# 软件学院 机器学习 课程实验报告

| 实验题目：随机森林 | | 学号：201700301166 |
| --- | --- | --- |
| 日期：2019/12/17 | 班级： 网安 17 | 姓名： 尹成林 |
| Email：sdyinruichao@163.com | | |

实验目的：

利用随机森林进行鸢尾花数据集的分类

实验软件和硬件环境：

Win10

Pycharm

使用语言：python

使用的库：numpy、math、csv、random

实验原理和方法：

决策树相关理论

特征选取的差异性：每个决策树的 n 个分类特征是在所有特征中随机选择的

投票机制

实验步骤：（不要求罗列完整源代码）

## 题目分析：

1.划分数据集，进行特征划分，给小树不同的特征进行训练，于是我专门写了一个 py 脚本，用于把数据集进行划分，划分出具有不同特征的样本，分别进行训练。

2.训练不同的决策树

3.得到测试集后对测试集进行测试，得到多个结果后运用投票机制，得到最终的结果

具体工作步骤：

1.数据集划分问题：我想到的方法是在原有数据具有四维特征的基础上，每一次都去掉一维特征，得到四组数据集，用于训练四个决策树。

```
def loadcsv(name):...
def write_csv_file(path, head, data):...
def dddvvv(num, dataset):...
# main function
if __name__=="__main__":
    name = "...\data\iris.csv"
    dataset = loadcsv(name)
    for i in range(4):
        fuck = dddvvv(i, dataset)
        write_csv_file("...\data\i{}.csv".format(i), None, fuck)
```

2.决策树的训练使用了上一次的代码，因为数据集没有发生改变，所以需要改的地方并不多，直接 for 循环读取四个数据集，训练出四棵决策树就可以了。

```
# 构建一个树
def buildTree(dataset, node_id):...


# 构建一个小小树
def buildTreeR(dataset, node_id, idd):...
```

3.投票时使用数组来保存不同决策树做出的决策，然后使用多数服从的机制，当出现 2 比 2 时，就在其中选择先进入数组的那个。

```
def get_panduan(vote):...

def use_vote(dataset):...

def vote(dataset, node):...
```

结论分析与问题：

实验结果：

```
..\data\i0.csv
验证得知，正确率为96.0%
..\data\i1.csv
验证得知，正确率为98.0%
..\data\i2.csv
验证得知，正确率为97.33333333333334%
..\data\i3.csv
验证得知，正确率为97.33333333333334%
验证得知投票后，正确率为99.33333333333333%
```

