HOMEWORK 2

DevMehrotra
9083593443

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.

   When a node only has training items with the same label one of the two things can happen

   1) Either we stop splitting

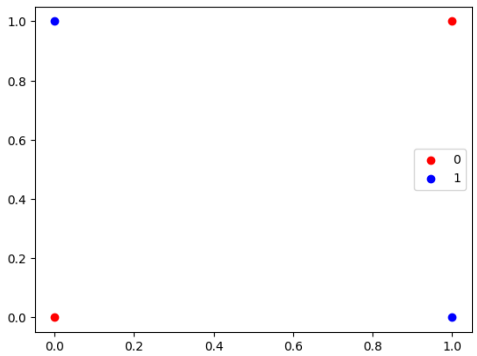
   2) We determine the split such that most items are correctly classified

   Now since all the training items have the same label the info gain will be zero and hence according to Occam's razor we choose the simpler one since both have the same result.

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.

   following data points will force the algothrim to not split.

x_1	x_2	Outcome
0	0	0
0	1	1
1	0	1
1	1	0



   Why only a handful of points.- let us consider a case, where the tree considers a split at x1. It will calculate the info gain before and after and both the times it will be -1 and hence there will be zero info gain and hence the tree will not split and we only need a handful of points.

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.

Condition	Information Gain
$x_1 \geq 0.1$	0.0321
$x_1 \geq 0$	0
$x_2 \geq -2.0$	0
$x_2 \geq -1.0$	0.0315
$x_2 \geq 0.0$	-0.0341
$x_2 \geq 1.0$	-0.0428
$x_2 \geq 2.0$	-0.0395
$x_2 \geq 3.0$	-0.03
$x_2 \geq 4.0$	-0.0148
$x_2 \geq 5.0$	0.0061
$x_2 \geq 6.0$	0.0400
$x_2 \geq 7.0$	-0.0342
$x_2 \geq 8.0$	0.1032

```
Split on x2>6.0 with Information Gain = 0.19958702318968735
    ├── Split on x2>0.0 with Information Gain = 0.2935644431995964
        ├── Split on x1>0.1 with Information Gain = 1.0
            ├── Leaf Node: Predicted Label = 1
            └── Leaf Node: Predicted Label = 0
        └── Leaf Node: Predicted Label = 0
    └── Split on x2>7.0 with Information Gain = 0.2516291673878229
        ├── Leaf Node: Predicted Label = 1
        └── Split on x2>8.0 with Information Gain = 1.0
            ├── Leaf Node: Predicted Label = 0
            └── Leaf Node: Predicted Label = 1
```

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.

```
Split on x1>10.0 with Information Gain = 0.3219280948873623
    ├── Split on x2>3.0 with Information Gain = 1.0
        ├── Leaf Node: Predicted Label = 0
        └── Leaf Node: Predicted Label = 1
    └── Leaf Node: Predicted Label = 1
```

if $x_1 \geq 10$:
assign label 1
else if $x_2 \geq 3$:
assign label 1
else:
assign label 0

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

[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.

