HOMEWORK 2

>>Yuanjun Ge<<
>>9082698615<<

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:

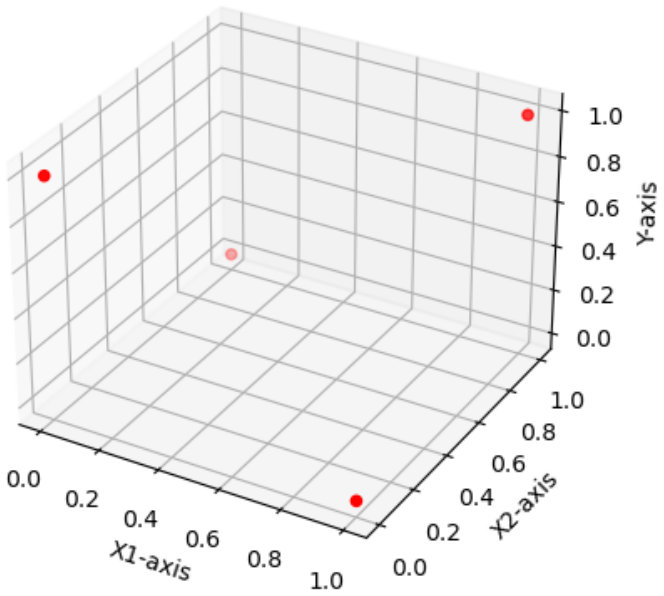
- Candidate splits (j, c) for numeric features should use a threshold c in feature dimension j in the form of $x_j \geq c$.
- c should be on values of that dimension present in the training data; i.e. the threshold is on training points, not in between training points. You may enumerate all features, 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 $y = 1$.

Click here: GitHub repo for this question and all questions below.

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.
   Yes, because our algorithm will not be able to find any candidates for next split. So, it must be 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.
   (with codes)
   Say we have data set containning four data points: [0,0,1], [1,1,0],[0,1,0],[1,0,0].



   After checking(see code file), we can see that this node is a leaf node. This is because our algorithm can only find splits that has zero gain ratio, which cannot split further and make this node a leaf node.
   However, when we manually force it to split using x1 as the threshould feature, say we have two subnodes containning [0,0.1],[0,1,0] and [1,0,0],[1,1,0]. We can see the splitted data set is not a leaf node(see codes) and will be further split. It is because when our algorithm can generate a positive gain ratio when it further splits two data points into two leaves respectively.

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.
   (with codes)

```
x1 has candidate cut 0.0 with information gain ratio 0,
    Information gain: 0.0 and entropy of the split: 0.0
x1 has candidate cut 0.1 with information gain ratio: 0.10051807676021828
x2 has candidate cut -2.0 with information gain ratio 0,
    Information gain: 0.0 and entropy of the split: 0.0
x2 has candidate cut -1.0 with information gain ratio: 0.10051807676021828
x2 has candidate cut 0.0 with information gain ratio: 0.055953759631263686
x2 has candidate cut 5.0 with information gain ratio: 0.11124029586339806
```

```
x2 has candidate cut 6.0 with information gain ratio: 0.23609960614360798
x2 has candidate cut 7.0 with information gain ratio: 0.055953759631263755
x2 has candidate cut 8.0 with information gain ratio: 0.4301569161309807
```

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.
(with codes)

```
        -True->|y=1
x2≥2.0
                        -True->|y=1
        -False-> x1≥10.0
                        -False->|y=0
```

The decision tree built on D3leaves.txt is above. When evaluating a data point [X1, X2, Y] from left to right in the tree, the following rules apply:
1. If X2 ≥ 2 is true, the data point follows the true branch, resulting in Y $= 1$.
2. If X2 ≥ 2 is false, the data point proceeds down the false branch, where it encounters the second threshold.
3. Following 2, if X1 ≥ 10 is true for our data point, then Y $= 1$; otherwise, Y $= 0$.

