HOMEWORK 2

RISHIDEEP REDDY RALLABANDI
9084949099

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.
https://github.com/Rishideep08/CS760

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: If the probability is exactly 0 or 1 for all possible outcomes, then the entropy is indeed 0. In this scenario, there is no uncertainty, and the entropy is already at its lowest possible value of 0, indicating there's no point in further decreasing it. This basically means that the Information Gain is zero.

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.

   Ans:

   Table 1: Dataset Representation

x_1	x_2	y_1
0	0	1
1	1	1
0	1	0
1	0	0

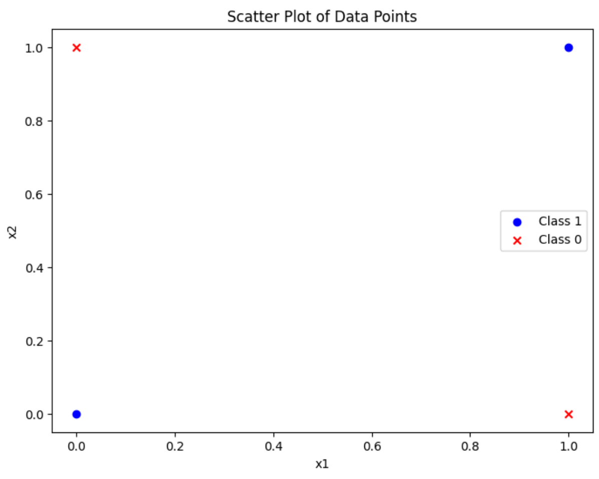


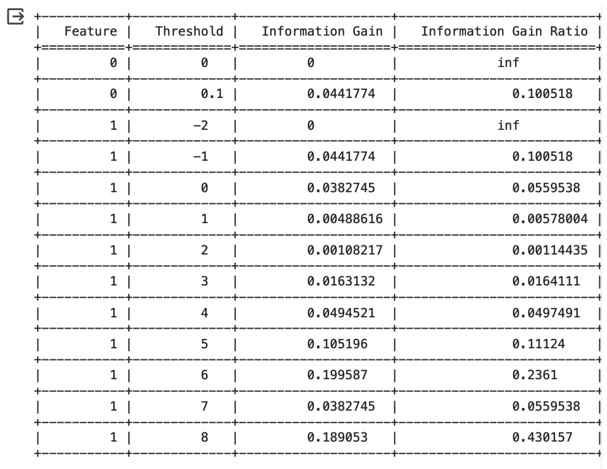
Figure 1: Handcraftdata

   In this dataset, all possible feature splits yield identical probabilities, resulting in equivalent entropy levels. However, a manual split based on the condition $\neg x_1 \oplus x_2$ can be applied to effectively separate the data.

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.

   Ans: We have calculated the information gain ratio. In the event of a candidate split resulting in zero, the information gain ratio is represented as "inf." For all other cases, the actual value is displayed. please refer to the infromation gain figure 2

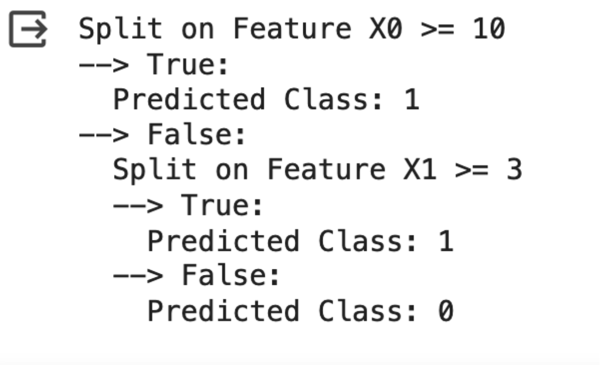
4. (The king of interpretability) [10 pts] Decision tree is not the most accurate classifier in general. However, it persists. This is largely due to its rumored interpretability: a data scientist can easily explain a tree to a



```
+-----------+-------------+--------------------+--------------------------+
|   Feature |   Threshold |   Information Gain |   Information Gain Ratio |
+===========+=============+====================+==========================+
|         0 |       0     |        0           |              inf         |
+-----------+-------------+--------------------+--------------------------+
|         0 |       0.1   |        0.0441774   |                0.100518  |
+-----------+-------------+--------------------+--------------------------+
|         1 |      -2     |        0           |              inf         |
+-----------+-------------+--------------------+--------------------------+
|         1 |      -1     |        0.0441774   |                0.100518  |
+-----------+-------------+--------------------+--------------------------+
|         1 |       0     |        0.0382745   |                0.0559538 |
+-----------+-------------+--------------------+--------------------------+
|         1 |       1     |        0.00488616  |                0.00578004|
+-----------+-------------+--------------------+--------------------------+
|         1 |       2     |        0.00108217  |                0.00114435|
+-----------+-------------+--------------------+--------------------------+
|         1 |       3     |        0.0163132   |                0.0164111 |
+-----------+-------------+--------------------+--------------------------+
|         1 |       4     |        0.0494521   |                0.0497491 |
+-----------+-------------+--------------------+--------------------------+
|         1 |       5     |        0.105196    |                0.11124   |
+-----------+-------------+--------------------+--------------------------+
|         1 |       6     |        0.199587    |                0.2361    |
+-----------+-------------+--------------------+--------------------------+
|         1 |       7     |        0.0382745   |                0.0559538 |
+-----------+-------------+--------------------+--------------------------+
|         1 |       8     |        0.189053    |                0.430157  |
+-----------+-------------+--------------------+--------------------------+
```