实验结果分析：

前八行位小决策树做出决策的正确率统计

最后一行是经过投票后，得到的正确率统计。

可以看出，使用集成算法后，比单个算法要好得多，而相较于上次实验，直接使用决策树，（上次的正确率位 98 左右）随机森林也有较大的提升。

实验代码：

```
# 201700301166 尹成林

import csv
import random
from math import log
import numpy as np

dict_iris = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
dict_siri = {0: 'Iris-setosa', 1: 'Iris-versicolor', 2: 'Iris-virginica'}


class treenode:
    def __init__(self, FenLei, FenLeiZhi, YeziPanDuan, node_id):
        self.FenLei = FenLei
        self.FenLeiZhi = FenLeiZhi
        self.YeziPanDuan = YeziPanDuan
        self.fenleiindex = [0, 1, 2]
        self.node_id = node_id
        '''if int(YeziPanDuan) == -1:
            print("产生中间节点{}，为第{}特征，分类值为{}".format(node_id, FenLei, FenLeiZhi))
        else:
            print("产生叶子节点{}，判断结果为{}".format(node_id, dict_siri[YeziPanDuan]))'''
    def popfenlei(self, fenlei):
        ii = 0
        for i in self.fenleiindex:
            if int(i) == int(fenlei):
                self.fenleiindex.pop(ii)
                return
            ii = ii+1

# 建立用来储存我们的树节点的
nodes = []


# 构建一个树
```

```
def buildTree(dataset, node_id):
    if get_ent(dataset) != 0:
        fenlei, fenleizhi = get_fenleidian(dataset)
        trnode = treenode(fenlei, fenleizhi, -1, node_id)
        trnode.popfenlei(fenlei)
        nodes.append(trnode)
        datasmall, databig = divide(dataset, fenlei, fenleizhi)
        #print(get_ent(datasmall))
        #print(get_ent(databig))
        buildTree(datasmall, 2 * node_id)
        #print("hhhhhhhhhhh{}".format(dataset[0][4]))
        buildTree(databig, 2 * node_id + 1)
    else:
        #print("hhhhhhhhhhh{}".format(dataset[0][4]))
        if len(dataset) == 0:
            #print("ffff")
            return
        panduan = dataset[0][3]
        trnode = treenode(0, 0, panduan, node_id)
        nodes.append(trnode)


# 构建一个小小树
def buildTreeR(dataset, node_id, idd):
    if get_ent(dataset) != 0:
        fenlei, fenleizhi = get_fenleidianR(dataset, idd)
        trnode = treenode(fenlei, fenleizhi, -1, node_id)
        trnode.popfenlei(fenlei)
        nodes.append(trnode)
        datasmall, databig = divide(dataset, fenlei, fenleizhi)
        #print(get_ent(datasmall))
        #print(get_ent(databig))
        buildTree(datasmall, 2 * node_id)
        #print("hhhhhhhhhhh{}".format(dataset[0][4]))
        buildTree(databig, 2 * node_id + 1)
    else:
        #print("hhhhhhhhhhh{}".format(dataset[0][4]))
        if len(dataset) == 0:
            #print("ffff")
            return
        panduan = dataset[0][3]
        trnode = treenode(0, 0, panduan, node_id)
        nodes.append(trnode)
```

```
# 加载函数
def loadcsv(name):
    f = csv.reader(open(name, 'r'))
    dataset = list(f)
    return dataset


# 把数据数字化
def digital(dataset):
    for i in range(len(dataset)):
        dataset[i][3] = dict_iris[dataset[i][3]]

# 把第 num 个数据进行排列,并取出可能的分割点
def get_mid(dataset,num):
    l1 = []
    l2 = []
    for i in range(len(dataset)):
        l1.append(dataset[i][num])
    l1 = list(set(l1))
    l1.sort()
    for i in range(len(l1)-1):
        l2.append(((float)(l1[i])+(float)(l1[i+1]))/2)
    return l2


# 分割函数
def randDivision(dataset , trainSize):
    copy = list(dataset)
    train = []
    while len(train)<trainSize:
        index = random.randrange(len(copy))
        train.append(copy.pop(index))
    return [train, copy]


'''
计算信息熵和信息增益
'''


# 划分数据集方便计算条件信息熵
def divide(dataset, index, dnum):
```

```
    dataset2 = list(dataset)
    dataset1 = []
    ii  = 0
    for i in dataset:
        if (float)(i[index])<(float)(dnum):
            dataset1.append(i)
            dataset2.pop(ii)
            ii = ii-1
        ii = ii+1
    return [dataset1, dataset2]


# 计算信息熵
def getP(dataset,num):
    p = 0.0
    a = len(dataset)
    if a == 0:
        return 0
    b = 0
    for i in dataset:
        if i[3] == num:
            b = b+1
    p = b/a
    #print(p)
    return p


def get_ent(dataset):
    p = []
    ent = 0.0
    for i in range(3):
        pp = getP(dataset, i)
        p.append(pp)
    for i in p:
        if i != 0:
            ent = ent - i*log(i, 2)
    return ent


# 计算信息增益
def get_gain(dataset, index, dnum):
    entD = get_ent(dataset)
    data1, data2 = divide(dataset, index, dnum)
    a1 = 0.0
```

```
    a2 = 0.0
    gain = 0.0
    '''print("data 的熵{}".format(entD))
    print("data1 的熵{}".format(get_ent(data1)))
    print("data2 的熵{}".format(get_ent(data2)))'''
    a2 = len(data2)/len(dataset)
    a1 = len(data1)/len(dataset)
    gain = entD - a1*get_ent(data1) - a2*get_ent(data2)
    return gain


# 得到单个特征最高的信息增益所在的分类点
def get_maxgain(dataset, index):
    max = -10000
    iii = 0
    dnums = get_mid(dataset, index)
    for i in dnums:
        gg = get_gain(dataset, index, i)
        if gg>=max:
            max = gg
            iii = i
    return iii


# 多个特征中选择一个分类点
'''
输入：数据集，各个特征的最佳分类点
输出：排序 list
'''
def get_maxtezhengR(dataset,node,idd):
    max = 0
    max_i = 0
    max_list = []
    for i in range(len(node)):
        if i==idd:
            print(i)
        else:
            a = get_gain(dataset, i, node[i])
            max_list.append(a)
            if max<a:
                max_i = i
                max = a
    return [max_list, max_i]
```

```
def get_fenleidianR(dataset, idd):
    divide_node = []
    for tezheng in range(3):

