HOMEWORK 2

William Powell
908 343 3244

Source code for this assignment is available at https://github.com/wgraysonp/CS760/tree/main/HW_2

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 a non-empty node contains only training items having the same label then the gain ratio of any split will be zero which meets the stopping criteria for making a leaf node. To see this let D be the training samples

contained in the node, $n = |D|$ be the number of samples, and Y the random variable representing the class label. By assumption Y is constant so

$$H_D(Y) = -\frac{n}{n}\log_2\left(\frac{n}{n}\right) - 0 \cdot \log_2(0) = 0.$$

Now let $S \in \{\texttt{then}, \texttt{else}\}$ represent an arbitrary split and let n_t and n_e represent the number of samples classified as `then` or `else` by the split

$$\begin{aligned} H_D(Y|S) &= H_D(Y|\texttt{then})P(\texttt{then}) + H_D(Y|\texttt{else})P(\texttt{else}) \\ &= \left(-\frac{n_t}{n_2}\log_2\left(\frac{n_t}{n_t}\right) - 0 \cdot \log_2(0)\right)\frac{n_t}{n} + \left(-\frac{n_e}{n_e}\log_2\left(\frac{n_e}{n_e}\right) - 0 \cdot \log_2(0)\right)\frac{n_e}{n} \\ &= 0. \end{aligned}$$

So the information gain $H_D(Y) - H_D(Y|S)$ is equal to zero, implying a gain ratio of zero.

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.

   Consider the dataset below consisting of four training samples.

x_1	x_2	y
-1	-1	1
1	1	1
1	-1	0
-1	1	0

   There are four possible candidate splits: $x_1 \geq -1$, $x_1 \geq 1$, $x_2 \geq -1$ and $x_2 \geq 1$. If we split on value -1 in either dimension, then the entropy of the split will be zero as all of the data will be sent in the same direction. If we instead choose to split on value 1, then the information gain will be equal to zero for both dimensions. This is because the conditional entropy

$$\begin{aligned} H(Y|X_i) &= H(Y|X_i = 1)P(X_i = 1) + H(Y|X_i = -1)P(X_i = -1) \\ &= \frac{1}{2}\left(-\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2})\right) + \frac{1}{2}\left(-\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2})\right) \\ &= 1 \end{aligned}$$

   is equal to the entropy

$$H(Y) = -\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1.$$

   See Figure 1 for a plot of this data.

   If we force a split, then either we split perfectly along the training labels and the algorithm will terminate with two leaf nodes, or we will have one leaf node with one training instance and another node containing three training instances. For the second node, splitting on either $x_1 \geq 1$ or $x_2 \geq 1$ will have non-zero entropy and information gain and the algorithm will split this node resulting in perfect training accuracy.

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.

   Information gain for each candidate split is listed below. Table 1 lists splits using the first feature dimension and table 2 list splits in the second feature dimension. Splits using the smallest value for each dimension have zero entropy as all training data is sent to the left child of the root.

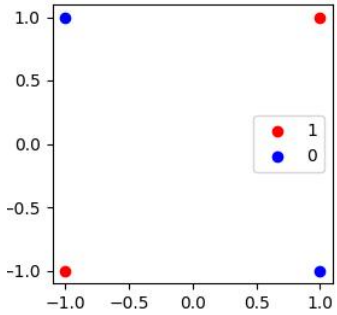

Figure 1: A training set that will not split

split value	gain ratio / mutual information
0	0*
0.1	0.1005

Table 1: Canditate splits in first feature dimension. A * denotes mutual information

split value	gain ratio / mutual information
-2	0*
-1	0.1005
0	0.0560
1	0.0058
2	0.0011
3	0.0164
4	0.0497
5	0.1112
6	0.2361
7	0.0559
8	0.4302

Table 2: Canditate splits in second feature dimension. A * denotes mutual information

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.

