HOMEWORK 2

>>Vishal V Naik<<
>>9084602235<<

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 : https://github.com/vvnaik2/ece760.git

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.

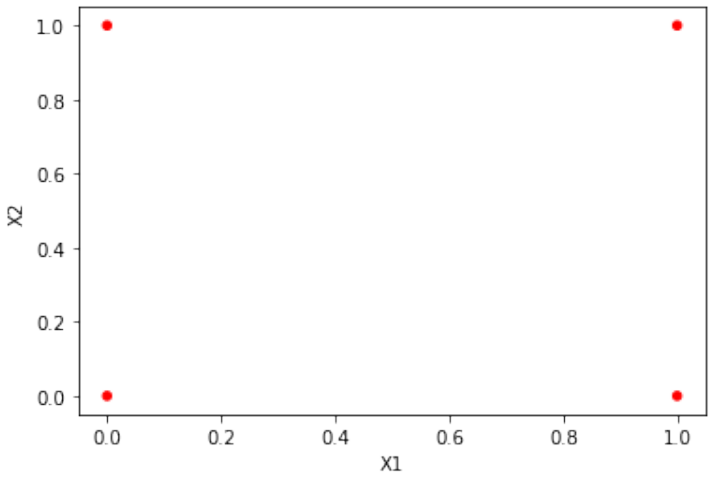
   Answer :
   The entropy of every candidate split for any feature is always equal to zero if all training items have the same class label, which implies that the probability of predicting the class label for any item in the training dataset is always 1. As a result, the node must be changed into a leaf in accordance with the stopping criteria for a decision tree building.
   On the other hand, if we force a split on any feature and keep going, every leaf in the sub-tree after that will always have the same class label. Since the parent node is now a leaf with the same class label, we can prune the subtree.

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.

   Answer :
   Lets take the example of XOR

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

   Here, any candidate split (0 or 1) has split entropy of 0 for both features X1 and X2. As a result, the building of the decision tree ends at the root node, turning the root into a leaf. The algorithm will continue to build a deeper tree with 3 levels and 100 percent training accuracy if we force a split on any of the features. For each possible combination of X1 and X2, there will be 4 boolean expressions. These pictures demonstrate this:



Let's say we force a split on X2 >=4 1. Further dataset splits are shown below:
Split 1:
X1 X2 Y
0 0 0
1 0 1
Split 2:
X1 X2 Y
0 1 1
1 1 0
Now, for 'Split 1', we have Y = X1 and for 'Split 2', we have Y = !X1, which can be used to split the tree further to form a complete binary tree with boolean expressions representing an XOR function.

```
   X1  X2  Y
0   1   0  1
1   0   1  1
2   1   1  0
3   0   0  0
Searching best split candidate for: X1
/home/vishaln/cs760/HW2/new.py:64: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  else_positive_count = len(else_dataset[dataset[CLASS_LABEL_COLUMN_NAME] == 1])
/home/vishaln/cs760/HW2/new.py:66: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  then_positive_count = len(then_dataset[dataset[CLASS_LABEL_COLUMN_NAME] == 1])
Best X1_split: 1, X1_entropy: 1.0, X1_gain_ratio: 0.0
Searching best split candidate for: X2
Best X2_split: 0, X2_entropy: 1.0, X2_gain_ratio: 0
X1 and X2 gain ratio is 0. Finding majority class and stopping
Level: 0
[Id: 1, label: 1, parent: root]
(base) [vishaln@royal-07] (45)$
```

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.

