HOMEWORK 2

BATCHU S S S S HARISH BABU
sbatchu2(NetID) - 9084603043

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/bharishb/ECE760_spring2022

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

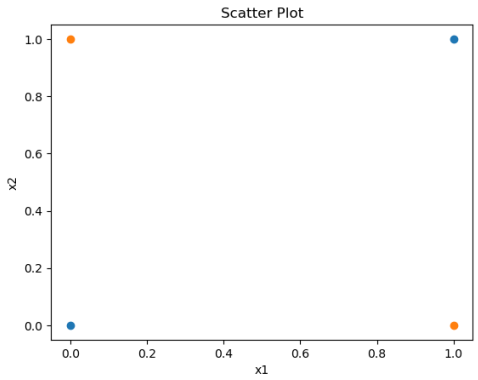
1. (Our algorithm stops at pure labels) [10 pts] If a node is not empty but contains training items with the same label, why is it guaranteed to become a leaf? Explain. You may assume that the feature values of these items are not all the same.
   If a non-empty node has training items with same label, then information gain,entropy on any candidate splits (H(Y)-H(Y/S)), H(Y), H(Y/S) are zero. So, there is no point of splitting. That's why it is considered to 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.
   If the training set is something like this, You will end up trying to split at points where gain ratios are all same which is zero. H(Y) = H(Y/S) since they will be equally distributed with either x1 greater than or equa1 to or x2 greather than or equal to 1.
   Training Set

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

   In textual form, we can force a split with something like this.
   if(x1=1):
   if(x2=1):
   (x1=1,x2=1) label
   else:
   (x1=1,x2=0) label
   else:
   if(x2=1):
   (x1=1,x2=1) label
   else: (x1=0,x2=0) label

   We can manually force a split by taking (X1 XNOR X2) as a single feature at the root and (X1 XNOR X2) as the feature in the next step, as X1 XOR X2 = (X1 XNOR X2) OR (X1 XNOR X2), and the branches of the decision tree are nothing but OR values



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: X1
Mutual Information: 0.0 If Split at: 0.0
Info Gain: 0.10051807676021852 If Split at: 0.1
Feature: X2
Mutual Information: 0.0 If Split at: -2
Gain Ratio: 0.10051807676021852 If Split at: -1
Gain Ratio: 0.055953759631263686 If Split at: 0
Gain Ratio: 0.005780042205152451 If Split at: 1
Gain Ratio: 0.0011443495172768668 If Split at: 2
Gain Ratio: 0.016411136842102245 If Split at: 3
Gain Ratio: 0.04974906418177866 If Split at: 4
Gain Ratio: 0.11124029586339812 If Split at: 5
Gain Ratio: 0.2360996061436081 If Split at: 6
Gain Ratio: 0.055953759631263686 If Split at: 7
Gain Ratio: 0.43015691613098095 If Split at: 8

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.

Logic Rule : x1, x2 are the Labels
if(x2 is greater than or equal to 2) assign Label 1 else
if(x1 is greater than or equal to 10) assign Label 0 else
assign Label 0

```
print2D(d.root)
```

```
                -> L 0
        -> 1 [10.]
                -> L 1
-> 2 [2.]
        -> L 1
```

```
Notation :
    Non Leaf Node : Feature_index_x1_or_x2 [split_thresh]
        Leaf Node : Label_0_or_1
```

(root , left node at bottom, right node at up)

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.

[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.

D1.txt Tree (root , left node at bottom, right node at up)

```
print2D(d.root)
```

```
        ->  L 0
->  2 [0.201829]
        ->  L 1
```

```
Notation :
    Non Leaf Node : Feature_index_x1_or_x2 [split_thresh]
        Leaf Node : Label_0_or_1
```

- 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.
  If X2 is greater than or equal to 0.201829, then y is most likely to be 1, otherwise it belongs to 0
- Build a decision tree on D2.txt. Show it to us.
  D2.txt Tree (root , left node at bottom, right node at up)

```
print2D(d.root)
```

