HOMEWORK 2

Saikumar Yadugiri
saikumar@cs.wisc.edu, 9083079468

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.

GitHub Link: `https://github.com/saikumarysk/cs760_hw2`

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.

   As all the labels are the same, the entropy of Y is zero, i.e, $H_D(Y) = 0$. Moreover, no matter how we split, the value of Y will be same. That is, given the splitting information S, the uncertainity in Y still remains 0. So, $H_D(Y|S) = 0$. Hence, the information gain ratio is 0 for any split which is a stopping criterion in the design of our Decision Tree algorithm. So, a non-empty node with all training items having same label will 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 complete XOR ($\oplus$) data set, i.e, $Y = X_1 \oplus X_2$ for every possible X_1 and X_2 (Dxor.txt). Here, we can see that $H_D(Y) = 1$. And, for any possible split, the information gain will be zero. In other words, for any possible split S, $H_D(Y|S) = 1$. For instance, consider the split $X_1 \geq 1$. We have that $H_D(Y|X_1 \geq 1) = 1$ and $H_D(Y|X_1 < 1) = 1$. Hence, $H_D(Y|S) = \frac{1}{2} \times 1 + \frac{1}{2} \times 1 = 1$. Similarly all other possible splits give zero information gain. So, the decision tree algorithm refuses to split and sets the root node to T and halts. However if we manually force a split, every data point will be a leaf node on its own.

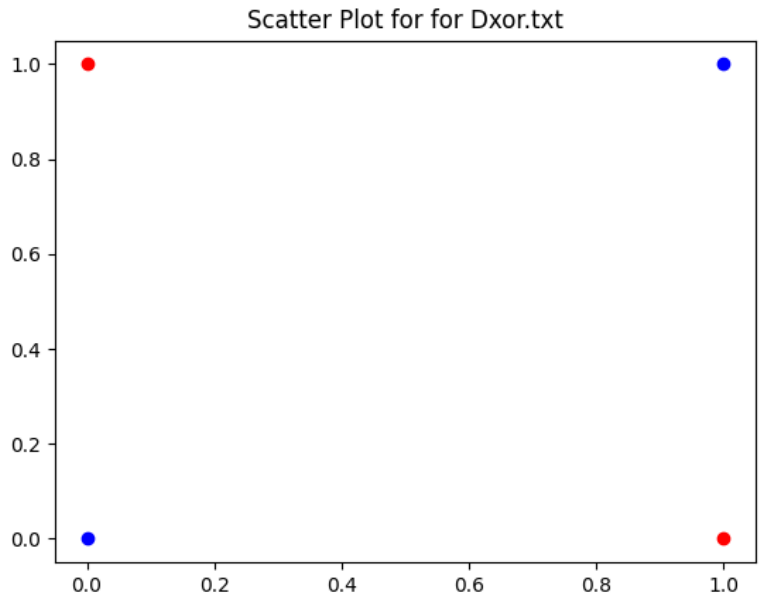


Figure 1: Scatter Plot for Dxor.txt

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.

   The required information can be found in table 1. My code doesn't explicitly provide these values. I had to use some debugging statements and workarounds to get these values.

Feature	Threshold (c)	$\mathbf{H_D(S)}$	Information Gain Ratio
X_1	0	0	0
X_1	0.1	0.4395	0.10052
X_2	-1	0.4395	0.10052
X_2	0	0.68404	0.05596
X_2	6	0.84535	0.2361
X_2	7	0.68404	0.05596
X_2	8	0.4395	0.43015

Table 1: Split Information for the root node in Druns.txt

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 resulted from my code is shown in figure 2

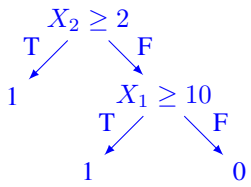


Figure 2: Decision Tree for D3leaves.txt

   It's logical formula is $(X_2 \geq 2) \vee (X_1 \geq 10)$ where X_i is the i-th feature.

