HOMEWORK 2

AkhilPerumalReddy
9084606913

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 $y = 1$.

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.

   Ans: Because all items possess identical labels, the entropy will be zero as P equals 1 due to the presence of only one label. Consequently, the Information Gain equals 0. This is evident as knowing there is only one label allows for a direct prediction, resulting in an Information Gain of 0. Thus, there is only one value within the node, ensuring it is a definite 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.

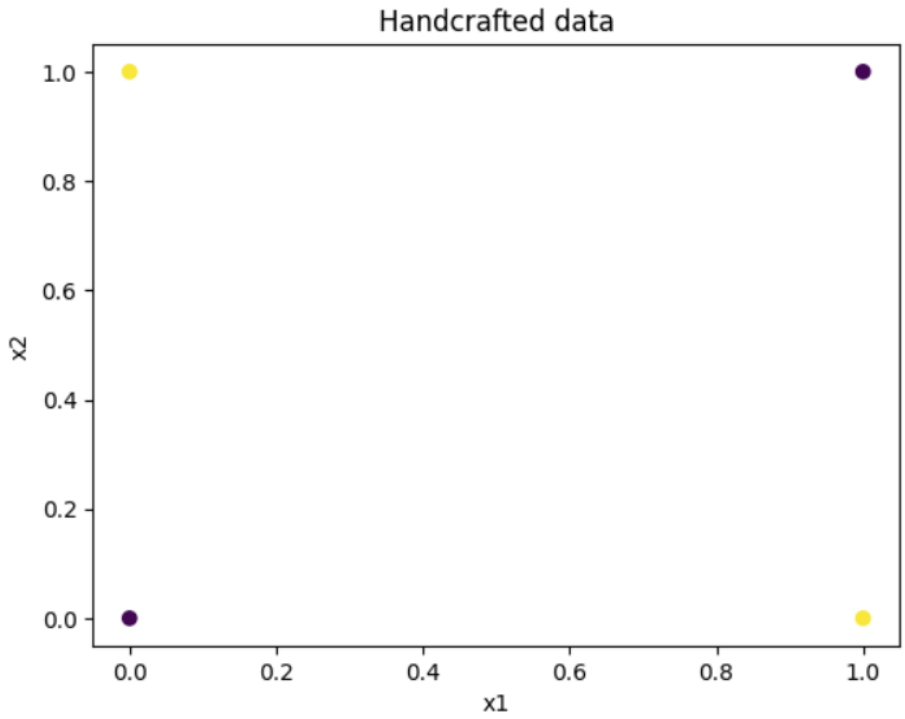

Figure 1

   Ans: When you split the data across any class at any point, the distribution of labels is same. So, H(Y) and H(Y/S) are equal making information gain zero. Hence, our root will be leaf.

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	Information Gain	Information Gain Ratio
0	0.0	0.0	
0	0.1		0.10051807676021828
1	-2.0	0.0	
1	-1.0		0.10051807676021828
1	0.0		0.0559537596312636
1	1.0		0.00578004220515232
1	2.0		0.0011443495172767494
1	3.0		0.016411136842102134
1	4.0		0.04974906418177849
1	5.0		0.11124029586339806
1	6.0		0.23609960614360798
1	7.0		0.055953759631263686
1	8.0		0.4301569161309807

Figure 2

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.

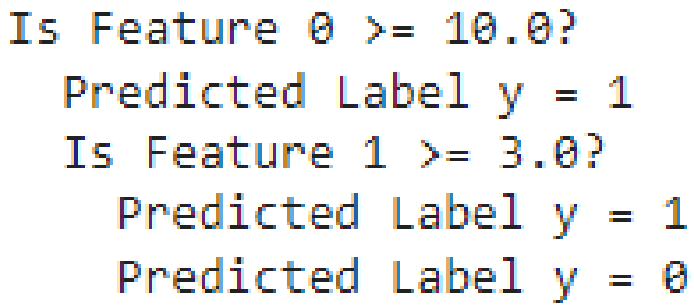

Figure 3

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.

[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.

