Homework 2: Decision Tree Learning for Classification

李翰韬 16711094 160324

I. Introduction

Decision tree induction is one of the simplest and yet most successful learning algorithms. A decision tree (DT) consists of internal and external nodes and the interconnections between nodes are called branches of the tree. An internal node is a decision-making unit to decide which child nodes to visit next depending on different possible values of associated variables. In contrast, an external node also known as a leaf node, is the terminated node of a branch. It has no child nodes and is associated with a class label that describes the given data. A decision tree is a set of rules in a tree structure, each branch of which can be interpreted as a decision rule associated with nodes visited along this branch.

II. Principle and Theory

Decision trees classify instances by sorting them down the tree from root to leaf nodes. This tree-structured classifier partitions the input space of the data set recursively into mutually exclusive spaces. Following this structure, each training data is identified as belonging to a certain subspace, which is assigned a label, a value, or an action to characterize its data points. The decision tree mechanism has good transparency in that we can follow a tree structure easily in order to explain how a decision is made. Thus interpret ability is enhanced when we clarify the conditional rules characterizing the tree.

Entropy of a random variable is the average amount of information generated by observing its value. Consider the random experiment of tossing a coin with probability of heads equal to 0.9, so that P(Head) = 0.9 and P(Tail) = 0.1. This provides more information than the case where P(Head) = 0.5 and P(Tail) = 0.5. Entropy is used to evaluate randomness in physics, where a large entropy value indicates that the process is very random. The decision tree is guided heuristically according to the information content of each attribute. Entropy is used to evaluate the information of each attribute; as a means of classification. Suppose we have *m* classes, for a particular attribute, we denoted it by pi by the proportion of data which belongs to class C_i where $i = 1, 2, \ldots m$.

The entropy of this attribute is then:

$$Entropy = \sum_{i=1}^{m} -p_i \cdot \log_2 p_i$$

We can also say that entropy is a measurement of the impurity in a collection of training examples: larger the entropy, the more impure the data is. Based on entropy, Information Gain (IG) is used to measure the effectiveness of an attribute as a means of discriminating between classes.

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where all examples *S* is divided into several groups (i.e.S_v for $v \in Values(A)$) according to the value of *A*. It is simply the expected reduction of entropy caused by partitioning the examples according to this attribute.

III. Objective

The goals of the experiment are as follows:

(1) To understand why we use entropy-based measure for constructing a decision tree.

(2) To understand how Information Gain is used to select attributes in the process of building a decision tree.

(3) To understand the equivalence of a decision tree to a set of rules.

(4) To understand why we need to prune the tree sometimes and how can we prune? Based on what measure we prune a decision tree.

(5) To understand the concept of Soft Decision Trees and why they are important extensions to classical decision trees.