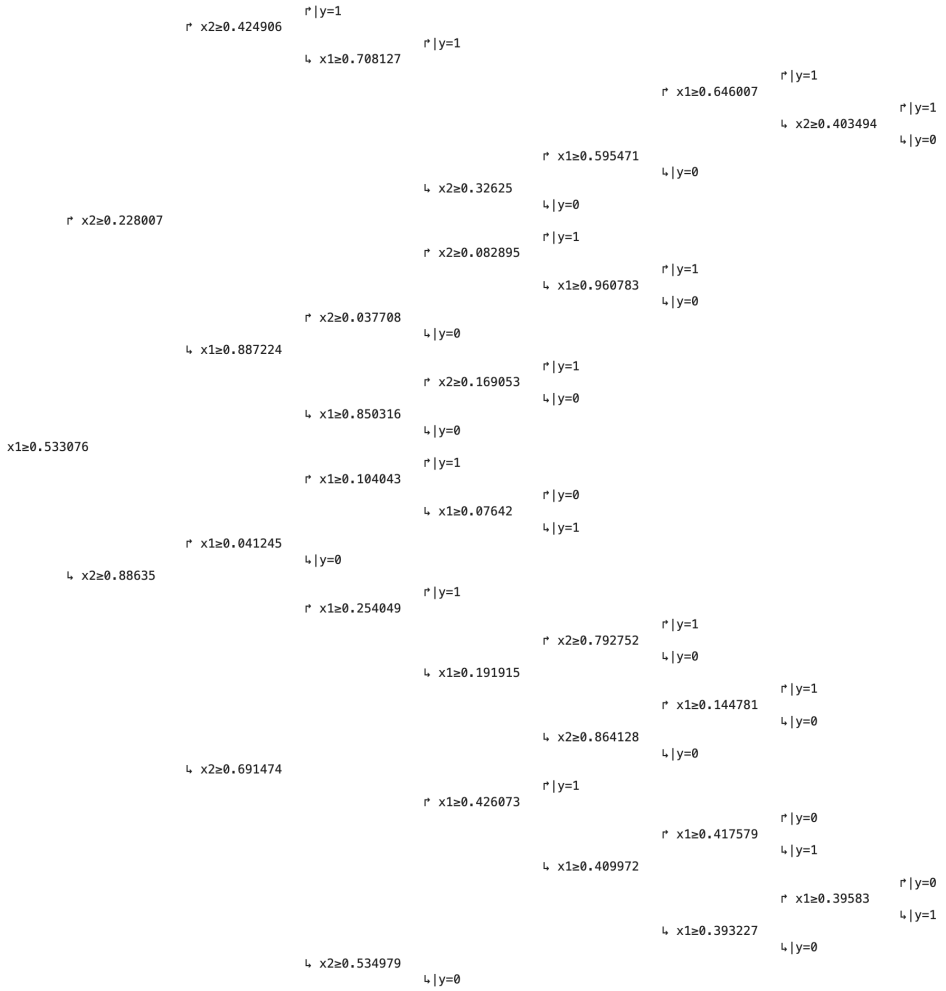
5. (Or is it?) [10 pts] For this question only, make sure you DO NOT VISUALIZE the data sets or plot your tree's decision boundary in the 2D $\mathbf{x}$ space. If your code does that, turn it off before proceeding. This is because you want to see your own reaction when trying to interpret a tree. You 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.

```
        -True->|y=1
x2≥0.201829
        -False->|y=0
```

   - 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 built on D1.txt is above. Our tree suggests that for any data point having x2$\geq$0.201829, it predicts y$=$ 1; otherwise, y$=$0;
   - Build a decision tree on D2.txt. Show it to us.
   see below.

[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.



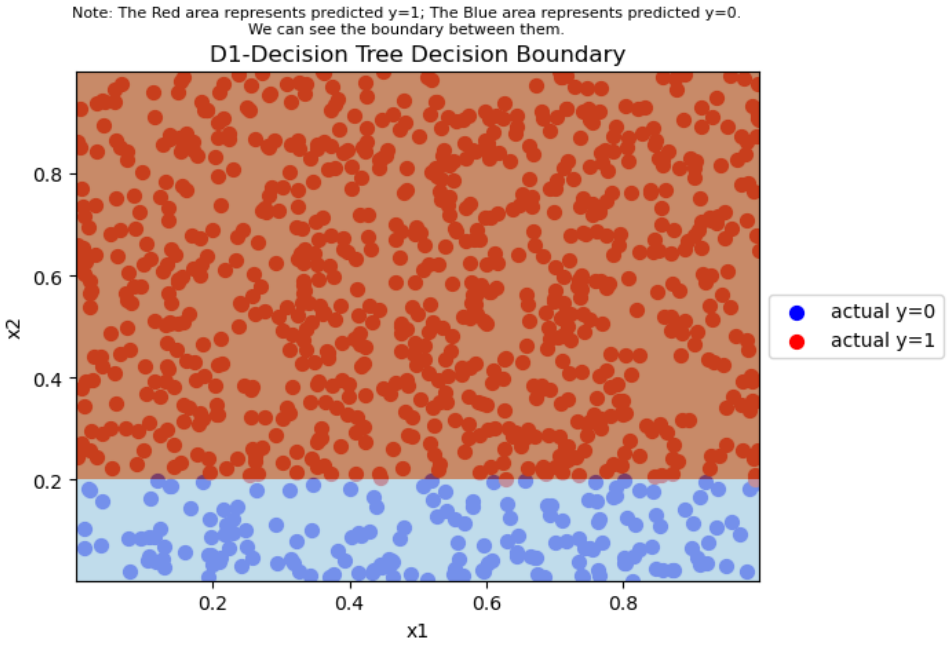
- Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
  The decision tree built on D2.txt is above. From left to right, we can count the depth of the tree is 9 (including root and leaf). It's quite difficult to interpret D2 desion tree when this tree is quite complex without visualization.