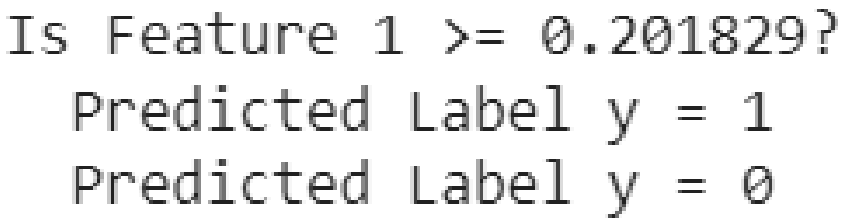

Figure 4

- 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. Ans: It is very clear that if feature 1 is greater than 0.2 then it is true else false.
- Build a decision tree on D2.txt. Show it to us.

```
Is Feature 0 >= 0.533076?
  Is Feature 1 >= 0.228007?
    Is Feature 1 >= 0.424906?
      Predicted Label y = 1
      Is Feature 0 >= 0.708127?
        Predicted Label y = 1
        Is Feature 1 >= 0.32625?
          Is Feature 0 >= 0.595471?
            Is Feature 0 >= 0.646007?
              Predicted Label y = 1
              Is Feature 1 >= 0.403494?
                Predicted Label y = 1
                Predicted Label y = 0
            Predicted Label y = 0
          Predicted Label y = 0
    Is Feature 0 >= 0.887224?
      Is Feature 1 >= 0.037708?
        Is Feature 1 >= 0.082895?
          Predicted Label y = 1
          Is Feature 0 >= 0.960783?
            Predicted Label y = 1
            Predicted Label y = 0
        Predicted Label y = 0
      Is Feature 0 >= 0.850316?
        Is Feature 1 >= 0.169053?
          Predicted Label y = 1
          Predicted Label y = 0
        Predicted Label y = 0
  Is Feature 1 >= 0.88635?
    Is Feature 0 >= 0.041245?
      Is Feature 0 >= 0.104043?
        Predicted Label y = 1
        Is Feature 0 >= 0.07642?
          Predicted Label y = 0
          Predicted Label y = 1
      Predicted Label y = 0
    Is Feature 1 >= 0.691474?
      Is Feature 0 >= 0.254049?
        Predicted Label y = 1
        Is Feature 0 >= 0.191915?
          Is Feature 1 >= 0.792752?
            Predicted Label y = 1
```


Figure 5

- Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
  Ans: It is very difficult to visualize the tree given the large depth of the tree.

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).
   - Build a decision tree on D2.txt. Show it to us.

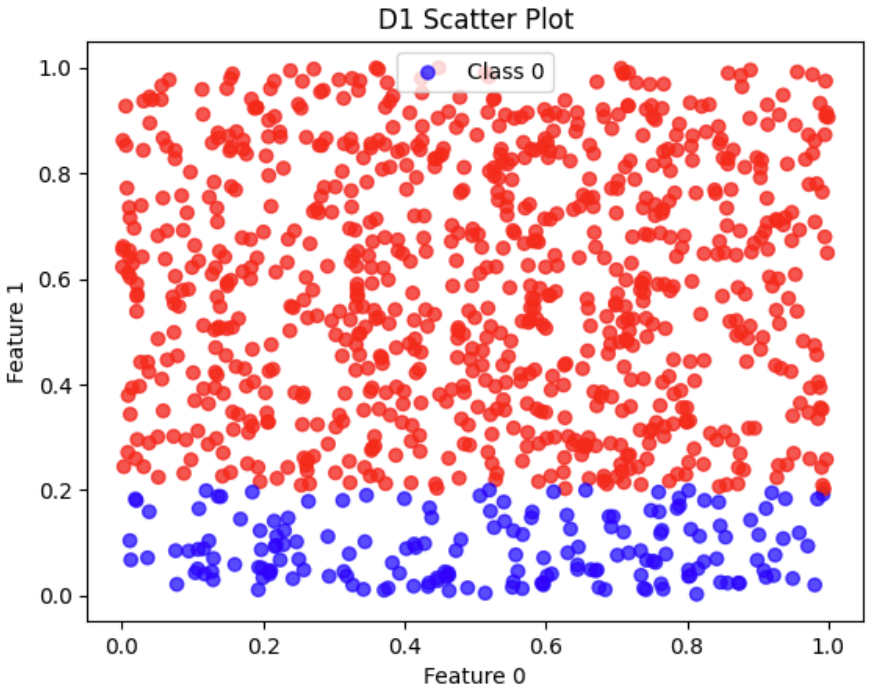

Figure 6

- Build a decision tree on D2.txt. Show it to us.