IV. Contents and Procedure

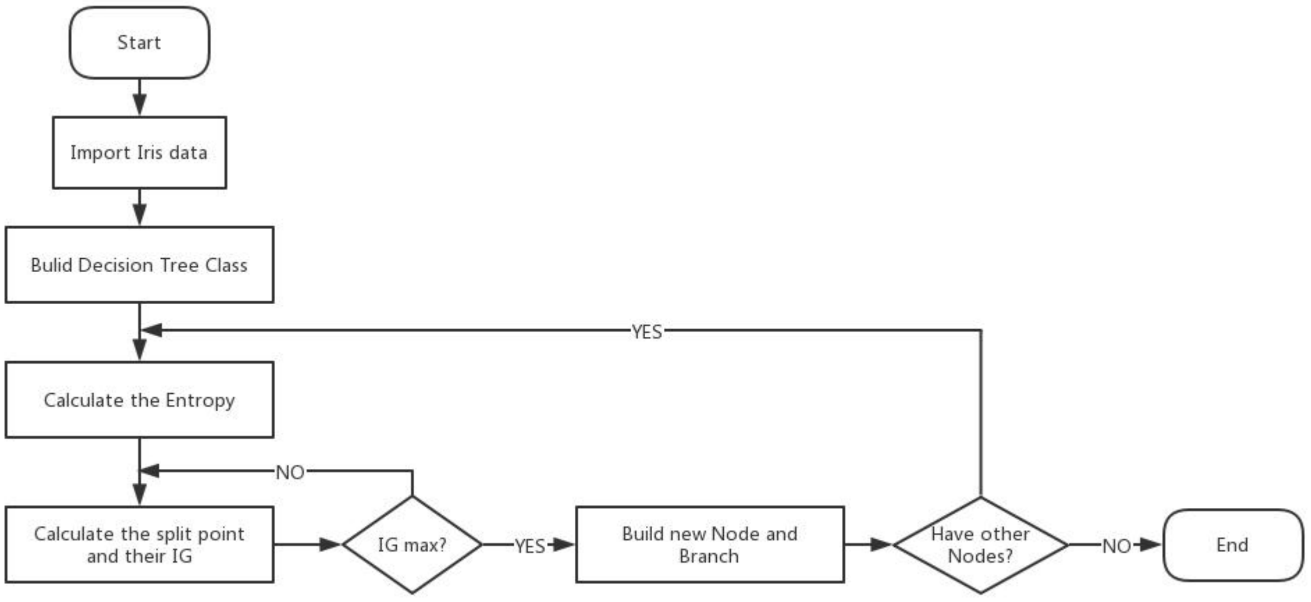
Stage 1:

(1) Select the most informative attribute from a given classification problem (e.g., we will be given the Iris Data set from the UCI Machine Learning Repository)

(2) Find an appropriate data structure to represent a decision tree. Building the tree from the root to leaves based on the principal discussed in section 3.2 by using Information Gain guided heuristics.

(3) According to the above principle and theory in section 3.2, implement the code to calculate the information entropy of each attribute.

In order to achieve the experimental goal, combined with the definition of the decision tree and the content learned in the class, the flow chart of the program written in this experiment is as follows:


Start
Import Iris data
Bulid Decision Tree Class
YES
Calculate the Entropy
NO
Calculate the split point and their IG
IG max?
YES
Build new Node and Branch
Have other Nodes?
NO
End

We can implement the above program with **Python**.

For the first step, we import Iris Data from the UCI data set:

```
from matplotlib import pyplot as plt
import numpy as np
import math
from sklearn.datasets import load_iris

data = load_iris()
x = data.get('data')
y = data.get('target')
training_data = np.c_[x, y]
names = ["Sepal length", "Sepal width", "Petal length", "Petal width", "label"]
```

Then, we can define the **Leaf Class** and the **Decision_Node Class** for the DT:

```
class Leaf:
    def __init__(self, rows):
        self.predictions = class_counts(rows)

class Decision_Node:
    def __init__(self, feature, value, true_branch, false_branch, entropy_Yes, entropy_No):
        self.feature = feature
        self.value = value
        self.true_branch = true_branch
        self.false_branch = false_branch
        self.entropy_Yes = entropy_Yes
        self.entropy_No = entropy_No
```

Next, using code below, we can calculate the **Entropy** of the data :

```
def class_counts(rows):   #Calculate the number of labels in the dataset
    counts = {}    #Use dic because it's more convinence for different features
    for row in rows:
        label = row[-1]
        if label not in counts:
            counts[label] = 0
        counts[label] += 1
    return counts

def Entropy(rows): #Calculate the Entropy of the dataset
    counts = class_counts(rows)
    entropy = 0.0
    for label in counts:
        p_i = counts[label] / float(len(rows))
        entropy = entropy - (p_i * math.log(p_i,2))
    return entropy
```