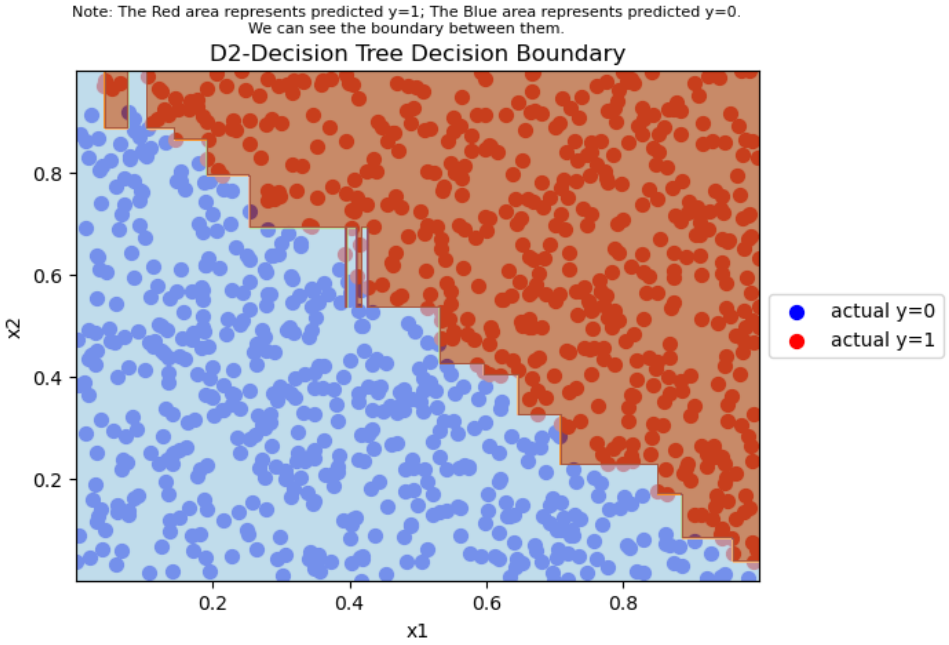
6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

   - 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.

   (Plots are in next page)
   For D1.txt, the decision boundary is a simple horizontal line, whereas for D2.txt, it forms a diagonal ladder-like pattern. It's evident that the size of the decision tree generated for D2 is much larger compared to D1. Our decision tree algorithm defines splits based on a single feature at each node and continues splitting until certain conditions are met. Specifically, it stops when a node becomes empty or when the entropy of all candidate splits becomes zero. For datasets like D1, our algorithm stops at the first split, showing the the hypothesis space of a decision tree could limit its effectiveness for some datasets.