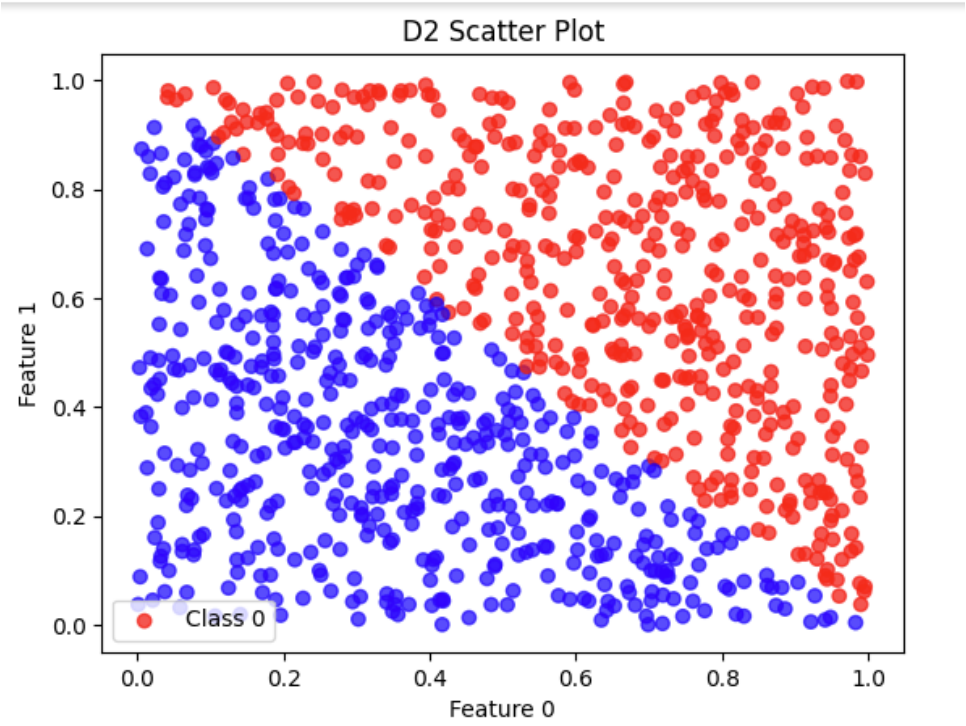

Figure 7

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, it is clear that for a single cut on Y axis, we can distinguish between both classes and decision tree is able to capture it. For D2, the decision tree is complex because it captures decision boundaries based on axis-aligned splits. This means it can make decisions based on individual features, but it struggles with diagonal decision boundaries like y=x.

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).

```
Training size: 32, n: 13, err_n: 0.11172566371681414
Training size: 128, n: 19, err_n: 0.06692477876106195
Training size: 512, n: 63, err_n: 0.04922566371681414
Training size: 2048, n: 129, err_n: 0.03816371681415931
Training size: 8192, n: 263, err_n: 0.020464601769911495
```

