成绩

Pattern Recognition and Machine Learning Experiment Report

院（系）名称 自动化科学与电气工程学院

专 业 名 称 自动化

学 生 学 号 15071129

学 生 姓 名 苗子琛

2018 年 6 月

Experiment3 Decision Tree Learning for Classification

3.1 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.

3.2 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, interpretability 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 Ci where i = 1, 2, … 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.

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

where all examples S is divided into several groups (i.e. Sv 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.

3.3 Objective

(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.

3.4 Contents and Procedures

Stage 1

(1) According to the above principles and theories in section 3.2, implement the code to calculate the information entropy of each attribute.

As shown in the equation below,

$$\text{Entropy} = \sum_{i=1}^{c} -p_i \cdot \log_2 p_i \tag{1}$$

where c is the number of classes and p_i is the likelihood that a sample falls in class *I*, I implement the calculation of entropy in python 3.6 in a function. I want to stress that when calculating, pi is actually the frequency of class.

(2) Select the most informative attribute from a given classification problem.

The information gain is derived as,

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

Where $Entropy(S)$ means the information entropy of samples belonging to class *v*, while $\frac{|S_V|}{|S|}$ performs normalization role. The information gain measures the effectiveness of an attribute with which we partition the data.

It is obvious that the larger the information gain of an attribute, the more informative the attribute is. To select the most informative attribute, I calculated information gain for all attributes given the dataset and select the one with the largest information gain.

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

Naively, I choose the classical data structure, tree, to represent a decision tree. For this task, I choose binary tree for classification, in which one parent node has two children nodes.

When it comes to create a tree, people always use recursive method to build the total structure Every step in this process is like this: first I calculate information gain of all attributes and select all the one with the largest information gain for partitioning the data. Then for the *Iris dataset,* the dataset with continuous value, I select the value resulting the largest information gain from all the values in one attribute and separate the data into two branches. After repeating this process recursively until all three classes can be clearly discerned, I finally create the decision tree for *Iris dataset.*

For visualization of this decision tree, I use the graph drawing software Graphviz 2.3.6, with the help of the python interface package Graphviz 0.7. The visualization is shown in **fig.1.**

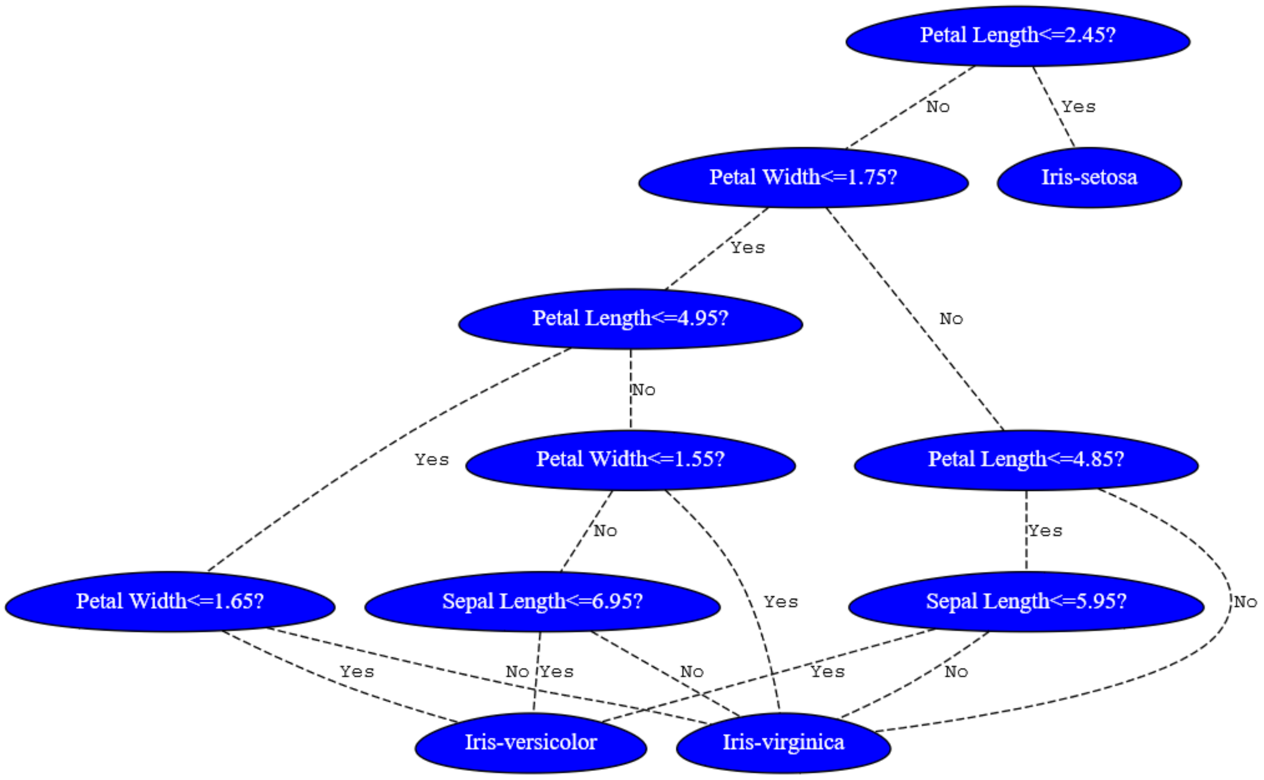


Fig.1 visualization of the decision tree for the Iris dataset

As shown in **fig.1,** petal length attribute effectively separate *Iris-setosa class* individually. And after five separation, all classes are discerned.

