Evan Justin 119010521
Ryan Christopher 119010507
Tantama Trisan 119010543

Team Project Computer Science 1001

I. Topic

This project objection is to make a program with the decision as the baseline. We are allowed to use any possible ways in order to meet the requirements of the dataset. For the dataset, there are two different datasets that we can choose either the Google Play Store or the Red Wine Quality. However, every dataset has their own tasks which are classification and regression. In this project, the aim of the classification method is to predict whether the user rating of the app is above 4.5 or less. If we choose the classification, we also need to give the accuracy report. In contrast, the regression method is used to predict the user rating of the app. If we choose the regression, we need to provide the report about the mean square error. However, for the classification, knowing the real value of the prediction is not necessary whereas for the regression is required.

In our project on the decision tree, we are interested in configuring the code of the second dataset which is the Red Wine Quality. Then, we also choose classification which is used to predict whether the user quality of the wine is above 6 or less. We also provide a report about the accuracy of the program. For us, this topic is seen as the most interesting to solve.

II. Approaches

Before we started to make the project, our team did some research in order to make some consideration about the method that we are going to use for the machine learning program. We searched the basic concept and logic of machine learning and these are what we found out; information gain, Entropy, Classification and Regression Tree (CART), GINI index, and decision tree to build the program.

The first step that we do is to find the class of the root. The purpose of this is to make the tree as well as the branches. As we find it, we are able to create the information gain which also allows us to create the tree and the branches. In order to construct a decision tree, information gain is the main point which has an aim to choose the most compatible class for the first branch of the root. It also decreases the ‘uncertainty’ of the result. In the matter of calculating the information gain, we first tried to use the entropy. Later then, we started to realize that entropy is not appropriate or not suitable for this task since it uses some complex mathematical functions.

Then we figure out another possible way to calculate the information gain. We then think about using other methods which later seem ineffective and inefficient. They also do not work well. However, we keep on trying to use another method and after some time, we finally found out that Gini index is the perfect solution to calculate the information gain.

III. Final Approach

The aim of our final program is to calculate the information gain using the Gini index. Gini index is the fastest method in the matter of computing the information gain since it has no complex mathematical functions. Despite the fast calculation, Gini index also provides users the most accurate amount and less than the entropy index. Moreover, the Gini index can also calculate about how often an element or data will be incorrectly labeled when it was chosen at random and randomly labeled according to the distribution of labels in the subset. So that Gini index is the best method among others. Thus, we use this method in our program. By using the Gini index, the information gain can be calculated in order to find the best class to be placed as the first root and the next one.

IV. Working Principle and Functions

Our program consisted of 3 classes which include Question, Leaf and Decision. It also has 14 functions.

- load_csv(filename)

First our program will run the dataset with the build up function “reader” inside function “load_csv(filename)”. Then our program changes all the data into float with 2D array.

```
def load_csv(filename):
    file = open(filename,"rt")
    lines = reader(file)
    dataset = list(lines)
    del dataset[0]
    return dataset

traindatadirectory = 'train.csv'
testdatadirectory = 'test.csv'
training_data = [[float(y) for y in x] for x in load_csv(traindatadirectory)]
testing_data = [[float(y) for y in x] for x in load_csv(testdatadirectory)]
```

- quality_classifier(dataset)

In the function “quality_classifier(dataset)”, we classify the quality of each data from training data to ease the prediction we separated the quality of red wine from below 6 and above 6 and represent both with number 0 and 1

```python
def quality_classifier(dataset):
    for i in range (len(dataset)):

        temp = dataset[i]

        for k in range(len(temp)):
            temp[k] = float(temp[k])

        if temp[-1] > 6:
            temp[-1] = 1
        else:
            temp[-1] = 0
```

- data_count(training_data)