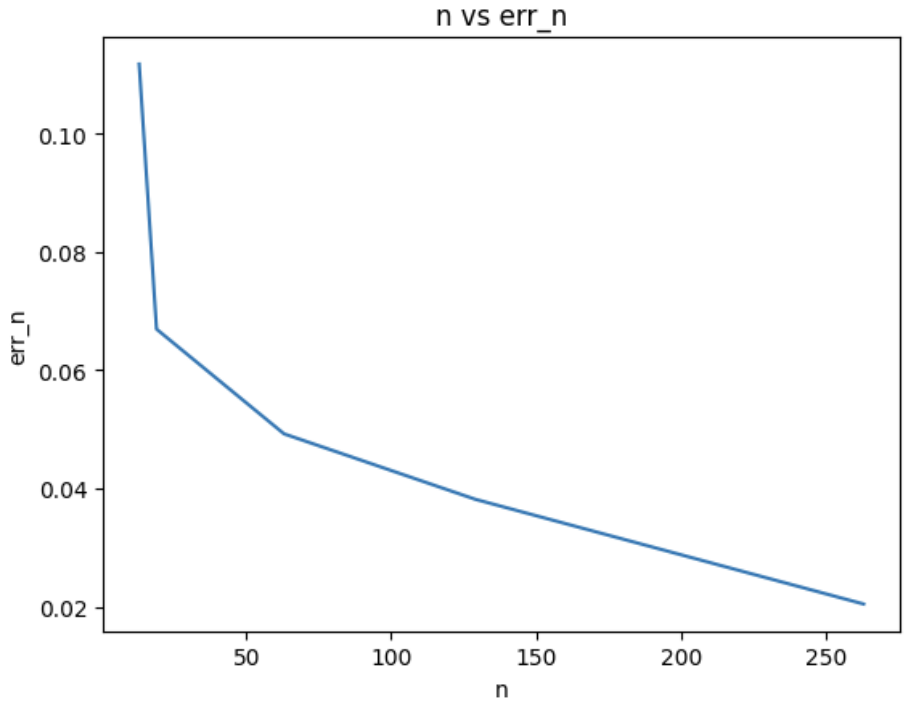

Figure 8

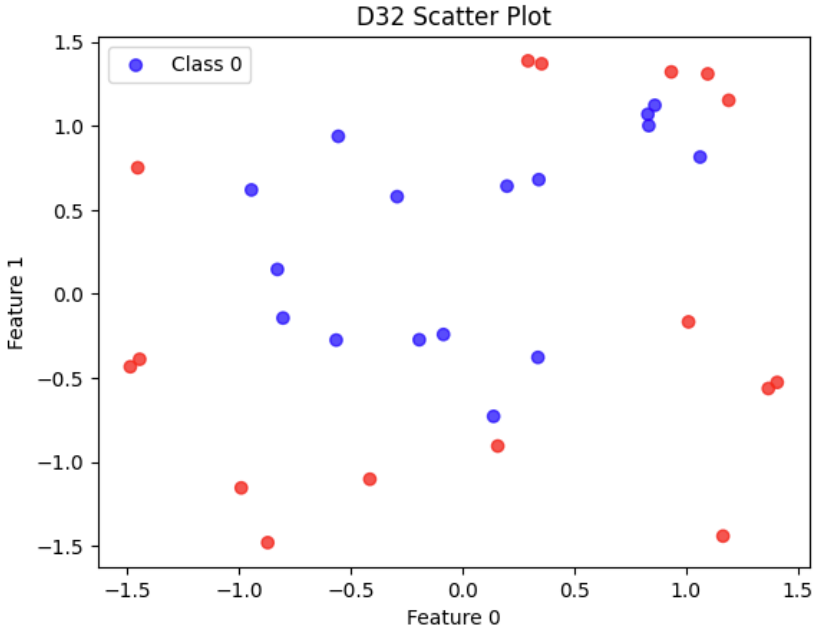

Figure 9

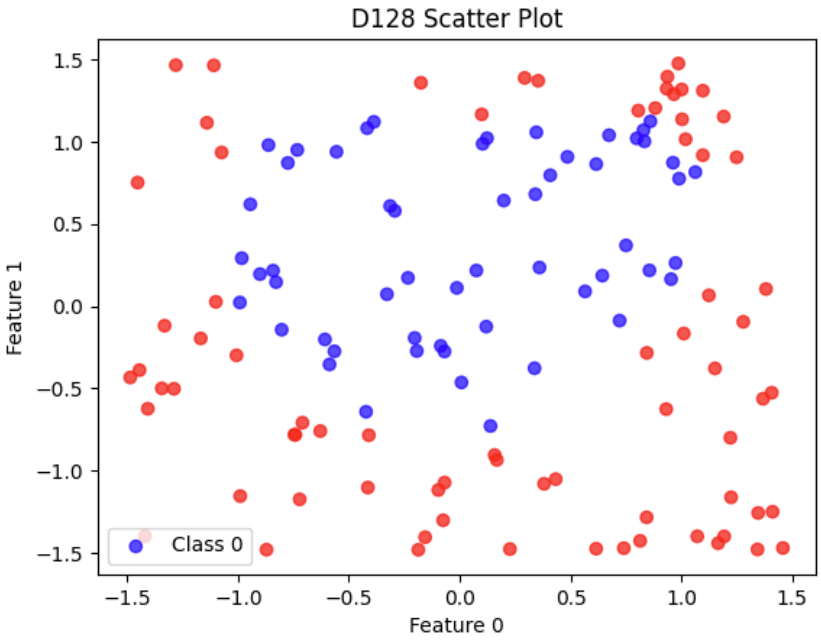

Figure 10

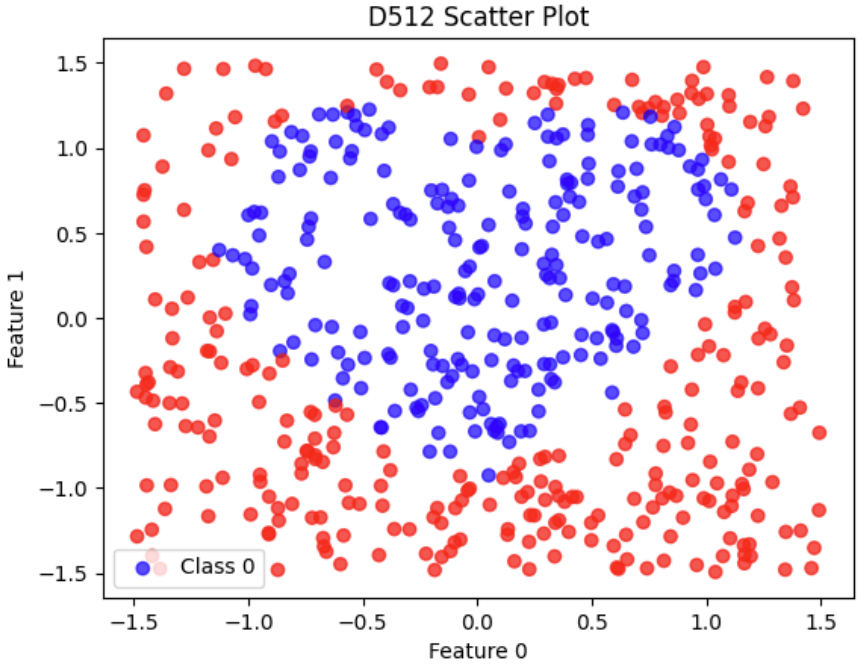

Figure 11

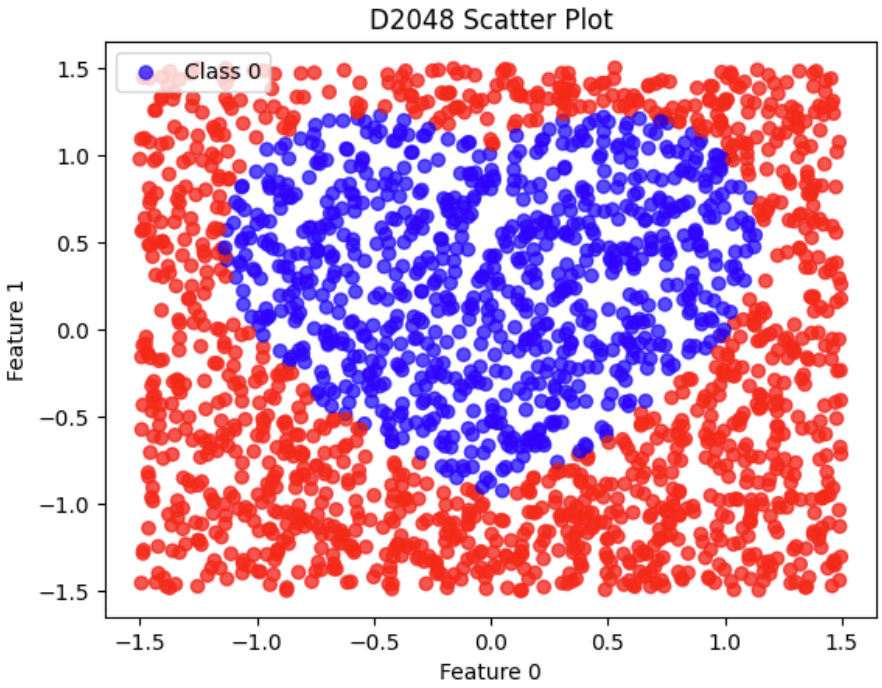

Figure 12

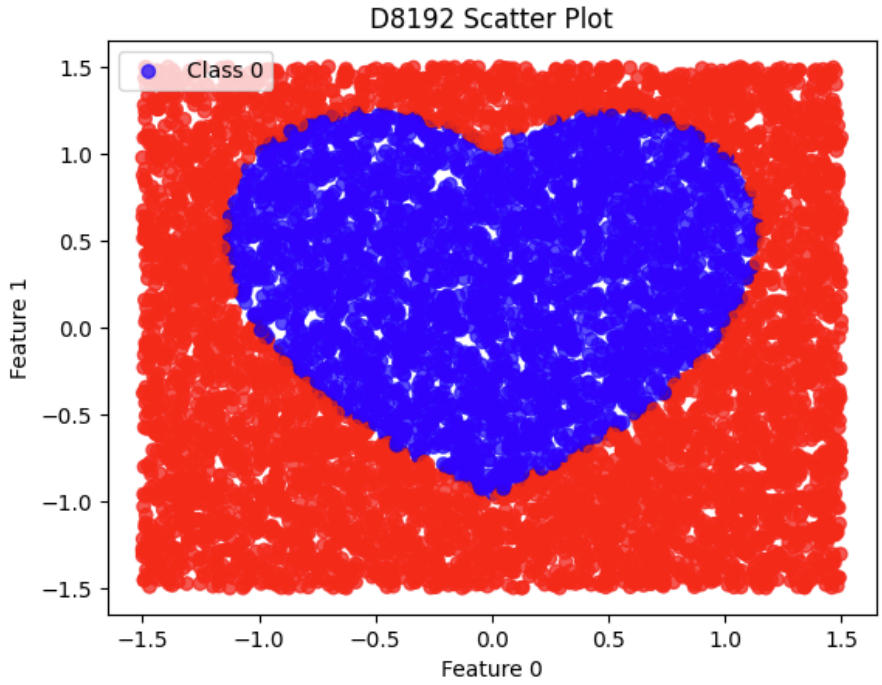

Figure 13

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.

```
Training size: 32, n: 15, err_n: 0.16869469026548672
Training size: 128, n: 27, err_n: 0.103429203539823
