HOMEWORK 2

Dario Placencio
907 284 6018

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.

- The Jupyter Noteebok userd for this homework can be found on this link: https://github.com/placenciohid/ECE760-Homework/blob/205d92d9c0aaa05ab6ed142faf0d3fecb14a08c5/Homework%202/Homework%202%20-%20Dario%20Placencio.ipynb

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 node contains training items with the same label, then the entropy of the node is 0. This is because entropy measures the amount of uncertainty or randomness in a set. If all the labels are the same, there is no uncertainty. The formula for entropy is:

   $$\text{entropy}(S) = -p_+ \log_2(p_+) - p_- \log_2(p_-)$$

   Where:

   - p_+ is the proportion of positive examples in S
   - p_- is the proportion of negative examples in S

   In the case where all examples have the same label, one of the proportions (either p_+ or p_-) will be 1, and the other will be 0. This results in an entropy of 0. Given our decision tree's stopping criterion, if the entropy of a node is 0, we make that node a leaf. This is because there is no benefit to splitting a node with zero entropy further: we already have a perfectly accurate classification for the training items in that 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.

   In the plotted training set, we have the two classes ('0' and '1'). Initially, if our algorithm tries to split based on the best information gain (or gain ratio), it would find that there's no effective split to separate the classes at the root since they're intertwined. Thus, the algorithm might refuse to split and decide to make the root a leaf.

   However, if we manually force a split (say, $x1 < 3$), then the algorithm can further split the data to produce a deeper tree. For the left child node with data points $\{$(1,1,0), (2,2,0)$\}$, a further split on $x1$ can perfectly separate these two points, and similarly for the right child node with data points $\{$(3,3,1), (4,4,1)$\}$.

   This example illustrates that the greedy approach of decision trees, which relies solely on metrics like information gain or gain ratio, may not always find the optimal tree, especially when the decision boundaries are not straightforward.

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.

   For the root node, considering the candidate cuts on the dataset `Druns.txt`, we can observe the following Information Gain Ratios (GR) and Information Gains (IG) for each threshold value:

   For Feature 1 (x1):

   - For a threshold of 0.1, the Information Gain (IG) is 0.0442.

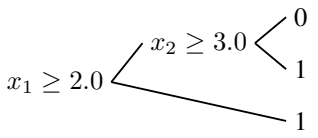
   For Feature 2 (x2):

   - For a threshold of -1.0, the Information Gain (IG) is 0.0442.
   - For a threshold of 0.0, the Gain Ratio (GR) is 0.0560.
   - For a threshold of 1.0, the Gain Ratio (GR) is 0.0058.
   - For a threshold of 2.0, the Gain Ratio (GR) is 0.0011.
   - For a threshold of 3.0, the Gain Ratio (GR) is 0.0164.

- For a threshold of 4.0, the Gain Ratio (GR) is 0.0497.
- For a threshold of 5.0, the Gain Ratio (GR) is 0.1112.
- For a threshold of 6.0, the Gain Ratio (GR) is 0.2361.
- For a threshold of 7.0, the Gain Ratio (GR) is 0.0560.
- For a threshold of 8.0, the Information Gain (IG) is 0.1891.

From the above results, we observe that the highest Gain Ratio (GR) is 0.2361 for a threshold of 6.0 on Feature 2. This would be the most promising candidate cut for the root node. It's important to note the instances where the Gain Ratio (GR) is not computed due to the split entropy being zero, and instead, the Information Gain (IG) was calculated.

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.



Rules of the Tree:

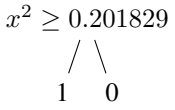
(a) IF $(x_1 \geq 2.0)$ THEN class = 1

(b) IF $(x_1 < 2.0)$ AND $(x_2 \geq 3.0)$ THEN class = 1

(c) IF $(x_1 < 2.0)$ AND $(x_2 < 3.0)$ THEN class = 0

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.



- 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.

  From the decision tree built on D1.txt, only a single split is made, with the rule being:

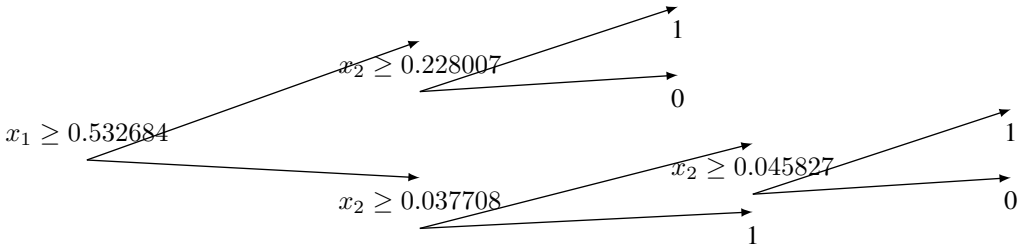
  (a) If X2 is greater than or equal to 0.201829, then the class is 1.

  (b) If X2 is less than 0.201829, then the class is 0.

  Regarding the decision boundary is difficult to interpret the total lack of knowledge of the magnitude of the features. However, I'm integred of why ended being a decimal number the threshold for the split.

- 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?

[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.



So, this case is the totally of oposit, with many splits being made. From here I could suggest that the tree is more complex, and sensitive to the features, hence the need for more splits. However, I'm not sure if this is the case, without visualizing the data, so it's difficult to interpret the decision tree.

1. (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.

   It can be observed the difference in complexity for D2 compared to D1, the distribution of the labels on D1 makes the tree quite simple, considering the strong effect of X2 on the label. On the case of D2 is the oposit, the distribution of the labels is more complex, and the effect of X1 and X2 is not as strong as in D1, hence the need for more splits.

2. (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).

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.

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?

From the test errors it can be observed an incredible size of the magnitude of the errors, almost astronomaically large, but it might be the nature of the interpolation method with a big number of points.

Repeat the experiment with zero-mean Gaussian noise ϵ added to x. Vary the standard deviation for ϵ and report your findings.

The magnitudes are still very large, but this is based on the nature of the function, and the interpolation method. But now the values are different between train and test, and the relationship with the standard deviation is positive, more deviation, more error.