At the same time, we can calculate the Entropy and the corresponding **IG** to find the best split point of all the points that can be used in all dimensions:

```
def splitting(rows, axis, value):
    true_rows, false_rows = [], []
    for row in rows:
        if row[axis] >= value:
            true_rows.append(row)
        else:
            false_rows.append(row)
    return true_rows, false_rows

def find_best_split(rows):  #Find the best split point in all J_i
    Base_entropy = Entropy(rows)
    best_gain = 0.0
    best_Feature = 0
    best_value = 0.0
    entropy_Yes = 0.0
    entropy_No = 0.0
    n_features = len(rows[0]) - 1    #The number of features
    for col in range(n_features):
        values = set([row[col] for row in rows])     #Unique values in the column
        for val in values:
            true_rows, false_rows = splitting(rows, col, val) # Splitting each possible points
            if len(true_rows) == 0 or len(false_rows) == 0:   # If no split occurs
                continue                                      # Try the next value
            p_1 = float(len(true_rows)) / (len(true_rows)+len(false_rows))
            p_2 = 1 - p_1
            entropy1 = Entropy(true_rows)
            entropy2 = Entropy(false_rows)
            gain = Base_entropy - p_1 * entropy1 - p_2 * entropy2
            if gain >= best_gain:            #Find better split point
                best_gain, best_Feature, best_value = gain, col, val
                entropy_Yes, entropy_No = entropy1, entropy2
    return best_gain, best_Feature, best_value, entropy_Yes, entropy_No
```

After that, we can use the data directly for decision tree classification operations:

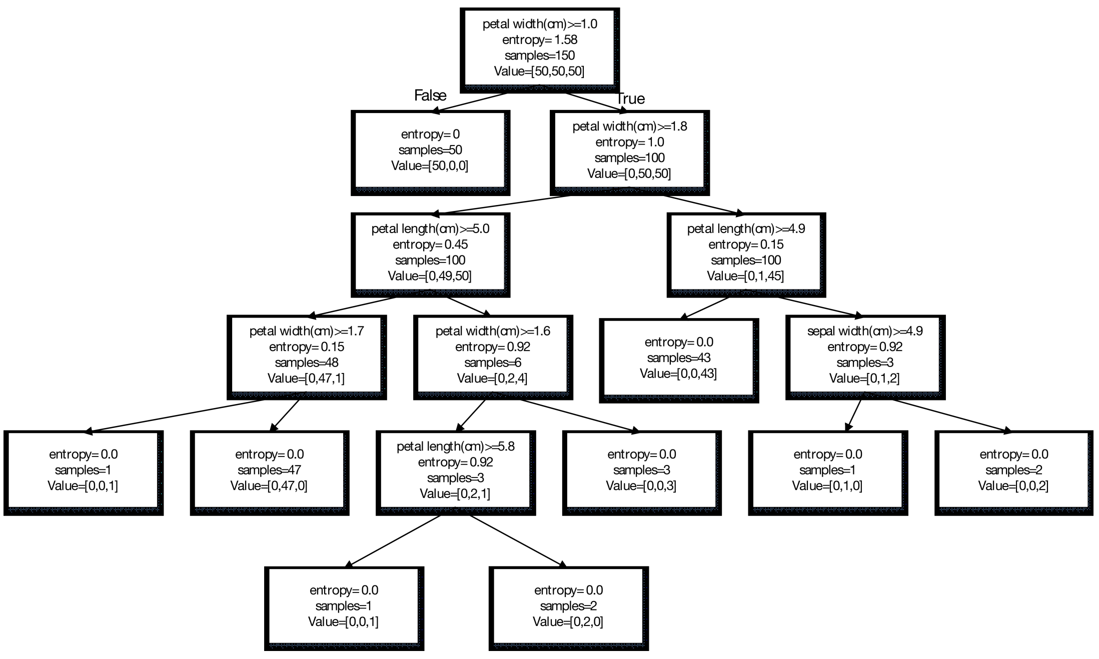
```python
def build_tree(rows):
    gain, feature, value, entropy_Yes, entropy_No = find_best_split(rows)
    if gain == 0:
        return Leaf(rows)
    true_rows, false_rows = splitting(rows, feature, value)
    true_branch = build_tree(true_rows)
    false_branch = build_tree(false_rows)
    return Decision_Node(feature, value, true_branch, false_branch, entropy_Yes, entropy_No)
```