Then “data_count(training_data)” used to store the dictionary for quality with key 1 and 0 and increase the value by 1 if it meets the condition

```python
def data_count(training_data):
    qualitydirectory = {0:0, 1:0}
    for k in training_data:
        if k[-1] == 0:
            qualitydirectory[0] += 1
        else:
            qualitydirectory[1] += 1
    return qualitydirectory
```

- values(rows,col) and num_check(value)

Then function “value(rows, col)” and “num_check(value)” are used to check each row and column to be int and float

```python
def values(rows, col):
    return set([row[col] for row in rows])


def num_check(value):
    return isinstance(value, int) or isinstance(value, float)
```

Evan Justin 119010521
Ryan Christopher 119010507
Tantama Trisan 119010543

- Class Question

Class “Question” is used to build a condition for the tree for each value in each column and row the question is based on the header chosen. Each column number and value stored are going to be compared in function match

```python
class Question:

    def __init__(self, column, value):
        self.column = column
        self.value = value

    def match(self, example):
        val = example[self.column]
        if num_check(val):
            return val >= self.value
        else:
            return val == self.value

    def __repr__(self):
        condition = "=="
        if num_check(self.value):
            condition = ">="
        return "Is %s %s %s?" % (
            header[self.column], condition, str(self.value))
```

- partition(rows, question)

Function “partition(rows, partition)” is used to check if each row match the question and separated into 2 group “Trow” and “Frow”

```python
def partition(rows, question):
    Trows, Frows = [], []
    for row in rows:
        if question.match(row):
            Trows.append(row)
        else:
            Frows.append(row)
    return Trows, Frows
```

- gini_index(rows)

Function “gini_Index(rows)” is the most crucial part for this program. This function is used to calculate the impurity inside the dataset given. It started by calculating the probability of labels.

```python
def gini_index(rows):
    counts = data_count(rows)
    impurity = 1
    for lbl in counts:
        prob_of_lbl = counts[lbl] / float(len(rows))
        impurity -= prob_of_lbl**2
    return impurity
```

- info_gain(left, right, current_uncertainty)

Function “info_gain(left, right, current_uncertainty)” is used to calculate the uncertainty node by using the result from “gini_index(rows)” calculation

```python
def info_gain(left, right, current_uncertainty):

    p = float(len(left)) / (len(left) + len(right))
    return current_uncertainty - p * gini_index(left) - (1 - p) * gini_index(right)
```

- best_split(rows)

Function “best_split(rows)” is used to find the question by iterating the feature and value. Then we split the dataset from the question and then calculate the new information gain after the split

```python
def best_split(rows):
    gain = 0
    question_limit = None
    current_uncertainty = gini_index(rows)
    n_features = len(rows[0]) - 1

    for col in range(n_features):
        values = set([row[col] for row in rows])
        for val in values:
            question = Question(col, val)
            Trows, Frows = partition(rows, question)
            if len(Trows) == 0 or len(Frows) == 0:
                continue
            gain = info_gain(Trows, Frows, current_uncertainty)
            if gain >= gain:
                gain, question_limit = gain, question

    return gain, question_limit
```

- Class Leaf

“Class Leaf” is to determine the prediction from function data count with the help of initializer function.

```python
class Leaf:

    def __init__(self, rows):
        self.predictions = list(data_count(rows))[0]
```

- Class Decision_Node

“Class Decision Node” is used to generate question for nodes, false branch and true branch each represented by “Fbranch” and “TBranch”

```python
class Decision_Node:

    def __init__(self,
                 question,
                 Tbranch,
                 Fbranch):
        self.question = question
        self.Tbranch = Tbranch
        self.Fbranch = Fbranch
```

- tree_function(rows)

Function “tree_function(rows)” is used to grow the decision tree from nodes given by the dataset header. It return the leaf of the tree and uses local variable named as “Trow” and “Frows”.

```python
def tree_function(rows):

    gain, question = best_split(rows)

    if gain == 0:
        return Leaf(rows)
    Trows, Frows = partition(rows, question)
    Tbranch = tree_function(Trows)

    Fbranch = tree_function(Frows)

    return Decision_Node(question, Tbranch, Fbranch)
```

- print_tree(node, spacing = “”)

Function “print_tree(node, spacing = “”)” is used to print the result of the tree from root to nodes until the leaf which is the prediction from the actual quality.

