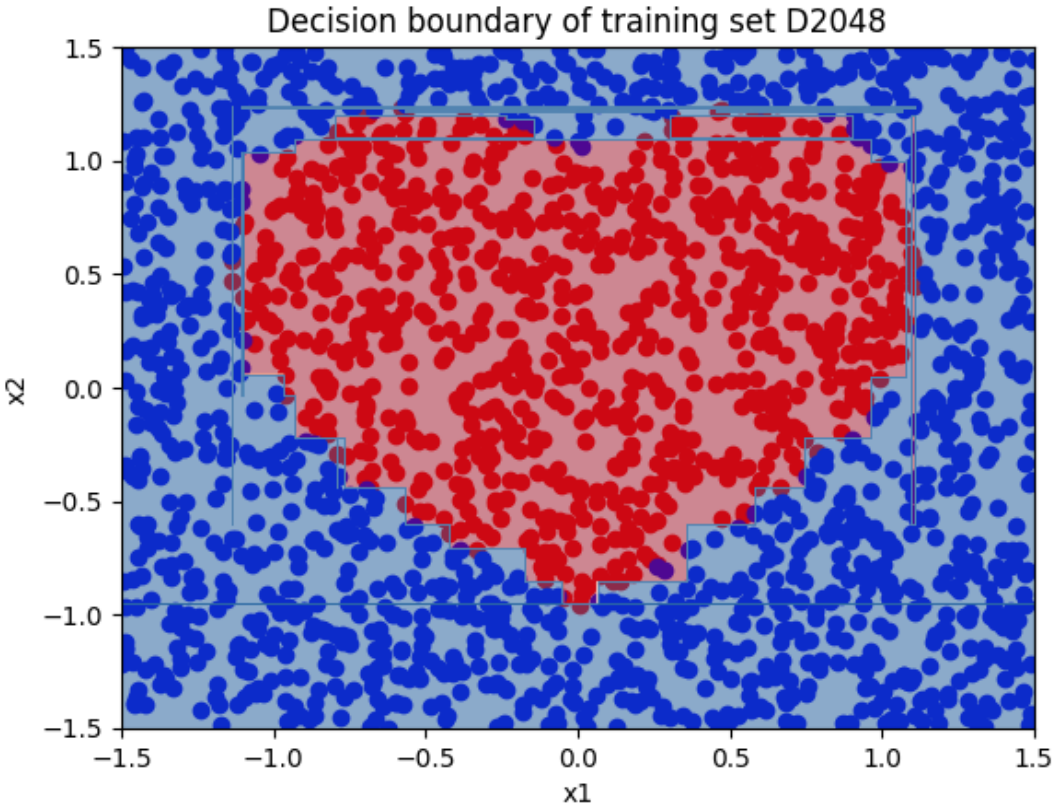
error	n
0.17215088282504012	6
0.11142625607779579	9
0.05459527824620573	28
0.03005533199195171	59
0.012168141592920354	136

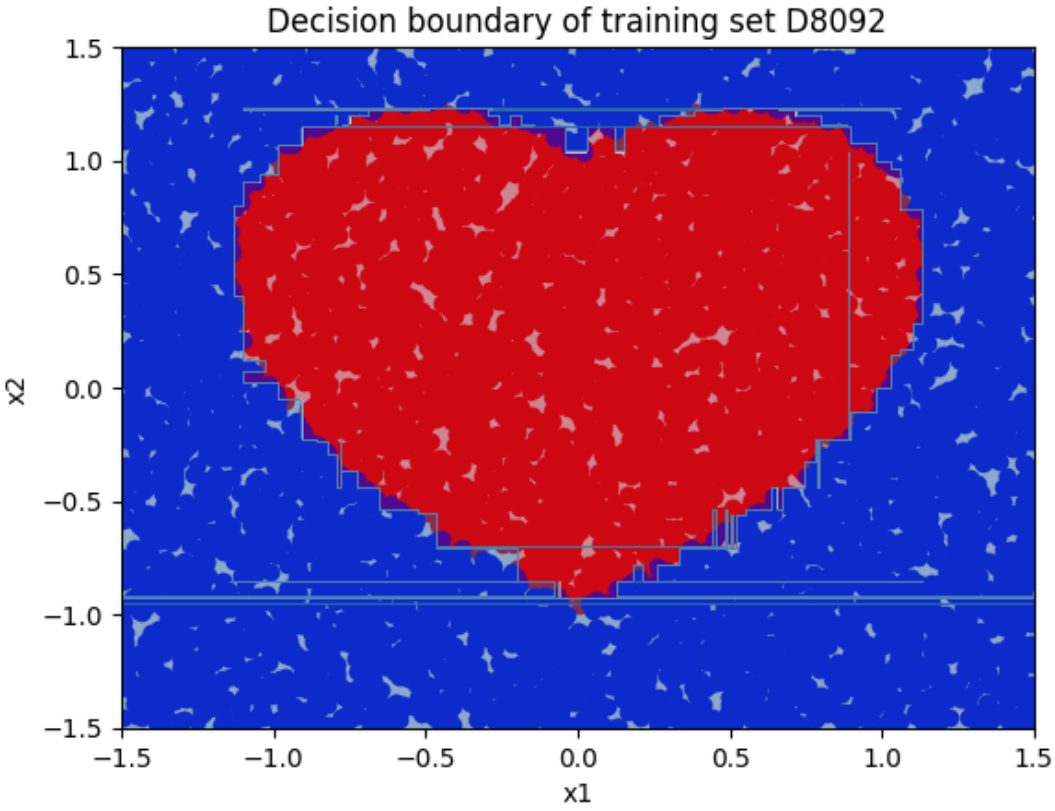


- **Considered decision tree's decision boundary of D32, D128, D512, D2048, D8092**






