HOMEWORK 2

Luke Neuendorf
907-608-0960

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 Repo: https://github.com/lneuendorf/CS760-Machine-Learning/tree/main/DecisionTreeClassifier

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:

$$\begin{array}{ccc} x_{11} & x_{12} & y_1 \\ & \dots & \\ x_{n1} & x_{n2} & y_n \end{array}$$

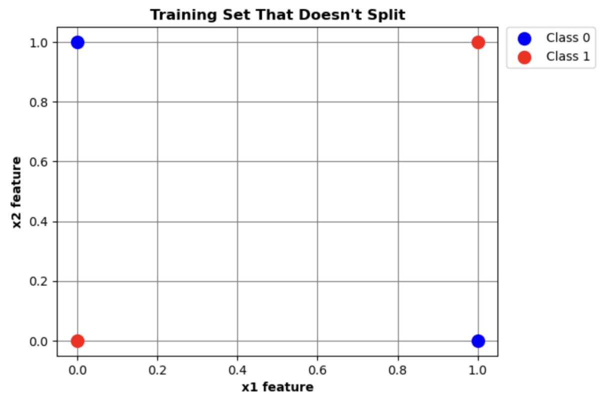
Your program should implement a decision tree learner according to the following guidelines:

- 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.
   A node containing training items all having the same label is guaranteed to become a leaf because it will have a zero gain ratio for all possible candidate splits, which is a stopping condition. The equation of gain ratio is as follows:
   $$GainRatio(D,S) = \frac{H_D(Y) - H_D(Y|S)}{H_D(S)} \qquad \textbf{[1]}$$
   Both the components in the numerator will be zero. $H_D(Y)$ is entropy of Y which is zero when Y is consists of one label. Furthermore, the formula for conditional entropy is as follows:
   $$H_D(Y|S) = -\sum_{s\in S, y\in Y} p(s,y) * log_2 \frac{p(s,y)}{p(s)} \qquad \textbf{[2]}$$
   We know $p(s,y) == p(s)$ because all of $y \in Y$ are the same label. Therefore, the logarithm becomes $log_2(1)$ which equals zero, and thus, $H_D(Y|S) = 0$.

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.
   A "square" training set consisting of four datapoints with corresponding labels at opposite ends of the "square" will not split. This is because the information gain of any split on the dataset is zero, due to fact that $H(Y)$ and $H(Y|S)$ are both one, so the difference of the two is zero. This causes the gain ratio to be zero as seen in equation [1]. If you force a split, then there would be two sets of data with a non-zero gain ratio that could be split further, leading to a tree with zero training error.



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.

```
X1 >= 0.1       Gain Ratio: 0.10052
X1 >= 0.0       Mutual Info: 0.00000
X2 >= -2        Mutual Info: 0.00000
X2 >= -1        Gain Ratio: 0.10052
X2 >= 0         Gain Ratio: 0.05595
X2 >= 1         Gain Ratio: 0.00578
X2 >= 2         Gain Ratio: 0.00114
X2 >= 3         Gain Ratio: 0.01641
X2 >= 4         Gain Ratio: 0.04975
X2 >= 5         Gain Ratio: 0.11124
X2 >= 6         Gain Ratio: 0.23610
X2 >= 7         Gain Ratio: 0.05595
X2 >= 8         Gain Ratio: 0.43016
```

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.

```
        X1>=10.0
      T/        \F
    Y=1          X2>=3.0
               T/      \F
              Y=1       Y=0
```

