HOMEWORK 2

Martino Boggs
ID: 907 810 3539

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 $x_{\cdot j} \geq c$.
- 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 $y = 1$.

See code on GitHub:

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 the training items have the same label y^*, then $Y = y^*$ is a constant random variable and thus is independent of any candidate split S. It follows that the empirical entropy of Y and $Y|S$ is

   $$H_D(Y) = -\sum_{y \in \mathcal{Y}} \mathbb{P}(Y = y) \log_2 \mathbb{P}(Y = y) = -\mathbb{P}(Y = y^*) \log_2 \mathbb{P}(Y = y^*) = 0$$

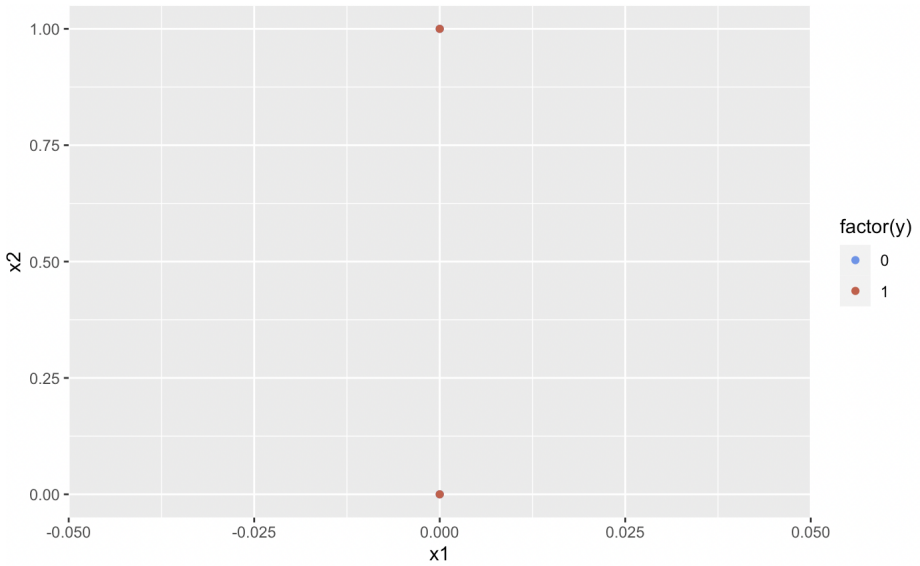
   $$H_D(Y|S) = -\sum_{s \in \mathcal{S}} \sum_{y \in \mathcal{Y}} \mathbb{P}(S = s)\mathbb{P}(Y = y|S = s) \log_2 \mathbb{P}(Y = y|S = s)$$

   $$= -\sum_{s \in \mathcal{S}} \mathbb{P}(S = s)\mathbb{P}(Y = y^*) \log_2 \mathbb{P}(Y = y^*) = 0$$

   since $\log_2 \mathbb{P}(Y = y^*) = \log_2(1) = 0$. Therefore, the information gain ratio of the candidate split is zero and our algorithm creates a leaf node.

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.

   The algorithm will turn the root into a leaf node if all candidate splits have zero gain ratio. In this example, our training set has both classes present, but the features x_1 and x_2 are non-informative. That is for each observation with label $y = 1$, there is an observation labelled $y = 0$ with the same features x_1, x_2. The candidate splits will have zero gain ratio and the algorithm will return the prediction $y = 1$.