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 resulted from my code is shown in figure 3

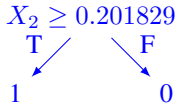
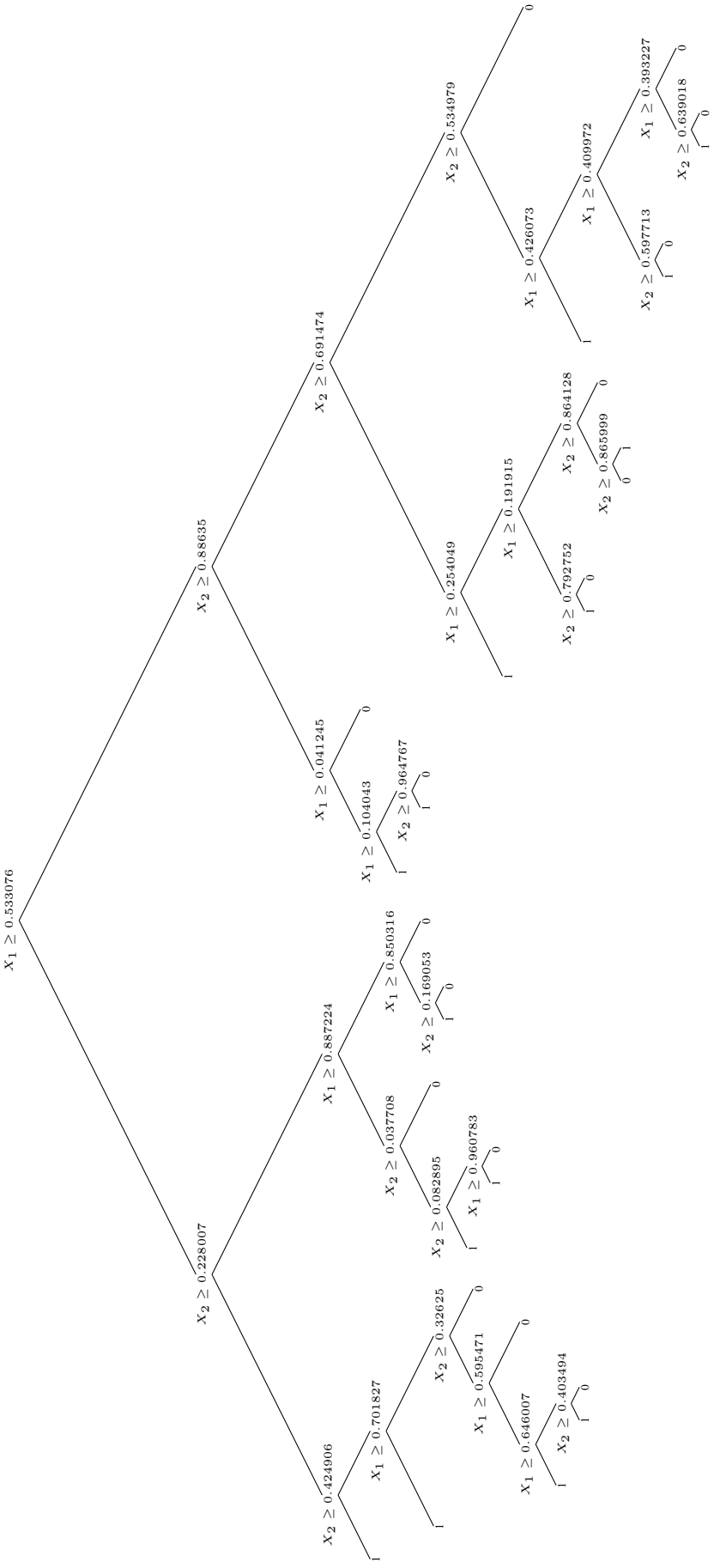


Figure 3: Decision Tree for D1.txt

   - 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 boundary will be a horizontal straight line in the (X_1, X_2) plane where the X_2-intercept will be 0.201829 and the X_1 intercept is ∞.
   - Build a decision tree on D2.txt. Show it to us.
     The decision tree resulted from my code is shown in the following tree (it's the horizontal page).

[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.

