CS 5350/6350: Machine Learining Fall 2021

Homework 1

Handed out: 7 Sep, 2021
Due date: 11:59pm, 24 Sep, 2021

- You are welcome to talk to other members of the class about the homework. I am more concerned that you understand the underlying concepts. However, you should write down your own solution. Please keep the class collaboration policy in mind.
- Feel free discuss the homework with the instructor or the TAs.
- Your written solutions should be brief and clear. You need to show your work, not just the final answer, but you do *not* need to write it in gory detail. Your assignment should be **no more than 20 pages**. Not that you do not need to include the problem description. Every extra page will cost a point.
- Handwritten solutions will not be accepted.
- *Your code should run on the CADE machines.* You should include a shell script, `run.sh`, that will execute your code in the CADE environment. Your code should produce similar output to what you include in your report.

  You are responsible for ensuring that the grader can execute the code using only the included script. If you are using an esoteric programming language, you should make sure that its runtime is available on CADE.
- Please do not hand in binary files! We will *not* grade binary submissions.
- The homework is due by **midnight of the due date**. Please submit the homework on Canvas.
- Note the bonus questions are for **both 5350 and 6350** students. If a question is mandatory for 6350, we will highlight it explicitly.

1 Decision Tree [40 points + 10 bonus]

1. [7 points] Decision tree construction.

   (a) [5 points] Use the ID3 algorithm with information gain to learn a decision tree from the training dataset in Table 1. Please list every step in your tree construction, including the data subsets, the attributes, and how you calculate the information gain of each attribute and how you split the dataset according to the selected attribute. Please also give a full structure of the tree. You can manually draw the tree structure, convert the picture into a PDF/EPS/PNG/JPG format and include it in your homework submission; or instead, you can represent the tree with a conjunction of prediction rules as we discussed in the lecture.

   Answer

x_1	x_2	x_3	x_4	y
0	0	1	0	0
0	1	0	0	0
0	0	1	1	1
1	0	0	1	1
0	1	1	0	0
1	1	0	0	0
0	1	0	1	0

Table 1: Training data for a Boolean classifier

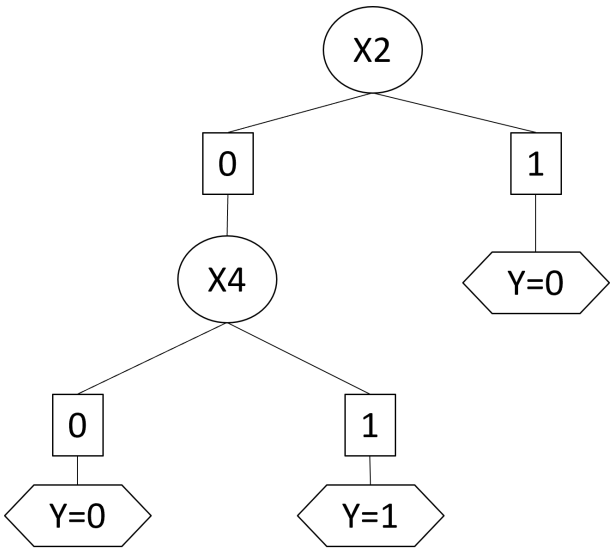

Figure 1: Decision tree for question 1a.

I used Python to create the information gain function. You can find the code in the GitHub repository in QuestionAnswers.part1.q1a for where I called the necessary functions, and you can find the functions themselves in DecisionTree.error_calcs. You can also find the functions used in the Appendix. In general, I calculate $H(y)$ by calculating the entropy of y (on which I would calculate the probabilities of y with the calc_discrete_probability function) with the calc_entropy function. Then, to find the gain, I use the calc_gain function with the given x_i and y. By default, calc_entropy is used to calculate the individual errors, so I did not specify that here. With this approach, I find the following results.