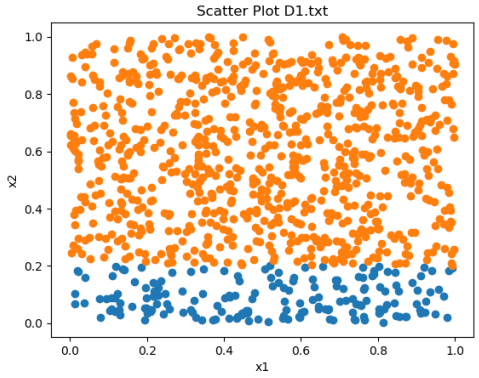
```
                                        ->  L 0
                    ->  2 [0.534079]
                                                ->  L 0
                                        ->  1 [0.393227]
                                                        ->  L 1
                                                ->  1 [0.39583]
                                                        ->  L 0
                                ->  1 [0.409972]
                                                ->  L 1
                                        ->  1 [0.417579]
                                                ->  L 0
                            ->  1 [0.426873]
                                    ->  L 1
                ->  2 [0.691474]
                                        ->  L 0
                                ->  2 [0.864128]
                                                ->  L 1
                                        ->  2 [0.865099]
                                                ->  L 0
                            ->  1 [0.191915]
                                        ->  L 0
                                ->  2 [0.792752]
                                        ->  L 1
                    ->  1 [0.254849]
                            ->  L 1
        ->  2 [0.88635]
                ->  L 0
            ->  1 [0.841245]
                                ->  L 1
                            ->  1 [0.87642]
                                    ->  L 0
                    ->  1 [0.104843]
                            ->  L 1
->  1 [0.533076]
                            ->  L 0
                    ->  1 [0.858316]
                                ->  L 0
                            ->  2 [0.169053]
                                    ->  L 1
                ->  1 [0.887224]
                            ->  L 0
                        ->  2 [0.037708]
                                                ->  L 0
                                        ->  1 [0.960783]
                                                ->  L 1
                                ->  2 [0.882895]
                                        ->  L 1
        ->  2 [0.228087]
                                        ->  L 0
                                ->  2 [0.32625]
                                        ->  L 0
                                    ->  1 [0.595471]
                                                            ->  L 0
                                                    ->  2 [0.403494]
                                                            ->  L 1
                                            ->  1 [0.646087]
                                                    ->  L 1
                    ->  1 [0.708127]
                            ->  L 1
                ->  2 [0.424986]
                        ->  L 1
```

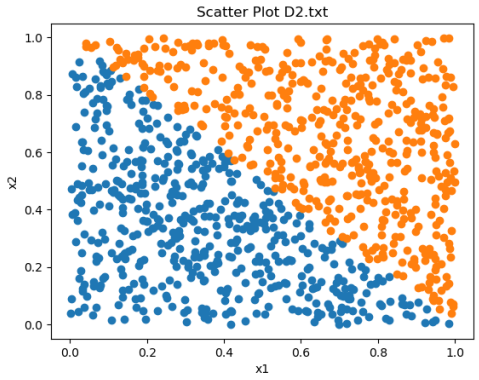
```
Notation :
    Non Leaf Node : Feature_index_x1_or_x2 [split_thresh]
        Leaf Node : Label_0_or_1
```

- Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
  For a big decision tree like this, initially it is somewhat possible to interpret. But once the control starts coming out of the first nested conditions, it becomes extremely difficult to imagine and keep track of the decision tree logic. Hence, visualization is needed for large and complex decision trees as number of features grows and more split conditions allowing for big trees

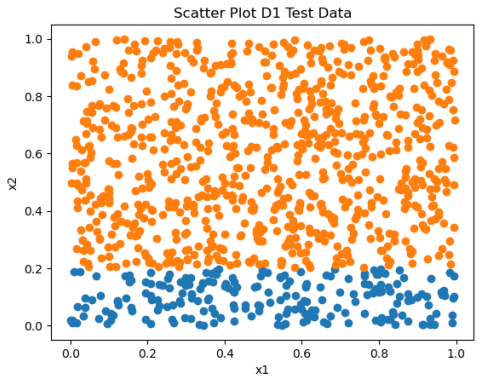
6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

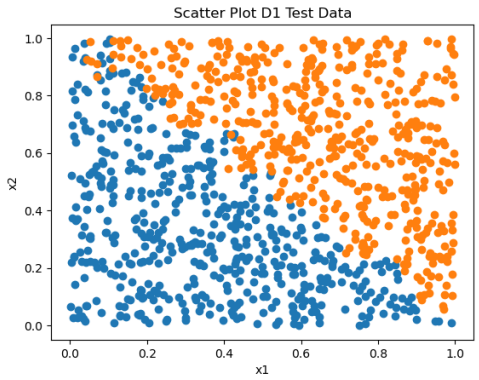
- Produce a scatter plot of the data set.
  TRAIN DATA