In order to visualize the obtained classification results, the following graphical decision tree module was simply written. Through the following subroutine, we can get the output results shown in the left figure below (because of the limited space, only a part of it is intercepted). In addition, we can also use the Graphviz to graph the decision trees.

```python
def print_tree(node, spacing=""):
    if isinstance(node, Leaf):
        print (spacing + "+++++++++++++++++++++")
        print (spacing + "| " + "value = ", str(node.predictions) + " |")
        print (spacing + "+++++++++++++++++++++\n")
        return
    print (spacing + "---------------------")
    print (spacing + "| " + names[node.feature] + '>=' +str(node.value) + '? |' )
    print (spacing + "---------------------")
    print (spacing + '       ||')
    print (spacing + '       VV')
    print (spacing + '     Yes:' + '->entropy={:.2f}'.format(node.entropy_Yes))
    print (spacing + '       ||')
    print (spacing + '       VV')
    print_tree(node.true_branch, spacing + "  ")
    print (spacing + ' =>=> No:' + '->entropy={:.2f}'.format(node.entropy_No))
    print (spacing + '       ||')
    print (spacing + '       VV')
    print_tree(node.false_branch, spacing + "  ")
```

```
----------------------
| Petal width>=1.0? |
----------------------
      ||
      VV
     Yes:->entropy=1.00
      ||
      VV
  ----------------------
  | Petal width>=1.8? |
  ----------------------
        ||
        VV
       Yes:->entropy=0.15
        ||
        VV
    ----------------------
    | Petal length>=4.9? |
    ----------------------
          ||
          VV
         Yes:->entropy=0.00
          ||
          VV
      ++++++++++++++++++++++
      | value =  {2.0: 43} |
      ++++++++++++++++++++++

     =>=> No:->entropy=0.92
          ||
          VV
      ----------------------
      | Sepal width>=3.2? |
      ----------------------
```



petal width(cm)>=1.0 entropy= 1.58 samples=150 Value=[50,50,50]
False
True
entropy= 0 samples=50 Value=[50,0,0]
petal width(cm)>=1.8 entropy= 1.0 samples=100 Value=[0,50,50]
petal length(cm)>=5.0 entropy= 0.45 samples=100 Value=[0,49,50]
petal length(cm)>=4.9 entropy= 0.15 samples=100 Value=[0,1,45]
petal width(cm)>=1.7 entropy= 0.15 samples=48 Value=[0,47,1]
petal width(cm)>=1.6 entropy= 0.92 samples=6 Value=[0,2,4]
entropy= 0.0 samples=43 Value=[0,0,43]
sepal width(cm)>=4.9 entropy= 0.92 samples=3 Value=[0,1,2]
entropy= 0.0 samples=1 Value=[0,0,1]
entropy= 0.0 samples=47 Value=[0,47,0]
petal length(cm)>=5.8 entropy= 0.92 samples=3 Value=[0,2,1]
entropy= 0.0 samples=3 Value=[0,0,3]
entropy= 0.0 samples=1 Value=[0,1,0]
entropy= 0.0 samples=2 Value=[0,0,2]
entropy= 0.0 samples=1 Value=[0,0,1]
entropy= 0.0 samples=2 Value=[0,2,0]

Stage 2:

(1) Now consider the case of with continuous attributes or mixed attributes (with both continuous and discrete attributes), how can we deal with the decision trees? Can you propose some approaches to do discretization?