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.

```
Split on x2>0.201829 with Information Gain = 0.6690158350565576
    ├── Leaf Node: Predicted Label = 0
    └── Leaf Node: Predicted Label = 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.

  We can see it easily that if x2 is greater than 0.201829 then it has the label 0 otherwise it has the label 1

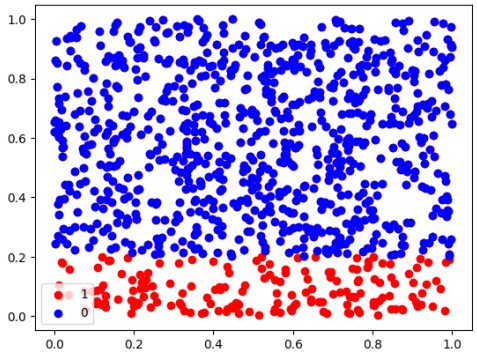
- Build a decision tree on D2.txt. Show it to us.

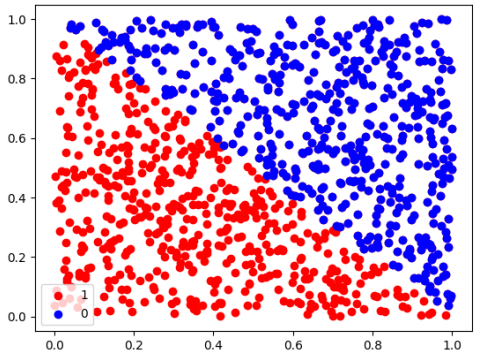
```
Split on x1>0.533076 with Information Gain = 0.18547466660807876
    ├── Split on x2>0.639018 with Information Gain = 0.34999649471050315
        ├── Split on x2>0.534979 with Information Gain = 0.0757438417334903
            ├── Leaf Node: Predicted Label = 0
            └── Split on x1>0.409972 with Information Gain = 0.04412566480449431
                ├── Leaf Node: Predicted Label = 0
                └── Split on x1>0.426073 with Information Gain = 0.11134785288180327
                    ├── Split on x1>0.417579 with Information Gain = 1.0
                        ├── Leaf Node: Predicted Label = 1
                        └── Leaf Node: Predicted Label = 0
                    └── Leaf Node: Predicted Label = 1
        └── Split on x1>0.111076 with Information Gain = 0.18697554574592323
            ├── Split on x2>0.964767 with Information Gain = 0.5435644431995964
                ├── Leaf Node: Predicted Label = 0
                └── Leaf Node: Predicted Label = 1
            └── Split on x2>0.861 with Information Gain = 0.19938026829205346
                ├── Split on x1>0.33046 with Information Gain = 0.06616910145902322
                    ├── Split on x2>0.745406 with Information Gain = 0.3212981208250194
                        ├── Leaf Node: Predicted Label = 0
                        └── Split on x1>0.254049 with Information Gain = 0.21206776641371627
                            ├── Split on x1>0.191915 with Information Gain = 0.3178113757536237
                                ├── Leaf Node: Predicted Label = 0
                                └── Split on x2>0.792752 with Information Gain = 1.0
                                    ├── Leaf Node: Predicted Label = 0
                                    └── Leaf Node: Predicted Label = 1
                            └── Leaf Node: Predicted Label = 1
                    └── Leaf Node: Predicted Label = 1
                └── Leaf Node: Predicted Label = 1
    └── Split on x2>0.383738 with Information Gain = 0.35833190797692244