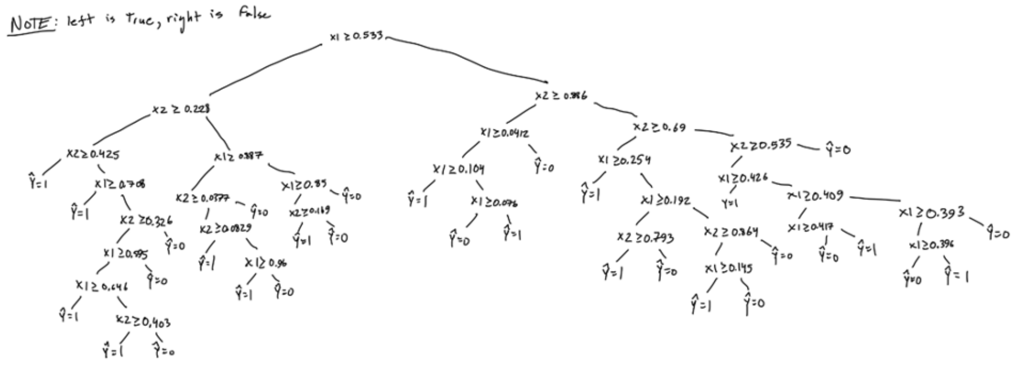
5. (Or is it?) [20 pts] For this question only, make sure you DO NOT VISUALIZE the data sets or plot your tree's decision boundary in the 2D $\mathbf{x}$ space. If your code does that, turn it off before proceeding. This is because you want to see your own reaction when trying to interpret a tree. You 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.

```
      X2>=0.201829
     T/           \F
   Y=1             Y=0
```

   - 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 is a line at X2=0.201829. Everything above the line is classified as Y=1 and everything below as Y=0. The feature is of no significance when making predictions with this model.

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



```
Preorder DFS:
        ['(root)X1>=0.533', '(internal)X2>=0.228', '(internal)X2>=0.425', '(leaf)Y=1', '(internal)X1>=0.708',
        '(leaf)Y=1', '(internal)X2>=0.326', '(internal)X1>=0.595', '(internal)X1>=0.646', '(leaf)Y=1',
        '(internal)X2>=0.403', '(leaf)Y=1', '(leaf)Y=0', '(leaf)Y=0', '(leaf)Y=0', '(internal)X1>=0.887',
        '(internal)X2>=0.038', '(internal)X2>=0.083', '(leaf)Y=1', '(internal)X1>=0.961', '(leaf)Y=1', '(leaf)Y=0',
        '(leaf)Y=0', '(internal)X1>=0.850', '(internal)X2>=0.169', '(leaf)Y=1', '(leaf)Y=0', '(leaf)Y=0',
        '(internal)X2>=0.886', '(internal)X1>=0.041', '(internal)X1>=0.104', '(leaf)Y=1', '(internal)X1>=0.076',
        '(leaf)Y=0', '(leaf)Y=1', '(leaf)Y=0', '(internal)X2>=0.691', '(internal)X1>=0.254', '(leaf)Y=1',
        '(internal)X1>=0.192', '(internal)X2>=0.793', '(leaf)Y=1', '(leaf)Y=0', '(internal)X2>=0.864',
        '(internal)X1>=0.145', '(leaf)Y=1', '(leaf)Y=0', '(leaf)Y=0', '(internal)X2>=0.535', '(internal)X1>=0.426',
        '(leaf)Y=1', '(internal)X1>=0.410', '(internal)X1>=0.418', '(leaf)Y=0', '(leaf)Y=1', '(internal)X1>=0.393',
        '(internal)X1>=0.396', '(leaf)Y=0', '(leaf)Y=1', '(leaf)Y=0', '(leaf)Y=0']
```

