HOMEWORK 2

>>SAI NIKHILESH KRISHNAMURTHY<<
>>9084624320<<

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 $x_{\cdot j} \geq c$.
- 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

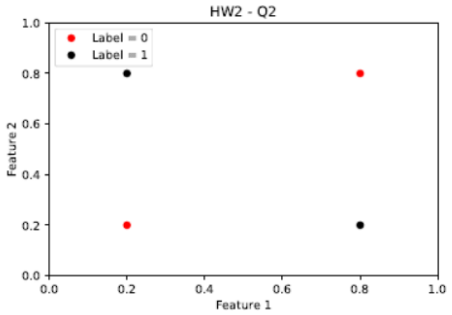
1. (Our algorithm stops at pure labels) [10 pts] If a node is not empty but contains training items with the same label, why is it guaranteed to become a leaf? Explain. You may assume that the feature values of these items are not all the same.

   Decision tree algorithm tries to maximize the information gain or minimizes entropy. If we try to force split in a node that contains items with same label, this will not be any different from any further splitting. The training items has the same labels and splitting it will result in information gain of zero. Since the information gain is 0, the algorithm will not continue and it will stop.

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.

   Assuming the dataset is similar to a square shape. This way the feature split will not be able to determine the information gain since its 50 percent chance of choosing the label .If both nodes have equal number of points of both class, it will result in information gain of zero. The algorithm will stop at the root as explained in previous question.

   x = [0.2, 0.8, 0.2, 0.8] y = [0.8, 0.2, 0.2, 0.8] z = [1, 1, 0, 0]



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.

   Candidate cuts and Mutual Information:
   "Feature 1 >= 0.1" - Mutual Information: 0.04418
   "Feature 2 >=-1.0" - Mutual Information: 0.04418
   "Feature 2 >= 0.0" - Mutual Information: 0.03827
   "Feature 2 >= 6" - Mutual Information: 0.19959
   "Feature 2 >= 7.0" - Mutual Information: 0.03827
   "Feature 2 >= 8.0" - Mutual Information: 0.18905

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.

Below is the tree representation

```
Is Feature2 >= 2.0?
--> True:
  Predict {1.0: 3}
--> False:
  Is Feature1 >= 10.0?
  --> True:
    Predict {1.0: 1}
  --> False:
    Predict {0.0: 1}
```

The root node I have taken is "Is Feature 2 (column 2) >= 2.0?". If this results is true, then it predicts the label 1 for 3 instances. But if it is false, then another node appears with the question "Is Feature 1 (column 1) >= 10.0?". If it is true, then predicts label 1 for 1 instance of the data. but if is false, take the last instance and predicts as label 0.

5. (Or is it?) [20 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.
   - 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 Decision tree for D1.txt is:

   Is Feature2 >= 0.201829?
   –>True:
   Predict 1.0: 825
   –>False:
   Predict 0.0: 175

   The Decision tree for D2.txt is:

   Is Feature1 >= 0.533076?

   –>True:

   Is Feature2 >= 0.383738?

   –>True:

[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.

Is Feature1 >= 0.550364?
–>True:
Predict 1.0: 280
–>False:
Is Feature2 >= 0.474971?
–>True:
Predict 1.0: 14
–>False:
Predict 0.0: 2
–>False:
Is Feature1 >= 0.761423?
–>True:
Is Feature2 >= 0.191206?
–>True:
Predict 1.0: 48
–>False:
Is Feature1 >= 0.90482?
–>True:
Is Feature2 >= 0.037708?
–>True:
Is Feature2 >= 0.061886?
–>True:
Predict 1.0: 17
–>False:
Is Feature2 >= 0.053702?
–>True:
Predict 0.0: 1
–>False:
Predict 1.0: 2
–>False:
Predict 0.0: 4
–>False:
Is Feature2 >= 0.169053?
–>True:
Is Feature2 >= 0.190692?
–>True:
Predict 0.0: 1
–>False:
Predict 1.0: 2
–>False:
Predict 0.0: 20
–>False:
Is Feature2 >= 0.301105?
–>True:

Is Feature1 >= 0.66337?
–>True:
Predict 1.0: 9
–>False:
Predict 0.0: 9
–>False:
Predict 0.0: 72
–>False:
Is Feature2 >= 0.639018?
–>True:
Is Feature1 >= 0.111076?
–>True:
Is Feature2 >= 0.861?
–>True:
Predict 1.0: 61
–>False:
Is Feature1 >= 0.33046?
–>True:
Predict 1.0: 31
–>False:
Is Feature2 >= 0.745406?
–>True:
Is Feature1 >= 0.254049?
–>True:
Predict 1.0: 10
–>False:
Is Feature2 >= 0.792752?
–>True:
Is Feature1 >= 0.191915?
–>True:
Predict 1.0: 3
–>False:
Predict 0.0: 3
–>False:
Predict 0.0: 7
–>False:
Predict 0.0: 14
–>False:
Is Feature2 >= 0.964767?
–>True:
Predict 1.0: 5
–>False:
Predict 0.0: 35
–>False:

Is Feature2 >= 0.534979?

–>True:

Is Feature1 >= 0.409972?

–>True:

Is Feature1 >= 0.426073?

–>True:

Predict 1.0: 9

–>False:

Is Feature2 >= 0.597713?

–>True:

Predict 1.0: 1

–>False:

Predict 0.0: 1

–>False:

Predict 0.0: 49

–>False:

Predict 0.0: 290

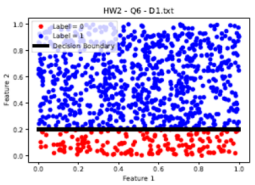
"Predict" here means how many instances of our data are in that leaf.

1.1) please see the D1.txt tree above
1.2) Thus all the points that are labeled/predicted 1 are above 0.201829 and there are 825 such instances. similarly the points below 0.201829 are labeled/predicted as 0 and there are 175 such instances.
1.3 and 1.4) Since this tree is very hard to interpret because of the large number of nodes. There are 55 nodes here. It is not possible to interpret without the visualization.

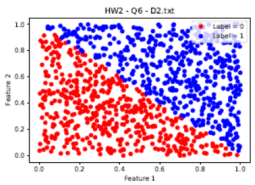
6. (Hypothesis space) [10 pts] For D1.txt and D2.txt, do the following separately:

   - 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.



D1 with its decision boundary



D2 with its dec. boundary noticed by the different colors

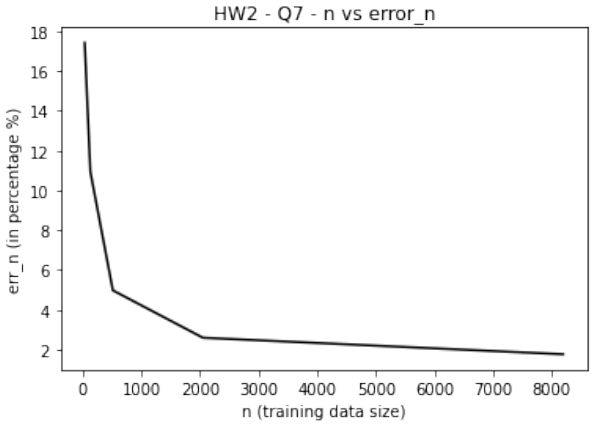
The decision boundary is the line in the plot.
The decision tree D1 is much smaller than D2 because there is just one node in D1.
For D2, the tree is way bigger since we need more nodes to do the splitting more properly. Thus the hypothesis space for D2 is much larger than D1.
we would need many splits to represent D2 as unlike D1 it has more nodes and something like a straight line cannot be used here as the tree of D2 is large/deep.

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).