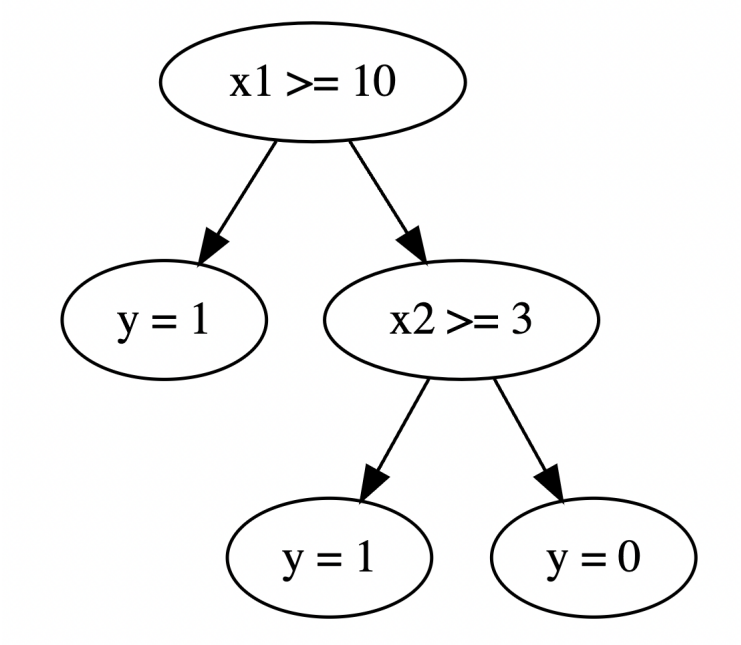


3. (Information gain ratio exercise) [10 pts] Use the training set Druns.txt. For the root node, list all candidate cuts and their information gain ratio. If the entropy of the candidate split is zero, please list its mutual information (i.e. information gain). Hint: to get $\log_2(x)$ when your programming language may be using a different base, use `log(x)/log(2)`. Also, please follow the split rule in the first section.

feature	threshold	igr	ig
1	0.1	0.100518077	0.044177392
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
1	0.0	0.000000000	0.000000000
2	-2.0	0.000000000	0.000000000
2	-1.0	0.100518077	0.044177392
2	0.0	0.055953760	0.038274522
2	1.0	0.005780042	0.004886164
2	2.0	0.001144350	0.001082166
2	3.0	0.016411137	0.016313166
2	4.0	0.049749064	0.049452073
2	5.0	0.111240296	0.105195532
2	6.0	0.236099606	0.199587023
2	7.0	0.055953760	0.038274522
2	8.0	0.430156916	0.189052669

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.

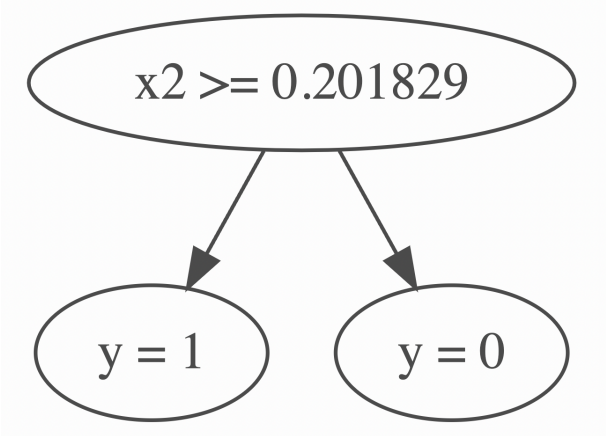
   If $x_1 \geq 10$, then predict $y = 1$; otherwise, predict $y = 1$ if $x_2 \geq 3$ or predict $y = 0$ if $x_2 < 3$.



[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.

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.

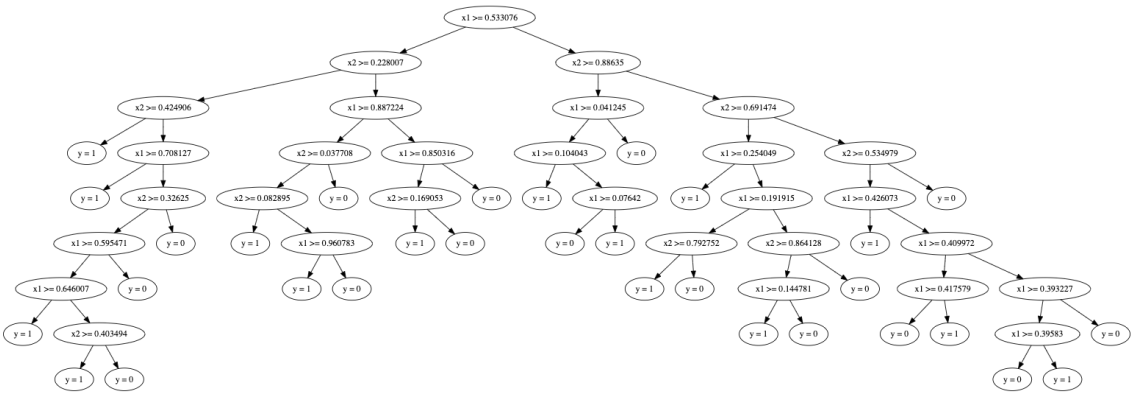
   - Build a decision tree on D1.txt. Show it to us in any format (e.g. could be a standard binary tree with nodes and arrows, and denote the rule at each leaf node; or as simple as plaintext output where each line represents a node with appropriate line number pointers to child nodes; whatever is convenient 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.

     For D1.txt, the decision boundary is the horizontal line $x_2 = 0.201829$. Training points in the dataset above this line are predicted to have label $y = 1$, and points below this line are predicted to have label $y = 0$.