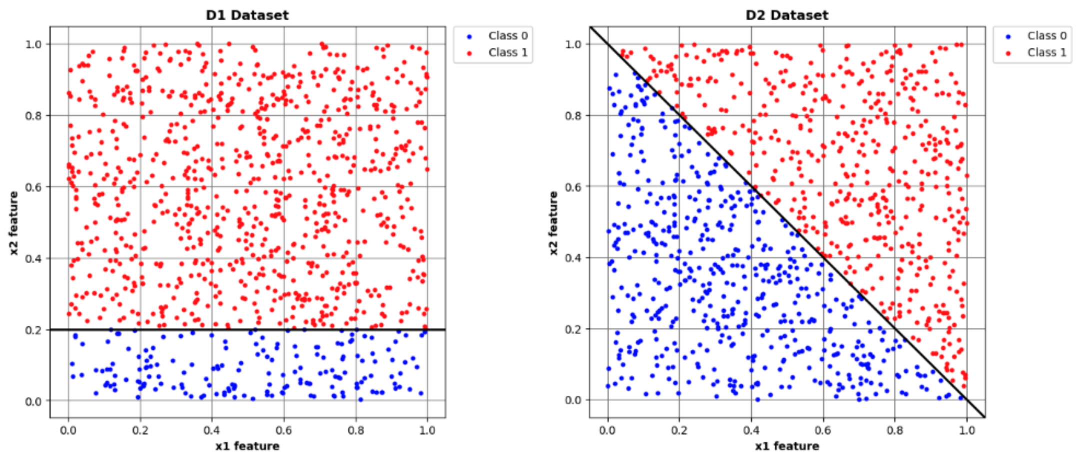
   - Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
     No, the tree has 61 nodes including leaf nodes, internal nodes, and the root node. It is too complicated to easily visualize. It is obvious this decision boundary is not a horizontal or vertical line at X2 or X1.

[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.

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.



D1 is a simple decision tree because it can be divided into two classes by splitting at $X2 = 0.2$. D2 is a complex decision tree because its decision boundary is a diagonal line crossing the X1 and X2 axes. Since we do not consider splits that are a linear combination of X1 and X2, the model must fit a staircase-like boundary to divide the D2 dataset classes. Therefore, they hypothesis space of models that accurately divide the D1 dataset is relatively small. However, the hypothesis space of models that accurately divide D2 is very large as you could take very short steps to mimic a negatively sloped line that is the decision boundary.

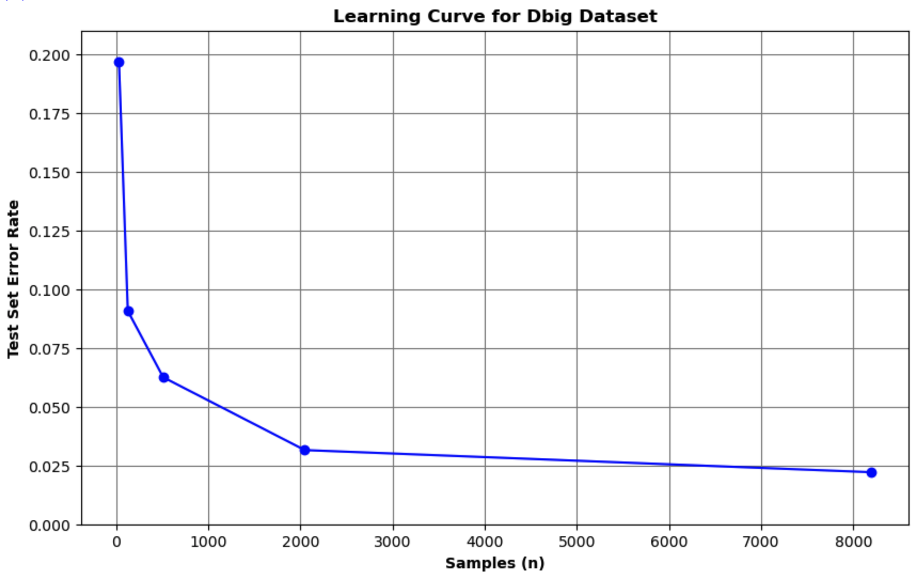
7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

    - 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).