In this example, the calculation method we use is the **ID3** algorithm, and we take the IG as a measure of purity. This method is suitable for the construction of decision trees with discrete attributes. For datasets with continuous attributes or mixed attributes, we cannot use the ID3 algorithm.

At this time, an improved decision tree algorithm **C4.5** algorithm can be used. This algorithm uses a two-separation dispersion scheme, which can process continuous numerical attributes. Its processing method for the continuity attribute is: Firstly, find the maximum and minimum values of the training sample on the continuity attribute, and set multiple equal breakpoints on the value interval defined by the maximum and minimum values. Secondly, the information gain values with these breakpoints as column points are calculated and compared. After that, the breakpoint with the largest information gain is the best split point. From this split point, the entire value interval is divided into two parts, and the record set is divided into two parts according to the value recorded on this attribute.

(2) Is there a trade off between the size of the tree and the model accuracy? Is there existing an optimal tree in both compactness and performance?

For a decision tree with weak generalization ability, the decision tree is relatively large. In contrast, for a decision tree with strong generalization ability, the decision tree is relatively small. When the decision tree is small, the training and testing errors are large, which is called **model underfitting**. The reason for the underfitting is that the model has not learned the true structure of the data yet. Therefore, the performance of the model on the training set and test set are not goof enough. As the number of nodes in the decision tree increases, the training and testing errors of the model will decrease accordingly.

However, once the size of the tree becomes too large, even though the training error continues to decrease, the test error starts to increase. This phenomenon is called **model overfitting**. Therefore, there is a trade-off between the size of the tree and the accuracy of the model. For a classifiable data set, there are decision trees that can be better classified in terms both of compactness and performance.

(3) For one data element, the classical decision tree gives a hard boundary to decide which branch to follow, can you propose a "soft approach" to increase the robustness of the decision tree?

Decision-making methods that use the collective creativity of experts are called soft methods of decision-making. **Deep neural networks (DNN)** are a very effective method for performing classification tasks. When the input data is high-dimensional, the relationship between input and output is extremely complicated, and the number of labeled training samples is very large, the performance of the deep neural network is very good.

If we can take full advantage of the knowledge gained from neural networks and express the same knowledge in a model that relies on hierarchical decision-making, it will be much easier to explain a particular decision. Thus, soft decision trees can therefore be created using trained neural networks.

(4) Compare to the Naïve Bayes, what are the advantages and disadvantages of the decision tree learning?

One of the biggest advantages of learning based on decision trees is that it does not require users to know a lot of background knowledge during the learning process. As long as the training examples can be expressed by attributes, the decision tree algorithm can be used for learning. Moreover, when the number of attributes is large, the decision tree has a better classification effect than the naive Bayes method.

However, the ID3 method in the decision tree cannot classify continuous features well, while the C4.5 method based on ID3 can classify continuous attributes but the accuracy is greatly reduced. In addition, for the relatively small attribute correlation, Naive Bayes can achieve a good classification effect, but the decision tree is difficult to get a compact result.

Stage 3:

Explore the questions in the previous section and design experiments to answer these questions. Complete and submit an experiment report about all experiment results with comparative analysis and a summary of experiences about this experiment study.

The advantages of decision tree algorithm are simple and easy to understand, easy to explain, visual, and wide applicability.

The disadvantage of it is that it is easy to overfit; unstable, which means small changes in the data will affect the results; the selection of each node is a greedy algorithm, which cannot guarantee a global optimal solution.

The above experiment result is obtained by the information gain (IG) method. The decision tree can also use the Gini coefficient instead of the information gain, which is called the **CART** decision tree classification algorithm. The result of this method will similar to the ID3 algorithm.

