

华中科技大学

机器学习导论作业报告

专业院系 ： 电子信息与通信学院
专业班级 ： 电信1805班
学生学号 ： U201813372
学生姓名 ： 朱浩然

TASK1:使用决策树预测隐形眼镜类型

一、软件设计

1. 设计目标

① 问题背景：眼科医生是如何判断患者需要佩戴的镜片类型的？

② 目标分析

2. 算法设计分析

①决策树算法

二、运行结果

结果截图1

结果截图二

结果截图三

三、总结体会

程序设计心得:

TASK2: 根据用户采集的WiFi信息采用决策树预测用户所在房间

一、软件设计

1. 设计目标

① 数据集讲解：

② 目标分析

2. 算法设计分析

①决策树算法

②数据处理

③决策树测试

二、运行结果

结果截图1

结果截图二

结果截图三

结果截图四

三、总结体会

程序设计心得:

TASK3: IMDB数据集电影评测分类（二分类问题）

一、软件设计

1. 设计目标

① 数据集讲解：

② 目标分析

2. 算法设计分析

①决策树算法

②数据处理

③文件操作

二、运行结果

结果截图1

结果截图二

结果截图三

结果截图四

三、总结体会

程序设计心得:

TASK1:使用决策树预测隐形眼镜类型

一、软件设计

1. 设计目标

① 问题背景：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

② 目标分析

- 设计一个决策树生成算法，对提供的数据集进行处理，得到关于眼镜类型的决策树，用该决策树对输入的一组数据进行判断，同时做到决策树的绘制

2. 算法设计分析

所用项目环境 IDE：VS code Python版本：python3.7.5

① 决策树算法

- 决策树学习过程：
 - **数据预处理**：数据预处理是指对获取或者提供的数据文件进行文件操作，提取有效信息，转化为适合特征选择的数据类型（如列表，字典等）
 - **特征选择**：特征选择是指从训练数据中众多的特征中选择一个特征作为当前节点的分裂标准，如何选择特征有着很多不同量化评估标准标准，从而衍生出不同的决策树算法
 - **决策树生成**：根据选择的特征评估标准，从上至下递归地生成子节点，直到数据集不可分则停止决策树停止生长。树结构来说，递归结构是最容易理解的方式。
 - **剪枝**：决策树容易过拟合，一般来需要剪枝，缩小树结构规模、缓解过拟合。剪枝技术有预剪枝和后剪枝两种

本次任务的数据量较小，决策树不需要进行剪枝操作

算法代码分析时，只粘贴了函数头与返回值，叙述代码细节，目的在于粘贴源码，篇幅过长

=====>> 数据预处理 <<=====

算法介绍：对提供的训练文件进行读取，并转化为所需要的列表或者字典格式

- **数据处理函数**：使用python的文件操作按行读取提供的lenses.txt文件，并将每一行转化为列表，得到二级列表的数据

```
def product():  
    ...  
    return lensesTree
```

- 代码细节：

(1) `.decode()`设置文件解码方式；`.strip()`去除每一行的开始结尾；`.split('\t')`按照\t将数据分片成为列表

=====>> 特征选择 <<=====

算法介绍：对处理好的数据进行熵，信息增益等的计算确定决策树的划分

- **熵计算函数：**对输入的数据信息熵的计算

```
def calcShannonEnt(dataSet):  
    ...  
    return shannonEnt
```

- **代码细节：**

(1) for featVec in dataSet 一段循环，目的在对输入的数据遍历，存储每一种特征的数量
(2) for key in labelCounts 一段循环，目的在于对于上个循环存储好的特征数量，编写对应的数学公式计算熵；概率公式： $L(X_i) = -\log_2 P(X_i)$ ；熵公式： $H = -\sum_{i=1}^n P(X_i) \log_2 P(X_i)$

- **数据集划分函数：**该函数是为了将指定划分的左右提取出来，方便进一步计算信息增益

```
# 按照给定特征划分数据集  
def splitDataSet(dataSet, axis, value):  
    ...  
    return retDataSet
```

- **代码细节：**

(1) 一次循环遍历数据，运用数组的剪切增加操作保存划分后的数据

- **最好划分特征选择函数：**该函数是计算每一种特征划分后的信息增益，选择信息增益最大的为划分特征

```
# 选择最好的数据集划分方式  
def chooseBestFeatureToSplit(dataSet):  
    ...  
    return bestFeature # 返回最佳特征值索引
```