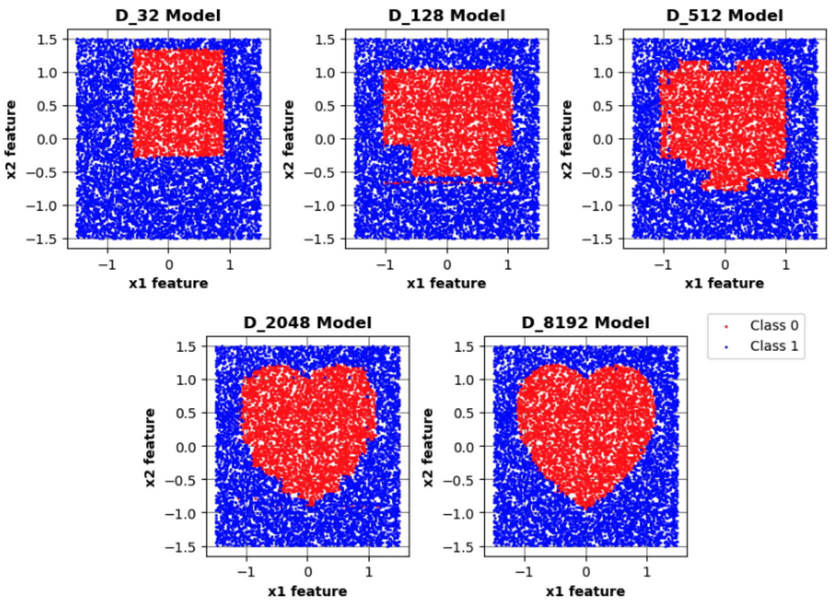
(1)

n	num_nodes	error_rate
32	9	0.196903
128	19	0.090708
512	53	0.062500
2,048	137	0.031527
8,192	259	0.022124

(2)



(3)



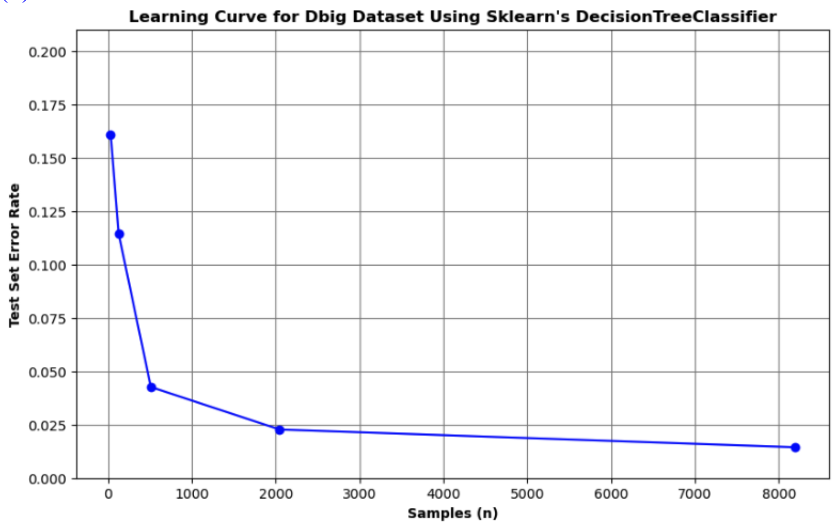
3 sklearn [10 pts]

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.

(1)

n	num_nodes	error_rate
32	9	0.160951
128	23	0.114491
512	43	0.042588
2,048	111	0.022677
8,192	233	0.014381

(2)

