HOMEWORK 2

Hailey Park
hpark353@wisc.edu
https://github.com/hparkailey/760ML.git

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:

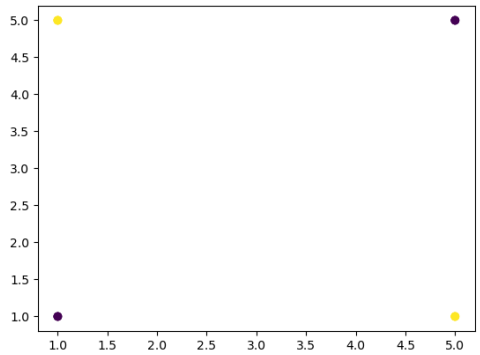
- Candidate splits (j, c) for numeric features should use a threshold c in feature dimension j in the form of $x_j \geq c$.
- c should be on values of that dimension present in the training data; i.e. the threshold is on training points, not in between training points. You may enumerate all features, 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.

   If the node has training items all with the same label, then any splits from that node will not result in any information gain and would have zero mutual information. Another reason is by Occam's razor, we prefer a simpler model.

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.



   Attempting to split this data will not result in any information gain (i.e. the information gain of any splits would be 0). Hence, our decision tree will stop at the root node and assign every instance as a leaf.

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.

   Please refer table 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.

   '$x1 <= 1$': 'left': 0, 'right': 1 If x1 is less than or equal to 1, then classify as 0, else as 1.

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 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

[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.

Feature	Candidate Split Threshold	InfoGain
x1	0.0	0.10051807676021828
x1	0.0	0.10051807676021828
x1	0.0	0.10051807676021828
x1	0.0	0.10051807676021828
x1	0.0	0.10051807676021828
x2	-1	0.055953759631263526
x2	0	0.00578004220515232
x2	6	0.055953759631263526
x2	7	0.4301569161309807
x1	0.0	0.07280247297910734
x1	0.0	0.07280247297910734
x1	0.0	0.07280247297910734
x1	0.0	0.07280247297910734
x2	-1	0.1206166388929235
x2	0	0.03661492441548469
x2	6	0.07280247297910734
x1	0.0	1.0
x2	6	0.04755786045881497

Table 1: Gain Info Table

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?

D1 decision tree is as follows. The tree can be interpreted as determine whether $x2 <= 0.199725$. If so, assign 0, otherwise assign 1.

```
{'x2<=0.199725': {'left': 0, 'right': 1}}
```

