HOMEWORK 2

Venkata Saikiranpatnaik Balivada
Net ID: vbalivada
Wisc ID: 9084550244

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 $y = 1$.

The code for the algorithm and the plots can be found at https://github.com/pao214/cs760_hw2.

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.
   To prove that a pure label node becomes a leaf, we prove that such a node satisfies the stopping criteria. Let's compute the entropy when all values have the same label. Without loss of generality, let's assume that $y = 1, \forall x$ in the node. Since, all points of the node are labelled 1, we have $P(Y = 1) = 1$ and $P(Y = 0) = 0$. The value of entropy is $-\log_2(P(Y = 1)^{P(Y=1)} P(Y = 0)^{P(Y=0)}) = -\log_2(1^1 \times 0^0) = -log_2(1) = 0$.

   Before looking at the gain ratio, let's first look at information gain $H_D(Y) - H_D(Y|S)$. We already know that $H_D(Y) = 0$ for nodes with items all having the same label. Now, let's prove the same for $H_D(Y|S)$. $H_D(Y|S) = \sum_{s \in S} P(S = s) H_D(Y|S = s)$. Either the split is empty in which case $P(S = s) = 0$ or the split contains all elements of the same label, i.e. $H_D(Y|S = s) = 0$. Hence, $H_D(Y|S) = 0$ and consequently the information gain is also $0 - 0 = \boxed{0}$.

   Hence, the gain ratio of any split is zero regardless of the split S. This triggers stopping criteria $\boxed{2}$ and we construct 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.
   Our training algorithm terminates when none of the candidate splits gain in information (zero gain ratio), ignoring splits that cause empty splits. Consider the training data as mentioned in figure 1. Let's first compute the entropy of the entire data set. The points are $(x1, x2, y) = (0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 0)$. $P(Y = 1) = \frac{2}{4} = \frac{1}{2}$ and $P(Y = 0) = \frac{2}{4} = \frac{1}{2}$. Thus, entropy is $-(P(Y = 1) \log_2 P(Y = 1) + P(Y = 0) \log_2 P(Y = 0)) = -(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}) = -log_2 \frac{1}{2} = log_2 2 = \boxed{1}$.

   Now, let's compute the entropy post-split. There is only one non-trivial split on $x1$ and one on $x2$. Without loss of generality, lets consider the split on x1, $x1 >= 1$. This splits the data into $(1, 0, 1), (1, 1, 0)$ on the then branch and $(0, 0, 0), (0, 1, 1)$ on the else branch. The entropy on each split is the same and equal to $P(Y|S) = -(P(Y = 1|S) \log_2 P(Y = 1) + P(Y = 0) \log_2 P(Y = 0)) = -(\frac{1}{2} \log_2 \frac{1}{2} + \frac{1}{2} \log_2 \frac{1}{2}) = -\log_2 \frac{1}{2} = \log_2 2 = \boxed{1}$. This is also true for the split on $x2 >= 1$. All other splits are trivial and will be ignored.

   Clearly, there is no information gain on any split since the entropy stays 1 and hence the algorithm will stop and then produce a leaf node. Instead, say the algorithm actually splits on $x1 >= 1$ regardless, then the algorithm can continue and split each of the branches using $x2 >= 1$ since the splits now has an information gain.

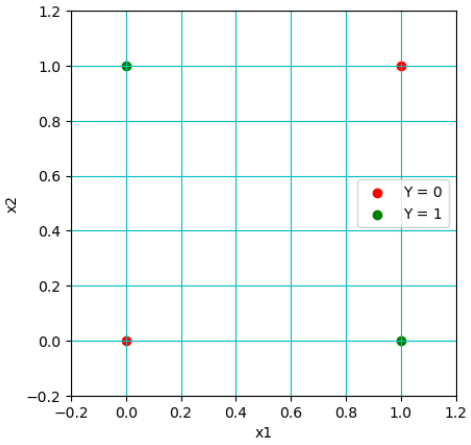


Figure 1: Our decision tree algorithm terminates immediately creating one node with label 1

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 candidate cuts j, c is listed along with the information gain (ratio with non-trivial splits) in table 1. We have zero split entropy for the smallest number on each feature dimension. It refers to the first two rows in the table. The information gain is zero, since one element in the split doesn't change anything in terms of entropy.

j	c	Gain [Ratio]	Zero Split Entropy?
0	0.1	0.101	False
0	0	0	True
1	-2	0	True
1	-1	0.101	False
1	0	0.056	False
1	1	0.006	False
1	2	0.001	False
1	3	0.016	False
1	4	0.05	False
1	5	0.111	False
1	6	0.236	False
1	7	0.056	False
1	8	0.43	False

Table 1: Information gain for each candidate split

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.