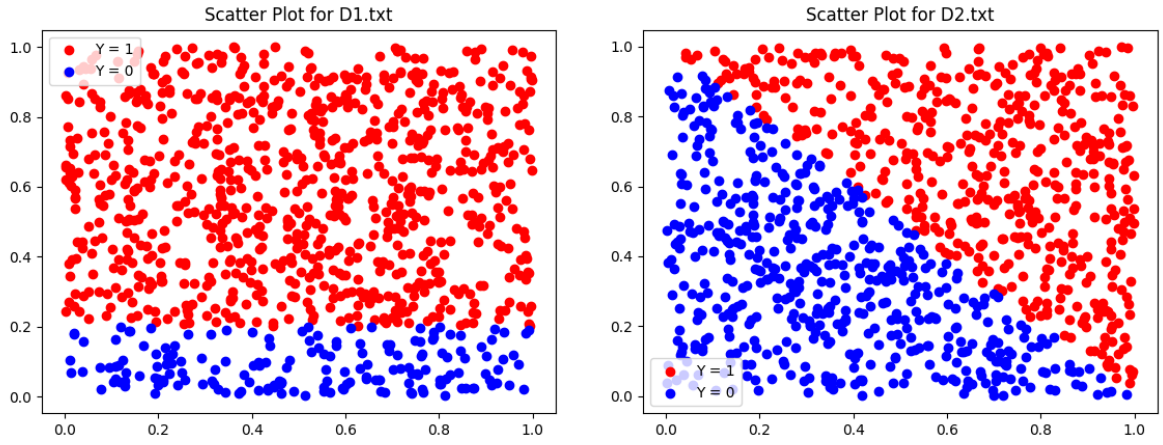

$x_1 \geq 0.533076$
$x_2 \geq 0.228007$
$x_2 \geq 0.88635$
$x_2 \geq 0.424906$
$x_1 \geq 0.701827$
$x_1 \geq 0.887224$
$x_1 \geq 0.041245$
$x_2 \geq 0.691474$
$x_1 \geq 0.254049$
$x_2 \geq 0.534979$
$x_2 \geq 0.32625$
$x_1 \geq 0.595471$
$x_2 \geq 0.037708$
$x_2 \geq 0.082895$
$x_1 \geq 0.960783$
$x_1 \geq 0.850316$
$x_2 \geq 0.169053$
$x_1 \geq 0.104043$
$x_2 \geq 0.964767$
$x_1 \geq 0.646007$
$x_2 \geq 0.403494$
$x_1 \geq 0.191915$
$x_2 \geq 0.792752$
$x_1 \geq 0.426073$
$x_1 \geq 0.409972$
$x_2 \geq 0.864128$
$x_2 \geq 0.865999$
$x_2 \geq 0.597713$
$x_1 \geq 0.393227$
$x_2 \geq 0.639018$

- Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
  We can interpret the decision tree as an SMT constraint but it is quite long. It is certainly hard without visualizing the tree. Something like this:

$$(X_1 \geq 0.533076 \wedge X_2 \geq 0.424906) \vee (0.228007 \leq X_2 < 0.424906 \wedge X_1 \geq 0.701827) \vee$$
$$(0.646007 \leq X_1 < 0.701827 \wedge 0.32625 \leq X_2 < 0.424906) \vee$$
$$(0.595471 \leq X_1 < 0.646007 \wedge 0.403494 \leq X_2 < 0.424906) \vee$$
$$(X_1 \geq 0.887224 \wedge 0.082895 \leq X_2 < 0.228007) \vee (X_1 \geq 0.960783 \wedge 0.037708 \leq X_2 < 0.082895) \vee$$
$$(0.850316 \leq X_1 < 0.887224 \wedge X_2 \geq 0.228007) \vee (0.104043 \leq X_1 < 0.533076 \wedge X_2 \geq 0.88635) \vee$$
$$(0.041245 \leq X_1 < 0.104043 \wedge X_2 \geq 0.964767) \vee$$
$$(0.254049 \leq X_1 < 0.533076 \wedge 0.691474 \leq X_2 < 0.88635) \vee$$
$$(0.191915 \leq X_1 < 0.254049 \wedge 0.792752 \leq X_2 < 0.88635) \vee$$
$$(X_1 < 0.191915 \wedge 0.864128 \leq X_2 < 0.865999) \vee (X_1 < 0.191915 \wedge 0.864128 \leq X_2 < 0.865999) \vee$$
$$(0.426073 \leq X_1 < 0.533076 \wedge 0.534979 \leq X_2 < 0.691474) \vee$$
$$(0.191915 \leq X_1 < 0.254049 \wedge 0.792752 \leq X_2 < 0.88635) \vee$$
$$(0.409977 \leq X_1 < 0.426073 \wedge 0.597713 \leq X_2 < 0.691474) \vee$$
$$(0.393227 \leq X_1 < 0.409972 \wedge 0.639018 \leq X_2 < 0.691474) \vee$$

