实 验 报 告

(2017 / 2018 学年 第 2 学期)

课程名称	机器学习导论
实验名称	Decision Tree
实验时间	2018 年 5 月 20 日
指导教师	王邦

姓名	游浩然	学号	U201515429

1 问题重述

- 对课堂上所讲的西瓜数据集，构造决策树。
- 根据用户采集的 WiFi 信息采用决策树预测用户所在房间。

2 西瓜决策树

2.1 Python 代码实现

```python
# -*- coding: utf-8 -*-
"""
@author : Haoran You

"""
from math import log
import operator
import matplotlib
import matplotlib.pyplot as plt

"""
Moudule - Create decision tress for datasets
:createTree(main)
:bestFeature
:calcShannonEnt
:splitDataset
:majorityCnt
"""
def createTree(dataset, label):
    classlist = [example[-1] for example in dataset]
    # no subtree
    if classlist.count(classlist[0]) == len(classlist):
        return classlist[0]
    # no feature
    if len(dataset[0]) == 1:
        return majorityCnt(classlist)
    # choose best feature
    bestFeat = bestFeature(dataset)
    bestFeatLabel = label[bestFeat]
    myTree = {bestFeatLabel:{}}
    del(label[bestFeat])
    # recursion for each subdataset
    feat_values = [example[bestFeat] for example in dataset]
    uniqueVals = set(feat_values)
    for value in uniqueVals:
        sub_label = label[:]
        myTree[bestFeatLabel][value] = createTree(splitDataset(dataset,bestFeat,value), sub_label)
    return myTree

def bestFeature(dataset):
    num_features = len(dataset[0]) - 1
    baseEntropy = calcShannonEnt(dataset)
    bestInfoGain = 0.0
    bestFeat = 0
    for i in range(0, num_features):
        featList = [example[i] for example in dataset]
        uniqueVals = set(featList)
        newEntropy = 0.0
        for value in uniqueVals:
            sub_dataset = splitDataset(dataset, i, value)
            prob = len(sub_dataset) / float(len(dataset))
            newEntropy += prob * calcShannonEnt(sub_dataset)
        infoGain = baseEntropy - newEntropy
        if infoGain > bestInfoGain:
            bestInfoGain = infoGain
            bestFeat = i
    return bestFeat

def calcShannonEnt(dataset):
    num_samples = len(dataset)
    labelCounts = {}
    for feat_vector in dataset:
        current_label = feat_vector[-1]
        if current_label not in labelCounts.keys():
            labelCounts[current_label] = 0
        labelCounts[current_label] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key]) / num_samples
        shannonEnt -= prob * log(prob, 2)
    return shannonEnt
```

```
72
73  def splitDataset(dataset, axis, value):
74      sub_dataset = []
75      for feat_vector in dataset:
76          if feat_vector[axis] == value:
77              reduce_feat_vector = feat_vector[:axis]
78              reduce_feat_vector.extend(feat_vector[axis+1:])
79              sub_dataset.append(reduce_feat_vector)
80      return sub_dataset
81
82  def majorityCnt(classlist):
83      classCount = {}
84      for vote in classlist:
85          if vote not in classCount.keys():
86              classCount[vote] = 0
87          classCount[vote] += 1
88      sortedClassCount = sorted(classCount.items(), key=operator.itemgetter(1), reverse=True)
89      return sortedClassCount[0][0]
90
91  """
92  Module - Plot decision tree constructed before
93  :createPlot(main)
94  :plotTree
95  :plotNode
96  :plotMidText
97  :getNumLeaves
98  :getTreeDepth
99  """
100 decisionNode = dict(boxstyle='sawtooth', fc='0.8')