   See figure 2 for the decision tree created by the algorithm. The rules are pretty simple. Classify a new data point as label 1 when $\boxed{x_1 >= 10}$ or when $\boxed{x_2 >= 3}$. Otherwise, we output label 0.

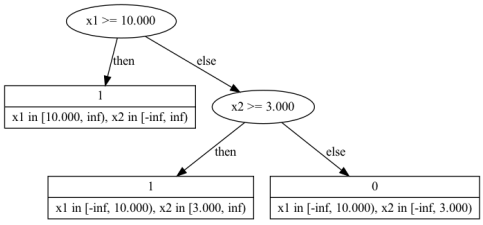


Figure 2: Decision tree on D3leaves.txt

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.

     See figure 3 for the decision tree constructed by the algorithm.

[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.

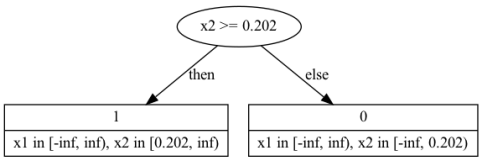


Figure 3: Decision tree on D1.txt

- 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.
  Data points with $x2$ less than 0.202 have a label of 0 or 1 otherwise. The semi-plane below the line $\boxed{x_2 = 0.202}$ is labelled 0, assuming $x1, x2$ are the x, y axes respectively.
- Build a decision tree on D2.txt. Show it to us.
  See figure 4 for the decision tree constructed by the algorithm.

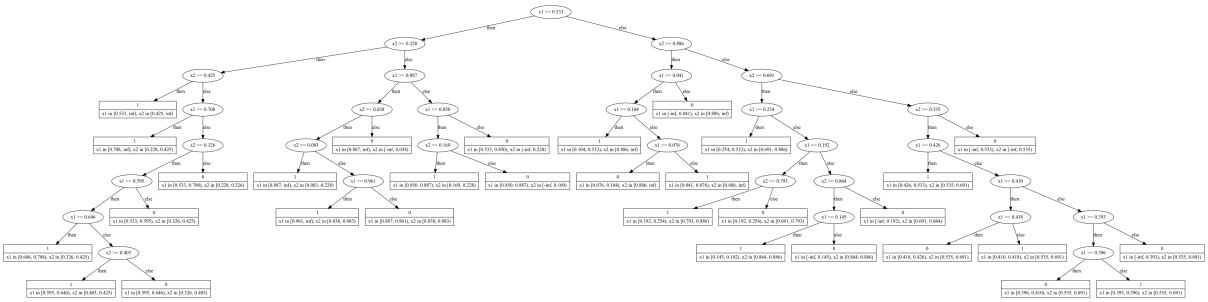


Figure 4: Decision tree on D2.txt

- Try to interpret your D2 decision tree. Is it easy or possible to do so without visualization?
  Let's pick a couple of examples to see what's happening. Some examples where the label is one are (a) $x1 \in [0.533, \infty), x2 \in [0.425, \infty)$, (b) $x1 \in [0.393, 0.396), x2 \in [0.535, 0.691)$. Some examples where the label is 0 include (a) $x1 \in (-\infty, 0.535), x2 \in (-\infty, 0.535)$, (b) $x1 \in (0.535, 0.850), x2 \in (-\infty, 0.228)$. From a cursory look, it looks like the sum of $x1, x2$ decides the label. Hence, $\boxed{x1 + x2 >= 1}$ is a good indicator for label 1, and the data takes label 0 otherwise.

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.

   See figure 5 for a plot of training data with $x1, x2$ on x, y axes respectively on training data D1.txt. The points with label 0 are colored red while the points with label 1 are colored green. The decision boundary separates the plane into red and green region for classification into label 0 and label 1 respectively. A similar plot for data D2.txt in figure 6.

   Our algorithm has a much easier time fitting D1.txt than D2.txt since our decision tree draws boundaries parallel to the axes. This follows from the fact that the decision tree nodes always branch based on one feature and such a decision cuts the hyper-space (2-D in this case) using a hyper-plane (line in this case) parallel to the remaining axes. The "true" decision boundary for D1.txt is simply $x2 >= 0.2$ which our algorithm finds pretty quickly. However, the decision boundary for D2.txt depends on both $x1, x2$ linearly such as $x1 + x2 >= 1$. Our algorithm cannot represent this since our hypothesis space only draws boundaries parallel to axes. Hence, it tries to make a best effort attempt to replicate a sloped line by drawing many $\boxed{rectangular}$ boundaries, making the depth of the tree a lot higher.

