HOMEWORK 2

>>Nevindu M. Batagoda<<
>>9081677594<<

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/Nevindu/CS760.HW2

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 node only has items of the same class, then the entropy of the split is 0 because the probability of one class is 1 and for the other is 0. Therefore, the information gain ratio is 0 for all possible splits. Therefore, the algorithm will stop at this node and make it a leaf.

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 following training set,

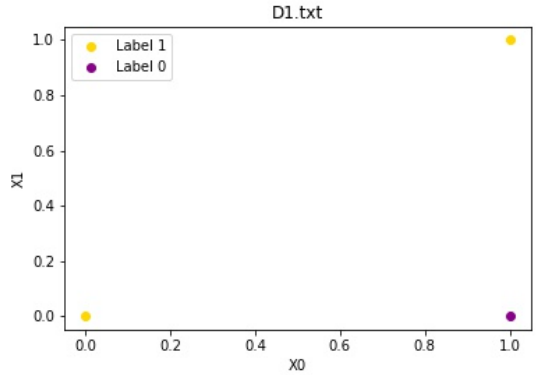


Figure 1: Example of when the algorithm stops at the root node and refuses to split

   Since we split on feature values, and data points from two classes lie on the same line the algorithm cannot cleanly split this training set. However, if we force the algorithm to split it will create a complicated tree with more nodes than training points as in the following Figure.(2).

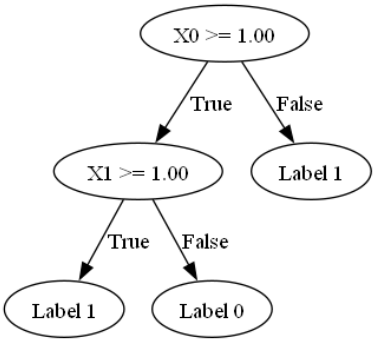


Figure 2: Decision tree for the training set in Figure.(1)

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.

The following Figure.(3) shows all the candidates for features x_0 and x_1 and their information gain ratio.

	x0 >= c	Gain Ratio	Split Entropy	InfoGain
0	0.0	0.000000	0.000000	0.000000
1	0.1	0.100518	0.439497	0.044177

(a) Information gain ratio for feature x_0

	x1 >= c	Gain Ratio	Split Entropy	InfoGain
0	0.0	0.055954	0.684038	0.038275
1	1.0	0.005780	0.845351	0.004886
2	2.0	0.001144	0.945660	0.001082
3	3.0	0.016411	0.994030	0.016313
4	4.0	0.049749	0.994030	0.049452
5	5.0	0.111240	0.945660	0.105196
6	6.0	0.236100	0.845351	0.199587
7	7.0	0.055954	0.684038	0.038275
8	8.0	0.430157	0.439497	0.189053
9	-1.0	0.100518	0.439497	0.044177
10	-2.0	0.000000	0.000000	0.000000

(b) Information gain ratio for feature x_1

Figure 3: Information gain ratio for the root node state

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.

The following Figure.(4) shows the decision tree for the given data set.

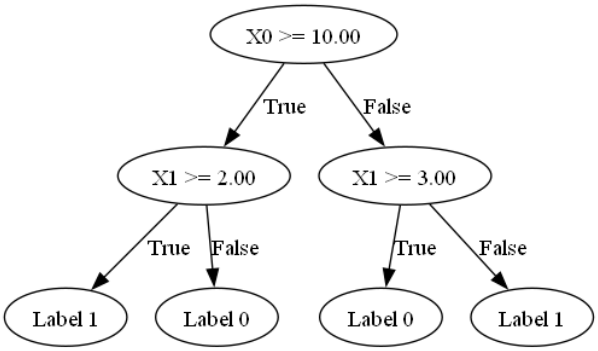


Figure 4: Decision tree for D3leaves.txt

The decision tree can be converted to the following set of rules,

- If $x_0 \geq 10$ and $x_1 \geq 2$ then $y = 1$
- If $x_0 \geq 10$ and $x_1 < 2$ then $y = 0$
- If $x_0 < 10$ and $x_1 \geq 3$ then $y = 0$
- If $x_0 < 10$ and $x_1 < 3$ then $y = 1$

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.