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

   - Produce a scatter plot of the data set.
     The scatter plots are shown in figure 4



(a) D1.txt (b) D2.txt

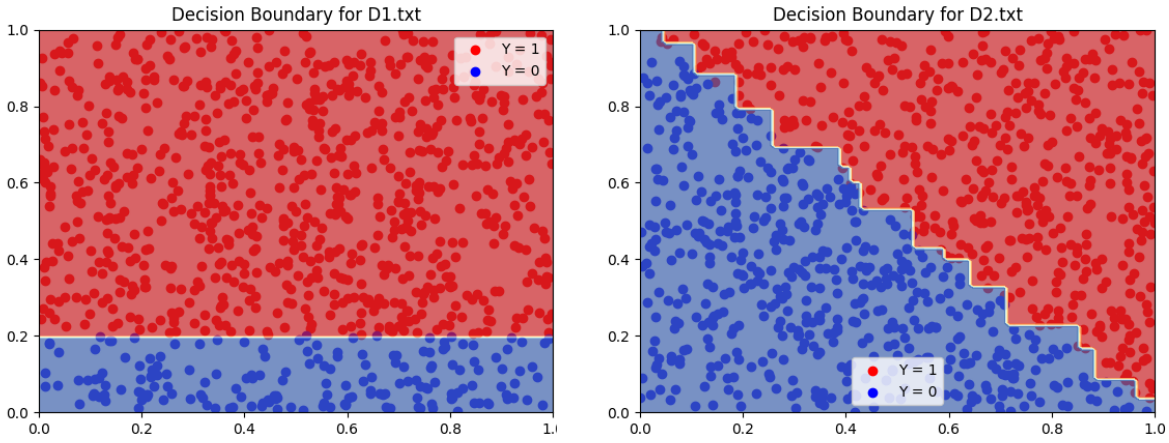
Figure 4: Scatter Plots for D1.txt and D2.txt

   - 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).
     The decision boundaries are shown in figure 5

   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 the data in D1.txt, the decision entirely depends on one of the feature, X_2. However, for D2.txt, we need to infer information from both X_1 and X_2 in a single node to predict accurately. That is, if we have a node whose split condition is like $X_1 + X_2 \geq 1$, then we can get a tree which looks like the decision tree for D1.txt.

   But, we are generating the decision tree by considering one feature per node, as dictated by the hypothesis bias of the decision tree learning method. Hence, D1.txt gives a shorter, accurate tree whereas D2.txt gives a complicated tree. The decision boundary for D2.txt's decision tree hypothesis will contain a lot of horizontal and vertical lines near the line $X_1 + X_2 = 1$. If we get more training instances near the said line,



(a) D1.txt (b) D2.txt

Figure 5: Approximate Decision Boundaries for D1.txt and D2.txt

we can be more accurately picture the decision boundary at the cost of a deeper decision tree and higher testing time.

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).

The nodes generated by my code and the err_n information is provided in the tabular form in table 2 and the plot is shown in 6. The decision boundaries are shown in figure 7.

n	# of nodes	$\mathbf{err_n}$
32	9	0.14767
128	17	0.10896
512	49	0.05862
2048	139	0.03207
8192	309	0.01493

Table 2: Number of nodes for each n

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.
The nodes generated by my code and the err_n information is provided in the tabular form in table 3 and the plot is shown in 8.

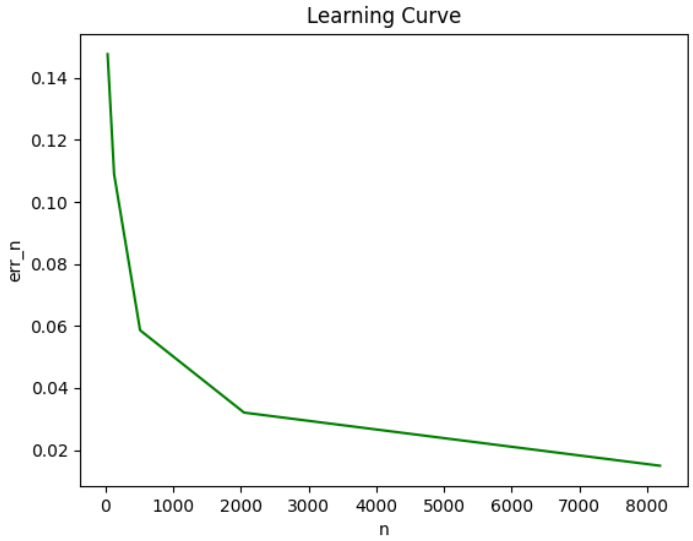


Figure 6: Learning Curve for Decision Tree by my code

n	# of nodes	$\mathbf{err_n}$
32	9	0.10674
128	15	0.11559
512	41	0.04480
2048	125	0.03097
8192	237	0.00940