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

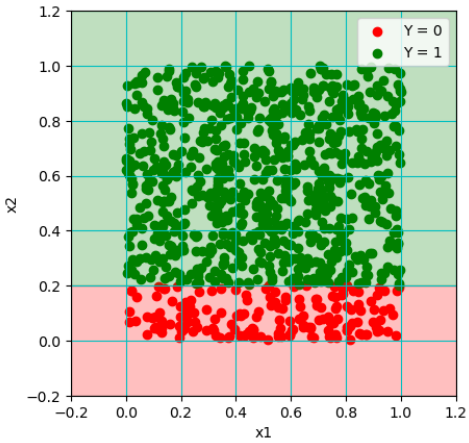


Figure 5: Decision boundary visualized on D1.txt

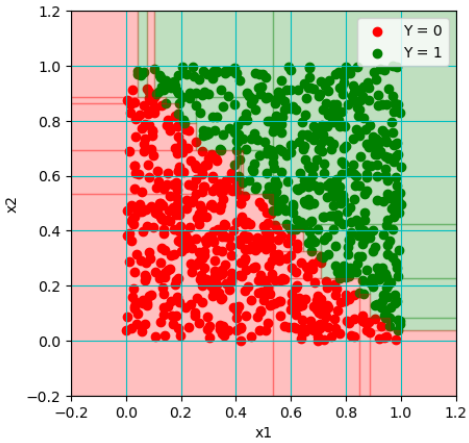


Figure 6: Decision boundary visualized on D2.txt

- 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).

The size of the tree and error is mentioned in table 2.

n	num nodes	err_n
32	7	0.190
128	23	0.101
512	53	0.040
2048	133	0.031
8192	295	0.017

Table 2: Scaling with sample size

The learning curve is plotted in the figure 7.

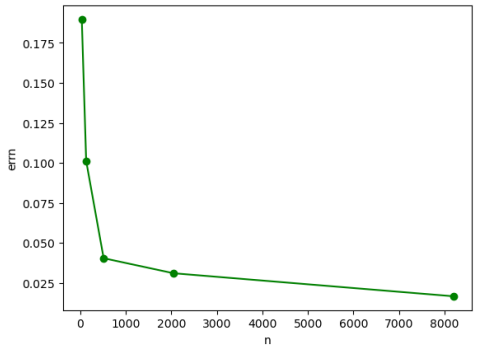
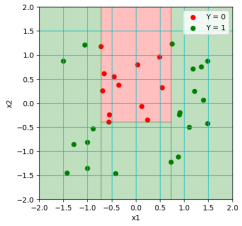
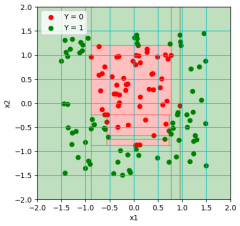


Figure 7: Learning curve

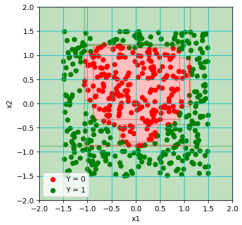
Figure 8 visualizes decision boundaries for each sample size.



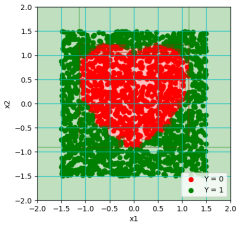
(a) Sample size 32 visualized



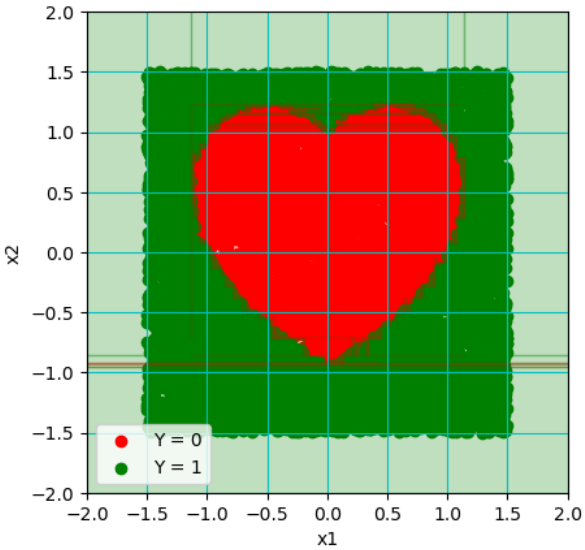
(b) Sample size 128 visualized



(c) Sample size 512 visualized



(d) Sample size 2048 visualized



(e) Sample size 8192 visualized

Figure 8: Decision boundaries for different sample sizes

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.
The size of the tree and error is mentioned in table 3.

n	num nodes	err_n
32	7	0.188
128	23	0.086
512	53	0.048
2048	115	0.024
8192	247	0.008

Table 3: Scaling skelarn with sample size

The learning curve is plotted in the figure 9.