- 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).
  Test Data Generated from Uniform distribution between 0 1





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.
D1 x2 is clearly independent of x1 (x2=constant), while D2 x2 seems to be dependent on values of x1 (similar to x1+x2=1 boundary equation). Therefore the decision space is constant for D1, and is variable for

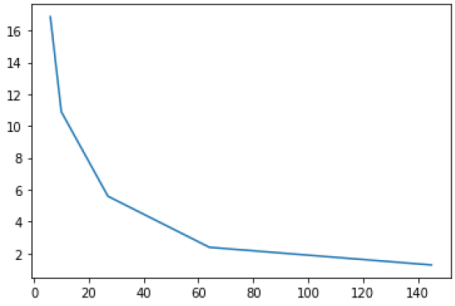
D2 with a boundary crossing both x and y axis. The decision hypothesis of D1 is not perfect as the nearby constant x2 could be discarded as irrelevant feature parameter. In D2, we need more feature variables for training as the Occams razor is followed automatically here because of the simple splitting based on the linear relationship between x1 x2.

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

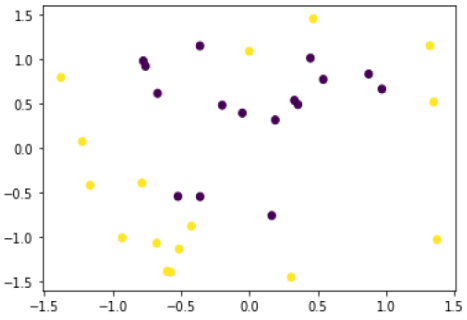
   - You will randomly split Dbig.txt into a candidate training set of 8192 items and a test set (the rest). Do this by generating a random permutation, and split at 8192.
     data$_shuffled = np.random.permutation(data)$

   - 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).

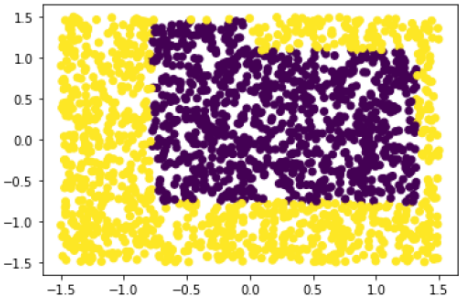
   D32, n=6, error: 16.87 percent
   D128, n=10, error: 10.9 percent
   D512, n=27, error: 5.59 percent
   D2048, n=64, error: 2.38 percent
   D8192, n=145, error: 1.27 percent