Answer :

```
Best X1_split: 0.1, X1_entropy: 0.8011735447551751, X1_gain_ratio: 0.10051807676021852
Best X2_split: 8, X2_entropy: 0.6562982680794203, X2_gain_ratio: 0.43015691613098095
X2_gain_ratio > X1_gain_ratio. Choosing X2 as split feature...
Best X1_split: 0.0, X1_entropy: 0.0, X1_gain_ratio: 0
Best X2_split: 8, X2_entropy: 0.0, X2_gain_ratio: 0
X1 and X2 gain ratio is 0. Finding majority class and stopping
Best X1_split: 0.1, X1_entropy: 0.6877840558577583, X1_gain_ratio: 0.07280247297910734
Best X2_split: 0, X2_entropy: 0.6348515545596771, X2_gain_ratio: 0.1206166388929235
X2_gain_ratio > X1_gain_ratio. Choosing X2 as split feature...
Best X1_split: 0.0, X1_entropy: 0.5435644431995964, X1_gain_ratio: 0
Best X2_split: 6, X2_entropy: 0.25, X2_gain_ratio: 0.36185425731195636
X2_gain_ratio > X1_gain_ratio. Choosing X2 as split feature...
Best X1_split: 0.0, X1_entropy: 1.0, X1_gain_ratio: 0
Best X2_split: 7, X2_entropy: 0.0, X2_gain_ratio: 1.0
X2_gain_ratio > X1_gain_ratio. Choosing X2 as split feature...
Best X1_split: 0.0, X1_entropy: 0.0, X1_gain_ratio: 0
Best X2_split: 7, X2_entropy: 0.0, X2_gain_ratio: 0
X1 and X2 gain ratio is 0. Finding majority class and stopping
Best X1_split: 0.0, X1_entropy: 0.0, X1_gain_ratio: 0
Best X2_split: 6, X2_entropy: 0.0, X2_gain_ratio: 0
X1 and X2 gain ratio is 0. Finding majority class and stopping
Best X1_split: 0.0, X1_entropy: 0.0, X1_gain_ratio: 0
Best X2_split: 0, X2_entropy: 0.0, X2_gain_ratio: 0
X1 and X2 gain ratio is 0. Finding majority class and stopping
Best X1_split: 0.1, X1_entropy: 0.0, X1_gain_ratio: 1.0
Best X2_split: -1, X2_entropy: 0.0, X2_gain_ratio: 1.0
X2_gain_ratio > X1_gain_ratio. Choosing X2 as split feature...
Best X1_split: 0.0, X1_entropy: 0.0, X1_gain_ratio: 0
Best X2_split: -1, X2_entropy: 0.0, X2_gain_ratio: 0
X1 and X2 gain ratio is 0. Finding majority class and stopping
Best X1_split: 0.1, X1_entropy: 0.0, X1_gain_ratio: 0
Best X2_split: -2, X2_entropy: 0.0, X2_gain_ratio: 0
X1 and X2 gain ratio is 0. Finding majority class and stopping
~
~
```

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.

   Answer:
   The below screenshot shows the decision tree constructed on D3leaves.txt. The tree is depicted in level order traversal format starting from level 0 till the deepest level. Each level has nodes, where each node has the following attributes:
   Id: 'Unique identifier' for every node
   Predicate: featurename $>=$ splitvalue' (the feature chosen to split the dataset at the node and the value on which the dataset is being split)
   Parent: 'root' if it is root, else 'parentid' followed by 'left' or 'right' indicating whether the node is the left or right child of the parent found using 'parentid'
   Label: 1 or 0, if node is leaf
   To obtain the logic rules for each leaf node, a depth first traversal starting from root is done storing all predicates. If the right branch is taken a ! (not) is added to the predicate indicating the else branch.
   The boolean expressions are:
   X2 $>= 2 -> 1$
   X2 < 2 and X1 $>= 10 -> 1$
   X2 < 2 and X1 $< 10 -> 0$

[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.

The tree levels are:

```
Level: 0
[Id: 1, predicate: X2 >= 2, parent: root]
Level: 1
[Id: 2, label: 1, parent: 1_left] [Id: 3, predicate: X1 >= 10, parent: 1_right]
Level: 2
[Id: 4, label: 1, parent: 3_left] [Id: 5, label: 0, parent: 3_right]
```

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.
   - 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?