101 leafNode = dict(boxstyle='round4', fc='0.8')
102 arrow_args = dict(arrowstyle='<-')
103
104 def createPlot(myTree):
105     fig = plt.figure(1, facecolor='white')
106     fig.clf()
107     axprops = dict(xticks=[], yticks=[])
108     createPlot.ax1 = plt.subplot(111, frameon=True)
109     plotTree.totalW = float(getNumLeaves(myTree))
110     plotTree.totalD = float(getTreeDepth(myTree))
111     plotTree.xOff = -0.5/plotTree.totalW
112     plotTree.yOff = 1.0
113     plotTree(myTree, (0.5, 1.0), '')
114     plt.show()
115
116 def plotTree(myTree, parentPt, nodeTxt):
117     num_leaves = getNumLeaves(myTree)
118     depth = getTreeDepth(myTree)
119     firstStr = list(myTree.keys())[0]
120     cntrPt = (plotTree.xOff + (1.0 + float(num_leaves))/2.0/plotTree.totalW, plotTree.yOff)
121     plotMidText(cntrPt, parentPt, nodeTxt)
122     plotNode(firstStr, cntrPt, parentPt, decisionNode)
123     secondDict = myTree[firstStr]
124     plotTree.yOff = plotTree.yOff - 1.0/plotTree.totalD
125     for key in secondDict.keys():
126         if type(secondDict[key]).__name__ == 'dict':
127             plotTree(secondDict[key], cntrPt, str(key))
128         else:
129             plotTree.xOff = plotTree.xOff + 1.0/plotTree.totalW
130             plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
131             plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
132     plotTree.yOff = plotTree.yOff + 1.0/plotTree.totalD
133
134 def plotNode(nodeTxt, centerPt, parentPt, nodeType):
135     createPlot.ax1.annotate(nodeTxt, xy=parentPt,  xycoords='axes fraction',
136              xytext=centerPt, textcoords='axes fraction',
137              va="center", ha="center", bbox=nodeType, arrowprops=arrow_args)
138
139 def plotMidText(cntrPt, parentPt, txtString):
140     # positon for text
141     xMid = (parentPt[0]-cntrPt[0])/2.0 + cntrPt[0]
142     yMid = (parentPt[1]-cntrPt[1])/2.0 + cntrPt[1]
143     createPlot.ax1.text(xMid, yMid, txtString, va="center", ha="center")
144
145 def getNumLeaves(myTree):
146     numLeaves = 0
147     firstStr = list(myTree.keys())[0]
148     secondDict = myTree[firstStr]
149     for key in secondDict.keys():
150         if type(secondDict[key]).__name__=='dict':
151             numLeaves += getNumLeaves(secondDict[key])
152         else:   numLeaves +=1
153     return numLeaves
154
155 def getTreeDepth(myTree):
156     maxDepth = 0
157     firstStr = list(myTree.keys())[0]
158     secondDict = myTree[firstStr]
159     for key in secondDict.keys():
```

```
        if type(secondDict[key]).__name__=='dict':
            thisDepth = 1 + getTreeDepth(secondDict[key])
        else:   thisDepth = 1
        if thisDepth > maxDepth: maxDepth = thisDepth
    return maxDepth

if __name__ == '__main__':
    # load dataset
    f = open('watermelon.txt', 'r', encoding='utf-8')
    label = f.readline()
    label = label.strip().split('  ')
    dataset = [inst.strip().split('  ') for inst in f.readlines()]
    print('feature ', label)
    print('dataset ', dataset)
    # tree structure
    myTree = createTree(dataset, label)
    print('myTree  ', myTree)
    # draw tree
    from pylab import *
    mpl.rcParams['font.sans-serif'] = ['SimHei']
    createPlot(myTree)
```