V. Code

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Sun Sat 21 15:27:47 2020

@author: hantao.li
"""

from matplotlib import pyplot as plt
import numpy as np
import math
from sklearn.datasets import load_iris

data = load_iris()
x = data.get('data')
y = data.get('target')
training_data = np.c_[x, y]
names = ["Sepal length", "Sepal width", "Petal length", "P
etal width", "label"]

def class_counts(rows):     #Calculate the number of
labels in the dataset
        counts = {}      #Use dic because it's more
convenience for different features
    for row in rows:
        label = row[-1]
        if label not in counts:
            counts[label] = 0
        counts[label] += 1
    return counts

def Entropy(rows): #Calculate the Entropy of the dataset
    counts = class_counts(rows)
    entropy = 0.0
    for label in counts:
        p_i = counts[label] / float(len(rows))
        entropy = entropy - (p_i * math.log(p_i,2))
    return entropy

def splitting(rows, axis, value):
      true_rows, false_rows = [], []
      for row in rows:
            if row[axis] >= value:
                  true_rows.append(row)
            else:
                  false_rows.append(row)
      return true_rows, false_rows

def find_best_split(rows):  #Find the best split point in all
J_i
      Base_entropy = Entropy(rows)
    best_gain = 0.0
    best_Feature = 0
    best_value = 0.0
    entropy_Yes = 0.0
    entropy_No = 0.0
      n_features = len(rows[0]) - 1      #The number of
features
    for col in range(n_features):
         values = set([row[col] for row in rows])       #Uniqu
e values in the column
        for val in values:
             true_rows, false_rows = splitting(rows, col, val)
# Splitting each possible points
               if len(true_rows) == 0 or len(false_rows) == 0:
# If no split occurs
                                                   continue
               # Try the next value
             p_1 = float(len(true_rows)) / (len(true_rows)+len(f
alse_rows))
                   p_2 = 1 - p_1
            entropy1 = Entropy(true_rows)
            entropy2 = Entropy(false_rows)
               gain = Base_entropy - p_1 * entropy1 - p_2 *
entropy2
             if gain >= best_gain:              #Find better
split point
                best_gain, best_Feature, best_value = gain, c
ol, val
                entropy_Yes, entropy_No = entropy1, entropy
2
     return best_gain, best_Feature, best_value, entropy_Y
es, entropy_No

class Leaf:
    def __init__(self, rows):
        self.predictions = class_counts(rows)

class Decision_Node:
     def __init__(self, feature, value, true_branch, false_bra
nch, entropy_Yes, entropy_No):
        self.feature = feature
        self.value = value
        self.true_branch = true_branch
        self.false_branch = false_branch
        self.entropy_Yes = entropy_Yes
        self.entropy_No = entropy_No

def build_tree(rows):
    gain, feature, value, entropy_Yes, entropy_No = find_b
est_split(rows)
    if gain == 0:
        return Leaf(rows)
    true_rows, false_rows = splitting(rows, feature, value)
    true_branch = build_tree(true_rows)
    false_branch = build_tree(false_rows)
    return Decision_Node(feature, value, true_branch, fals
e_branch, entropy_Yes, entropy_No)

def print_tree(node, spacing=""):
    if isinstance(node, Leaf):
            print (spacing + "+++++++++++++++++++++++")
                    print (spacing + "|  " +
"value = ", str(node.predictions) + " |")
            print (spacing + "+++++++++++++++++++++++\n")
        return
    print (spacing + "---------------------")
      print (spacing + "|  "
+ names[node.feature] + '>=' +str(node.value) + '? |' )
      print (spacing + "---------------------")
      print (spacing + '          ||')
      print (spacing + '          VV')
                    print (spacing + '
Yes:' + '->entropy={:.2f}'.format(node.entropy_Yes))
      print (spacing + '          ||')
      print (spacing + '          VV')
    print_tree(node.true_branch, spacing + "  ")
                    print (spacing + '
=>=> No:' + '->entropy={:.2f}'.format(node.entropy_No))
      print (spacing + '          ||')
      print (spacing + '          VV')
    print_tree(node.false_branch, spacing + "  ")

my_tree = build_tree(training_data)
print_tree(my_tree)
```