Figure 2: InformationGain

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.
Ans: please refer to the figure 3



```
Split on Feature X0 >= 10
--> True:
  Predicted Class: 1
--> False:
  Split on Feature X1 >= 3
  --> True:
    Predicted Class: 1
  --> False:
    Predicted Class: 0
```

Figure 3: D3Leaves

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.

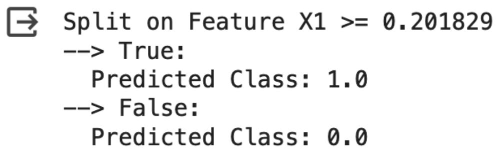
   - Build a decision tree on D1.txt. Show it to us in any format (e.g. could be a standard binary tree with nodes and arrows, and denote the rule at each leaf node; or as simple as plaintext output where each line represents a node with appropriate line number pointers to child nodes; whatever is convenient 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.
   - 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.
   - Build a decision tree on D2.txt. Show it to us.
   - Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?

Ans:

[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.

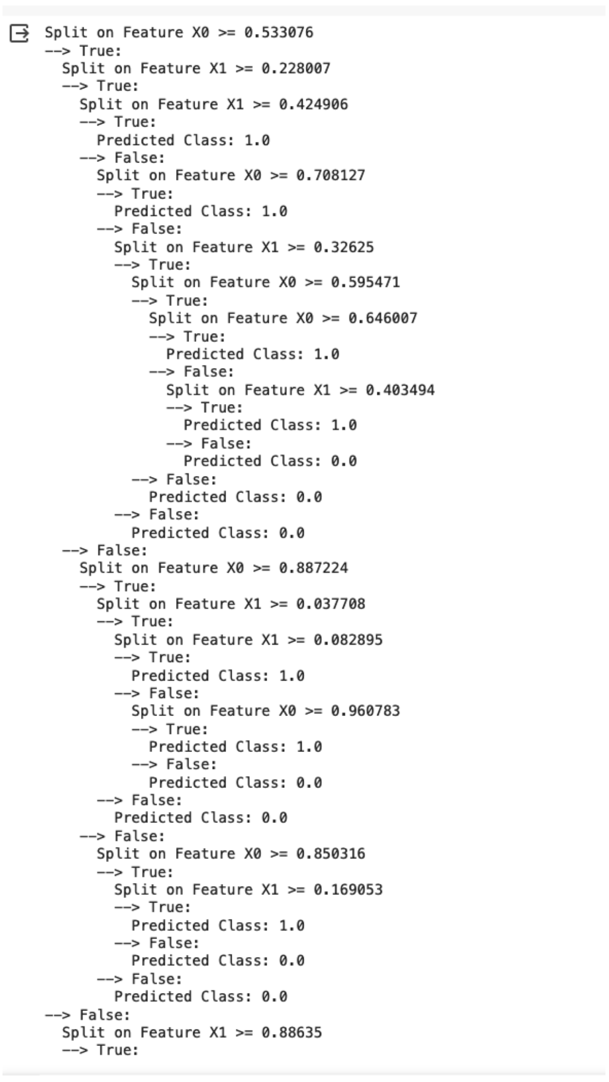
For D1 - please refer to the figure 4. If the first feature (X1) is greater than or equal to 0.201829 then the output class or label is 1 else it is 0.