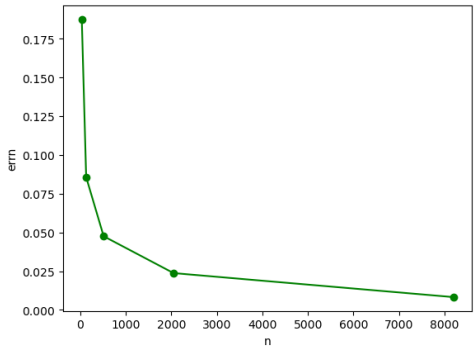


Figure 9: sklearn Learning curve

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.
Consider the following setting. We pick 100 points in the range $[-\pi, \pi]$ uniformly at random. Let's use this set of points for testing and also sample another 100 points from the same range which we are later going to spread by adding noise. The standard deviation for this noise is varied from 0 to 140 in increments of 10. We then plot both the training and test error (here error refers to log of mean squared error) in figure 10.

We make the following observations. Both the test and training error is high without noise (corresponds to noise spread = 0). This is easily explained by the numerical instability of our Lagrange polynomial fit as warned by scipy API itself. For the same reason, the training error remains high throughout. In fact, we plot the training data set, test data set, their true labels and the predicted labels for various spreads in figure 11. For the plot with no spread, see figure 11a we observe that a lot of predicted values (red dots) towards the tails cannot even be seen in the plot because they go out of bounds of the plot.

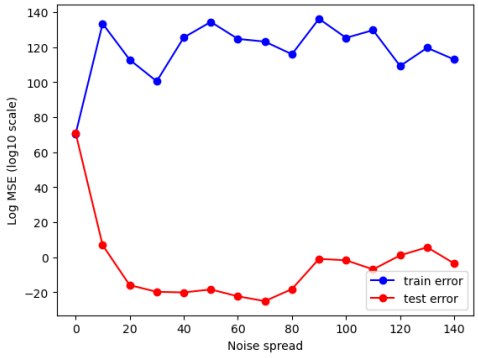
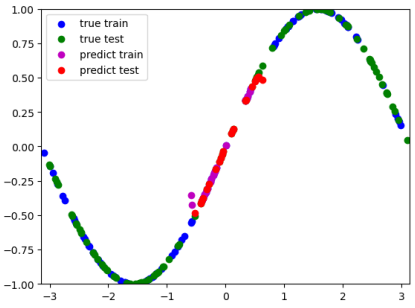


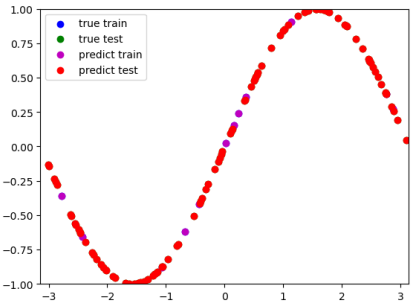
Figure 10: Train and Test error

From the testing error plot 10, we can see that the error really starts becoming minute once we have 20 iterations or so. The dataset is visualized in figure 11b and we can see right away that we can no longer see the true test values (green dots) since the predicted values (red dots) perfectly overwrite the former. We explain this sudden shift by claiming that the noise makes the polynomial function smooth near the origin.

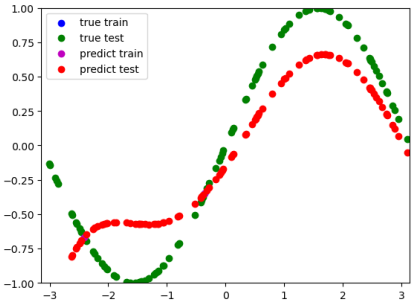
Another important point that changes the error drastically is when the number of iterations reaches and cross 90. The reason becomes apparent when we visualize this dataset in 11c. There are hardly any training points in the plot which implies that the polynomial is unable to fit the sinusoidal curve around the origin because there are hardly any points in that region to fit the polynomial properly. So, the predictions slowly start deviating from the true labels with increased variance in noise. This issue is exacerbated in figure 11d where the predicted values (red dots) are completely out of bounds.



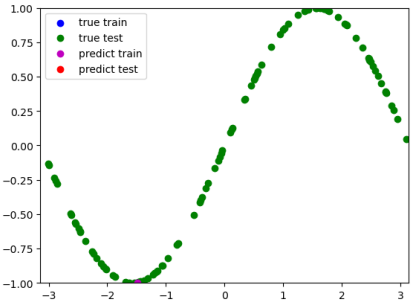
(a) Added Gaussian noise with std dev 0



(b) Added Gaussian noise with std dev 20



(c) Added Gaussian noise with std dev 90



(d) Added Gaussian noise with std dev 130

Figure 11: Visualizing the Lagrange polynomials for increasing values of noise spread.