2.2 结果图

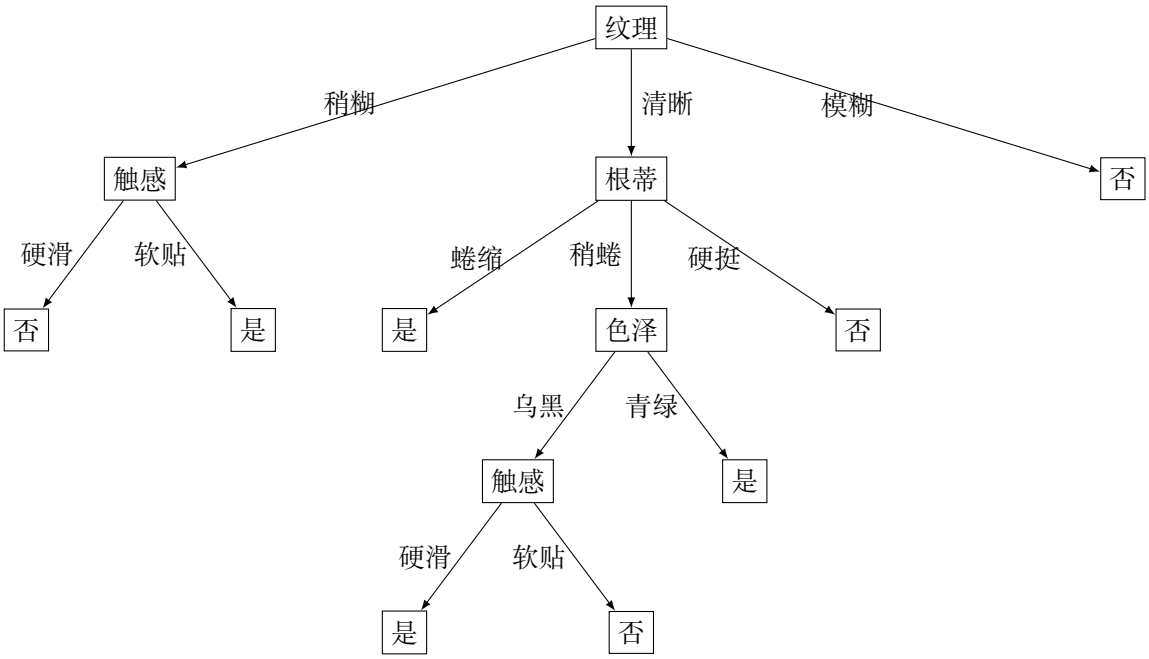

Figure 1: A decision tree for evaluating watermelon.

3 房间决策树

3.1 Python 代码实现

```
# -*- coding: utf-8 -*-
"""
@author : Haoran You

"""
from math import log
import csv
import os
import random
import matplotlib.pyplot as plt
from Task1 import *

def devidetraincsv():
    trainDT = csv.reader(open('TrainDT.csv', 'r'))
    dataset = []
```

```
    for line in trainDT:
        dataset.append(line)
    del(dataset[0])
    time_list = list(set([int(example[-1]) for example in dataset]))
    BSSID_list = list(set([example[0] for example in dataset]))
    label_dict = {}
    for item in dataset:
        label_dict[item[-1]] = item[2]
    pre_whole_train = []
    for i in range(0, len(time_list)):
        new_dict = {}
        for j in range(0, len(BSSID_list)):
            new_dict[BSSID_list[j]] = 0
        pre_whole_train.append(new_dict)
    for item in dataset:
        pre_whole_train[int(item[-1])-1][item[0]] = float(item[1])
    whole_train = []
    fin_label = 0
    for item in pre_whole_train:
        fin_label += 1
        new_list = []
        for keys in item.keys():
            new_list.append(item[keys])
        new_list.append(label_dict[str(fin_label)])
        whole_train.append(new_list)
    num_train = int(0.9*len(whole_train))
    num_val = len(whole_train) - num_train
    train = random.sample(whole_train, num_train)
    val = random.sample(whole_train, num_val)
    return train, val, BSSID_list

def dividetestcsv(feature):
    testDT = csv.reader(open('TestDT.csv', 'r'))
    dataset = []
    for line in testDT:
        dataset.append(line)
    del (dataset[0])
    time_list = list(set([int(example[-1]) for example in dataset]))
    BSSID_list = feature
    pre_whole_test = []
    for i in range(0, len(time_list)):
        new_dict = {}
        for j in range(0, len(BSSID_list)):
            new_dict[BSSID_list[j]] = 0
        pre_whole_test.append(new_dict)
    for item in dataset:
        pre_whole_test[int(item[-1]) - 1][item[0]] = float(item[1])
    whole_test = []
    fin_label = 0
    for item in pre_whole_test:
        fin_label += 1
        new_list = []
        for keys in item.keys():
            new_list.append(item[keys])
        whole_test.append(new_list)
    return whole_test

def discretization(dataset):
    for i in range(0, len(dataset[0])-1):
        rss_list = [example[i] for example in dataset]
        num = len(rss_list)
        for k in range(0, num):
            if rss_list[k] == 0.0:
                dataset[k][i] = 0
            else:
                dataset[k][i] = 1
    return dataset

def predict_val(dataset, feature, tree):
    label = [example[-1] for example in dataset]
    i = 0
    total_num = len(label)
    corr_num = 0
    for item in dataset:
        for keys in tree.keys():
            index = feature.index(keys)
            sub_tree = tree[keys][item[index]]
            if isinstance(sub_tree, dict):
                pred = find_leaf(item, feature, sub_tree)
            else:
                pred = sub_tree
        if pred == label[i]:
            corr_num += 1
        i += 1
    acc = corr_num / total_num
    return acc

def predict(dataset, feature, tree):
```

```
    if os.path.exists('results.csv'):
        os.remove('results.csv')
    f = open('results.csv', 'a', newline='')
    csv_write = csv.writer(f, dialect='excel')
    i = 1
    for item in dataset:
        for keys in tree.keys():
            index = feature.index(keys)
            sub_tree = tree[keys][item[index]]
            if isinstance(sub_tree, dict):
                pred = find_leaf(item, feature, sub_tree)
            else:
                pred = sub_tree
        i += 1
        result = []
        result.append(i)
        result.append(pred)
        csv_write.writerow(result)

def find_leaf(item, feature, tree):
    for keys in tree.keys():
        index = feature.index(keys)
        sub_tree = tree[keys][item[index]]
        if isinstance(sub_tree, dict):
            pred = find_leaf(item, feature, sub_tree)
        else:
            pred = sub_tree
    return pred

if __name__ == '__main__':
    train_dataset, val_dataset, feature = devidetraincsv()
    test_dataset = dividetestcsv(feature)
    print('feature   ', feature)
    print('num_train ', len(train_dataset))
    print('num_val   ', len(val_dataset))
    train_dataset = discretization(train_dataset)
    myTree = createTree(train_dataset, feature.copy())
    print(myTree)
    createPlot(myTree)
    val_dataset = discretization(val_dataset)
    acc = predict_val(val_dataset, feature.copy(), myTree)
    print('accuracy  ', acc)
    test_dataset = discretization(test_dataset)
    predict(test_dataset, feature.copy(), myTree)
```