   Answer :

   The level order traversal of the decision tree on D1.txt looks as shown below

```
Level: 0
[Id: 1, predicate: X2 >= 0.201829, parent: root]
Level: 1
[Id: 2, label: 1, parent: 1_left] [Id: 3, label: 0, parent: 1_right]
```

   Interpretation:
   If X2 >= 0.201829 for any data point, then the class label for such data points is 1, else it is 0.
   This is also represented by the logic rules of the leaves below:
   X2 >= 0.201829 == 1
   !(X2 >= 0.201829) == 0

   The level order traversal of the decision tree on D2.txt is as below.
   it is very hard to interpret this decision tree without visualizing the decision boundary of the dataset. The tree has 8 levels and around 60 nodes

```
Level: 0
[Id: 1, predicate: X1 >= 0.533076, parent: root]
Level: 1
[Id: 2, predicate: X2 >= 0.228007, parent: 1_left] [Id: 29, predicate: X2 >= 0.88635, parent:
1_right]
Level: 2
[Id: 3, predicate: X2 >= 0.424906, parent: 2_left] [Id: 16, predicate: X1 >= 0.887224, parent:
2_right] [Id: 30, predicate: X1 >= 0.041245, parent: 29_left] [Id: 37, predicate: X2 >= 0.691474,
parent: 29_right]
Level: 3
[Id: 4, label: 1, parent: 3_left] [Id: 5, predicate: X1 >= 0.708127, parent: 3_right] [Id: 17,
predicate: X2 >= 0.037708, parent: 16_left] [Id: 24, predicate: X1 >= 0.850316, parent: 16_right]
[Id: 31, predicate: X1 >= 0.104043, parent: 30_left] [Id: 36, label: 0, parent: 30_right] [Id:
38, predicate: X1 >= 0.254049, parent: 37_left] [Id: 49, predicate: X2 >= 0.534979, parent:
37_right]
Level: 4
[Id: 6, label: 1, parent: 5_left] [Id: 7, predicate: X2 >= 0.32625, parent: 5_right] [Id: 18,
predicate: X2 >= 0.082895, parent: 17_left] [Id: 23, label: 0, parent: 17_right] [Id: 25,
predicate: X2 >= 0.169053, parent: 24_left] [Id: 28, label: 0, parent: 24_right] [Id: 32, label:
1, parent: 31_left] [Id: 33, predicate: X2 >= 0.964767, parent: 31_right] [Id: 39, label: 1,
parent: 38_left] [Id: 40, predicate: X1 >= 0.191915, parent: 38_right] [Id: 50, predicate: X1 >=
0.426073, parent: 49_left] [Id: 61, label: 0, parent: 49_right]
Level: 5
[Id: 8, predicate: X1 >= 0.595471, parent: 7_left] [Id: 15, label: 0, parent: 7_right] [Id: 19,
label: 1, parent: 18_left] [Id: 20, predicate: X1 >= 0.960783, parent: 18_right] [Id: 26, label:
1, parent: 25_left] [Id: 27, label: 0, parent: 25_right] [Id: 34, label: 1, parent: 33_left] [Id:
35, label: 0, parent: 33_right] [Id: 41, predicate: X2 >= 0.792752, parent: 40_left] [Id: 44,
predicate: X2 >= 0.864128, parent: 40_right] [Id: 51, label: 1, parent: 50_left] [Id: 52,
predicate: X1 >= 0.409972, parent: 50_right]
Level: 6
[Id: 9, predicate: X1 >= 0.646007, parent: 8_left] [Id: 14, label: 0, parent: 8_right] [Id: 21,
label: 1, parent: 20_left] [Id: 22, label: 0, parent: 20_right] [Id: 42, label: 1, parent:
41_left] [Id: 43, label: 0, parent: 41_right] [Id: 45, predicate: X2 >= 0.865999, parent:
44_left] [Id: 48, label: 0, parent: 44_right] [Id: 53, predicate: X2 >= 0.597713, parent:
52_left] [Id: 56, predicate: X1 >= 0.393227, parent: 52_right]
Level: 7
[Id: 10, label: 1, parent: 9_left] [Id: 11, predicate: X2 >= 0.403494, parent: 9_right] [Id: 46,
label: 0, parent: 45_left] [Id: 47, label: 1, parent: 45_right] [Id: 54, label: 1, parent:
53_left] [Id: 55, label: 0, parent: 53_right] [Id: 57, predicate: X2 >= 0.639018, parent:
56_left] [Id: 60, label: 0, parent: 56_right]
Level: 8
[Id: 12, label: 1, parent: 11_left] [Id: 13, label: 0, parent: 11_right] [Id: 58, label: 1,
parent: 57_left] [Id: 59, label: 0, parent: 57_right]
```