$H(y) = 0.863120568566631$
$Gain(x_1) = 0.061743357932800724$
$Gain(x_2) = 0.46956521111470695$
$Gain(x_3) = 0.0059777114237739015$
$Gain(x_4) = 0.46956521111470695$

$x_2 = x_4$, for maximum gain, so we choose x_2 as it comes first. Examining the table, we see that if $x_2 = 1$, then $y = 0$. However, when $x_2 = 0$, there is a split in the data. This split is laid out in Table 2, where $x_2 = 0$

x_1	x_3	x_4	y
0	1	0	0
0	1	1	1
1	0	1	1

Table 2: Data subset where $x_2 = 0$

$H(y) = 0.9182958340544896$
$Gain(x_1) = 0.2516291673878229$
$Gain(x_3) = 0.2516291673878229$
$Gain(x_4) = 0.9182958340544896$

Clearly want to split on x_4. When $x_4 = 0$ then $y = 0$, and when $x_4 = 1$ then $y = 1$. The decision tree is depicted in Figure 1.

(b) [2 points] Write the boolean function which your decision tree represents. Please use a table to describe the function — the columns are the input variables and label, i.e., x_1, x_2, x_3, x_4 and y; the rows are different input and function values.

Answer

x_2	x_4	y
0	0	0
0	1	1
1	N/A	0

Table 3: Function for Decision Tree model

Function summary is that if $x_2 = 1$ then $y = 0$. However, if $x_2 = 0$, then check x_4. If $x_4 = 0$ then $y = 0$, but if $x_4 = 1$ then $y = 1$. Function is also summarized in Table 3.

2. [17 points] Let us use a training dataset to learn a decision tree about whether to play tennis (**Page 43, Lecture: Decision Tree Learning**, accessible by clicking the link http://www.cs.utah.edu/~zhe/teach/pdf/decision-trees-learning.pdf). In the class, we have shown how to use information gain to construct the tree in ID3 framework.

(a) [7 points] Now, please use majority error (ME) to calculate the gain, and select the best feature to split the data in ID3 framework. As in problem 1, please list every step in your tree construction, the attributes, how you calculate the gain of each attribute and how you split the dataset according to the selected attribute. Please also give a full structure of the tree.

Answer

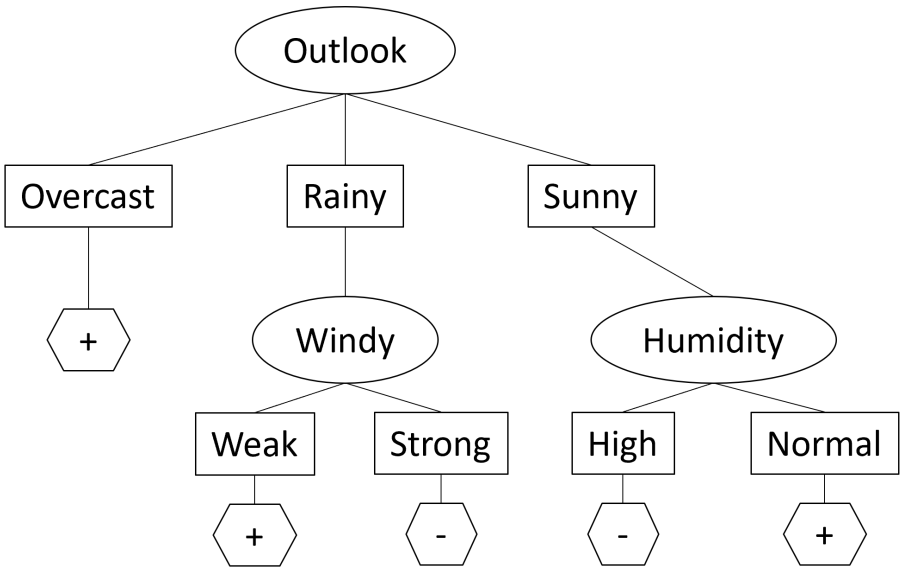

Figure 2: Decision tree for questions 2a, 2b, and 3d. Despite all three questions using different methods, they end up with the same tree.

The approach is similar as in 1a, except we will pass the calc_majority_error function into the calc_gain function. The calc_majority_error function requires a little explanation, as opposed to the calc_entropy or calc_gini_index functions, whose code explain themselves sufficiently. Majority error (ME), as a concept, means that all we care about is what is which feature has the maximum probability. If we know that, then because all probabilities MUST sum to 1, we can simply take the difference to calculate the majority error. If there is only one available probability, we will assume it is 1, and simply return 0 as the majority error. This 'if' statement is probably not strictly required in the code, since if a set of probabilities has only one result, then one would expect it to be a probability of 1, but we will keep it here for now. These results can be replicated with QuestionAnswers.part1.q2a.

play? ME: 0.35714285714285715
outlook gain: 0.0714285714285714
temperature gain: 5.551115123125783e-17
humidity gain: 0.07142857142857145
windy gain: 5.551115123125783e-17

Can split on either outlook or humidity, but will split on Outlook for consistency

Data from Outlook split:
Overcast: [1 1 1 1]
Rainy: [1 1 0 1 0]

Sunny: [0 0 0 1 1]
Clearly, when outlook = overcast, then play = 1, i.e. +.