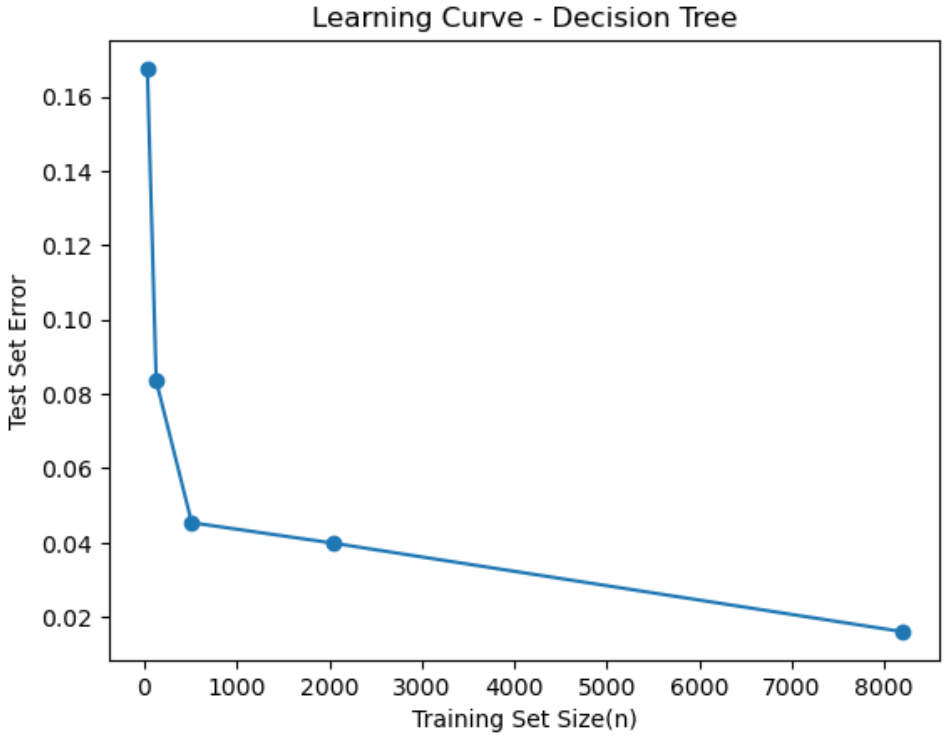


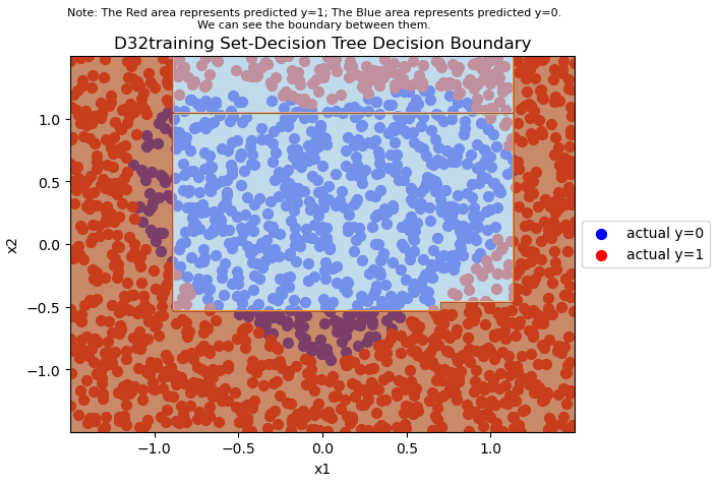
7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

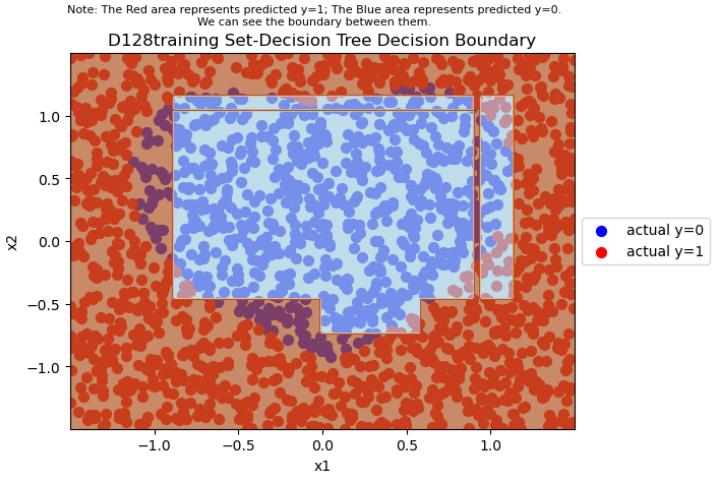
   - You will randomly split Dbig.txt into a candidate training set of 8192 items and a test set (the rest). Do this by generating a random permutation, and split at 8192.
   - Generate a sequence of five nested training sets $D_{32} \subset D_{128} \subset D_{512} \subset D_{2048} \subset D_{8192}$ from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