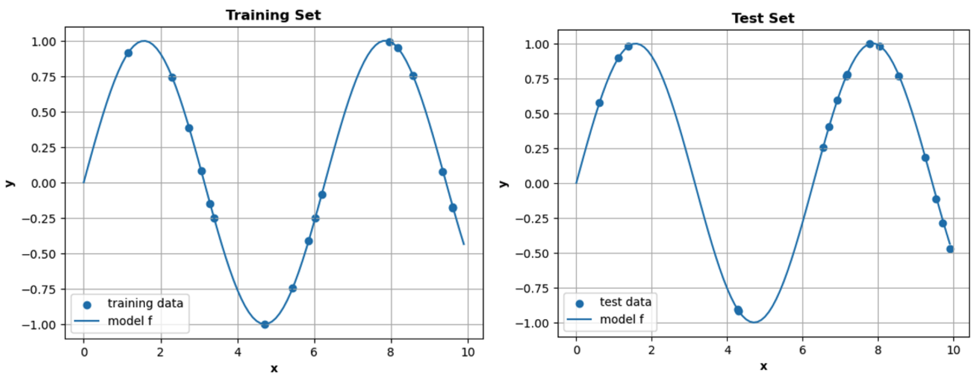


4 Lagrange Interpolation [10 pts]

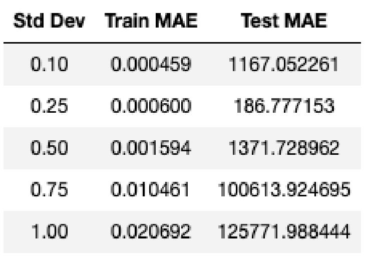
Fix some interval $[a, b]$ and sample $n = 100$ points x from this interval uniformly. Use these to build a training set consisting of n pairs (x, y) by setting function $y = sin(x)$.

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 problem, I uniformly sampled 100 points from the interval [0,10]. Them, I sampled 17 testing and 17 training points from the 100 points because the library scipy.interpolate.lagrange is unstable with anything over 20 datapoints as referenced in a piazza post.The Lagrange interpolating polynomial fits both the test and training data well as seen in the plots below. The train MAE (mean-absolute-error) is 0.003127 and the test MAE is 0.004501. The model performs worse on the test set than the train set which is logical.

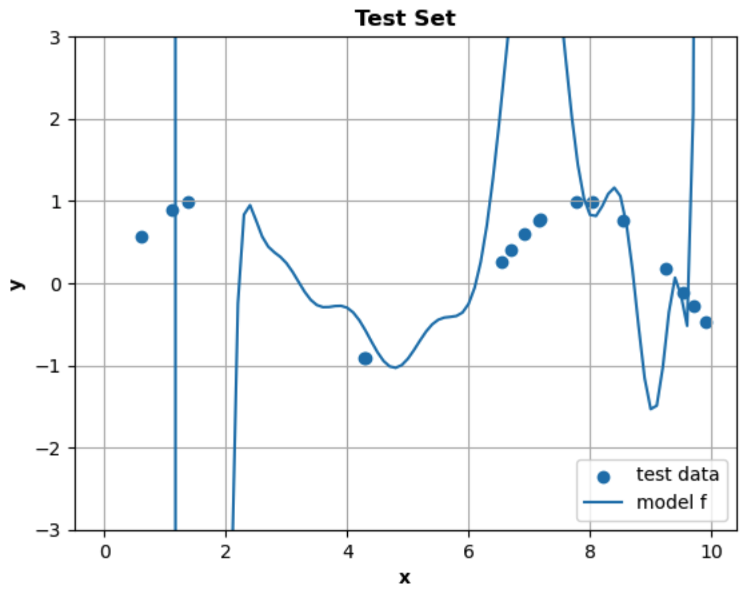


Next, I added zero-mean gaussian noise to x with standard deviations as follows: 0.1, 0.25, 0.5, 0.75, 1.0. The resulting MAE scores are shown in the table below.



Std Dev	Train MAE	Test MAE
0.10	0.000459	1167.052261
0.25	0.000600	186.777153
0.50	0.001594	1371.728962
0.75	0.010461	100613.924695
1.00	0.020692	125771.988444

The model fits well to the training data, but fits the testing data VERY poorly. This is because the polynomial becomes erratic when noise is added to the sign wave. We can see why the test MAE is so bad when looking at a plot of the test points against the Lagrange polynomial generating from the training data with noise added of standard deviation 0.1


Test Set
y
x
test data
model f