- **代码细节：**

(1) 遍历所有特征，并使用`set()`去重，调用熵计算函数与数据集划分函数，计算信息增益
(2) `bestInfoGain`保存最好信息增益，`infoGain`保存每一次运算时的信息增益，每次计算完`infoGain`后于最好增益比较，更大则更换`bestInfoGain`

- **叶子节点确定函数：**如果数据集已经处理了所有属性，但是类标签依然不是唯一的，采用多数表决的方法决定该叶子节点的分类

```
def majorityCnt(classList):  
    ...  
    return sortedClassCount[0][0] # 出现次数最多的元素（类标签）
```

- 代码细节:

(1) 对classList类标签列表进行遍历, 计算每种类标签的数量

=====>> 决策树生成 <<=====

算法介绍: 处理好的数据生成决策树, 并将其保存, 可视化

- **决策树生成函数:** 调用上面叙述的几种函数, 对处理好的数据进行决策树的绘制, 保存为多层字典的形式

```
def createTree(dataSet, labels):    # dataSet:数据集 labels:分类属性标签
    ...
    return myTree    # 返回决策树
```

- 代码细节:

(1) 采用递归的思路递归构建决策树, 对每一次函数运行执行熵计算, 数据集划分, 最好划分特征选取等操作, 建立好一层之后将其下一层采用递归同样构建。

(2) 递归部分: myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value), subLabels)

- **决策树可视化函数:** 采用graphviz包中的Digraph绘制生成好的决策树, 并保存为pdf

```
dot = Digraph(comment='The Test Table')
def treeplot(tree, node_num)
    ...
def plot_view(myTree):
    ...
    treeplot(myTree, "0")
    dot.render('task1/result/mytree.gv', view=True)
```

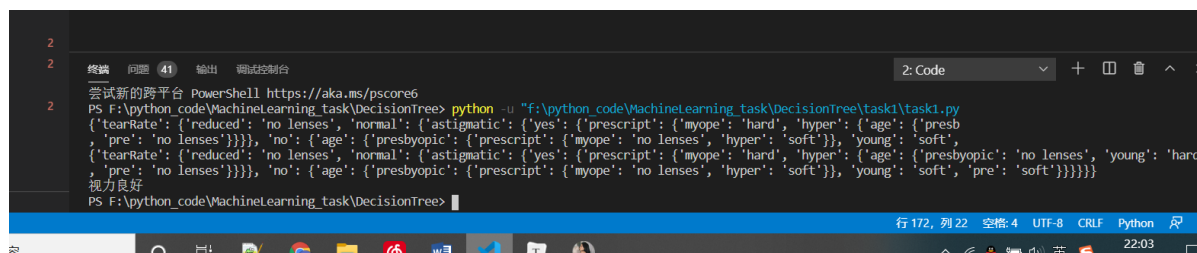
- 代码细节:

(1) 采用递归方式遍历字典决策树, dot.node()创建节点, dot.edge()创建节点之间的关系, dot.render将决策树保存为pdf文件

(2) 递归部分: treeplot(tree[first_label][i], root)