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

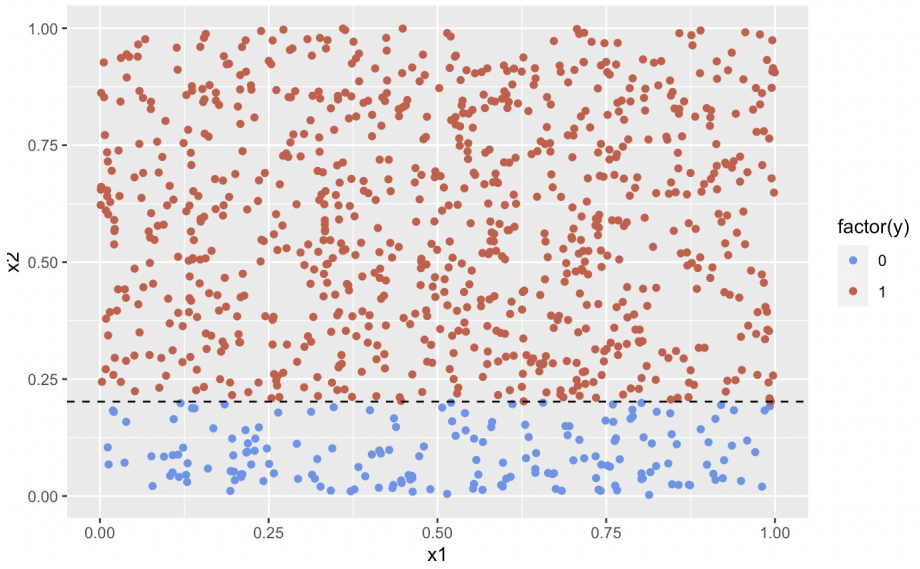


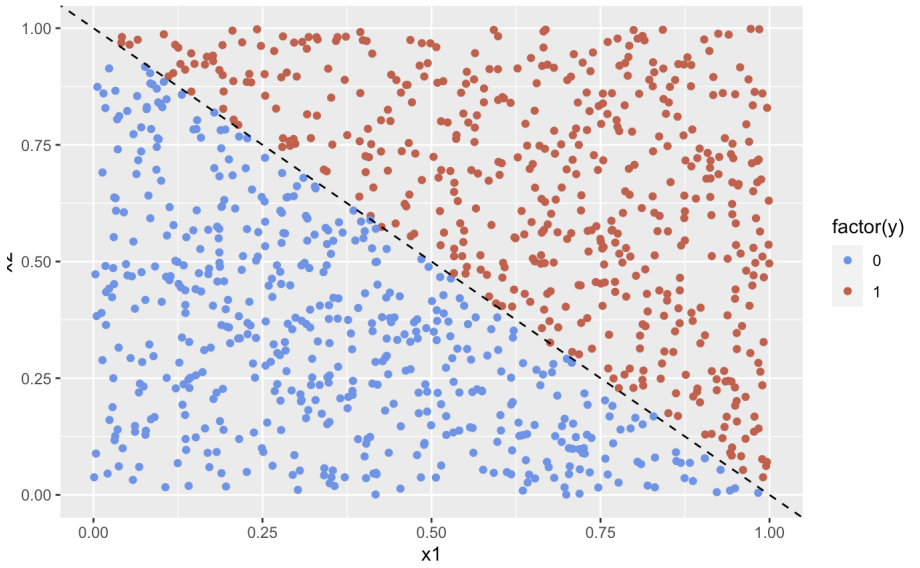
   - Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?

     The decision tree for 'D2.txt' is much larger than the previous one and is more challenging to interpret without visualization. The decision boundary involves both x_1 and x_2 (i.e., not a horizontal or vertical line). Training points in this dataset seem to be predicted to have label $y = 1$ if either x_1 or x_2 are sufficiently large. For example, the decision tree predicts $y = 1$ for both training points $(x_1, x_2) = (.97, .05)$ and $(x_1, x_2) = (.11, .88)$. It also predicts $y = 1$ for $(x_1, x_2) = (0.54, 0.43)$, but predicts $y = 0$ for $(x_1, x_2) = (0.53, 0.53)$. The decision boundary may be a line with a negative slope (e.g., close to $x_2 = 1 - x_1$).

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.

