华中科技大学

《机器学习导论》实验报告

姓名：陈东升

班级：电信 1604 班

学号：U201613325

指导老师：王邦

实验名称：决策树算法与实现

任务一：使用决策树预测隐形眼镜类型

一、 任务描述

眼科医生是如何判断患者需要佩戴的镜片类型的？隐形眼镜数据集是非常著名的数据集，它包含了很多患者眼部状况的观察条件以及医生推荐的隐形眼镜类型。隐形眼镜类型包括硬材质（hard）、软材质（soft）以及不适合佩戴隐形眼镜（no lenses）。以下为该数据集的部分数据，包括年龄、近视 or 远视类型，是否散光，是否容易流泪，最后 1 列为应佩戴眼镜类型：

```
young   myope   no      reduced no lenses
young   myope   no      normal  soft
young   myope   yes     reduced no lenses
young   myope   yes     normal  hard
young   hyper   no      reduced no lenses
young   hyper   no      normal  soft
young   hyper   yes     reduced no lenses
young   hyper   yes     normal  hard
pre     myope   no      reduced no lenses
```

准备数据：用 Python 解析文本文件，解析 tab 键分割的数据行；
分析数据：快速检查数据，确保正确地解析数据内容；
训练算法：采用决策树分类算法，获得预测隐形眼镜类型的决策树。

二、 问题分析

这是一个很规范的有监督类型的分类问题，我们根据数据集提供供的信息建立决策树即可，算法源自 C4.5。

三、 编程环境

① 编程语言：python

② 编译环境：python3+Pycharm

四、 构建决策树的模块及其代码（含注释）

① 计算数据集的香农熵

```
def calcShannonEnt(dataSet):  # 计算熵
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet:
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys():
            labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
        prob = float(labelCounts[key]) / numEntries
        shannonEnt -= prob * log(prob, 2)
    return shannonEnt
```

② 选择最佳划分属性

```
def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1  # feature个数
    baseEntropy = calcShannonEnt(dataSet)  # 整个dataset的熵
    bestInfoGainRatio = 0.0
    bestFeature = -1
    for i in range(numFeatures):
        featList = [example[i] for example in dataSet]  # 每个feature的list
        uniqueVals = set(featList)  # 每个list的唯一值集合
        newEntropy = 0.0
        splitInfo = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)  # 每个唯一值对应的剩余feature的组成子集
            prob = len(subDataSet) / float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
            splitInfo += -prob * log(prob, 2)
        infoGain = baseEntropy - newEntropy  # 这个feature的infoGain
        if (splitInfo == 0):  # fix the overflow bug
            continue
        infoGainRatio = infoGain / splitInfo  # 这个feature的infoGainRatio增益率
        if (infoGainRatio > bestInfoGainRatio):  # 选择最大的gain ratio
            bestInfoGainRatio = infoGainRatio
            bestFeature = i  # 选择最大的gain ratio对应的feature
    return bestFeature
```

③ 划分数据子集

```
def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:  # 只看当第i列的值=value时的item
```

```
            reduceFeatVec = featVec[:axis]  # featVec的第i列给除去
            reduceFeatVec.extend(featVec[axis + 1:])
            retDataSet.append(reduceFeatVec)
    return retDataSet
```

④ 构建决策树

```
def createTree(dataSet, labels):
    classList = [example[-1] for example in dataSet]  # ['N', 'N', 'Y', 'Y', 'Y', 'N', 'Y']
    if classList.count(classList[0]) == len(classList):
        # classList所有元素都相等，即类别完全相同，停止划分
        return classList[0]  # splitDataSet(dataSet, 0, 0)此时全是N，返回N
    if len(dataSet[0]) == 1:  # [0, 0, 0, 0, 'N']
        # 遍历完所有特征时返回出现次数最多的
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)  # 0 -> 2
    # 选择最大的gain ratio对应的feature
    bestFeatLabel = labels[bestFeat]  # outlook -> windy
    myTree = {bestFeatLabel: {}}
    # 多重字典构建树{'outlook': {0: 'N'
    del (labels[bestFeat])  # ['temperature', 'humidity', 'windy'] -> ['temperature', 'humidity']
    featValues = [example[bestFeat] for example in dataSet]  # [0, 0, 1, 2, 2, 2, 1]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]  # ['temperature', 'humidity', 'windy']
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels)
        # 划分数据，为下一层计算准备
    return myTree
```