For D2 - refer figure 5. Actual D2 is very big it is not able to fit in the single image. So, for D2 it is not possible to do without visualization.



```
Split on Feature X1 >= 0.201829
--> True:
  Predicted Class: 1.0
--> False:
  Predicted Class: 0.0
```

Figure 4: D1



```
Split on Feature X0 >= 0.533076
--> True:
  Split on Feature X1 >= 0.228007
  --> True:
    Split on Feature X1 >= 0.424906
    --> True:
      Predicted Class: 1.0
    --> False:
      Split on Feature X0 >= 0.708127
      --> True:
        Predicted Class: 1.0
      --> False:
        Split on Feature X1 >= 0.32625
        --> True:
          Split on Feature X0 >= 0.595471
          --> True:
            Split on Feature X0 >= 0.646007
            --> True:
              Predicted Class: 1.0
            --> False:
              Split on Feature X1 >= 0.403494
              --> True:
                Predicted Class: 1.0
              --> False:
                Predicted Class: 0.0
          --> False:
            Predicted Class: 0.0
        --> False:
          Predicted Class: 0.0
  --> False:
    Split on Feature X0 >= 0.887224
    --> True:
      Split on Feature X1 >= 0.037708
      --> True:
        Split on Feature X1 >= 0.082895
        --> True:
          Predicted Class: 1.0
        --> False:
          Split on Feature X0 >= 0.960783
          --> True:
            Predicted Class: 1.0
          --> False:
            Predicted Class: 0.0
      --> False:
        Predicted Class: 0.0
    --> False:
      Split on Feature X0 >= 0.850316
      --> True:
        Split on Feature X1 >= 0.169053
        --> True:
          Predicted Class: 1.0
        --> False:
          Predicted Class: 0.0
      --> False:
        Predicted Class: 0.0
--> False:
  Split on Feature X1 >= 0.88635
  --> True:
```

Figure 5: D2

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.

Ans: For D1 - please refer to the figure 6 For D2 - please refer to the figure 7

By inspecting the visualization, it can be inferred that for dataset $D1$, an effective separation is attained through a horizontal cut at the threshold of 0.208. However, in the case of $D2$, a diagonal cut is required, resulting in a more intricate decision boundary. The size of the decision tree for $D2$ is considerably larger, as decision trees typically favor vertical or horizontal splits. Accommodating a diagonal cut introduces multiple cases, contributing to the increased complexity of the tree.

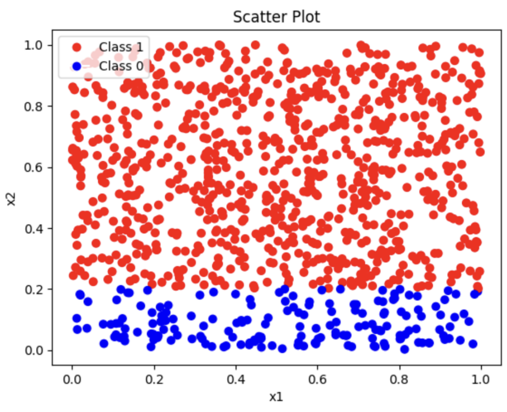


Figure 6: Decision-D1

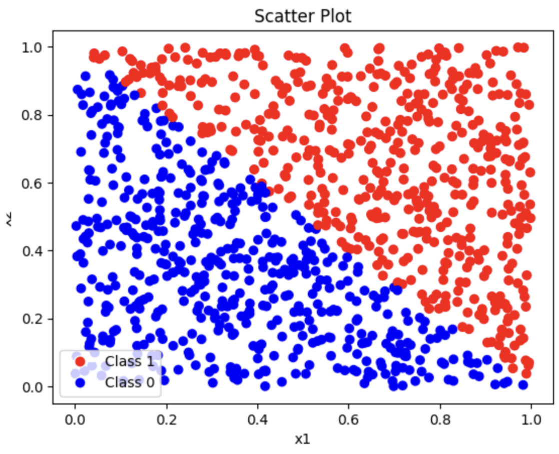


Figure 7: Decision-D2

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).