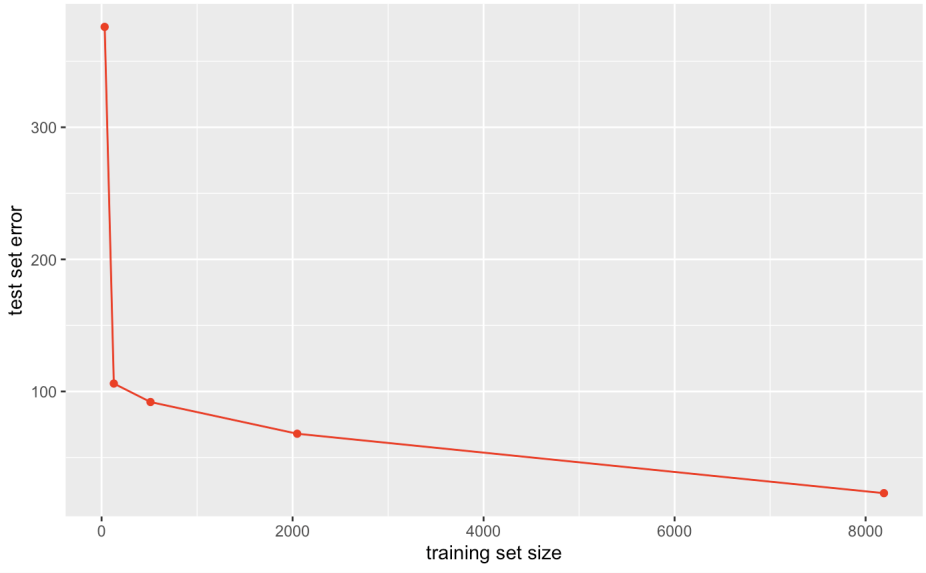
For D1, we only need to split on the feature x_2 since it alone determines the label y. The size of the decision tree on D2 is larger since the label y depends on both x_1 and x_2. The hypothesis space for our decision tree algorithm is the set of all possible decision trees using candidate splits on x_1 and x_2. Since x_1 provides no information gain, the hypothesis space for the decision tree on D1 is constrained than that of decision tree on D2.

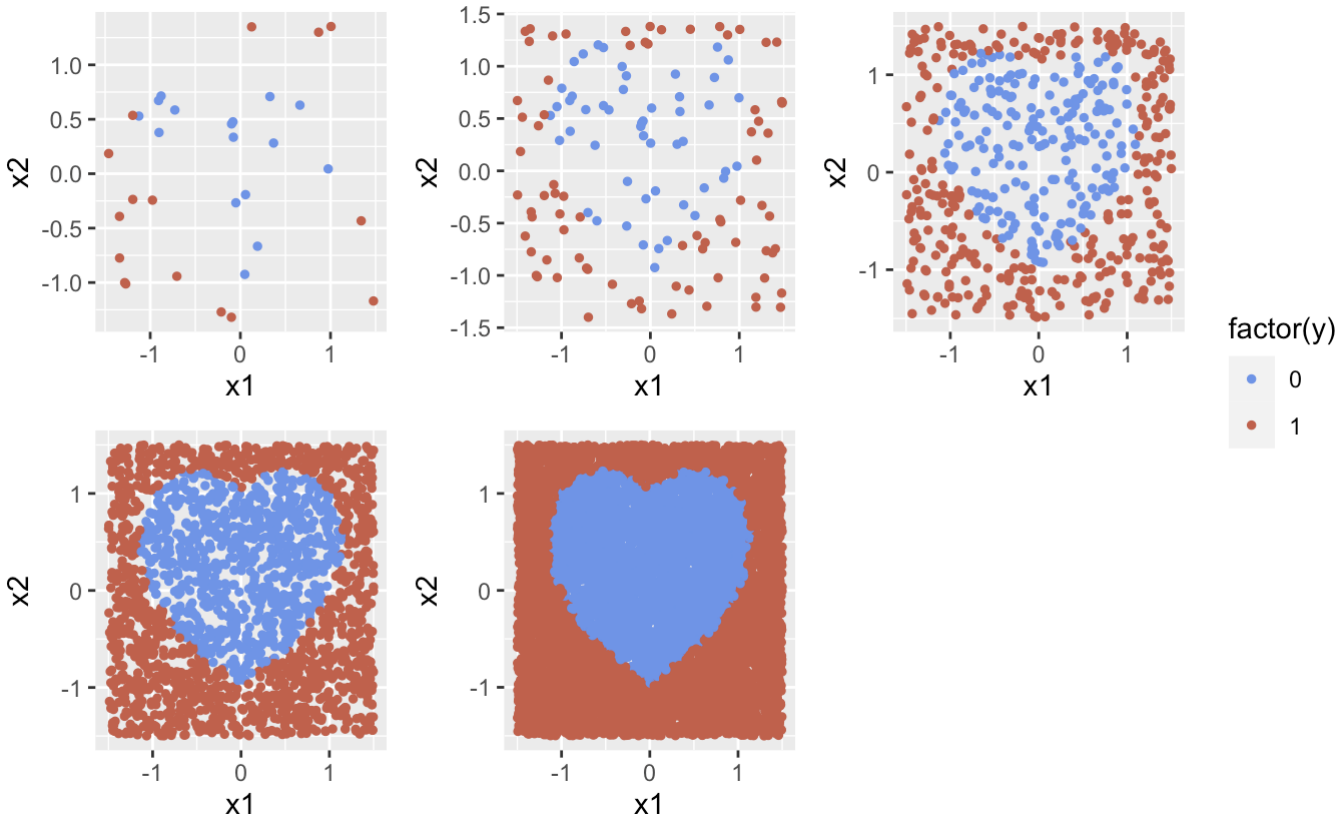
7. (Learning curve) [20 pts] We provide a data set Dbig.txt with 10000 labeled items. Caution: Dbig.txt is sorted.