Training size: 512, n: 61, err_n: 0.05530973451327434
Training size: 2048, n: 123, err_n: 0.030973451327433628
Training size: 8192, n: 231, err_n: 0.018805309734513276
```

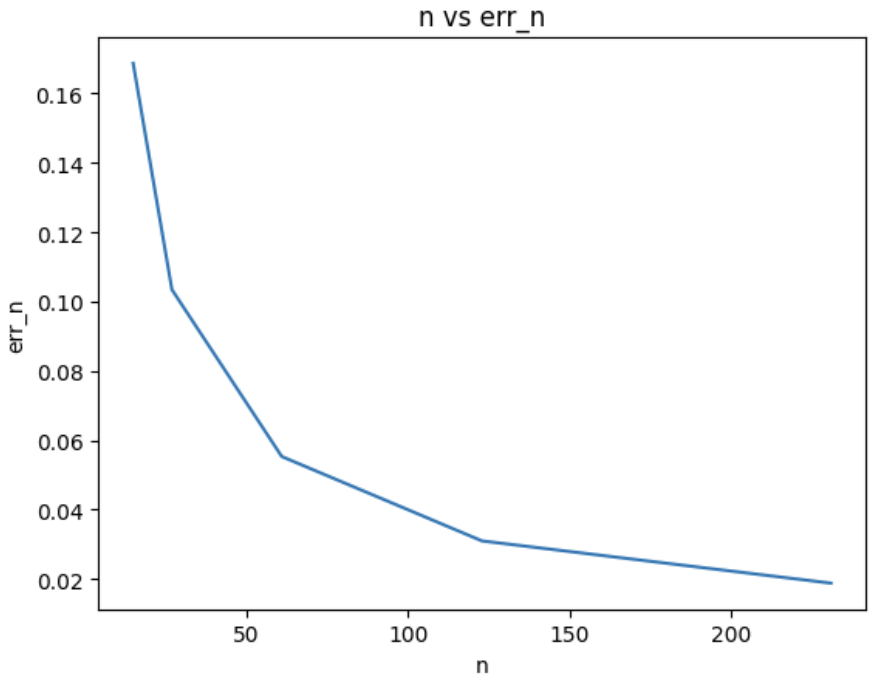

Figure 14

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.
Ans: The test error reduced as we added Gaussian noise to the training data. This is because when we use Lagrange interpolation with no noise the model fits extremely bad in the bounds (1e36 range) but when we add some noise of standard deviation 20 we observe that the error in the bounds has reduced (1e32 range) as we give more importance to the middle region of [-100,100]. When we increase the standard deviation further we observe test error to reduce because of the outlier behaviour in the bounds but it becomes bad in the central region as well.

```
Log Mean Train Error: 162.36357116902286
Log Mean Test Error: 161.91929421725322
```