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 discretization?

To discretize the continuous attribute s in dataset B, I follow the steps below,

1) Sort the dataset in ascending order of this attribute and get $\{s_1, s_2, \dots, s_n\}$.
2) For each of the k non-repetitive values in $\{s_1, s_2, \dots, s_n\}$, *i.e.* $\{s_1, s_2, \dots, s_k\}$, I choose it as partition threshold t and separate the dataset into two parts: $G_1 = \{s_i \le t\}$,and $G_2 = \{s_i > t\}$

3) For *k* possible t, I calculate their information gain and select the one with the largest information gain to partition dataset.

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

There is a tradeoff between tree size and model accuracy. A small decision tree is not able to capture the important structural information of the dataset, which performs poorly. To enhance the capacity of the model, one has to enlarge the tree.

However, with a tree too large, it tends to learn noises of the training set, which causes overfit. In such circumstances, the performance on the test set may not be desirable, although the accuracy on the training set can be extremely high.

Pruning is a typical way to keep balance between compactness and performance. Pruning can occur in a top-down or bottom-up fashion, based on several different criteria, such as reduced error and cost complexity.

(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?

The soft decision tree[1], proposed a method for higher classification accuracy. Unlike the hard internal node, which only select one child for further decision, soft decision tree select all children with different probabilities. It follows all the paths to all the leaves and all leaves contribute to the final decision but with different probabilities.

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

Advantage:

1) It is easy to understand and explain.

2) It can process both the data type and the general type property at the same time.

3) It will deal with data with many attributes.

4) It will give feasible and effective results for large data sources in a relatively short period of time.

Disadvantage:

1) It ignores the correlation between attributes.

2) It will cause the emergence of the problem of over-fitting

3) It is difficult to deal with missing data.

4) The results of Information Gain tend to lean the attributes with more values in a data in which it has different sample sizes for each attribute.