        divide_node.append(get_maxgain(dataset, tezheng))
        #print("**********")
        #print("所以应该第{}个特征选择{}作为分割点".format(tezheng, get_maxgain(dataset,
tezheng)))
        #print("**********")
    max_list, max_i = get_maxtezhengR(trainSet, divide_node, idd)
    #print(max_i)
    #print(max_list)
    return [max_i, divide_node[max_i]]

def get_fenleidian(dataset):
    divide_node = []
    for tezheng in range(3):
        divide_node.append(get_maxgain(dataset, tezheng))
        #print("**********")
        #print("所以应该第{}个特征选择{}作为分割点".format(tezheng, get_maxgain(dataset,
tezheng)))
        #print("**********")
    max_list, max_i = get_maxtezheng(trainSet, divide_node)
    #print(max_i)
    #print(max_list)
    return [max_i, divide_node[max_i]]


def find_(node_id):
    for node in range(len(nodes)):
        if nodes[node].node_id == node_id:
            return node+1
    return -1


votes = []

# 验证函数
def classify(dataset, nodes):
    corr = 0    # 记录正确的数目

    allof = len(dataset)    # 总数
    for item in dataset:
```

```
        aa = 1
        #print(aa)
        nnn = nodes[aa-1]
        while int(nnn.YeziPanDuan) == int(-1):  # 在主节点之间遨游
            if float(item[int(nnn.FenLei)]) <= float(nnn.FenLeiZhi):
                aa = 2*aa
                #print("左")
                nnn = nodes[find_(aa)-1]   # 左子树
            else:
                #print("
                aa = 2*aa+1
                nnn = nodes[find_(aa)-1]     # 右子树
        #print("树结果为{}， 实际为{}".format(nnn.YeziPanDuan, item[4]))
        if nnn.YeziPanDuan == item[3]:  # 判断是否正确
            #print("树结果为{}， 实际为{}".format(nnn.YeziPanDuan, item[4]))
            corr = corr+1
        aa = 1
    bibi = 0.0
    bibi = corr/allof
    return bibi


def get_panduan(vote):
    kkk = [0, 0, 0]
    for i in vote:
    max_index = kkk.index(max(kkk))
    # print(vote)
    return max_index


def use_vote(dataset):
    corr = 0
    fins = []
    for i in range(len(votes[0])):
        kkkk = []
        for vote in votes:
            kkkk.append(vote[i])
        fin = get_panduan(kkkk)
        fins.append(fin)
    for item in range(len(dataset)):
        if fins[item] == dataset[item][3]:  # 判断是否正确
            #print("树结果为{}， 实际为{}".format(fins[item], dataset[item][3]))
            corr = corr + 1
    allof = len(dataset)
```

```
    bibi = corr / allof
    return bibi


def vote(dataset, node):
    corr = 0  # 记录正确的数目
    vote = []
    allof = len(dataset)  # 总数
    for item in dataset:
        aa = 1
        # print(aa)
        nnn = nodes[aa - 1]
        while int(nnn.YeziPanDuan) == int(-1):  # 在主节点之间遨游
            if float(item[int(nnn.FenLei)]) <= float(nnn.FenLeiZhi):
                aa = 2 * aa
                # print("左")
                nnn = nodes[find_(aa) - 1]  # 左子树
            else:
                # print("
                aa = 2 * aa + 1
                nnn = nodes[find_(aa) - 1]  # 右子树
        # print("树结果为{},  实际为{}".format(nnn.YeziPanDuan, item[4]))
        vote.append(nnn.YeziPanDuan)
        if nnn.YeziPanDuan == item[3]:  # 判断是否正确
            # print("树结果为{},  实际为{}".format(nnn.YeziPanDuan, item[4]))
            corr = corr + 1
        aa = 1
    bibi = 0.0
    #print(vote)
    votes.append(vote)
    bibi = corr / allof
    return bibi


# main function
if __name__=="__main__":
    noddds = [[], [], [], []]
    for i in range(4):
        name = "..\data\i{}.csv".format(i)
        print(name)
        dataset = loadcsv(name)
        digital(dataset)
        trainSet, testSet = randDivision(dataset, 100)
        #print(get_fenleidian(trainSet))
```

```
        #set1, set2 = divide(trainSet, 3, 0.8)
        #tr = treenode(0,0,0,0)
        buildTree(trainSet, 1)
        noddds[i] = nodes
        #print(nodes)
        a = vote(dataset, noddds[i])
        print("验证得知，正确率为{}%".format(a * 100))
        nodes = []
    a = use_vote(dataset)
    print("验证得知投票后，正确率为{}%".format(a*100))
    #aaaaa = FiveFordCV(dataset, 150, 5)
    #print("经五折交叉验证后得知，正确率为{}%".format(aaaaa*100))

import csv
import random
from math import log
import numpy as np


dict_iris = {'Iris-setosa': 0, 'Iris-versicolor': 1, 'Iris-virginica': 2}
dict_siri = {0: 'Iris-setosa', 1: 'Iris-versicolor', 2: 'Iris-virginica'}


# 加载函数
def loadcsv(name):
    f = csv.reader(open(name, 'r'))
    dataset = list(f)
    return dataset


# 把数据数字化
def digital(dataset):
    for i in range(len(dataset)):
        dataset[i][4] = dict_iris[dataset[i][4]]


def write_csv_file(path, head, data):
    try:
        with open(path, ) as csv_file:
            writer = csv.writer(csv_file, dialect='excel')
            if head is not None:
                writer.writerow(head)
            for row in data:
                writer.writerow(row)
```

```
            print("Write a CSV file to path %s Successful." % path)
    except Exception as e:
        print("Write an CSV file to path: %s, Case: %s" % (path, e))


def dddvvv(num, dataset):
    finalset = []
    for a in dataset:
        aa = []
        for i in range(5):
            if i!=num:
                aa.append(a[i])
        finalset.append(aa)
    return finalset



# main function
if __name__=="__main__":
    name = "..\data\iris.csv"
    dataset = loadcsv(name)
    for i in range(4):
        fuck = dddvvv(i, dataset)
        write_csv_file("..\data\i{}.csv".format(i), None, fuck)
```