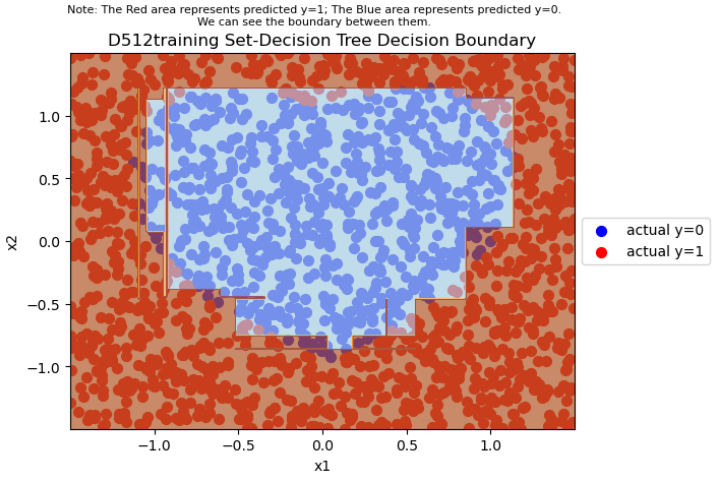
answer: (1) List n, number of nodes in that tree, err_n. (2) Plot n vs. err_n. This is known as a learning curve (a single plot). (3) Visualize your decision trees' decision boundary (five plots).

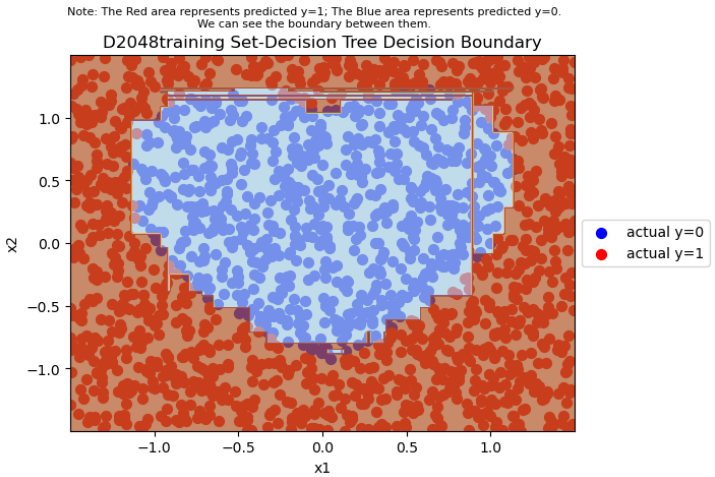
n:32, number of nodes in the tree:17, errn:0.16758849557522124
n:128, number of nodes in the tree:23, errn:0.08351769911504425
n:512, number of nodes in the tree:63, errn:0.04535398230088496
n:2048, number of nodes in the tree:137, errn:0.03982300884955752
n:8192, number of nodes in the tree:263, errn:0.016039823008849558

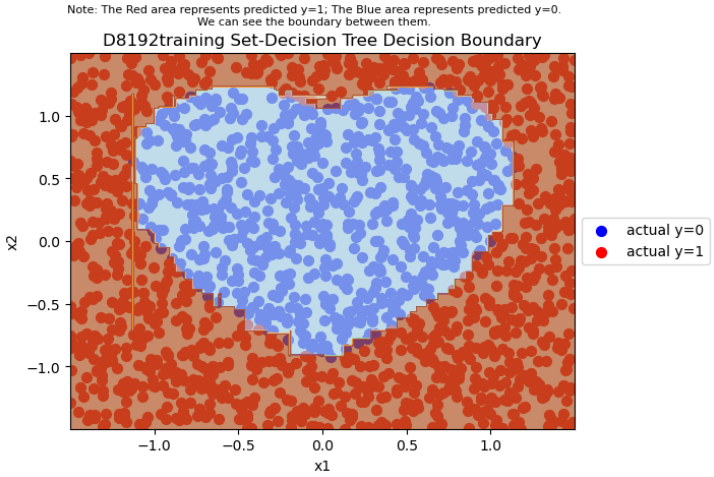





Note: The Red area represents predicted y=1; The Blue area represents predicted y=0.
We can see the boundary between them.
D128training Set-Decision Tree Decision Boundary
x2
x1
1.0
0.5
0.0
−0.5
−1.0
actual y=0
actual y=1