Ans:

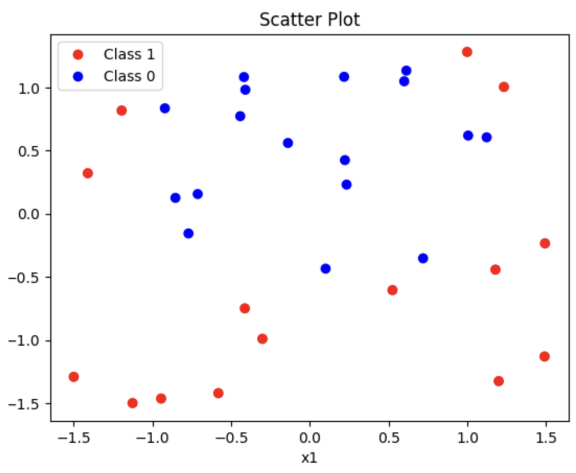


Figure 8: Decision-D32

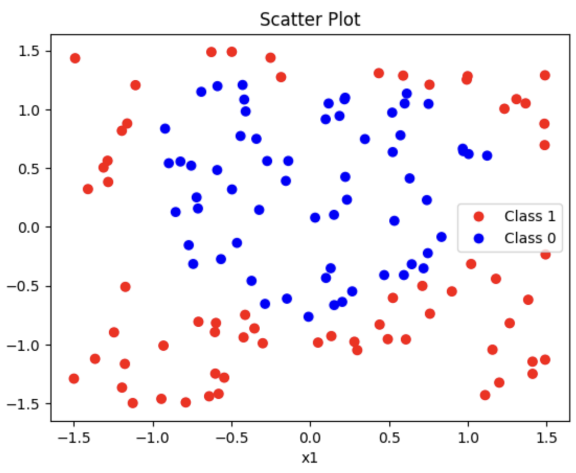


Figure 9: Decision-D128

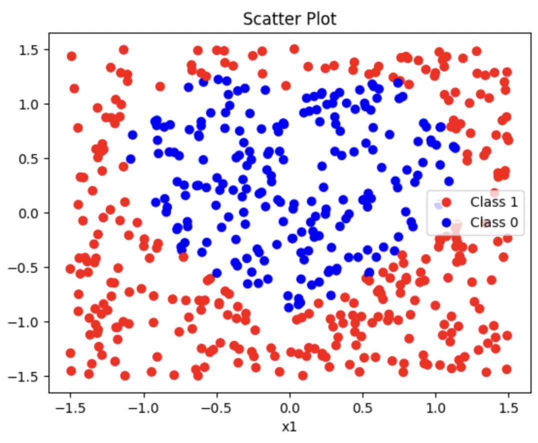


Figure 10: Decision-D512

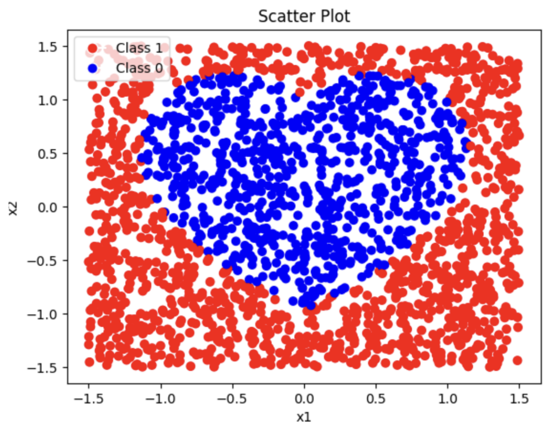


Figure 11: Decision-D2048

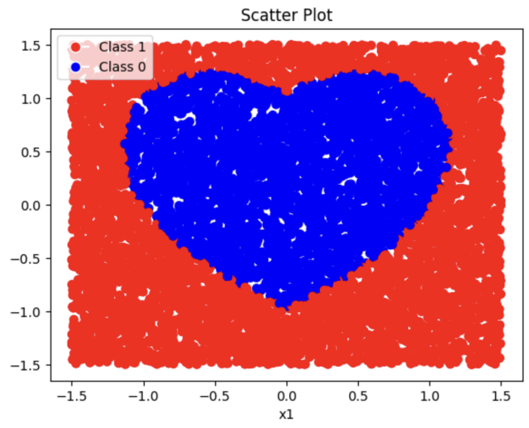


Figure 12: Decision-D8192

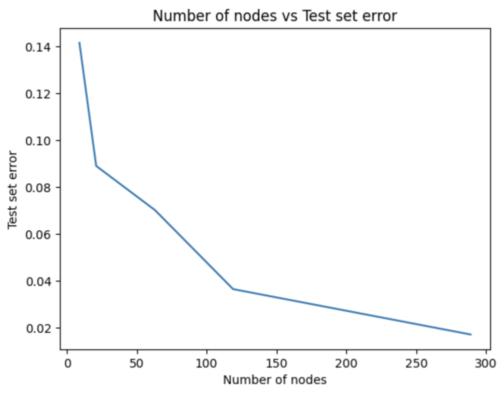


Figure 13: Nodes vs test error

Table 2: Tree count and Test Errors

Tree D	Number of Nodes	Test Set Error
32	9	0.1415929203539823
128	21	0.08904867256637172
512	63	0.07024336283185839
2048	119	0.036504424778761035
8192	289	0.017146017699115057

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.
Ans:

Table 3: Tree nodes and Test Errors using sklearn

Training Size	Number of Nodes	Test Set Error
32	9	0.10508849557522124
128	21	0.08628318584070796
512	51	0.06028761061946902
2048	95	0.024336283185840708
8192	235	0.010508849557522125

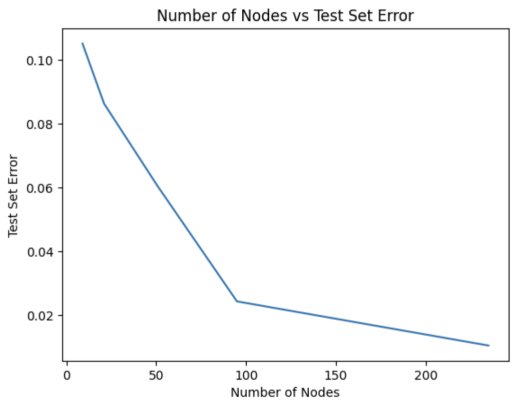


Figure 14: Nodes vs test error using sklearn

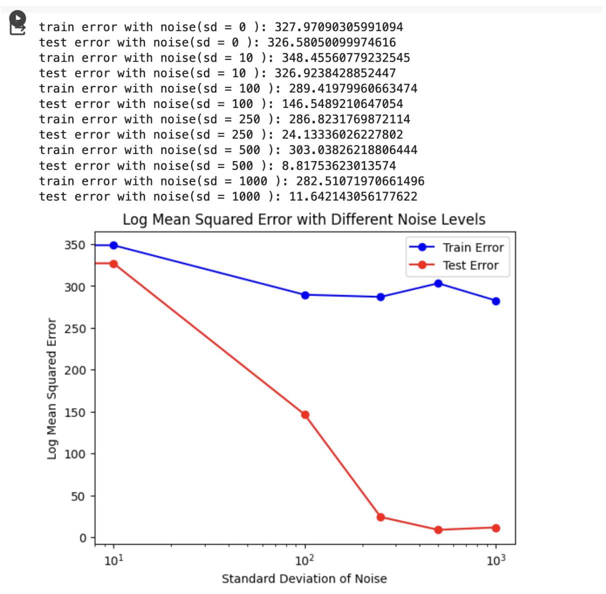
4 Lagrange Interpolation [10 pts]

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: Across various standard deviations, as illustrated in Figure 15, it is evident that when the standard deviation is zero, indicating the absence of noise, the test and train errors exhibit close similarity. However, with an increase

in the standard deviation, a noticeable trend emerges. Both the train and test errors demonstrate a decrease, yet the gap between them widens. Notably, the test error diminishes at a more pronounced rate compared to the train error.



```
train error with noise(sd = 0 ): 327.97090305991094
test error with noise(sd = 0 ): 326.58050099974616
train error with noise(sd = 10 ): 348.45560779232545
test error with noise(sd = 10 ): 326.9238428852447
train error with noise(sd = 100 ): 289.41979960663474
test error with noise(sd = 100 ): 146.5489210647054
train error with noise(sd = 250 ): 286.8231769872114
test error with noise(sd = 250 ): 24.13336026227802
train error with noise(sd = 500 ): 303.03826218806444
test error with noise(sd = 500 ): 8.81753623013574
train error with noise(sd = 1000 ): 282.51071970661496
test error with noise(sd = 1000 ): 11.642143056177622
```

Figure 15: Lagrange