        ├── Split on x1>0.761423 with Information Gain = 0.14646969122407372
            ├── Split on x2>0.301105 with Information Gain = 0.2689955935892812
                ├── Leaf Node: Predicted Label = 0
                └── Split on x1>0.66337 with Information Gain = 0.12115608152133561
                    ├── Leaf Node: Predicted Label = 0
                    └── Leaf Node: Predicted Label = 1
            └── Split on x2>0.191206 with Information Gain = 0.35601142235001465
                ├── Split on x1>0.90482 with Information Gain = 0.18978438611872506
                    ├── Split on x2>0.169053 with Information Gain = 0.3064509395127765
                        ├── Leaf Node: Predicted Label = 0
                        └── Split on x1>0.850316 with Information Gain = 0.9182958340544896
                            ├── Leaf Node: Predicted Label = 0
                            └── Leaf Node: Predicted Label = 1
                    └── Split on x2>0.037708 with Information Gain = 0.49962073521348693
                        ├── Leaf Node: Predicted Label = 0
                        └── Split on x1>0.930371 with Information Gain = 0.1486525820077829
                            ├── Split on x1>0.927522 with Information Gain = 0.9182958340544896
                                ├── Leaf Node: Predicted Label = 1
                                └── Leaf Node: Predicted Label = 0
                            └── Leaf Node: Predicted Label = 1
                └── Leaf Node: Predicted Label = 1
        └── Split on x1>0.550364 with Information Gain = 0.02782390816998806
            ├── Split on x2>0.474971 with Information Gain = 0.5435644431995964
                ├── Leaf Node: Predicted Label = 0
                └── Leaf Node: Predicted Label = 1
            └── Leaf Node: Predicted Label = 1
```

- Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
  No it no easy or maybe impossible by a human to follow the chain of the tree and intrpret the results successfully as the tree has a lot of nodes and hence i t would be very difficult to track them!

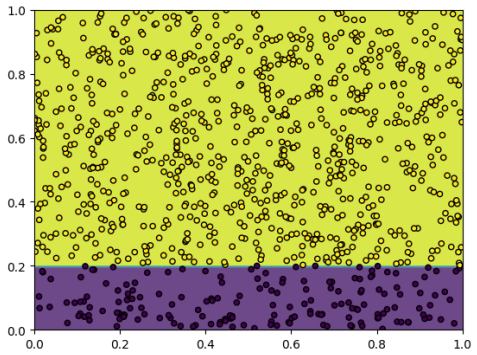
6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

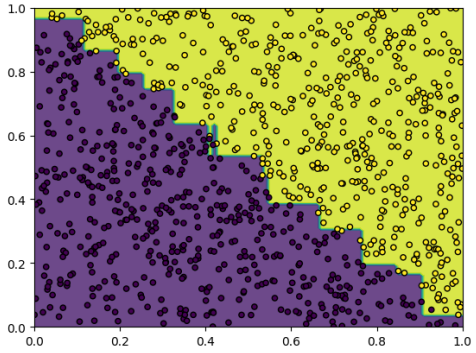
   - Produce a scatter plot of the data set.





- 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).





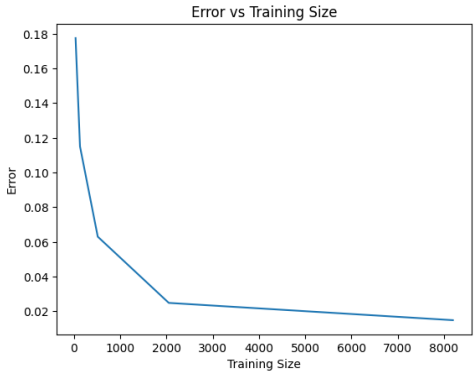
Then discuss why the size of your decision trees on D1 and D2 differ. Relate this to the hypothesis space of our decision tree algorithm.