Table 3: Number of nodes for each n for sklearn

Note: sklearn uses random selection. So, I ran each n multiple times and took the smallest error possible. I used sklearn version 1.3.0.

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.

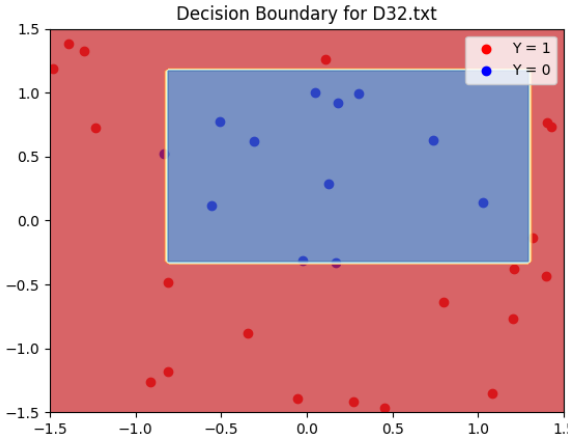
Generate a test set using the same distribution as your test set. Compute and report the resulting model's train and test error. What do you observe? Repeat the experiment with zero-mean Gaussian noise ϵ added to x. Vary the standard deviation for ϵ and report your findings.
For this question, I chose $n = 400$ samples from $[a, b] = [0, 4]$ as my test data. The data is present in Dsin.txt in the same format as data files from section 2. I built the Lagrange polynomial and my training error was 0.0. I used mean-squared error for calculating the error. This is because the training will fit a polynomial that goes through all of the (x, y) in the training data [2]. Thus all the points are accurately predicted by the Lagrange polynomial. Hence, training error will be 0. The testing error I got was $2.3991067263715886e + 40$ which is very high. This is because the polynomial might not go through all the points on the sine curve as the Taylor-series expansion of sine curve is an infinite polynomial.
In the next part of the experiment, I sampled x from an uniform distribution from $[0, 4]$ and the error from a zero-mean Gaussian distribution. I varied the standard deviation from $\sigma = 0.1 to \sigma = 3$ in irregular steps and set $y = \sin(x)$, where x implicitly includes the zero-mean Gaussian error (no clarification was provided on this aspect of the experiment. So, I chose to go with $y = \sin(x + \epsilon)$). The training error remained zero (as expected)

[2] The model doesn't require a training phase as you can directly test using the training values, similar to k-NN learning model

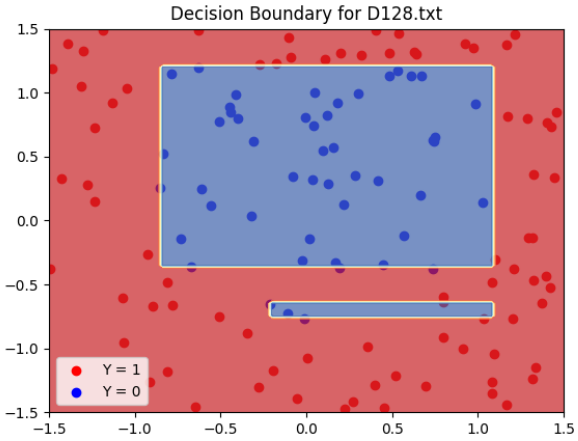
in all variations of σ. However, the testing error decreased dramatically as the data deviated from the sin-curve distribution and become less comprehensible. I used the same testing set as the previous experiment as mentioned in the The results are produced in the table 4.