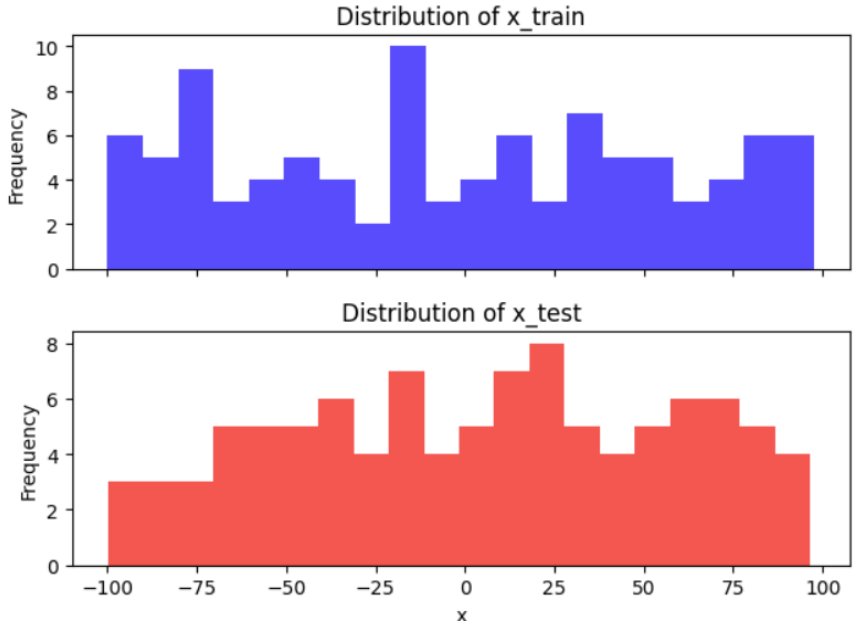

Figure 15

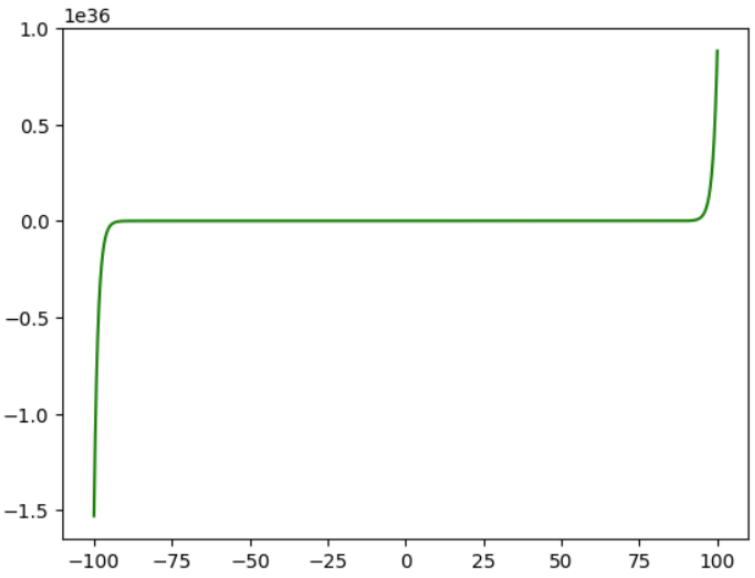

Figure 16

with standard deviation - 20

```
Log Mean Train Error: 178.60547049594857
Log Mean Test Error: 143.81409233566976
```