Decision boundary of training set D2048
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
x2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x1

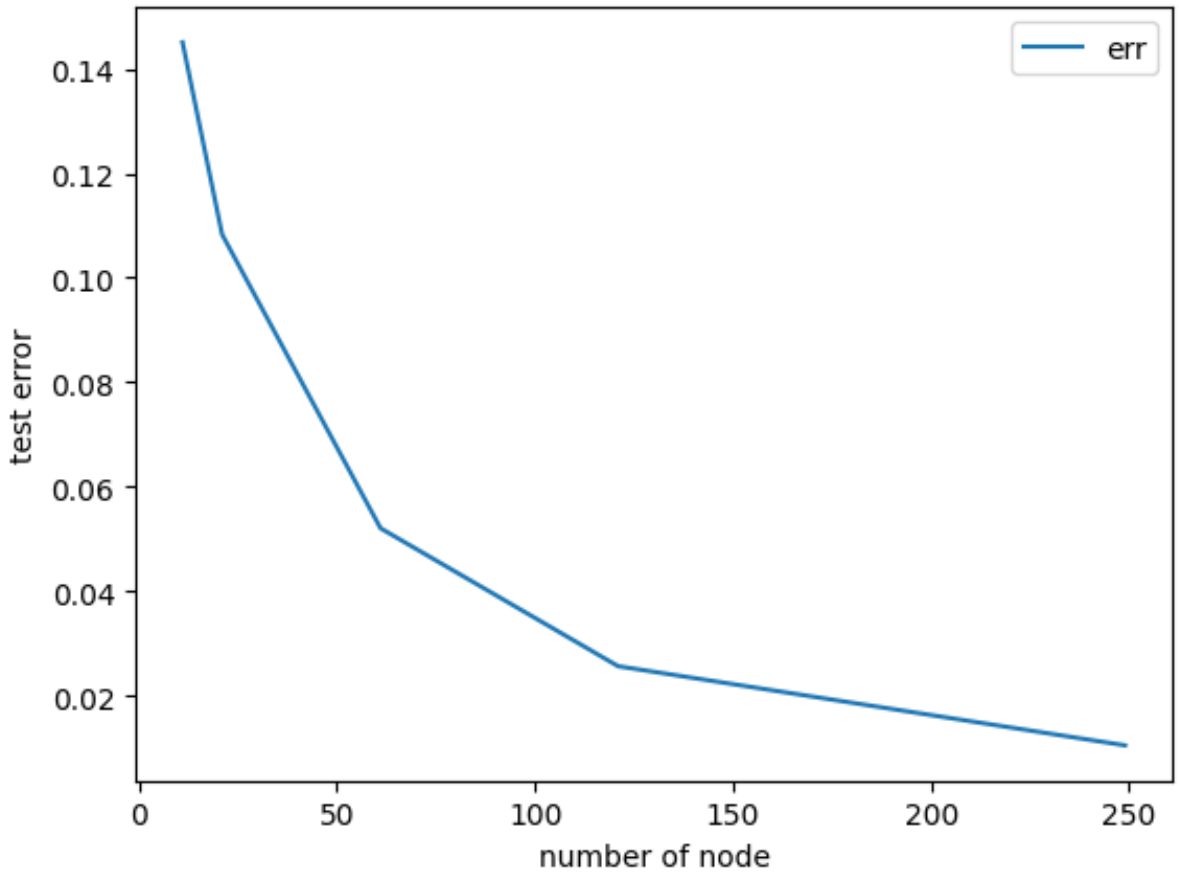

Decision boundary of training set D8092
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
x2
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x1

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.

Table 3: Sklearn Decision Tree's number of nodes vs their test errors

Number of nodes	Test Error
11	0.1450642054574639
21	0.10828606158833065
61	0.05206576728499157
121	0.025653923541247514
249	0.01050884955752207



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.

Table 4: Mean Squared Errors of Models f using Lagrange Interpolation

Standard Deviation of noise	train error	test error
0.0	2.764987948841503e+139	4.448900234046315e+139
0.05	2.5372300086359705e+146	9.397726153150905e+145
0.1	1.8348342655720676e+150	2.812304416389374e+148
0.15	1.0271350471838772e+152	4.371386765257422e+147
0.2	2.0902736336674595e+153	3.895983632576597e+147
0.25	1.1474712203801628e+156	1.8219155307474113e+149
0.3	4.445414052031636e+155	7.97238633803938e+147
0.35	1.820988310623467e+153	7.771471652870264e+144

- As we observe, these errors are significantly huge (this is because the difference between the predicted and the actual f(x) is significantly small). The mean squared training errors of models are relatively increasing according to the standard deviation of the noise. And for each value of noise, the mean squared test errors are relatively smaller than that of the trained error (mean that their difference before squaring is higher than that of the training error)