2.7.1) after splitting at 8192
For n = 32 - number of nodes = 7 - err = 17.42
For n = 128 - number of nodes = 33 - err = 10.95
For n = 512 - number of nodes = 43 - err = 4.98
For n = 2048 - number of nodes = 93 - err = 2.60
For n = 8192 - number of nodes = 219 - err = 1.77

2.7.2)

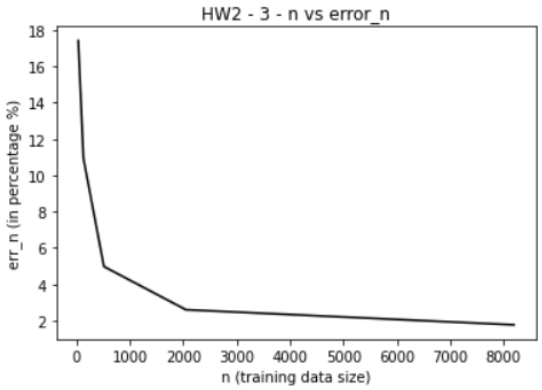


2.7.3)
please find all the plots at the end of the file
n = 32
n= 128
n=512
n=2048
n=8192

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.
Training the classifier with default hyperparameters and randomstate=0
The number of nodes are and errors are as follows:

For n = 32 - number of nodes = 9 - err = 14.15
For n = 128 - number of nodes = 15 - err = 7.79
For n = 512 - number of nodes = 63 - err = 5.97
For n = 2048 - number of nodes = 119 - err = 3.15
For n = 8192 - number of nodes = 229 - err = 1.54



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.
assuming the [a,b] is [0,10] and n=100
x = np.random.uniform(0, 10, 100)
we calculate y =sin(x)
calculating the lagrange using the scipy function, we store it in a list
similarly, this time we add noise of 0.1 to all elements in x y = sin(x) we calculate new lagrange using scipy and we store it in a list
the MSE between the two lagrange function is 1.35* 10^40
I am not sure what is the predicted label to calculate the testing and training scores.
I have however added the noise and saw that light addition of noise yields exponential and unstable results.
code is attached in the github repo.
2.7.3) The plots to 2.7.3 are below