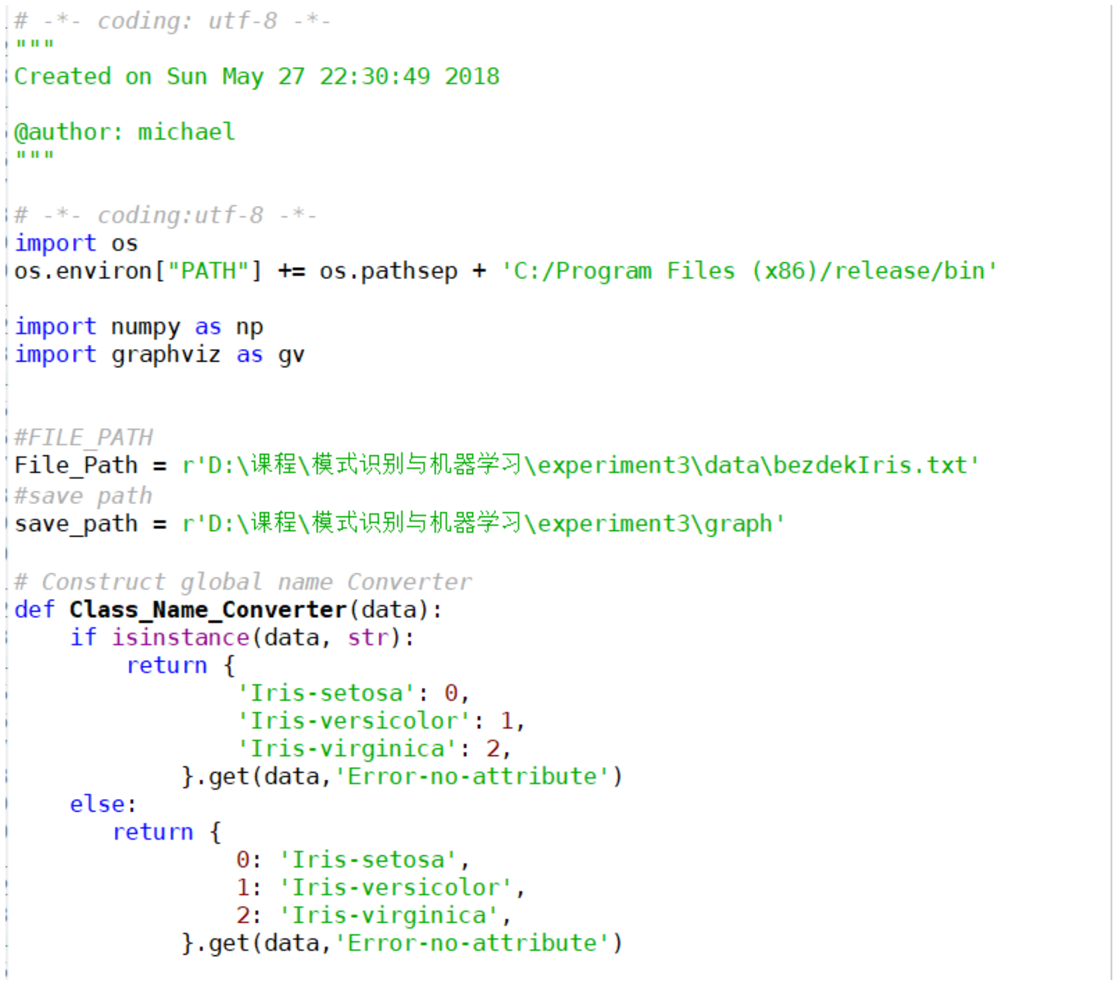
3.6 References

[1] Soft Decision Trees. O. Irsoy, O. T. Yildiz, E. Alpaydin ICPR 21, Tsukuba, Japan.

3.7 Part of Code

The whole version code can be seen on:

https://github.com/michaelHaha/BUAA_PRML_Experiment



```
# -*- coding: utf-8 -*-
"""
Created on Sun May 27 22:30:49 2018

@author: michael
"""

# -*- coding:utf-8 -*-
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files (x86)/release/bin'

import numpy as np
import graphviz as gv


#FILE_PATH
File_Path = r'D:\课程\模式识别与机器学习\experiment3\data\bezdekIris.txt'
#save path
save_path = r'D:\课程\模式识别与机器学习\experiment3\graph'

# Construct global name Converter
def Class_Name_Converter(data):
    if isinstance(data, str):
        return {
                'Iris-setosa': 0,
                'Iris-versicolor': 1,
                'Iris-virginica': 2,
            }.get(data,'Error-no-attribute')
    else:
       return {
                0: 'Iris-setosa',
                1: 'Iris-versicolor',
                2: 'Iris-virginica',
            }.get(data,'Error-no-attribute')
```

```python
def Attribute_Name_Converter(data):
    return {
            0: 'Sepal Length',
            1: 'Sepal Width',
            2: 'Petal Length',
            3: 'Petal Width',
        }.get(data,'Error-no-attribute')


#load data
def Load_Data(File_path):
    data = []

    with open(File_Path, 'r') as f:
        for line in f.readlines():
            line = line.split(sep = ',')
            tmp = [float(i) for i in line[:-1]]
            tmp.append(Class_Name_Converter(line[-1].replace('\n','')))
            data.append(tmp)

    return data

#Calculate Entropy
def Calc_Entropy(dataset):
    NumInputs = len(dataset)
    Labels = {}

    for single_Input in dataset:
        currentLabel = single_Input[-1]
        Labels[currentLabel] = Labels.get(currentLabel,0) +1

    Entropy = 0.0
    for key in Labels.keys():
        Probablity = float(Labels[key])/NumInputs
        Entropy -= Probablity * np.log2(Probablity)
    return Entropy
```

```
#Calculate Information Gain
def calInformGain(dataset, attribute,  threshold):
    dataset.sort(key = lambda x: x[int(attribute)])
    base_Entropy = Calc_Entropy(dataset)
    # Find the threshold position
    Thres_pos = 0
    while (dataset[Thres_pos][int(attribute)] < threshold):
        Thres_pos += 1
    InformGain = base_Entropy-Calc_Entropy(dataset[:Thres_pos])*Thres_pos/len(da
    InformGain -= Calc_Entropy(dataset[Thres_pos:])*(len(dataset)-Thres_pos)/len
    return InformGain

def calculateThreshold(dataset, attribute):
    final_threshold = 0.0
    # Sort data according to attribute
    dataset.sort(key = lambda x : x[attribute])
    threshold_list = [data[attribute] for data in dataset]
    entropy_gain_max = 0
    for i in range(1, len(threshold_list)):
        entropy_gain_current = calInformGain(dataset, attribute, threshold_list[
        if entropy_gain_current > entropy_gain_max:
            entropy_gain_max = entropy_gain_current
            final_threshold = np.mean([threshold_list[i], threshold_list[i - 1]]

    return final_threshold

# Select attribute with maximum entropy gain and its threshold
def slecAttribute(dataset, attributes):
    attribute_selected = 0
    InformGainMax_attri = 0
    threshold_selected = 0.0

    for attribute in attributes:
        #calculate information gain for this attribute
        threshold = calculateThreshold(dataset, attribute)
        Informgain = calInformGain(dataset, attribute, threshold)
        #select the optimal attributes according to information gain of differe
        if Informgain > InformGainMax_attri:
            InformGainMax_attri = Informgain
            attribute_selected = attribute
            threshold_selected = threshold
```