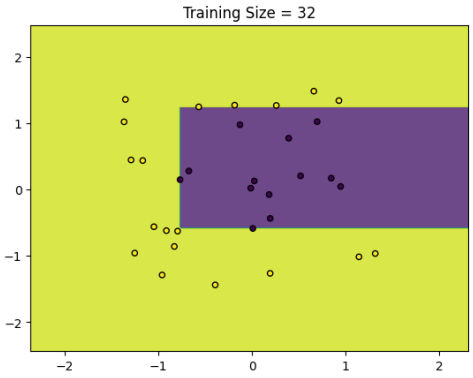
I think for D1 it was trying to model y= -0.2 essentially and hence it was able to create a simple decesion tree but d2 had a linear relationship and decesiopn tree work on better on non linear data hence we can see the difference in the decesion boundaries and the size because it was trying to overfit a linear relation ship.

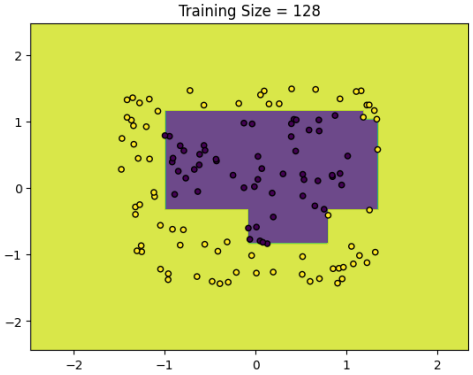
7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

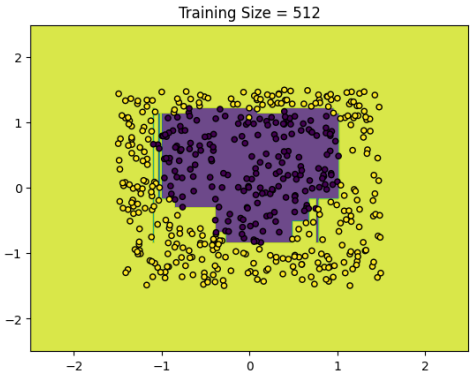
   - You will randomly split Dbig.txt into a candidate training set of 8192 items and a test set (the rest). Do this by generating a random permutation, and split at 8192.
   - Generate a sequence of five nested training sets $D_{32} \subset D_{128} \subset D_{512} \subset D_{2048} \subset D_{8192}$ from 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).

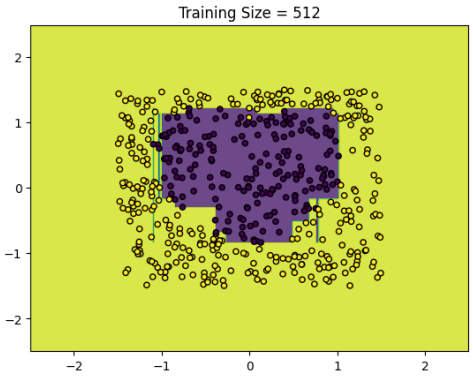
training-size	nodes
32	4
128	7
512	17
2048	49
8192	118

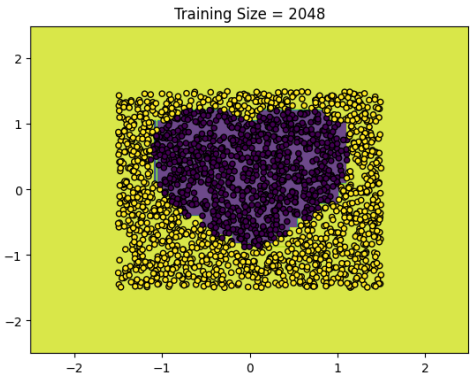


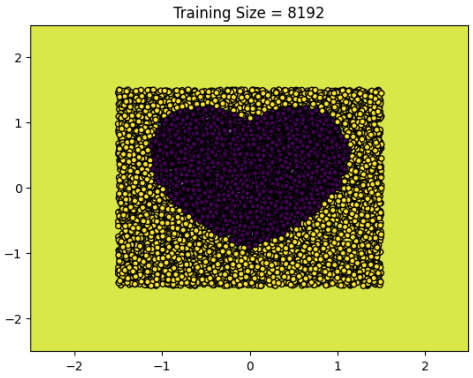

Training Size = 32
2
1
0
−1
−2
−2
−1
0
1
2


Training Size = 128
2
1
0
−1
−2
−2
−1
0
1
2


Training Size = 512
2
1
0
−1
−2
−2
−1
0
1
2



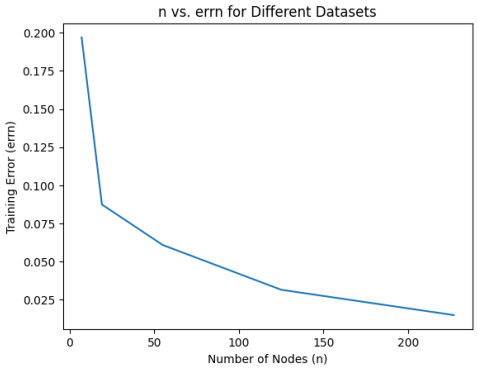




3 sklearn [10 pts]

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	nodes
32	11
128	35
512	59
2048	127
8192	245

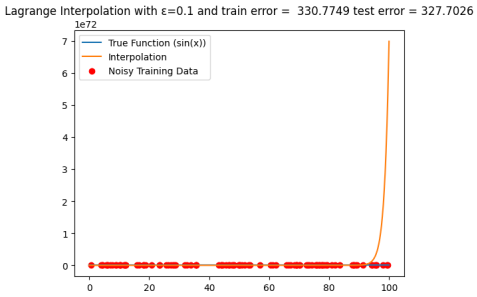


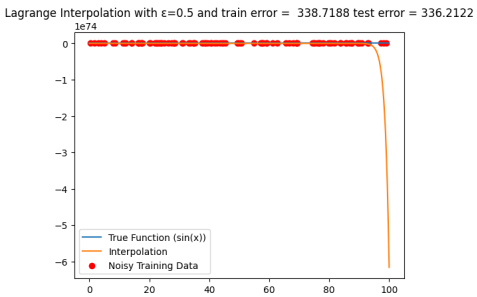
4 Lagrange Interpolation [10 pts]

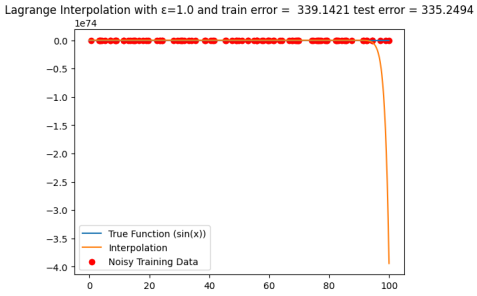
Fix some interval $[a, b]$ and sample $n = 100$ points x from this interval uniformly. Use these to build a training set consisting of n pairs (x, y) by setting function $y = sin(x)$.

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?
Training Error = -7.584071961222803e+69 Test Error = -1.0693718174155704e+70
We observe a very low both error as the data fits the model exactly
Repeat the experiment with zero-mean Gaussian noise ϵ added to x. Vary the standard deviation for ϵ and report your findings.
WIth noise we see the eroor rise as we make the sd higher which makes sense as noise increseases the eroor is also expected to increase as we can see in the grpahs.
We also a higher test error than training as i think the model tries to over fit the noisy data






Lagrange Interpolation with ε=1.0 and train error = 339.1421 test error = 335.2494
1e74
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
−3.0
−3.5
−4.0
0
20
40
60
80
100
True Function (sin(x))
Interpolation
Noisy Training Data