3.2 结果图

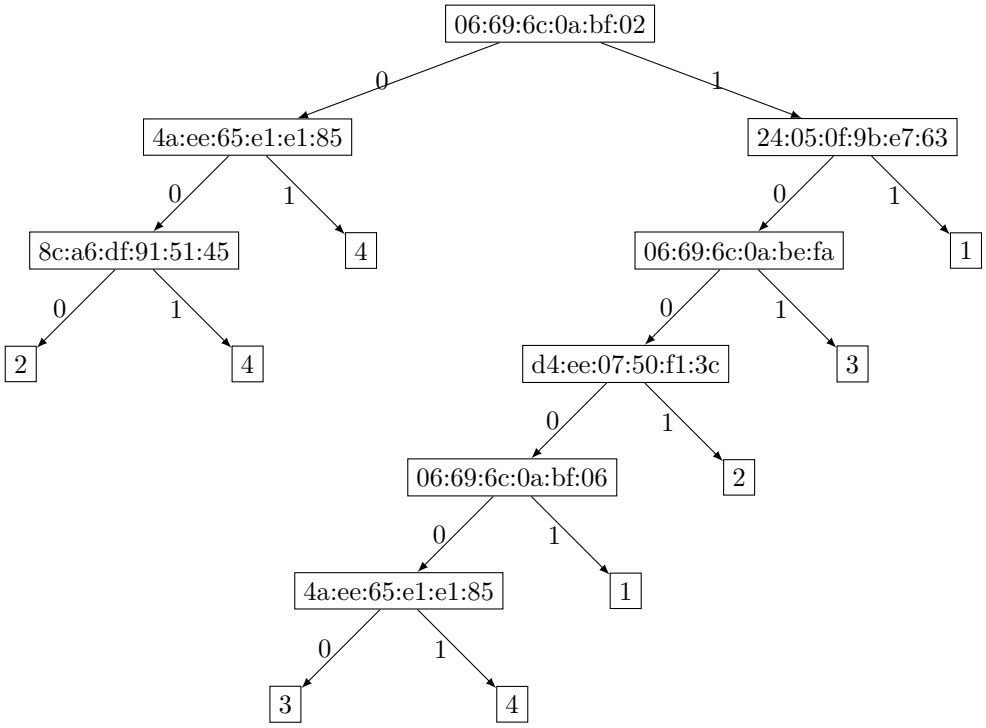

Figure 2: A decision tree for evaluating roomlabel. 0 presents no signal received, 1 means inversely.