   To generate the decision tree views in Figures 2, 3 and 4 I am using a helper function written by stackexchange user "Joel" to determine the position of the nodes for plotting using networkx so that the tree has a hierarchical structure. This function was written as an answer to the post https://stackoverflow.com/a/29597209/2966723 and is also cited as a comment in my code.

   Figure 2 displays the decision tree learned using the data in D3leaves.txt. Each node represents a decision of the form $x_i \geq c$ where c is the spliting threshold and i is the feature dimension. Leaf nodes display $y = 0$ or $y = 1$ if the leaf corresponds to a prediction of zero or one respectively.

[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.

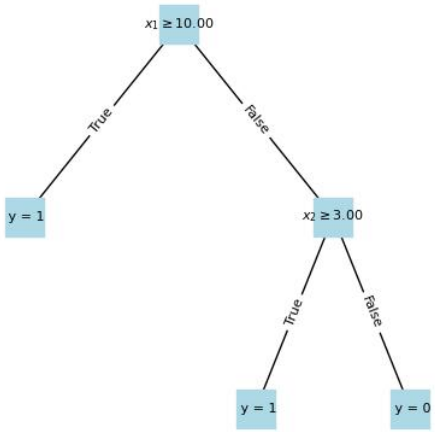

Figure 2: Learned decision tree using training data D3leaves.txt

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 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.

     The decision tree learned from the data in D1.txt is displayed in Figure 3.

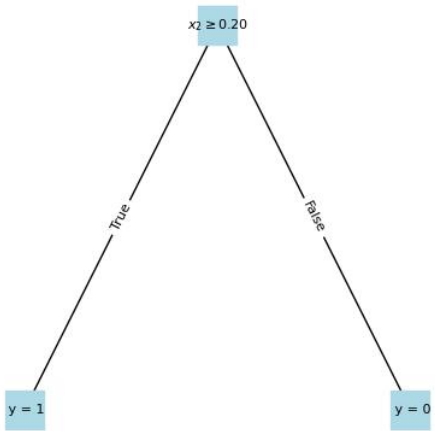

Figure 3: Learned decision tree using data in D1.txt

   - 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.

Because the tree only has one split and two leaf nodes, it is easy to interpret. The decision boundary will be the line $x_2 = 0.2$. Data for which $x_2 \geq 0.2$ will be classified as "1" and otherwise will be classified as "0".

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

  The decision tree learned from the data in D2.txt is displayed in Figure 4. It is much more complex that than that learned from the data in D1.txt.

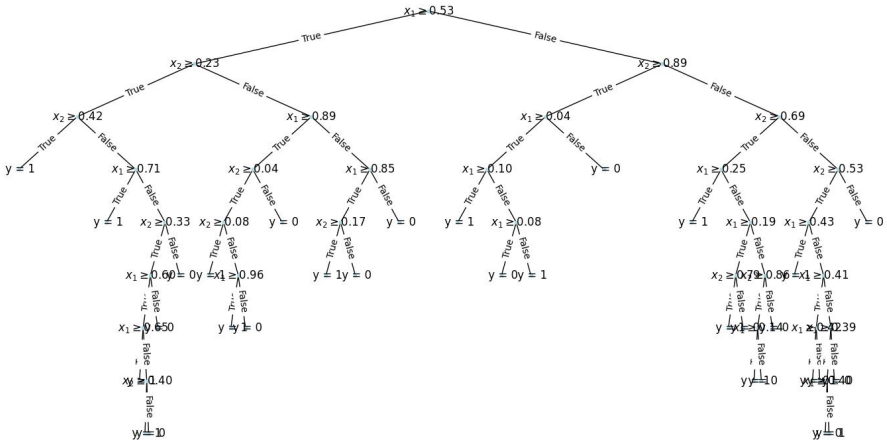

Figure 4: Learned decision tree using data in D2.txt

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

  Given the number of splits in the tree, it is very difficult to understand the decision boundary it generates. Without visualization, it is possible to interpret without visualization but much more cumbersome.