Evan Justin 119010521
Ryan Christopher 119010507
Tantama Trisan 119010543

```python
def print_tree(node, spacing=""):

    if isinstance(node, Leaf):
        print (spacing + "Predict", node.predictions)
        return

    print (spacing + str(node.question))
    print (spacing + '--> True:')
    print_tree(node.Tbranch, spacing + "  ")
    print (spacing + '--> False:')
    print_tree(node.Fbranch, spacing + "  ")
```

- classify(row, node)

Function “classify(row, node)” is a recursion function to return the classification from each row and nodes which represent by “node.Tbranch” or “node.Fbranch”.

```python
def classify(row, node):

    if isinstance(node, Leaf):
        return node.predictions

    if node.question.match(row):
        return classify(row, node.Tbranch)
    else:
        return classify(row, node.Fbranch)
```

- main function

Main function is a function to set the test sample of the program, calculate the prediction of the program. And lastly print the result of the tree from root to leaf, print total test sample, print correct prediction and calculate the accuracy. It also gave a comparison of the actual quality with the predicted quality with 0 and 1.

```python
if __name__ == '__main__':

    my_tree = tree_function(training_data)
    print_tree(my_tree)
    test_sample = 0
    correct_prediction=0

    for row in testing_data:
        prediction = classify(row,my_tree)
        real = (row[-1])
        print('Real', real, '|', prediction,'Predict')
        if prediction == real:
            test_sample+=1
            correct_prediction+=1
        else:
            test_sample+=1
    print("Results:")
    print("Total sample data: " ,test_sample)
    print("Total correct prediction: " ,correct_prediction)
    accuracy = (correct_prediction/test_sample)*100
    print("Accuracy: ",accuracy)
```

V. Summary of the result

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL
Is alcohol >= 11.5?
--> True:
  Is sulphates >= 0.7?
  --> True:
    Is sulphates >= 0.84?
    --> True:
      Predict 0
    --> False:
      Is sulphates >= 0.71?
      --> True:
        Is sulphates >= 0.83?
        --> True:
          Predict 0
        --> False:
          Is sulphates >= 0.72?
          --> True:
            Is sulphates >= 0.77?
            --> True:
              Predict 0
            --> False:
              Is pH >= 3.44?
              --> True:
                Predict 0
              --> False:
                Predict 0
          --> False:
            Predict 0
      --> False:
        Predict 0
  --> False:
    Is sulphates >= 0.59?
    --> True:
      Predict 0
    --> False:
      Is sulphates >= 0.55?
      --> True:
        Predict 0
```

Tree from root to leaf

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

Real 0 | 0 Predict
Real 0 | 0 Predict
Real 0 | 0 Predict
Real 1 | 0 Predict
Real 0 | 0 Predict
Real 1 | 0 Predict
Real 0 | 0 Predict
Real 0 | 0 Predict
Real 0 | 0 Predict
Real 0 | 0 Predict
Real 0 | 0 Predict
Real 1 | 0 Predict
Real 0 | 0 Predict
```

Comparison for actual quality and the code's prediction

```
Results:
Total sample data:  480
Total correct prediction:  416
Accuracy:  86.66666666666667
```

The result from the program's code

VI. Conclusion

It can be seen clearly that our program has 86,667 on the accuracy based on the data that we provide. In the matter to predict whether the user quality of the wine is above 6 or less, our program is accurate enough. Thus, we can state that our program is a successful program.

VII. Workload

Evan Justin - 119010521 ---------- 33,3%

Ryan Christopher - 119010507 ---------- 33,4%

Evan Justin	Ryan Christopher	Tantama Trisan
119010521	119010507	119010543

Tantama - 119010543 ---------- 33,3%