Leaf when outlook = rainy
play? when outlook = rainy ME: 0.4
temperature gain: 0.0
humidity gain: 0.0
windy gain: 0.4

Clearly, split on windy which yields the following data.

Strong: [0, 0]
Weak: [1, 1, 1]

Can see from data, when windy = Weak, then play = 1, and when windy = Strong, then play = 0

Leaf when outlook = sunny

play? when o = s ME: 0.4
temperature gain: 0.2
humidity gain: 0.4
windy gain: 0.0

Clearly, split on humidity

High: [0, 0, 0]
Normal: [1, 1]

Can see from data, when humidity = High, then play = 0, and when humidity = Normal, then play = 1

The decision tree is summarized in Figure 2.

(b) [7 points] Please use gini index (GI) to calculate the gain, and conduct tree learning with ID3 framework. List every step and the tree structure.

Answer

The approach is similar as in 1a and 2a, except we will pass the calc_gini_index function into the calc_gain function. The results can be replicated with QuestionAnswers.part1.q2b.

play? GI: 0.4591836734693877

outlook gain: 0.11632653061224485
temperature gain: 0.018707482993197244
humidity gain: 0.09183673469387743
windy gain: 0.030612244897959162

Outlook has highest gain, so split there.

Overcast: [1 1 1 1]
Rainy: [1 1 0 1 0]
Sunny: [0 0 0 1 1]

Clearly, when outlook = overcast, then play = 1.

Leaf when outlook = rainy
play? when o = r GI: 0.4
temperature gain: 0.013333333333333308
humidity gain: 0.013333333333333308
windy gain: 0.48

Clearly, split on windy

Strong: [0, 0]
Weak: [1, 1, 1]

Can see from data, when windy = w, then play? = 1, and when windy = s, then play = 0.

Leaf when outlook = sunny
play? when o = s ME: 0.4
temperature gain: 0.27999999999999997
humidity gain: 0.48
windy gain: 0.013333333333333308

Clearly, split on humidity

High: [0, 0, 0]
Normal: [1, 1]

Can see from data, when humidity = High, then play = 0, and when humidity = Normal, then play = 1.

Final decision tree is summarized in Figure 2.

(c) [3 points] Compare the two trees you just created with the one we built in the class (see Page 62 of the lecture slides). Are there any differences? Why?

Answer

The trees seem to be the same, which to me indicates that these three approaches (Entropy, Gini Index, Majority Error) are equivalent. The specific numerical results are, of course, different, but they seem to all serve well as methods of information gain, and so the question then becomes which ones will work better, on average? Which ones seem to be more scalable? This question will be explored more in Part 2.

3. [16 points] Continue with the same training data in Problem 2. Suppose before the tree construction, we receive one more training instance where Outlook's value is missing: {Outlook: Missing, Temperature: Mild, Humidity: Normal, Wind: Weak, Play: Yes}.

(a) [3 points] Use the most common value in the training data as the missing value, and calculate the information gains of the four features. Note that if there is a tie for the most common value, you can choose any value in the tie. Indicate the best feature.

Answer

Determining the most common value can be done simply by putting the data into a Pandas Series and using the mode function then selecting the first index of the output. Then, simply calculate the information gain in Python as we have done before. This is done in QuestionAnswers.part1.q3a. Doing so, we find the following results.

$Gain(outlook) = 0.2273273022811375$
$Gain(temperature) = 0.032498735534292944$
$Gain(humidity) = 0.168621667532054$
$Gain(windy) = 0.0597731301493174$

As before, the best feature is obviously Outlook.

(b) [3 points] Use the most common value among the training instances with the same label, namely, their attribute "Play" is "Yes", and calculate the information gains of the four features. Again if there is a tie, you can choose any value in the tie. Indicate the best feature.

Answer

The method here is largely the same as in 3a, except we do those same calculations on the subset of values with the same y value as in the "missing" index. This yields the following results. This is done in QuestionAnswers.part1.q3b.