D2 decision tree is as follows. The root node is splitting at $x1 <= 0.532664$ so x1 seems to be the prominent feature, then it consequently splits at $x2 <= 0.881731$ if the root condition is satisfied, followed by $x2 \leq 0.690829$, $x2 <= 0.501454$, and so on. It's extremely hard to interpret the D2 decision tree and patterns for different label assignments without visualizing it.

```
{'x1<=0.532664': {'left': {'x2<=0.881731':
{'left': {'x2<=0.690829':
{'left': {'x2<=0.501454': {'left': 0,
    'right': {'x1<=0.426073':
    {'left': {'x1<=0.409972': {'left': {'x1<=0.343437': {'left': 0,
          'right':
          {'x1<=0.409387':
          {'left': {'x2<=0.609262':
          {'left': 0,
              'right': 1}},
            'right': 1}}}},
       'right': {'x1<=0.41206': {'left': 0,
          'right':
          {'x2<=0.570223': {'left': 0, 'right': 1}}}}}},
    'right':
    {'x2<=0.527033': {'left': 0, 'right': 1}}}}}},
```

```
      'right':
      {'x1<=0.254049':
      {'left': {'x1<=0.130649': {'left': 0,
          'right':
          {'x2<=0.827252':
          {'left': {'x2<=0.783115': {'left': 0,
              'right': {'x1<=0.178711': {'left': 0, 'right': 1}}}},
            'right': 1}}}},
        'right': 1}}}},
    'right': {'x1<=0.104094':
    {'left': {'x2<=0.917219': {'left': 0,
        'right': 1}},
      'right': 1}}}},
  'right': {'x2<=0.221983':
  {'left': {'x1<=0.830388': {'left': 0,
      'right': {'x2<=0.091239':
      {'left': {'x1<=0.941524': {'left': 0,
          'right':
          {'x2<=0.014066': {'left': 0, 'right': 1}}}},
        'right': 1}}}},
    'right': {'x2<=0.226254':
    {'left': {'x1<=0.567315': {'left': 0,
        'right': {'x1<=0.574443':
        {'left': 0, 'right': 1}}}},
      'right': {'x2<=0.421367':
      {'left': {'x1<=0.707502': {'left': {'x2<=0.291518': {'left': 0,
            'right': {'x1<=0.66337':
            {'left': {'x1<=0.646007':
            {'left': {'x1<=0.582574': {'left': 0,
                'right': {'x2<=0.403494': {'left': {'x1<=0.619262': {'left': {'x2<
                  'right': 1}},
                'right': 0}},
              'right': 1}}}},
            'right': 1}},
          'right': 1}}}},
        'right': 1}},
      'right': 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.

   The size of decision trees on D1 and D2 differ because the hypothesis space of a decision tree algorithm is only limited to horizontal or vertical straight lines and cannot generate diagonal lines. Since the decision boundary of D1 is a straight horizontal line, it was able to make the decision based on a handful of nodes, but for D2, since the decision boundary resembles a diagonal line, D2 tree had to draw several linear lines across the diagonal, resulting in a larger tree.

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.

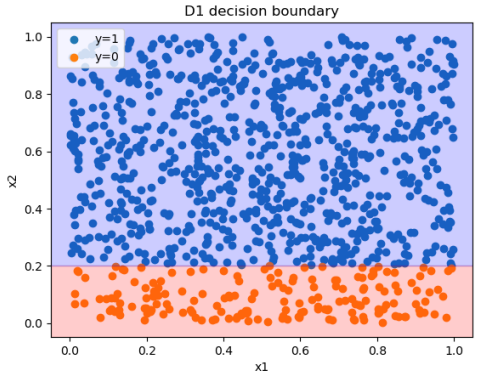


Figure 2: D1 decision boundary

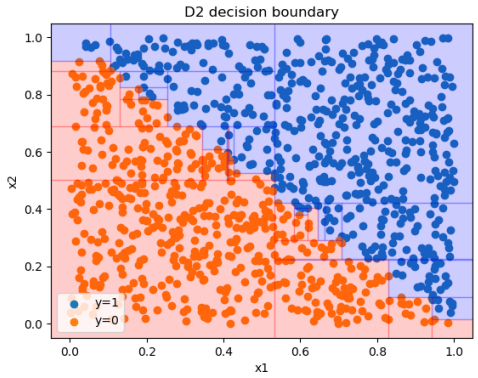


Figure 3: D2 decision boundary

- 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).

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.
Table 2 and Figure 9.

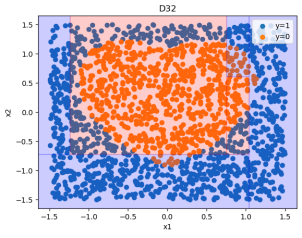


Figure 4: D32

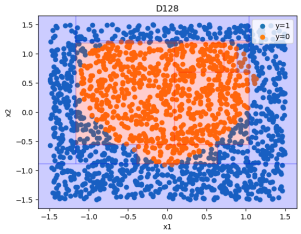


Figure 5: D128

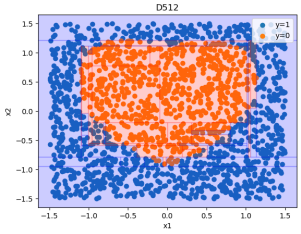


Figure 6: D512

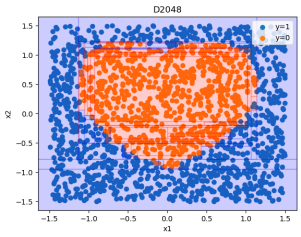


Figure 7: D2048

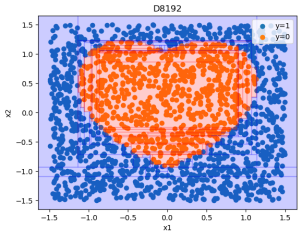


Figure 8: D8192

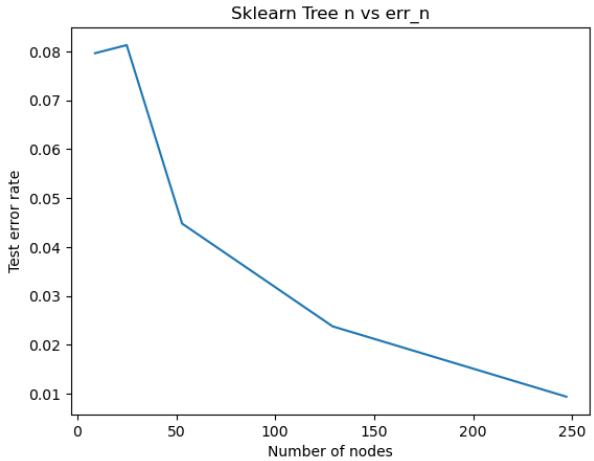


Figure 9: Number of nodes vs err_n

N	Number of nodes	Err_n
32	11	0.0796
128	21	0.0813
512	41	0.0448
2048	137	0.0237
8192	229	0.0094

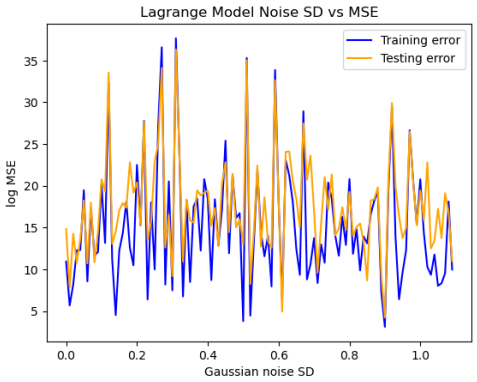
Table 2: Sklearn tree predictions

3 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.
The Larange interpolation model over [0,30] with 100 points have 868608.2054840629 training MSE and 340744.2914838607 test MSE. As the standard deviation of the Gaussian noise added to the training set x increases, both the training and the testing error get noisier. The MSE of the training and the testing error seem to be getting increasingly more unstable as the sd of the noise increases.



Lagrange Polynomial MSE