⑤ 绘制决策树

```
def plotNode(Nodename, centerPt, parentPt, nodeType):
    createPlot.ax1.annotate(Nodename, xy=parentPt, xycoords='axes fraction', xytext=centerPt,
                            textcoords='axes fraction', va="center", ha="center", bbox=nodeType, arrowprops=arrow_args)
```

```
def getNumLeafs(myTree):  # 获取叶节点的数目
    numLeafs = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__ == 'dict':
            numLeafs += getNumLeafs(secondDict[key])  # 递归
        else:
            numLeafs += 1
    return numLeafs
```

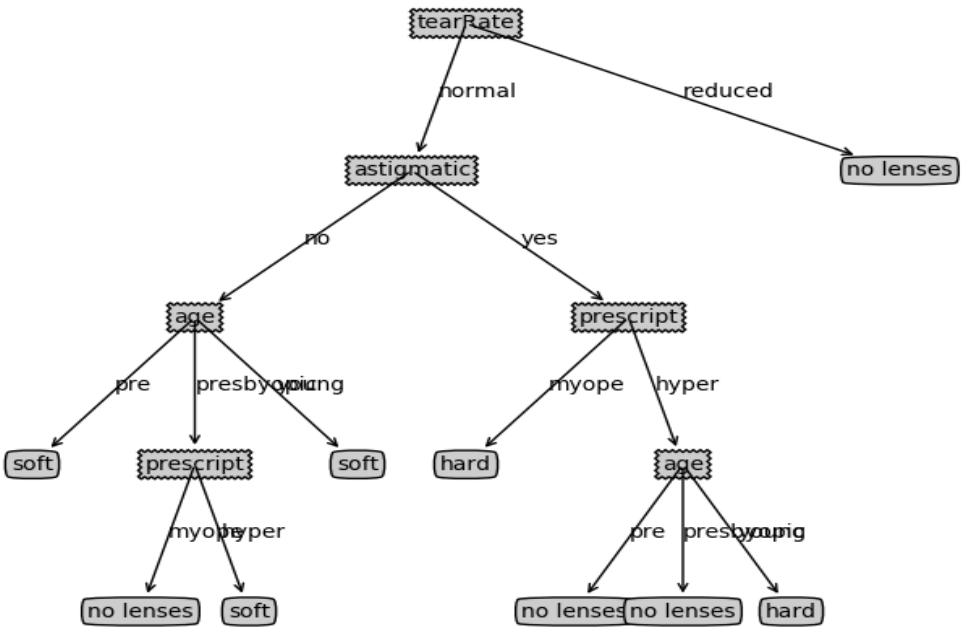
```
def getTreeDepth(myTree):  # 获取决策树的高度
    maxDepth = 0
    firstStr = list(myTree.keys())[0]
    secondDict = myTree[firstStr]
    for key in secondDict.keys():
        if type(secondDict[key]).__name__ == 'dict':
            thisDepth = 1 + getTreeDepth(secondDict[key])  # 递归
        else:
            thisDepth = 1
        if thisDepth > maxDepth:
            maxDepth = thisDepth
    return maxDepth
```

```
def plotMidText(cntrPt, parentPt, txtString):
    xMid = (parentPt[0] - cntrPt[0]) / 2.0 + cntrPt[0]
    yMid = (parentPt[1] - cntrPt[1]) / 2.0 + cntrPt[1]
    createPlot.ax1.text(xMid, yMid, txtString)
def plotTree(myTree, parentPt, nodeTxt):
    numLeafs = getNumLeafs(myTree)  # this determines the x width of this tree
    # 计算树的高
    depth = getTreeDepth(myTree)
    # 第一个关键字为第一次划分数据集的类别标签，附带的取值表示子节点的取值
    firstStr = list(myTree.keys())[0]  # the text label for this node should be this
    # 下一个节点的位置
    cntrPt = (plotTree.xOff + (1.0 + float(numLeafs)) / 2.0 / plotTree.totalW, plotTree.yOff)
    # 计算父节点和子节点的中间位置，并在此处添加简单的文本信息
    plotMidText(cntrPt, parentPt, nodeTxt)
    # 绘制此节点带箭头的注解
    plotNode(firstStr, cntrPt, parentPt, deciNode)
    # 新的树，相当于脱了一层皮
    secondDict = myTree[firstStr]
    # 按比例减少全局变量plotTree.yOff
    plotTree.yOff = plotTree.yOff - 1.0 / plotTree.totalD
    #
    for key in secondDict.keys():
        # 判断子节点是否为字典类型
        if type(secondDict[key]).__name__ == 'dict':
            # 是的话表明该节点也是一个判断节点，递归调用plotTree()函数
            plotTree(secondDict[key], cntrPt, str(key))
        else:
            # 不是的话更新x坐标值
            plotTree.xOff = plotTree.xOff + 1.0 / plotTree.totalW
            # 绘制此节点带箭头的注解
            plotNode(secondDict[key], (plotTree.xOff, plotTree.yOff), cntrPt, leafNode)
            # 绘制此节点带箭头的注解
            plotMidText((plotTree.xOff, plotTree.yOff), cntrPt, str(key))
    # 按比例增加全局变量plotTree.yOff
    plotTree.yOff = plotTree.yOff + 1.0 / plotTree.totalD
```

```
def createPlot(inTree):
    fig = plt.figure(1, facecolor='white')
    fig.clf()
    axprops = dict(xticks=[], yticks=[])
    createPlot.ax1 = plt.subplot(111, frameon=False, **axprops)
    plotTree.totalW = float(getNumLeafs(inTree))
    plotTree.totalD = float(getTreeDepth(inTree))
    plotTree.xOff = -0.5 / plotTree.totalW
    plotTree.yOff = 1.0
    plotTree(inTree, (0.5, 1.0), '')
    plt.show()
```

```
fr = open('C:\\Users\\chend\\Desktop\\MLproject\\deciTree\\Task1\\lenses.txt')
lenses = [inst.strip().split('\t') for inst in fr.readlines()]
lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
lensesTree = myTree.createTree(lenses, lensesLabels)
createPlot(lensesTree)
```