$Gain(outlook) = 0.27099543775137724$
$Gain(temperature) = 0.032498735534292944$
$Gain(humidity) = 0.168621667532054$

$Gain(windy) = 0.0597731301493174$

Again, Outlook is obviously the best feature.

(c) [3 points] Use the fractional counts to infer the feature values, and then calculate the information gains of the four features. Indicate the best feature.

Answer

By default, I have coded the calc_gain and calc_discrete_probability functions to use fractional counts in the cases where nulls are not handled before passing in the x an y arrays. These functions can be found in Listing 1 of the Appendix.

Basically, I check for the number of null values in the X array, and calculate for each unique x value (excluding nulls) what fraction of the X array that value represents. I then multiply the number of null values by the fraction to get a normalized number for the nulls filled in with the fractional result. Again, for a full understanding of how this works out, please check the calc_gain and calc_discrete_probability functions in Listing 1 of the Appendix.

This question's results are calculated in QuestionAnswers.part1.q3c, yielding the following results.

$Gain(outlook) = 0.22444415770980708$
$Gain(temperature) = 0.032498735534292944$
$Gain(humidity) = 0.168621667532054$
$Gain(windy) = 0.0597731301493174$

Yet again, Outlook is the best feature.

(d) [7 points] Continue with the fractional examples, and build the whole free with information gain. List every step and the final tree structure.

Answer

Based on 3c, we clearly want to split on outlook. By subsetting the table, we can see that when Outlook is overcast, then we will play. However, if the outlook is rainy or sunny, then we need to split again. The code to generate these results can be found in QuestionAnswers.part1.q3d.

For the sunny node, we find the following gains.
$Gain(temperature) = 0.5709505944546686$
$Gain(humidity) = 0.9709505944546686$
$Gain(windy) = 0.01997309402197489$

Clearly, split on humidity.

High: [0, 0, 0]

Normal: [1, 1]

Can see from data, when humidity = High, then play = 0, and when humidity = Normal, then play = 1.

For the rainy node, we find the following gains.
$Gain(temperature) = 0.01997309402197489$
$Gain(humidity) = 0.01997309402197489$
$Gain(windy) = 0.9709505944546686$

Clearly, split on windy.

Strong: [0, 0]
Weak: [1, 1, 1]

Can see from data, when windy = w, then play = 1, and when windy = s, then play = 0.

The Decision Tree is summarized in Figure 2.

4. [**Bonus question 1**] [5 points]. Prove that the information gain is always non-negative. That means, as long as we split the data, the purity will never get worse! (Hint: use convexity)

   To do this, simply find the Hessian. If we can show it is positive semi-definite, then obviously the information gain is always non-negative.

$$H(X) = -\sum_{i=1}^{n} x_i ln(x_i)$$

$$\frac{dH(X)}{dx_i} = -\frac{x_i}{x_i} - ln(x_i) = -(1 + ln(x_i))$$

$$\frac{d^2H(X)}{dx_i^2} = -\frac{1}{x_i}$$

$$\frac{d^2H(X)}{dx_i x_j} = 0$$

   Therefore, $trace(H(X)) = -\sum_{i=1}^{n} \frac{1}{x_i}$, and all other values are 0. This is concave, so as x increases, the entropy decreases.

   ANSWER INCOMPLETE.

5. [**Bonus question 2**] [5 points]. We have discussed how to use decision tree for regression (i.e., predict numerical values) — on the leaf node, we simply use the average of the (numerical) labels as the prediction. Now, to construct a regression tree, can you invent a gain to select the best attribute to split data in ID3 framework?

   Answer

   My initial instinct would be to use Variance or Euclidean Distance.

   ANSWER INCOMPLETE.

2 Decision Tree Practice [60 points]

1. [5 Points] Starting from this assignment, we will build a light-weighted machine learning library. To this end, you will first need to create a code repository in Github.com. Please refer to the short introduction in the appendix and the official tutorial to create an account and repository. Please commit a README.md file in your repository, and write one sentence: "This is a machine learning library developed by **Your Name** for CS5350/6350 in University of Utah". You can now create a first folder, "DecisionTree". Please leave the link to your repository in the homework submission. We will check if you have successfully created it.