[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.

- 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.
- 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 following Figure.(5) shows the decision tree for D1.txt.

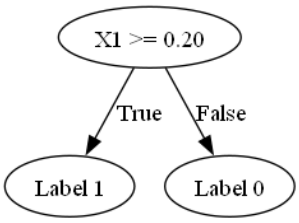


Figure 5: Decision tree for D1.txt

In the decision tree for D1.txt, every data point that has its x_1 value greater than 0.2 is classified as 1. However, every data point that has its x_1 value less than 0.2 is classified as 0. It is a very simple tree.

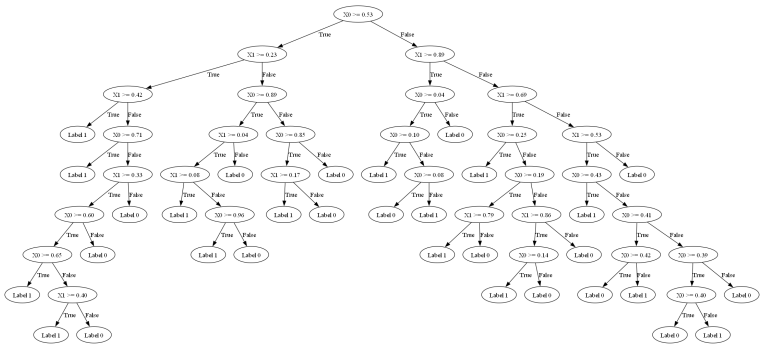


Figure 6: Decision tree for D2.txt

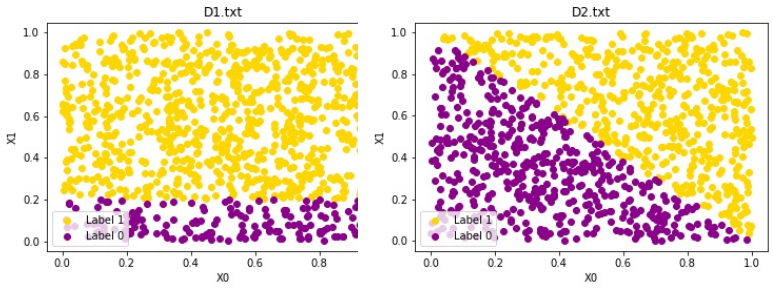
Unlike the decision tree for D1.txt which had only three total nodes, the tree for D2.txt (Figure.6) is a much complex tree with about 30 total nodes. Trying to interpret the tree by keepng track of the many branching in such a large tree is difficult. As such it is hard to get a good idea on what the decision tree is doing without the aid of visualization.

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.

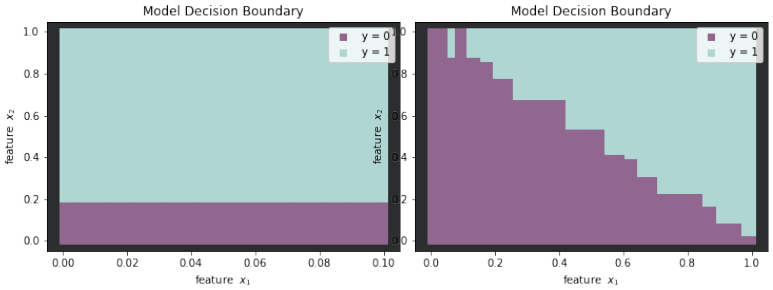
The scatter plots for D1.txt and D2.txt are as follows (Figure.7),



(a) Scatter plot for D1.txt (b) Scatter plot for D2.txt

Figure 7: Scatter plots for D1.txt and D2.txt

Their corresponding decision boundaries are as follows (Figure.8),



(a) Decision boundary for D1.txt (b) Decision boundary for D2.txt

Figure 8: Decision boundaries for D1.txt and D2.txt

The hypothesis space for our deision tree algorithm asssumes trees with single-feature and axis paralle splits. That is our decision tree tries to classify the input space by splitting it using horizontal or vertical lines based on the feature value of a single feature. According to the scatter plot of D1.txt (Figure.7a), the data is separated by a straight line. As such our decision tree can easiy separate it using a single horizontal split as seen by the decision boundary plot (Figure.8a).