五、 运行结果



六、 实验总结

在通过编写代码将 C4.5 决策树算法实现了一遍的过程中，我加深了对算法原理的理解，同时也提高了动手能力和对 python 语言的熟练运用程度。

任务二：根据用户采集的 WiFi 信息采用决策树预测用户所在房间

一、任务描述

（1）数据集讲解

数据集：训练集存于 TrainDT.csv 中；测试集存于 TestDT.csv 中。

BSSIDLabel：BSSID 标识符，每个 AP（接入点，如路由器）拥有 1 个或多个不同的 BSSID，但 1 个 BSSID 只属于 1 个 AP；

RSSLabel：该 BSSID 的信号强度，单位 dbm；

RoomLabel：该 BSSID 被采集时所属的房间号，为类标签，测试集中也含该标签，主要用于计算预测准确度；

SSIDLabel：该 BSSID 的名称，不唯一；

finLabel：finLabel 标号相同，表示这部分 BSSID 在同一时刻被采集到；我们将在同一时刻采集的所有 BSSID 及其相应 RSS 构成的矢量称为一个指纹 $f_i = [BSSID_1: RSS_1, BSSID_2: RSS_2, \dots, \text{RoomLabel}]$；由于 BSSID 的 RSS 在不同位置大小不同，因此指纹可以唯一的标识一个位置。

BSSIDLabel	RSSLabel	RoomLabel	SSIDLabel	finLabel
06:69:6c:0a:bf:02	-56	1	HUST_WIRELESS	1
20:76:93:3a:ae:78	-69	1	HC5761	1
4a:69:6c:07:a1:e7	-69	1	HUST_WIRELESS_AUTO	1
0e:69:6c:0a:bf:02	-63	1	HUST_WIRELESS_AUTO	2
4a:69:6c:07:a1:e7	-66	1	HUST_WIRELESS_AUTO	2
2a:69:6c:05:c5:25	-67	1	HUST_WIRELESS_AUTO	2
08:57:00:7b:63:16	-72	1	TL-WTR9500	2

（2）注意：

1. 连续值处理：一方面，可以将每个特征划分为两个属性，未接收到 RSS 用 0 表示，接收到 RSS 用 1 表示，则一个样本可表示为

$$f_i = [BSSID_1: 1, BSSID_2: 0, \dots]$$

另一方面可采用二分法对连续属性进行处理，计算每个划分点的信息增益；

2. 特征构造：不同样本中 BSSID 集合不尽相同，因此可以采用所有样本 BSSID 集合的并集作为特征，如指纹f_i的 BSSID 集合为$B_i = \{BSSID_j | BSSID_j \in f_i\}$，则特征可表示为$B_u = \bigcup_{i=1}^{N} B_i$。