   - 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).

Note that error is number of misclassified data points in test set (out of 1808).

n	num_nodes	error
32	13	376
128	23	106
512	59	92
2048	127	68
8192	263	23



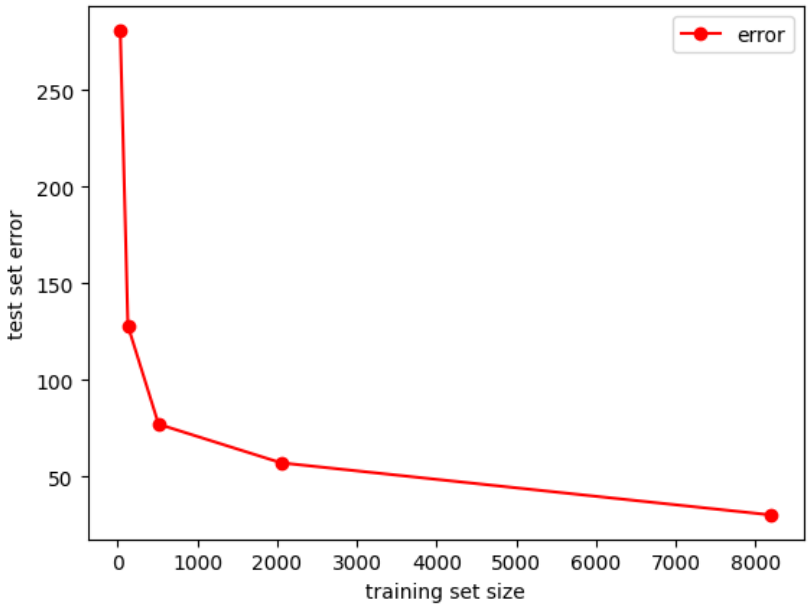


3 sklearn [10 pts]

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.

Note that error is number of misclassified data points in test set (out of 1808).

	n	num_nodes	error
0	32	13	281
1	128	35	128
2	512	63	77
3	2048	105	57
4	8192	241	30



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.

We sample $n = 19$ points uniformly from $[0, 4]$ and set $y_i = \sin(x_i)$ for $i = 1, \ldots, 19$. We fit a model using Lagrange interpolation. (Note that fitting a model with more than nineteen points using 'lagrange' from the 'scipy' package results in numerical errors and absurdly large MSE.) The MSEs computed on the training set and test set were 0.001 and 0.011, respectively. The model fits the sine curve fairly well (barring issues with numerical errors).

Let $x_i \sim U(0, 4)$, $y_i = \sin(x_i)$, and fit a model using Lagrange interpolation on $(x_i + \epsilon_i, y_i)$ where $\epsilon_i \sim \mathcal{N}(0, 1)$ for $i = 1, \ldots, 19$. The MSEs computed on the training set and test set (with no Gaussian noise) were 0.056 and 1.57×10^8. The test error is much larger than the train error setting $\sigma = 1$. Since our model was fit using noisy data, it doesn't capture the true distribution of (x_i, y_i). Increasing σ by a factor of 100 seems to reduce both the train and test error, although the test error is still much larger than the train error and our model doesn't capture the sine curve we are trying to learn. Reducing σ by a factor of 100 also tends to reduce both the train and test error (and are closer in magnitude).