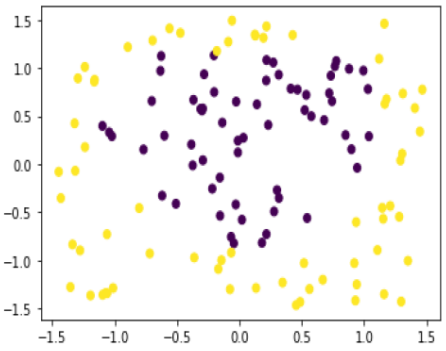
ERROR : Percent Error vs n



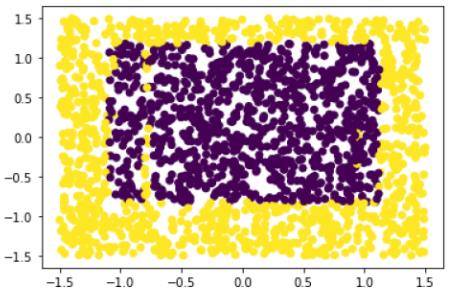
D32 TRAIN



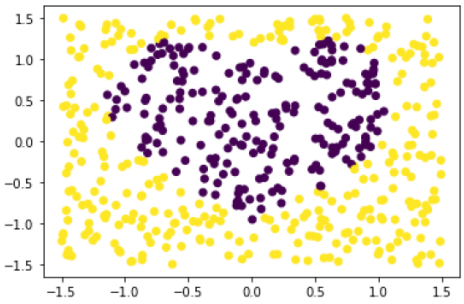
D32 TEST



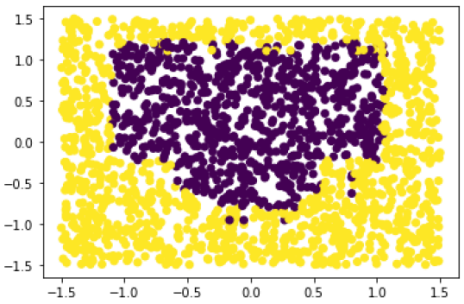
D128 TRAIN



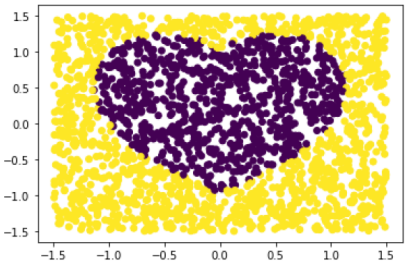
D128 TEST



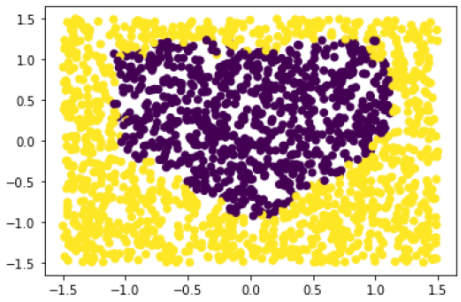
D512 TRAIN



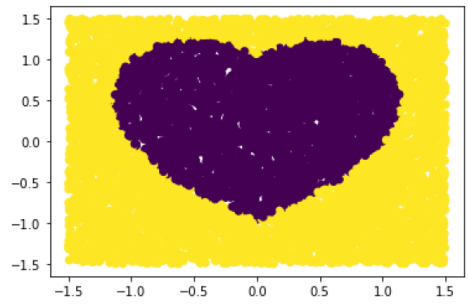
D512 TEST



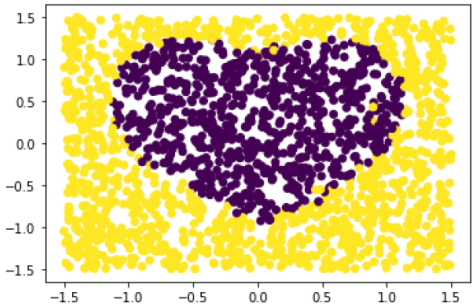
D2048 TRAIN



D2048 TEST



D8192 TRAIN

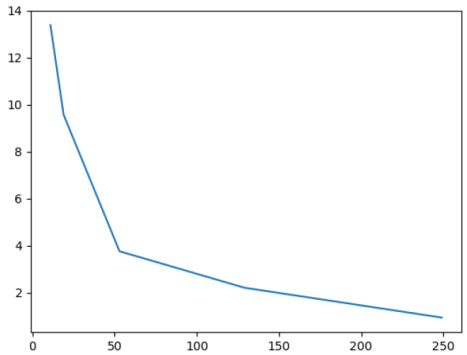


D8192 TEST

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.

32, n=11, error: 13.38 percent
128, n=19, error: 9.57 percent
512, n=53, error: 3.76 percent
2048, n=129, error: 2.21 percent
8192, n=249, error: 0.94 percent



Percent Error vs n

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.
Took 100 points from $[a, b] = [-2 * pi, 2 * pi]$
When tried to build a lagrange polynomial with degree 99, both training error and test error coming out to be huge.
Mean square Error is used as an error metric
$poly = lagrange(x, y)$
$y_predict = poly(x)$
$mse = ((np.square(y - y_predict))).mean(axis = 0)$
$TrainingError, mse = 1.0416124531011865e + 60$
When chosen a polynomial of degree 19, training and test are fitting well.
$mse = 5.803243952774228e - 12$ (Training Error)
$mse = 2.4288089242683953e - 12$ (Test Error with another data set of same distribution)
So, polynomial of degree 19 is fitting well with regularization.
When added Gaussian Noise with standard deviation e:
$x_e = np.random.normal(0, e, 100)$
$x + x_e$ is the new input to Lagrange
Below are the test errors with various polynomial degrees, different standard deviation e

e	polyDegree19	polyDegree49	polyDegree99
0.1	0.00643848	4.17539596e+13	4.28512475e+60
0.15	0.01001333	1.63658781e+13	2.18797297e+59
0.20	0.02471265	8.04466532e+13	2.54584615e+6
0.25	0.02633184	3.35663836e+14	5.30191230e+61