3. 缺失值处理：本身缺失值也可以作为特征属性；若采用功能二分法则可以填补特殊值-100 等。

4. 举例说明：

$$f_1 = [BSSID_1: 1, BSSID_2: 0, BSSID_3: 1, BSSID_4: 1,0]$$
$$f_2 = [BSSID_1: 1, BSSID_2: 1, BSSID_3: 1, BSSID_4: 0,1]$$

则f_1本身只接收到$BSSID_1$、$BSSID_3$和$BSSID_4$共 3 个$BSSID$；f_2本身只接收到$BSSID_1$、$BSSID_2$和$BSSID_3$共 3 个$BSSID$；特征为所有样本$BSSID$的并集$B_u = \{BSSID_1, BSSID_2, BSSID_3, BSSID_4\}$；接收到的$BSSID$其值用 1 表示，缺失值用 0 填充；最后一列表示样本类标签，f_1属于房间 0，f_2属于房间 1。

二、问题分析

按照题目中的提示，我们在编写程序之前需要先对数据进行处理，我们将在同一时刻采集的所有 BSSID 及其相应 RSS 构成的矢量称为一个指纹$f_i = [BSSID_1: RSS_1, BSSID_2: RSS_2, \dots, \text{RoomLabel}]$；将训练集数据得到的所有的指纹中的 BSSID 进行取并集，得出属性集，存放在一个 list 中，之后使用二分法判断出最佳划分点，然后将他们的 RSS 进行二值化处理：未接受到 RSS 用 0 表示，接收到 RSS 用 1 表示。同理在训练的过程中，我们对测试集也进行同样的操作。这里我们使用 csv 模块对 csv 文件进行处理，使用 sklearn 模块对处理后的数据构建决策树，最后后计算训练精度。

三、源代码

```
import csv
from sklearn import tree

with open('C:\\Users\\chend\\Desktop\\MLproject\\deciTree\\Task2\\TrainDT.csv', 'r')
as f:
    a = set()
    b = []
    reader = csv.reader(f)
    fieldnames = next(reader)
    csv_reader = csv.DictReader(f, fieldnames=fieldnames)
    for row in csv_reader:
        d = {}
        for k, v in row.items():
            d[k] = v
            if k == 'BSSIDLabel':
                a.add(v)
        b.append(d)

    c = list(a)
    num = len(c)
    d = [0] * num
    e = []
    x = int(b[0]['finLabel'])

    for i in b:
        if int(i['finLabel']) == x:
            d[c.index(i['BSSIDLabel'])] = 1
        else:
            e.append(d)
            d = [0] * num
```

```
            d[c.index(i['BSSIDLabel'])] = 1
            x = x + 1
    e.append(d)

    f = []
    y = int(b[0]['finLabel'])
    f.append(int(b[0]['RoomLabel']))
    for i in b:
        if int(i['finLabel']) != y:
            f.append(int(i['RoomLabel']))
            y = int(i['finLabel'])

with open('C:\\Users\\chend\\Desktop\\MLproject\\deciTree\\Task2\\TestDT.csv', 'r') as
f1:
    p = set()
    q = []
    reader1 = csv.reader(f1)
    fieldnames1 = next(reader1)
    csv_reader1 = csv.DictReader(f1, fieldnames=fieldnames1)
    for row in csv_reader1:
        d = {}
        for k, v in row.items():
            d[k] = v
            if k == 'BSSIDLabel':
                p.add(v)
        q.append(d)

    w = list(p)
    num = len(a)
    d = [0] * num
    t = []
    y = int(q[0]['finLabel'])

    for i in q:
        if int(i['finLabel']) == y:
            d[w.index(i['BSSIDLabel'])] = 1
        else:
            t.append(d)
            d = [0] * num
            d[w.index(i['BSSIDLabel'])] = 1
            y = y + 1
    t.append(d)

    h = []
```

```
    y = int(q[0]['finLabel'])
    h.append(int(q[0]['RoomLabel']))
    for i in q:
        if int(i['finLabel']) != y:
            h.append(int(i['RoomLabel']))
            y = int(i['finLabel'])

clf = tree.DecisionTreeClassifier(max_depth=30)
clf = clf.fit(e, f)

print(clf.predict(t))
print(clf.score(t, h))
```