σ	**Testing Error**
0.1	4.323348808761835e+34
0.2	6.464341563037188e+61
0.3	2.1122050373236405e+40
0.5	1.7255371955631576e+37
0.7	7.247605580034653e+51
0.9	6.160835860455152e+30
1	2.34853471287732e+30
1.5	3.947203987693241e+17
2	0.0006167642993747171
2.5	1.1984034812979894e-06
3	1.377963569529834e-05

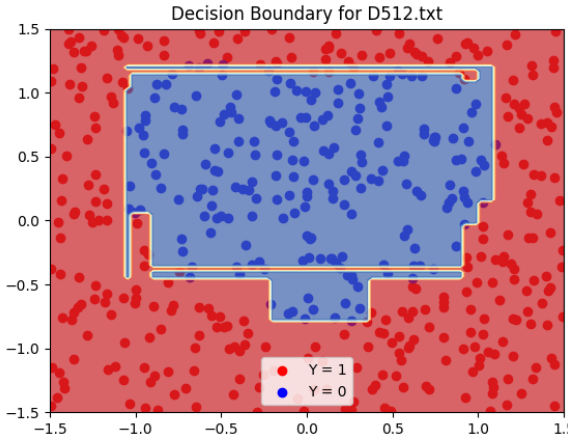
Table 4: Standard Deviation vs. Testing Error for Lagrange Interpolation of Noisy sin data



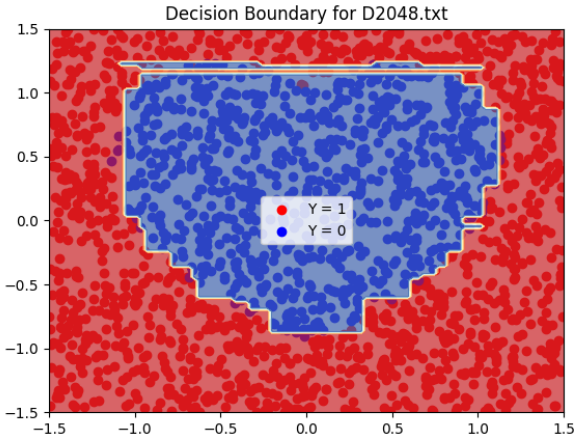
(a) D32.txt



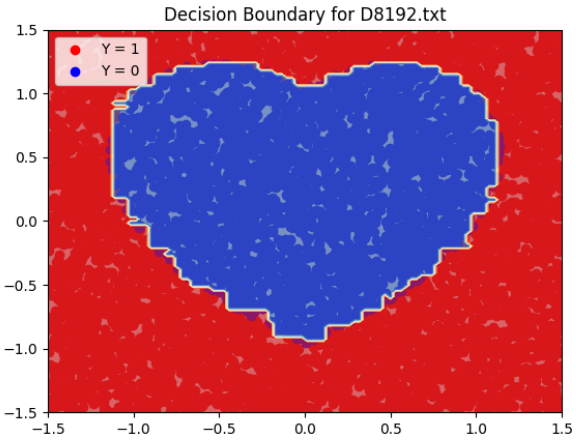
(b) D128.txt



(c) D512.txt



(d) D2048.txt



(e) D8192.txt

Figure 7: Approximate Decision Boundaries for D32.txt, D128.txt, D512.txt, D2048.txt, and D8192.txt

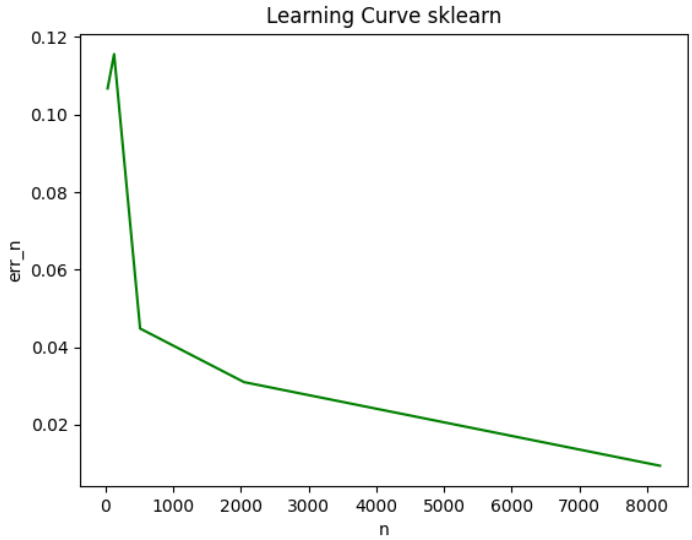


Figure 8: Learning Curve for Decision Tree by sklearn