   Answer

   Can find repository here: `https://github.com/Paul-Wissler/cs-6350-hw1`

2. [30 points] We will implement a decision tree learning algorithm for car evaluation task. The dataset is from UCI repository(`https://archive.ics.uci.edu/ml/datasets/car+evaluation`). Please download the processed dataset (car.zip) from Canvas. In this task, we have 6 car attributes, and the label is the evaluation of the car. The attribute and label values are listed in the file "data-desc.txt". All the attributes are categorical. The training data are stored in the file "train.csv", consisting of $1,000$ examples. The test data are stored in "test.csv", and comprise 728 examples. In both training and test datasets, attribute values are separated by commas; the file "data-desc.txt" lists the attribute names in each column.

   Note: we highly recommend you to use Python for implementation, because it is very convenient to load the data and handle strings. For example, the following snippet reads the CSV file line by line and split the values of the attributes and the label into a list, "terms". You can also use "dictionary" to store the categorical attribute values. In the web are numerous tutorials and examples for Python. if you have issues, just google it!

```
with open(CSVfile, 'r') as f:
    for line in f:
        terms = line.strip().split(',')
        process one training example
```

(a) [15 points] Implement the ID3 algorithm that supports, information gain, majority error and gini index to select attributes for data splits. Besides, your ID3 should allow users to set the maximum tree depth. Note: you do not need to convert categorical attributes into binary ones and your tree can be wide here.

Answer

Code for the ID3 algorithm can be found in the DecisionTree folder of the provided repository. Functions for calculating information gain, gini index, and majority error can be found in error_calcs.py, and code for the ID3 algorithm itself can be found in make_decision_tree.py. I chose to use native Python dictionaries to store the decision tree structure, reasoning that this would likely have the fastest look-up speed and most efficient storage, not to mention that it would be trivial to export it to a JSON file for long-term storage. The decision tree is constructed by a Python class called DecisionTreeModel, which constructs the model with the make_decision_tree method.

(b) [10 points] Use your implemented algorithm to learn decision trees from the training data. Vary the maximum tree depth from 1 to 6 — for each setting, run your algorithm to learn a decision tree, and use the tree to predict both the training and test examples. Note that if your tree cannot grow up to 6 levels, you can stop at the maximum level. Report in a table the average prediction errors on each dataset when you use information gain, majority error and gini index heuristics, respectively.

Answer

These results can be replicated with QuestionAnswers.part2.q2b.

Data	En	GI	ME
Training	0.1915	0.1928	0.1957
Test	0.2214	0.2218	0.2353

Table 4: Q2b. Average errors returned from decision tree with maximum depth varied from 1 to 6 for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches.

(c) [5 points] What can you conclude by comparing the training errors and the test errors?

Answer

In all cases, the model is able to predict the training data better than the test data. This makes sense because conceptually we expect the model will fit the noise we see in the training data. In fact, if you check Table 7 in the Appendix, we see that at the maximum depth, the model fits the training data perfectly (100%, no errors).

3. [25 points] Next, modify your implementation a little bit to support numerical attributes. We will use a simple approach to convert a numerical feature to a binary one. We choose the media (NOT the average) of the attribute values (in the training set) as the threshold, and examine if the feature is bigger (or less) than the threshold. We

will use another real dataset from UCI repository(`https://archive.ics.uci.edu/ml/datasets/Bank+Marketing`). This dataset contains 16 attributes, including both numerical and categorical ones. Please download the processed dataset from Canvas (bank.zip). The attribute and label values are listed in the file "data-desc.txt". The training set is the file "train.csv", consisting of $5,000$ examples, and the test "test.csv" with $5,000$ examples as well. In both training and test datasets, attribute values are separated by commas; the file "data-desc.txt" lists the attribute names in each column.

(a) [10 points] Let us consider "unknown" as a particular attribute value, and hence we do not have any missing attributes for both training and test. Vary the maximum tree depth from 1 to 16 — for each setting, run your algorithm to learn a decision tree, and use the tree to predict both the training and test examples. Again, if your tree cannot grow up to 16 levels, stop at the maximum level. Report in a table the average prediction errors on each dataset when you use information gain, majority error and gini index heuristics, respectively.

Answer

Results can be replicated with QuestionAnswers.part2.q3a.