四、训练结果及精度

```
C:\Users\chend\Documents\ML\venv\Scripts\python.exe C:/Users/chend/Documents/ML/task2
[2 1 2 2 1 1 2 1 2 1 2 1 1 2 2 2 2 2 2 2 1 1 2 3 1 1 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 3 3 3 3 2 3 3 3 2 2 3 3 3 3 2
 3 3 3 3 3 3 3 3 2 3 3 3 3 3 2 3 2 3 3 2 3 2 3 3 2 3 3 3 3 3 3 3 3 3]
0.5688073394495413
```

上面图片是程序的输出结果，我们得出使用建立的决策树对测试集进行测试，得到的训练精度只有 57%左右。

五、结论与反思

上面结果显示我的模型的训练精度只有 57%左右，这个精度远远不够理想，也说明了我的决策树训练的不够好，由于时间有限，我决定课下再抽空分析原因并且对其进行改进。

任务三：IMDB 数据集电影评测分类（二分类问题）

一、任务描述

（1）数据集讲解:

该数据集是IMDB电影数据集的一个子集，已经划分好了测试集和训练集，训练集包括25000条电影评论，测试集也有 25000 条，该数据集已经经过预处理，将每条评论的具体单词序列转化为词库里的整数序列，其中每个整数代表该单词在词库里的位置。例如，整数 104 代表该单词是词库的第 104 个单词。为实验简单，词库仅仅保留了 10000 个最常出现的单词，低频词汇被舍弃。每条评论都具有一个标签，0 表示为负面评论，1 表示为正面评论。

训练数据在 train_data.txt 文件下，每一行为一条评论，训练集标签在 train_labels.txt 文件下，每一行为一条评论的标签；测试数据在 test_data.txt 文件下，测试数据标签未给出。

（2）思路：

这里提供一个最简单的思路，还有其它思路同学们可以自行思考。

首先，需要将每条评论转换为特征向量，这里可以采用 one-hot 编码，举个例子：这里采用的词库大小为 10000，因此转换的 one-hot 编码也是 10000 维的，如某条评论为[3, 5]，则转换得到的 one-hot 编码的 10000 维向量，只有索引为 3 和 5 的元素为 1，其余全部为 0。

将每条评论都转换为 one-hot 编码后，再采用决策树算法进行分类。

二、问题分析

我们使用题目中的方法，使用 one-hot 编码对原.txt 文件进行处理，最终的训练集构成了一个 25000*10000 的矩阵，结果标签是一个长度为 25000 的向量，然后将其运用到任务一中的程序里面进行训练，最后用测试集进行测试，并将结果导出到.txt 文件中。

三、部分代码

```
import myTree
import gc

fr = open('C:\\Users\\chend\\Desktop\\MLproject\\deciTree\\Task3\\train\\train_data.txt') # 读取训练集中的字符个数
fa = open('C:\\Users\\chend\\Desktop\\MLproject\\deciTree\\Task3\\train\\train_labels.txt')
trainMat = [inst.strip().split(' ') for inst in fr.readlines()]
labelMat = [inst.strip().split('    ') for inst in fa.readlines()]
lenoftrainMat=len(trainMat)
MyData = [['0' for i in range(10001)] for j in range(25000)] # 定义一个10001维的数组，共有25000个向量
for i in range(lenoftrainMat):
   for j in trainMat[i]:
      MyData[i][int(j)] = '1'
```

```
    MyData[i][-1] = labelMat[i][0]
    gc.collect()
myLabels = [str(i) for i in range(10000)]
task3Tree = myTree.createTree(MyData, myLabels)
print(task3Tree)
```

四、结论与反思

将测试集运行到训练集所产生的决策树中进行测试，我得到了三组.txt 文件。程序实现起来不难，主要是这里的训练数据集有 25000 个 10000 维的向量，这会造成非常大的计算量，计算非常慢，我只有借用同学的云服务器才能进行计算。我初步有所体会，当数据非常大的时候，训练起来真的不只是程序本身的问题了，更重要的是设备硬件条件要足够。总之，这次实验的体验还是很不错的，同时也加强了自己的实践能力。