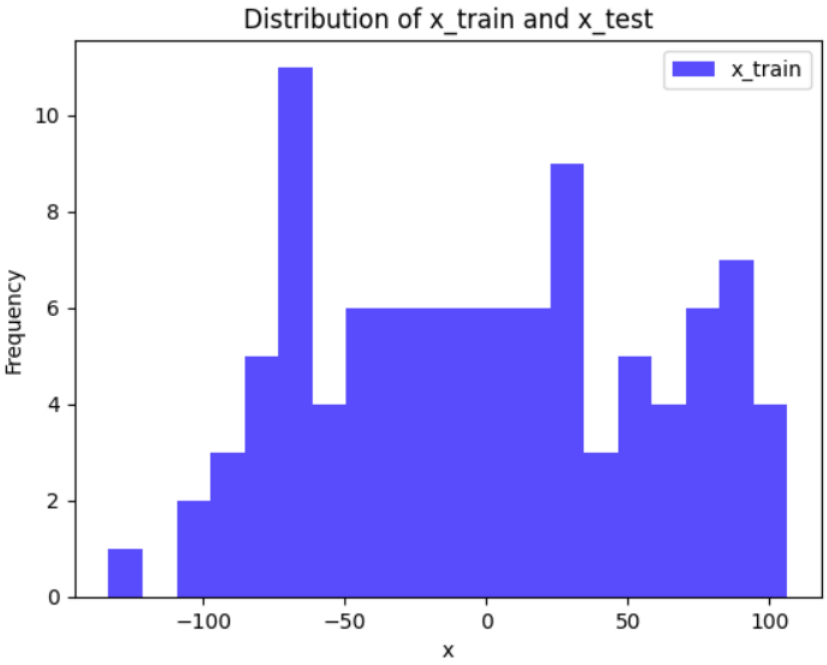

Figure 17

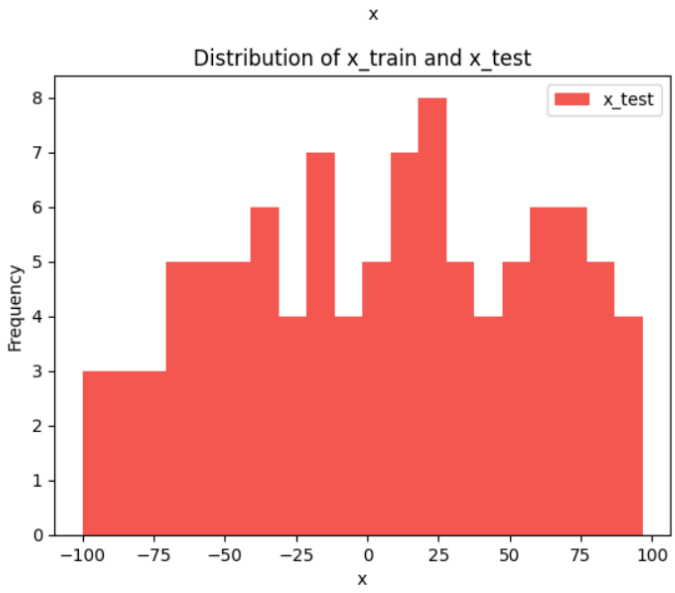

Figure 18

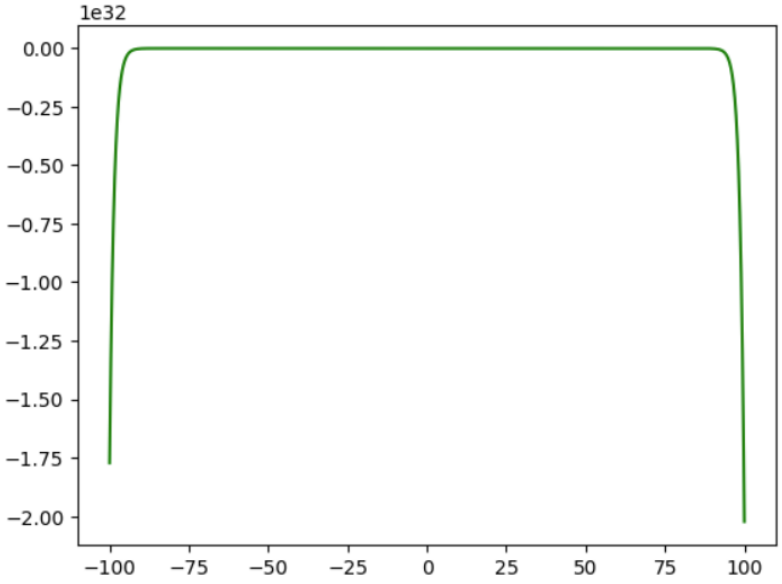
1e32
0.00
−0.25
−0.50
−0.75
−1.00
−1.25
−1.50
−1.75
−2.00
−100
−75
−50
−25
0
25
50
75
100

Figure 19