However, the scatter plot of D2.txt (Figure.7b) shows that the data is separated by a sloped line. As such our decision tree cannot separate it using a single horizontal or vertical split. Therefore, it has to use multiple splits to separate the data. This can be seen in the decision boundary plot for D2.txt (Figure.8b).This is why the decision tree for D2.txt is much larger than the decision tree for 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).

(i) The following Table.(1) lists the number of nodes in the trees,

D_n	**Num. Nodes**
D_{32}	8
D_{128}	22
D_{512}	44
D_{2048}	142
D_{8192}	258

Table 1: Number of nodes in the decision trees

(ii) The following Figure.(9) shows the learning curve for the decision tree,

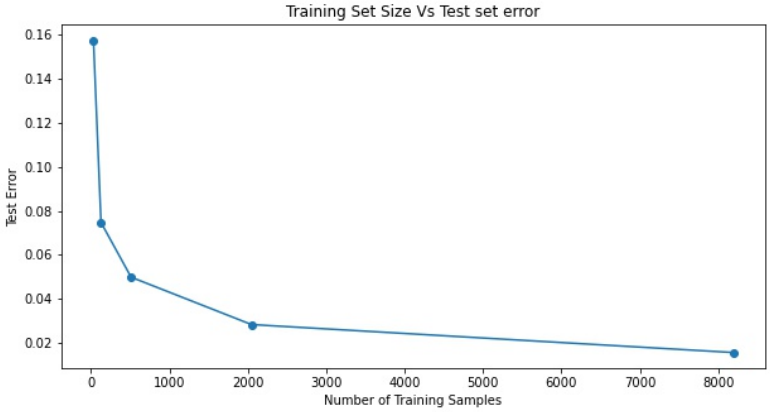
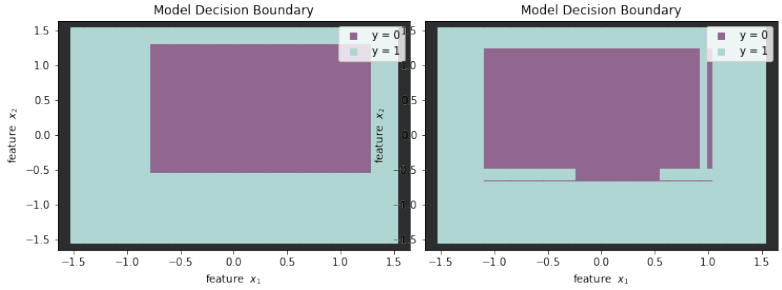
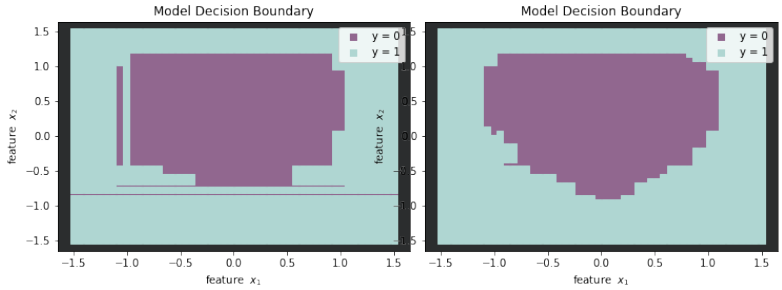


Figure 9: Learning curve for the decision trees.