Data	En	GI	ME
Training	0.0459	0.0440	0.0492
Test	0.1617	0.1695	0.1737

Table 5: Q3a. Average errors returned from decision tree with maximum depth varied from 1 to 16 for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches when "unknown" is considered to be an attribute value.

(b) [10 points] Let us consider "unknown" as attribute value missing. Here we simply complete it with the majority of other values of the same attribute in the training set. Vary the maximum tree depth from 1 to 16 — for each setting, run your algorithm to learn a decision tree, and use the tree to predict both the training and test examples. Report in a table the average prediction errors on each dataset when you use information gain, majority error and gini index heuristics, respectively.

Answer

Results can be replicated with QuestionAnswers.part2.q3b.

Data	En	GI	ME
Training	0.0531	0.0521	0.0566
Test	0.1560	0.1607	0.1659

Table 6: Q3b. Average errors returned from decision tree with maximum depth varied from 1 to 16 for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches when "unknown" is replaced with the mode of the attribute.

(c) [5 points] What can you conclude by comparing the training errors and the test errors, with different tree depths, as well as different ways to deal with "unknown" attribute values?

Answer

Keeping the "unknown" as attribute values meant that the model performed better on the training data, but worse on the test data. This is because the "unknown" values were noise which the model was fitting. This clearly demonstrates that handling the "unknown" values will result in better model performance. As seen in the Appendix, Tables 9, 10, 11, 12 show that there is not a strictly linear increase in model performance with increasing tree depth, rather the model's performance on test data would peak at some depth (usually 3 or 4) and then dip in performance. This is likely because after some point, the specificity of the model began to fit the noise of the training data. In other words, we can see that to make the model generally powerful, we need to cap off its depth and not simply allow it to go as deep as possible.

3 Appendix

3.1 Code

Listing 1: DecisionTree.error_calcs

```
import math
import numpy as np
import pandas as pd


def calc_entropy(set_array: pd.Series, unique_outcomes=2) -> float:
    h = list()
    for p in set_array:
        msg = f'Probability was {p}. Probabilities may not exceed 1.'
        if p > 1:
            raise ValueError(msg)
        elif p == 0:
            h.append(0)
        else:
            h.append(-p * logn(p, 2))
    return sum(h)


def calc_gini_index(set_array: pd.Series, unique_outcomes=2) -> float:
    h = list()
    for p in set_array:
        msg = f'Probability was {p}. Probabilities may not exceed 1.'
```

```
        if p > 1:
            raise ValueError(msg)
        else:
            h.append(-p**2)
    return 1 + sum(h)


def calc_majority_error(set_array: pd.Series, unique_outcomes=2) -> float:
    if len(set_array) == 1:
        return 0
    return 1 - set_array.max()


def calc_gain(x: pd.Series, y: pd.Series, f=calc_entropy) -> float:
    if isinstance(x, np.ndarray):
        x = pd.Series(x)

    if isinstance(y, np.ndarray):
        y = pd.Series(y)

    is_y_nan = y.isna()
    x = x[~is_y_nan]
    y = y[~is_y_nan]

    H_y = f(calc_discrete_probability(y),
        unique_outcomes=len(np.unique(y)))
    s = len(y)
    e = list()

    x_no_nans = x[~x.isna()]
    count_x_no_nans = len(x_no_nans)
    for v in np.unique(x_no_nans):
        is_x_eq_v = np.where(x==v)
        frac_of_x = len(x.loc[is_x_eq_v]) / count_x_no_nans
        null_x_v = len(x[x.isna()]) * frac_of_x

        null_y_v = y.loc[x.isna()]

        s_v = len(x.loc[is_x_eq_v]) + null_x_v
        y_v = y.loc[is_x_eq_v]
        prob_y_v = calc_discrete_probability(y_v,
            null_x_v=null_y_v, null_frac=frac_of_x)
        e.append(s_v / s * f(prob_y_v,
            unique_outcomes=len(np.unique(y))))
    return H_y - sum(e)
```

```
def calc_discrete_probability(x: np.array, null_x_v=pd.Series([]),
        null_frac=0) -> float:
    unique_vals = np.unique(x)
    null_size = len(null_x_v) * null_frac
    p = pd.Series([])
    for i in unique_vals:
        num = sum(x==i) + null_frac * sum(null_x_v==i)
        p[i] = num / (len(x) + null_size)
    return p


def logn(x, n) -> float:
    return math.log(x) / math.log(n)
```

3.2 Misc. Tables

Depth	En	GI	ME
1	0.698	0.698	0.698
2	0.698	0.698	0.698
3	0.698	0.698	0.698
4	0.811	0.803	0.790
5	0.946	0.946	0.942
6	1.000	1.000	1.000
Avg	0.8085	0.80716667	0.8043332

Table 7: Q2b. Data returned from decision tree for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches. Predicted on Training data.