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.

   Figures 5 and 6 show scatter plots of the training data and the resulting decision boundary for trees trained with the data in D1.txt and D2.txt respectively.

   The decision tree on D1 is much smaller as the training data can be separated by a single plane $x_2 = 0.2$. It is important to notice that the right hand side of this equation is a constant. As each candidate split is of the form $x_i \geq c$ for some constant c, the training data in D1 is well-described by a model within our hypothesis space.

   Conversely, the data in D2 is divided by the line $x_2 + x_1 = 1$. However, our chosen hypothesis space does not include candidate splits along lines such as this. Therefore, our tree requires more splits to approximate this boundary.

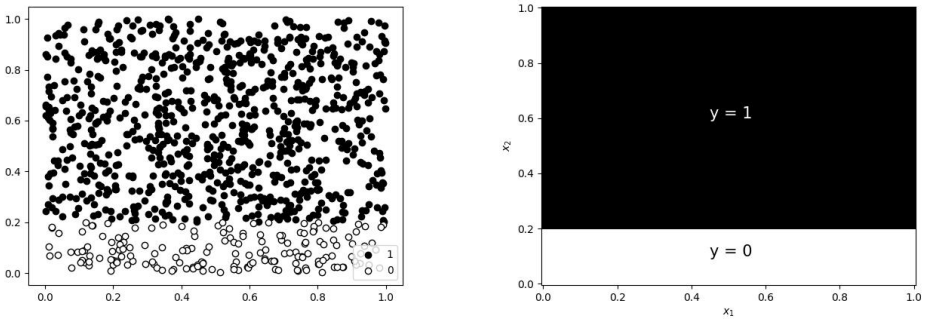

Figure 5: A scatter plot (left) of the training data and decision boundary (right) for the tree trained with data in D1.txt

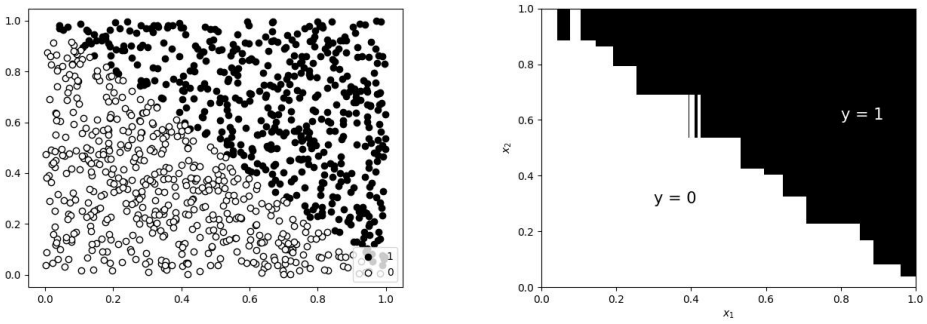

Figure 6: A scatter plot (left) of the training data and decision boundary (right) for the tree trained with data in D1.txt

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).

   Results for decision tree training with various test set sizes are below. Table 3 shows the number of nodes in the trained decision tree for each training set size. Figures 7 and 8 show the evolving decision boundary and training curve respectively.