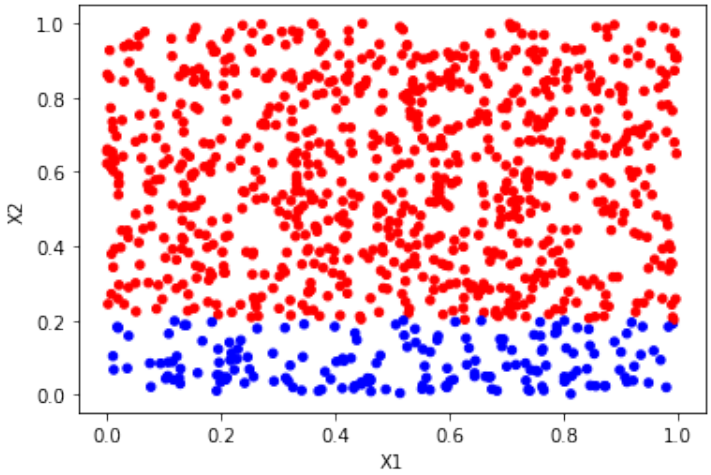
6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

   - 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).

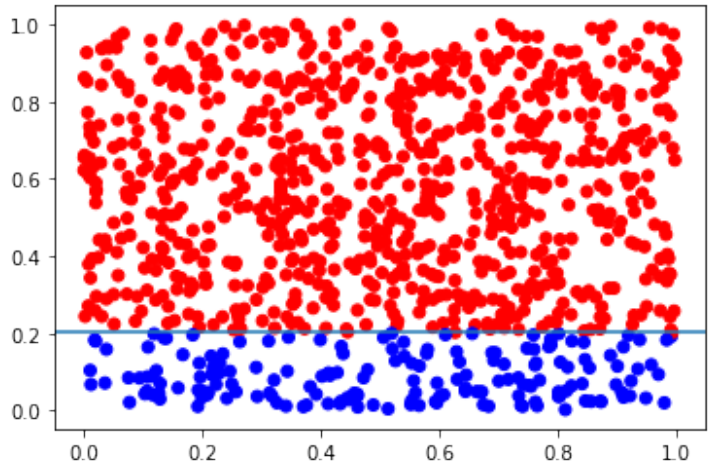
   Then discuss why the size of your decision trees on D1 and D2 differ. Relate this to the hypothesis space of our decision tree algorithm.

   Answer :

   D1 Scatter plot:

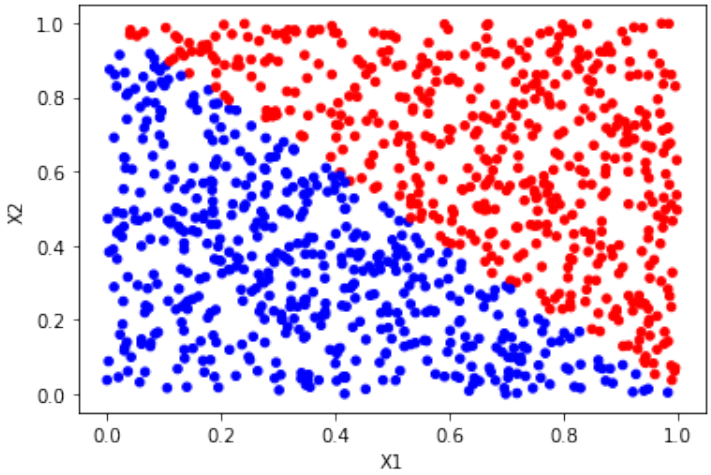


D1 decision boundary:

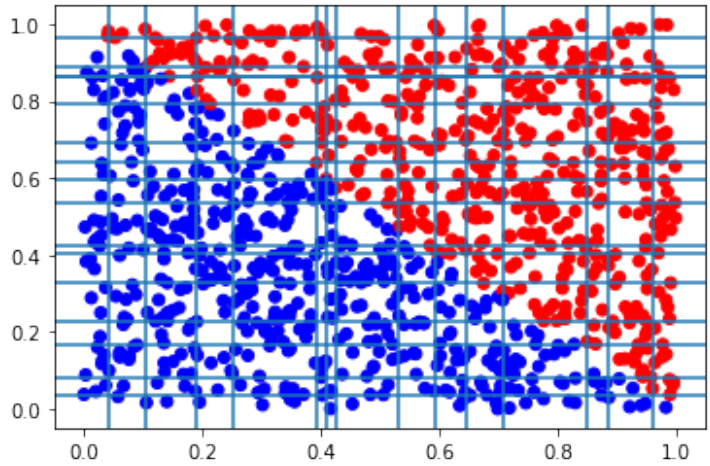


The decision boundary is a straight line represented by X2 = 0.201829. The data points above this line are classified with label 1 (shown as red) and the data points below this line are labeled 0 (shown as blue dots).

D2 Scatter plot:



D2 decision boundary:



Above is a scatter plot and decision boundary for the D2 decision tree. The decision boundary appears more like a step function. The step function cannot be written as a simple boolean expression of X1 or X2, which makes the D2 decision tree exceedingly large and difficult to grasp without the picture below (like it can be done for D1). As can be seen in the logic rules presented in 'Q5' for the D2 decision tree, it is a combination of many boolean expressions involving X1 and X2.
The hypothesis space of the decision tree algorithm comprises all possible trees of varying sizes. The best fit is the tree which maximizes classification accuracy which in this case is a very large decision tree. For D1, this is represented as a tree with just 2 levels, the first level containing the root node splitting on X2 $>=$ 0.201829 and the second level containing the left branch leaf (label: 1) and right branch leaf (label: 0)

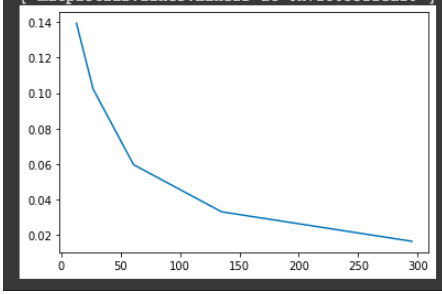
7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

   - 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).

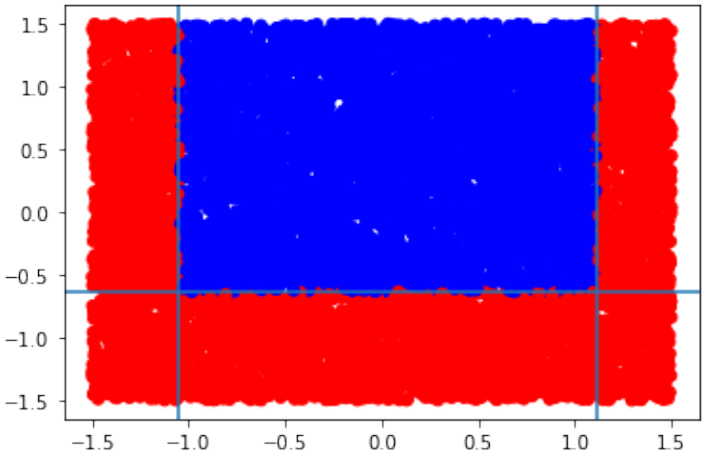
   Answer :

   Below is the screenshot of n and errn lists followed by the plot of n vs errn:

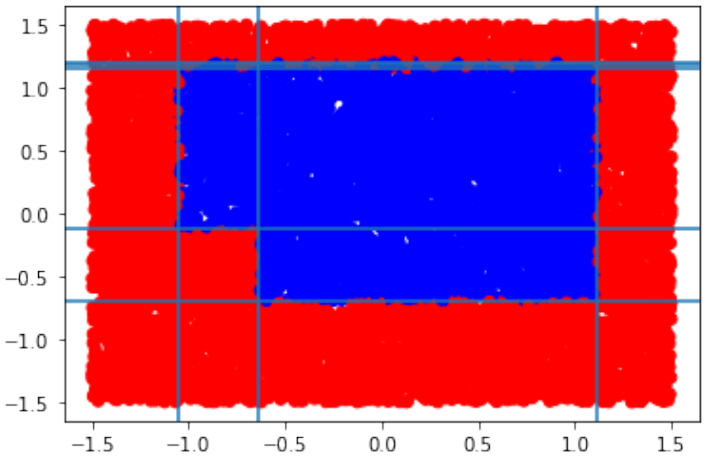
```
Printing n and errn lists respectively:
[13, 27, 27, 61, 135, 295]
[0.13938053097345132, 0.10232300884955753, 0.10232300884955753, 0.059734513274336286, 0.033185840707964605, 0.016592920353982302]
[<matplotlib.lines.Line2D at 0x7f8035ffeaf0>]
```



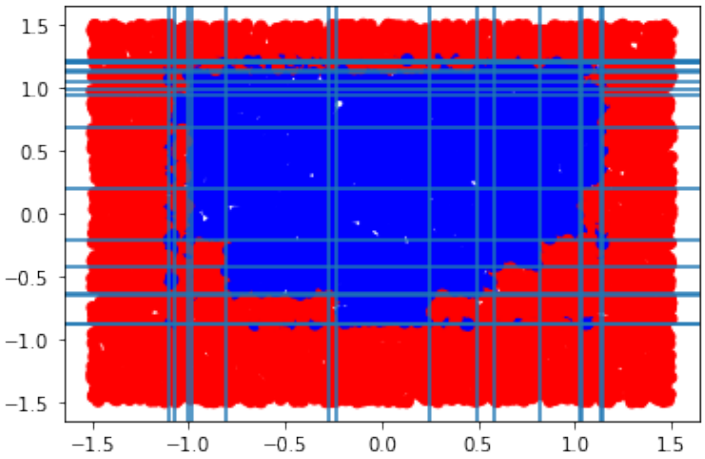
   D32 decision boundary:



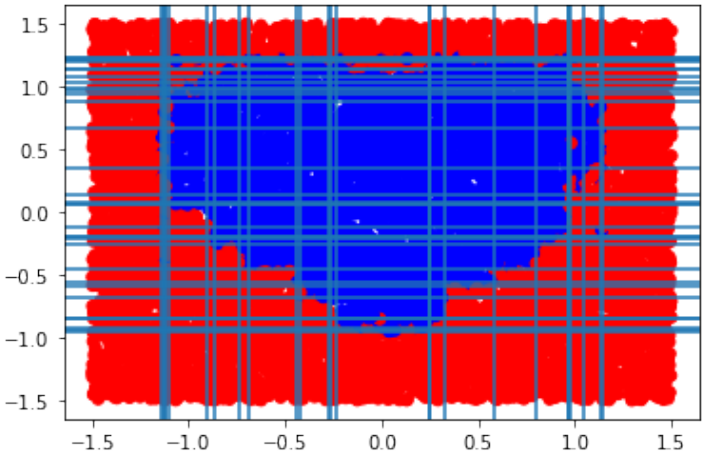
D128 decision boundary:



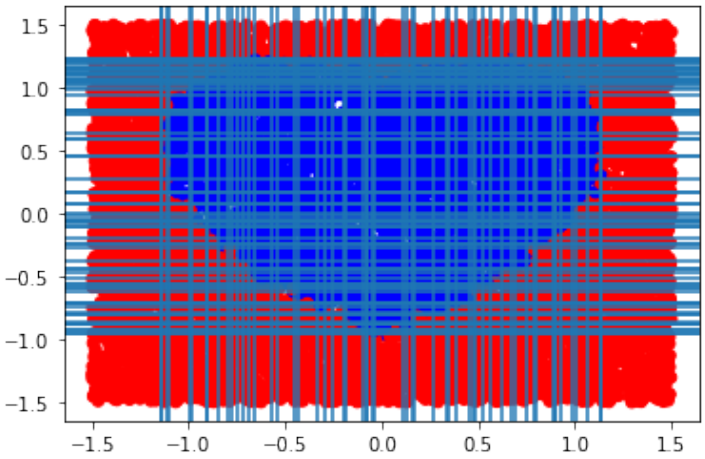
D512 decision boundary:



D2048 decision boundary:



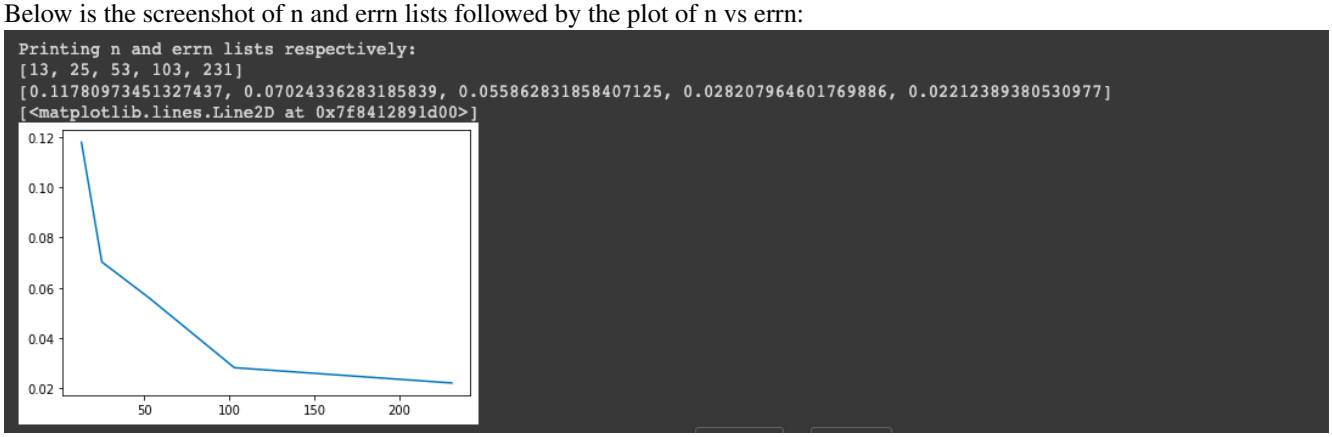
D8192 decision boundary:



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.
Answer :

Implemented using sklearn
Below is the screenshot of n and errn lists followed by the plot of n vs errn:



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.

Answer :

Uniform :
Train error: 1700806403111.0098 (0 if the point for which lagrange is calculated is included in the lagrange function)
Test error: 1100677.0316331435

Gaussian (sigma = pi/6) : Train error: 1.067385941921518e+58 (0 if the point for which lagrange is calculated is included in the lagrange function)
Test error: 1.1809409536427378e+54

Gaussian (sigma = pi/4) : Train error: 1.1861805350437908e+55 (0 if the point for which lagrange is calculated is included in the lagrange function)
Test error: 7.102691120150103e+46

Gaussian (sigma = pi/2) :
Train error: 1.3884450232724312e+47 (0 if the point for which lagrange is calculated is included in the lagrange function)
Test error: 1.191591564428518e+31

Observations :

The error magnitude, which tends to have very large numbers, seems to suggest that the lagrange interpolation does not suit the data distribution well (both for train and test set)

Additionally, it is noted that the lagrange interpolation appears to fit the uniform distribution better than the normal distribution as the error for the sample from the uniform distribution is smaller than the error for the samples from the normal distributions.

The last finding shows that the error lowers as the standard deviation of the normal distribution rises, suggesting that the lagrange interpolation appears to suit data with a higher spread or deviation better.