Depth	En	GI	ME
1	0.703297	0.703297	0.703297
2	0.703297	0.703297	0.703297
3	0.703297	0.703297	0.703297
4	0.809066	0.806319	0.791209
5	0.877747	0.877747	0.846154
6	0.875000	0.875000	0.840659
Avg	0.778617216	0.7781593	0.764652

Table 8: Q2b. Data returned from decision tree for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches. Predicted on Test data.

Depth	En	GI	ME
1	0.8808	0.8808	0.8808
2	0.8808	0.8912	0.8912
3	0.8950	0.9006	0.8988
4	0.9138	0.9202	0.9134
5	0.9370	0.9386	0.9304
6	0.9500	0.9516	0.9422
7	0.9640	0.9650	0.9508
8	0.9698	0.9714	0.9608
9	0.9760	0.9784	0.9672
10	0.9814	0.9820	0.9726
11	0.9852	0.9848	0.9784
12	0.9862	0.9862	0.9826
13	0.9864	0.9864	0.9840
14	0.9864	0.9864	0.9862
15	0.9864	0.9864	0.9864
16	0.9864	0.9864	0.9864
Avg	0.9541	0.956025	0.9507625

Table 9: Q3a. Data returned from decision tree for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches. Predicted on Training data.

Depth	En	GI	ME
1	0.8752	0.8752	0.8752
2	0.8752	0.8834	0.8834
3	0.8866	0.8856	0.8870
4	0.8742	0.8712	0.8758
5	0.8628	0.8552	0.8586
6	0.8514	0.8424	0.8482
7	0.8366	0.8270	0.8342
8	0.8312	0.8196	0.8190
9	0.8266	0.8136	0.8100
10	0.8224	0.8100	0.8038
11	0.8154	0.8062	0.7960
12	0.8130	0.8018	0.7910
13	0.8132	0.8018	0.7872
14	0.8098	0.7982	0.7838
15	0.8098	0.7982	0.7842
16	0.8098	0.7982	0.7842
Avg	0.838325	0.830475	0.82634999

Table 10: Q3a. Data returned from decision tree for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches. Predicted on Test data.

Depth	En	GI	ME
1	0.8808	0.8808	0.8808
2	0.8808	0.8912	0.8912
3	0.8948	0.8964	0.8978
4	0.9104	0.9090	0.9104
5	0.9262	0.9238	0.9218
6	0.9396	0.9388	0.9304
7	0.9528	0.9532	0.9412
8	0.9602	0.9614	0.9496
9	0.9674	0.9692	0.9572
10	0.9726	0.9748	0.9640
11	0.9760	0.9772	0.9680
12	0.9776	0.9780	0.9726
13	0.9780	0.9780	0.9750
14	0.9780	0.9780	0.9780
15	0.9780	0.9780	0.9780
16	0.9780	0.9780	0.9780
Avg	0.94695	0.9478625	0.943375

Table 11: Q3b. Data returned from decision tree for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches. Predicted on Training data.

Depth	En	GI	ME
1	0.8752	0.8752	0.8752
2	0.8752	0.8834	0.8834
3	0.8860	0.8868	0.8860
4	0.8760	0.8766	0.8776
5	0.8674	0.8646	0.8716
6	0.8578	0.8566	0.8614
7	0.8460	0.8402	0.8432
8	0.8410	0.8340	0.8294
9	0.8352	0.8270	0.8194
10	0.8292	0.8214	0.8140
11	0.8246	0.8170	0.8080
12	0.8214	0.8130	0.8044
13	0.8214	0.8132	0.7984
14	0.8160	0.8066	0.7910
15	0.8160	0.8066	0.7910
16	0.8160	0.8066	0.7910
Avg	0.844025	0.839299	0.8340624999

Table 12: Q3b. Data returned from decision tree for Information Gain (En), Gini Index (GI), and Majority Error (ME) approaches. Predicted on Test data.