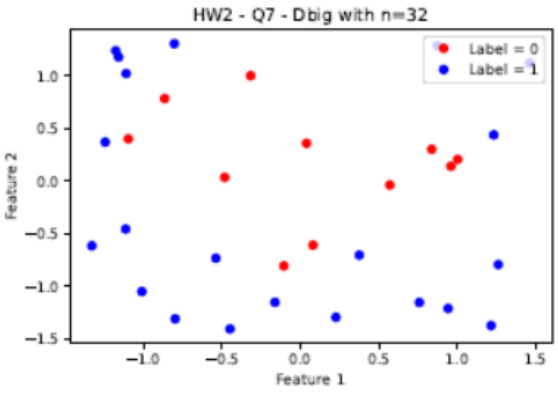
n=32

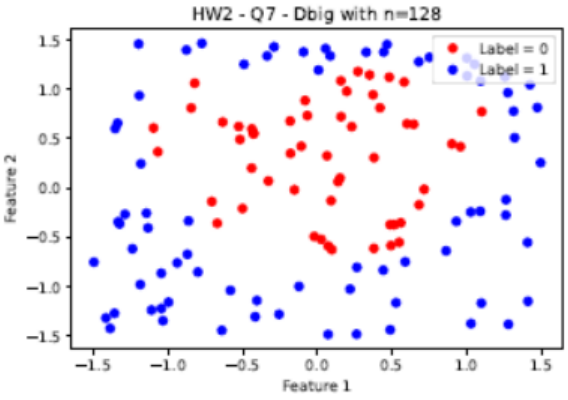
n =128

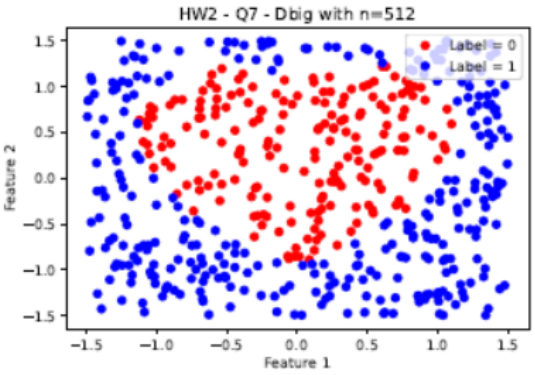
n = 512

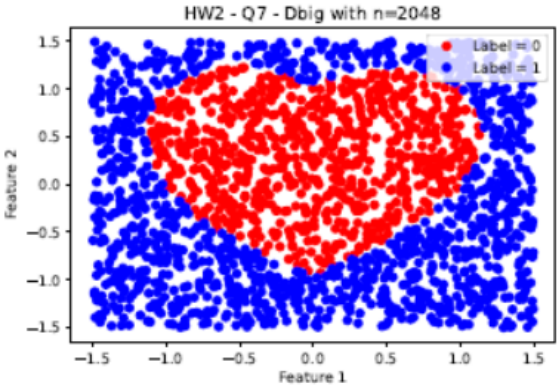
n =2048

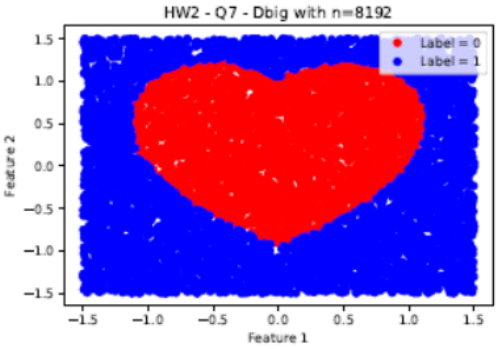
n =8192


HW2 - Q7 - Dbig with n=32
Label = 0
Label = 1
Feature 2
Feature 1
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


HW2 - Q7 - Dbig with n=128
Label = 0
Label = 1
Feature 2
Feature 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5


HW2 - Q7 - Dbig with n=512
Label = 0
Label = 1
Feature 2
Feature 1
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5






```
[214] poly1d([-1.13986037e+09, -3.55326724e+09,  5.56822096e+10,  8.15543967e+10,
              -1.37215400e+12, -2.99977905e+11,  2.08500087e+11, -8.35005149e+12,
              -5.37336996e+13,  2.02674018e+14, -8.98452144e+14, -3.03411212e+15,
               9.35977397e+15,  1.23462919e+16, -3.15652401e+16, -4.07550939e+16,
               1.27402344e+17,  1.22568986e+17, -3.66236619e+16, -3.88333934e+17,
              -2.38263381e+16, -1.15400154e+17,  8.13232033e+17, -3.81437989e+18,
               1.50923082e+18, -5.02303707e+18, -7.69397677e+18,  1.76544395e+19,
               2.25573735e+19,  2.54160360e+19, -1.15473892e+19, -7.49925219e+19,
               6.12943761e+19,  1.11425913e+20, -3.32923901e+20, -1.42780980e+20,
               6.08603975e+20,  7.02507652e+19, -4.19046336e+20, -6.95690900e+19,
               3.15870909e+20,  1.08384285e+20, -3.04066022e+20, -1.77186516e+20,
               1.29413026e+20,  3.51970115e+19, -1.16984698e+20, -7.37454012e+19,
               1.69816129e+19,  5.38661377e+19, -4.37155693e+19, -1.80585364e+19,
               1.85763701e+19,  5.46764177e+18, -1.24707893e+19, -3.37380016e+18,
               4.88922453e+18,  1.06003522e+18, -1.35130952e+18,  9.48070356e+16,
               4.94034666e+17, -2.36457271e+17, -7.25554963e+16,  6.00516765e+16,
               1.85241281e+16, -1.03792651e+16, -1.38251943e+15,  1.59187837e+15,
               9.64688886e+13, -4.12180672e+13, -3.22717869e+12,  1.09066498e+13,
               5.81312190e+12, -8.78876980e+11, -5.08541493e+11,  1.50517640e+11,
               1.07199260e+10, -4.48105687e+09,  1.67694854e+09,  8.96383375e+08,
               5.61771815e+06, -1.50802362e+07,  3.31368365e+06,  1.49111308e+06,
              -1.25081187e+05, -3.75609753e+04,  1.94799512e+04,  8.10234770e+02,
              -3.50419252e+02,  4.23255979e+01,  4.75797181e+00, -1.04990848e+00,
              -1.05447539e-01,  8.91052805e-01,  5.59068463e-01, -3.46519892e+00,
              -1.46519374e+00,  5.44273642e+00,  1.15068582e+00, -1.59056507e+00])
```

```
[218] from sklearn.metrics import mean_squared_error
      training_error = mean_squared_error(poly2,poly1)
      print(training_error)

      1.3536238979704357e+40
```