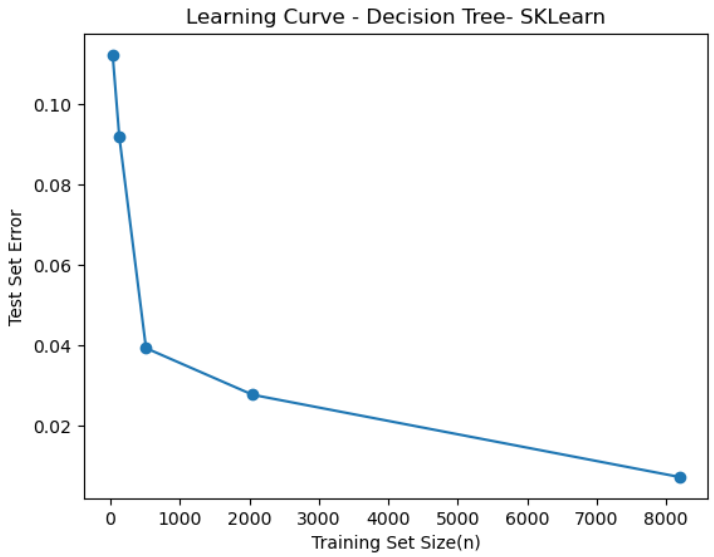

Note: The Red area represents predicted y=1; The Blue area represents predicted y=0.
We can see the boundary between them.
D512training Set-Decision Tree Decision Boundary
x2
x1
1.0
0.5
0.0
−0.5
−1.0
actual y=0
actual y=1


Note: The Red area represents predicted y=1; The Blue area represents predicted y=0.
We can see the boundary between them.
D2048training Set-Decision Tree Decision Boundary
x2
x1
1.0
0.5
0.0
−0.5
−1.0
actual y=0
actual y=1



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.
n:32, number of nodes in the tree:17, errn:0.1122787610619469
n:128, number of nodes in the tree:23, errn:0.0918141592920354
n:512, number of nodes in the tree:57, errn:0.039269911504424826
n:2048, number of nodes in the tree:105, errn:0.027654867256637128
n:8192, number of nodes in the tree:229, errn:0.007190265486725633



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 $y = sin(x)$.

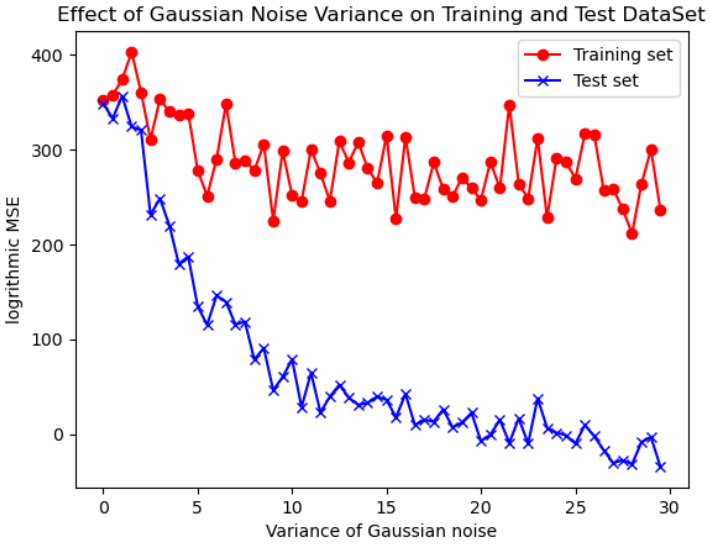
Build a model f by using Lagrange interpolation, check more details in https://en.wikipedia.org/wiki/Lagrange_polynomial and https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.lagrange.html.

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.

Before adding noise:
Train error(Log of MSE) : 351.2223662211015
Test error(Log of MSE) : 347.9679126631317

We can observe that both error are super high which is not nomal. It's because we trained too many samples for the Lagrange Interpolation model as scipy warns "Do not expect to be able to use more than about 20 points even if they are chosen optimally". A small change of x can result in huge fluctuations in the polynomial coefficients, which may be a reason of high MSE.

After adding noise with different variance:



When the variance of Gaussian nose is larger, our trainning set is "fatter" distributed, which lead to the both training set and test set reduce on MSE (significantly on test set). It probably indicates that modeling errors or fluctuations due to the many equidistant data points used in Lagrange interpolation could be masked by noise to some extent.