Training Samples	Nodes
32	11
128	27
512	59
2048	129
8192	309

Table 3: Number of Decision Tree nodes for each training set size

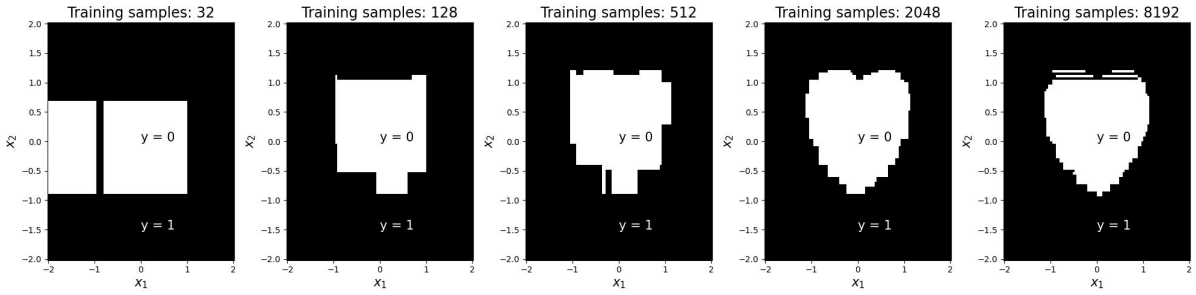

Figure 7: Learned decision boundaries for various training set sizes

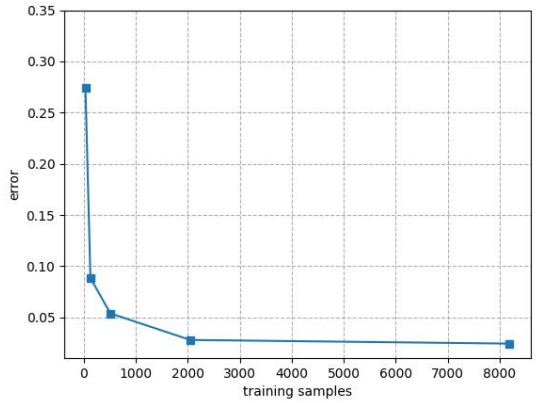

Figure 8: Training curve on Dbig

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 Samples	Nodes
32	11
128	25
512	49
2048	113
8192	237

Table 4: Number of Decision Tree nodes for each training set size

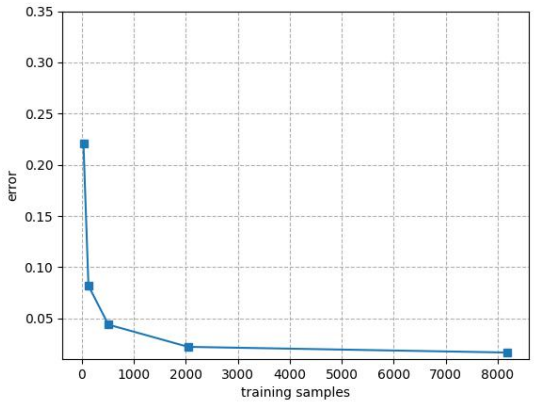

Figure 9: Training curve on Dbig using sklearn

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.

The model's train and test error are both very large without noise added to x. The log mean square error values are 325, and 328 respectively. As noise is added x, there is little improvement in the training accuracy. Remarkably however, the training accuracy improves drastically. Results are plotted in Figure 10.

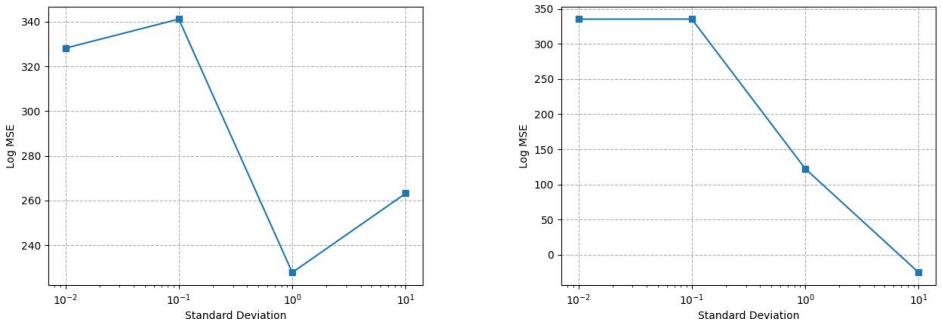

Figure 10: Plot of training accuracy (left) and test accuracy (right) as noise level is varied