(iii) The following Figures (12) shows the decision boundaries for the decision trees,



(a) Decision boundary for D_{32} (b) Decision boundary for D_{128}



(a) Decision boundary for D_{512} (b) Decision boundary for D_{2048}

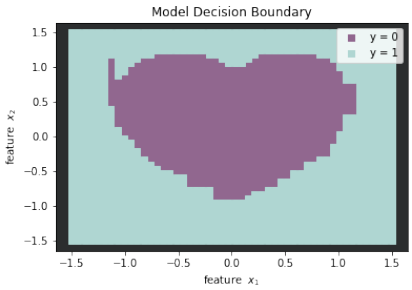


Figure 12: Decision boundary for D_{8192}

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.

(i) The following Table.(2) lists the number of nodes in the trees,

D_n	**Num. Nodes**
D_{32}	9
D_{128}	19
D_{512}	45
D_{2048}	133
D_{8192}	237

Table 2: Number of nodes in the decision trees

(ii) The following Figure.(13) shows the learning curve for the decision tree,

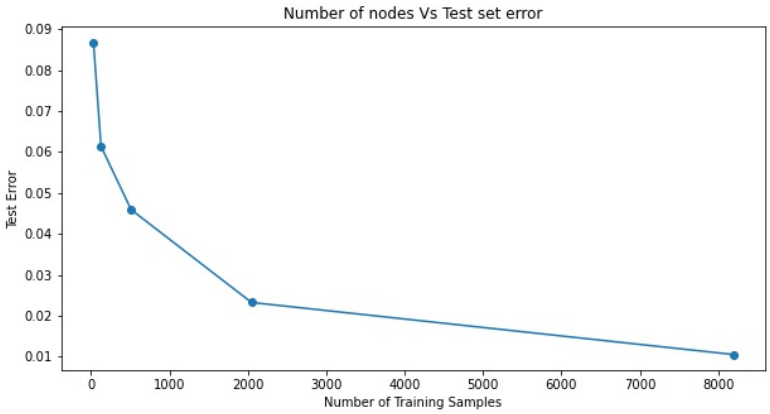


Figure 13: Learning curve for the decision trees.

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.
Before adding Gaussian noise, the test set error was $\approx 6.5e + 64$. This is because we try to ft a lagrange polynomial using $n = 100$ points. However, the lagrange interpolation is numerically unstable for $n > 20$ resulting in a very high test set error. After running the same experiment with Gaussian noise of zero mean and varying standard deviation added to the training data, we see the following test set error rates (Figure.14),

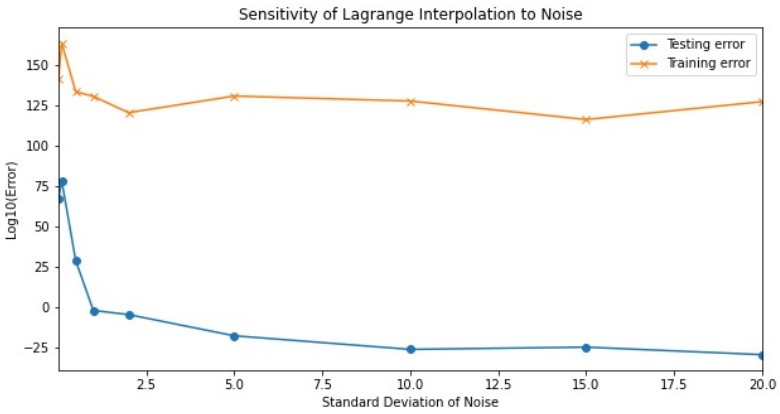


Figure 14: Test set error rates for added Gaussian noise with different standard deviations

Even with added noise, the training set error is still very high. This is because the lagrange interpolation is numerically unstable for $n > 20$. However, you can see a bit of a decreasing trend for training set error as the standard deviation of the noise increases.

For a test set of $n = 20$, we see that it starts off with a very high test set error but as the standard deviation of the added noise increases, the test set error decreases. With very high standard deviation, the test set error is very low, fitting the test set very well.