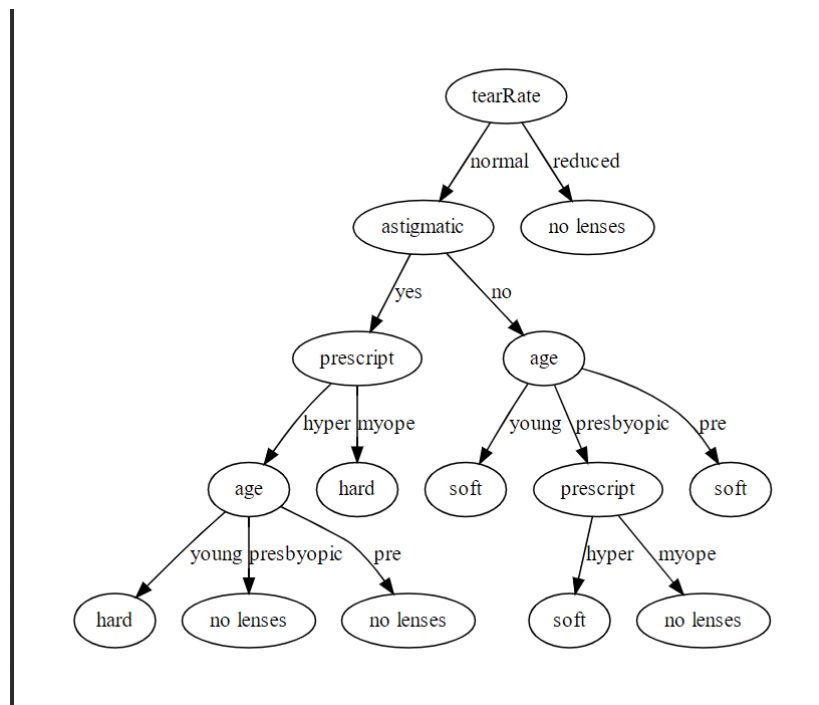
二、运行结果

结果截图1



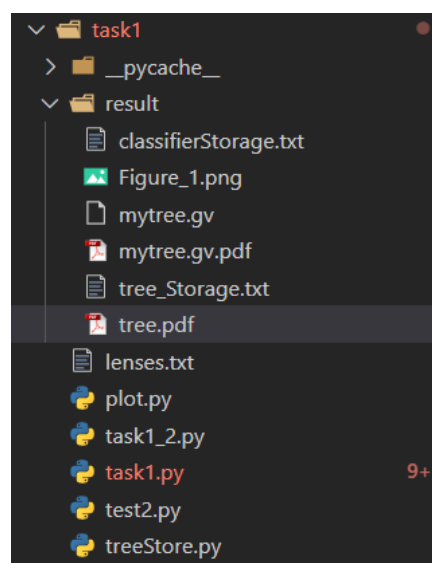
- 图片分析: 上图运行结果, 控制台打印决策树运行良好, 给定一段数据运用决策树计算结果正常, 程序设计成功

结果截图二



- 图片分析：决策树绘制如图所示，与控制台打印出的决策树保持一致，决策树可视化设计成功

结果截图三



- 图片分析：上图为任务一的文件目录，程序结构整体如上图所示

三、总结体会

程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 软件工程量较大，建议采用模块化设计，同时不同的内容可以分为不同的文件，优先设计函数头，确定参数及返回值后具体设计逻辑
切记不要一个文件写完，那样导致调试修改复杂，不利于后续修改。

3. 决策树算法的构建，重点在于对于决策树的结构理解，同时学会了递归的相关使用，能大大减少代码的复杂度，但递归参数的输入较容易出错，可以先单步调试，找到问题一步步修改

TASK2: 根据用户采集的WiFi信息采用决策树预测用户所在房间

一、软件设计

1. 设计目标

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

$f_i = [BSSID_1: RSS_1, BSSID_2: RSS_2, \dots, RoomLabel]$ ；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 目标分析

- 对给出的训练集进行数据处理，将同一时刻的构成一个矢量，用列表存储处理好的数据
- 再使用task1中写好的决策树生成算法进行决策树生成
- 根据决策树构建一个分类器，对测试集的数据进行测试，得出测试集的精度

2. 算法设计分析

所用项目环境 IDE：VS code Python版本：python3.7.5

① 决策树算法

TASK1中详细描述了决策树算法的实现，TASK2中调用相同的算法，下面就不再一一叙述

[跳转到task决策树算法](#)

②数据处理

=====>> 数据处理 <<=====

算法介绍：将训练集的数据进行处理，按照采集时间的不同生成不同矢量

$$f_i = [BSSID_1:RSS_1, BSSID_2:RSS_2, \dots, RoomLabel]$$

- 训练集数据处理算法：

```
def train_data(dataset_Path):    # 文件地址
    dataset, BSSID_List, fin_List = data_Get(dataset_Path) # 读取文件数据
    ...
    text_save('task2/result/testdata.txt', lenses) # 将处理好的数据存储为txt文本
    ...
    return lenses, bssidlist, roomlist # 处理后的数据， bssid去重后的集合， 每条数据的
    room集合

def text_save(filename, data): # filename为写入txt文件的路径，data为要写入数据列表。
    file = open(filename, 'w')
    ...
    file.close() # 保存文件函数
```

- 代码细节：

- (1) 读取文件数据后，对dataset进行遍历，每一次读取bssid, room, fin信息，然后将lenses每一行对应位置填1，使其成为01序列
- (2) lenses需要初始化一个二维数组，bssidlist是用来求lenses填1位置的索引
- (3) text_save()函数需要将列表转化为txt，每一个元素之间填\t分割，行末换行

③决策树测试

=====>> 决策树生成测试 <<=====

算法介绍：利用得到的决策树生成相对应的分类器，对测试集的数据进行测试

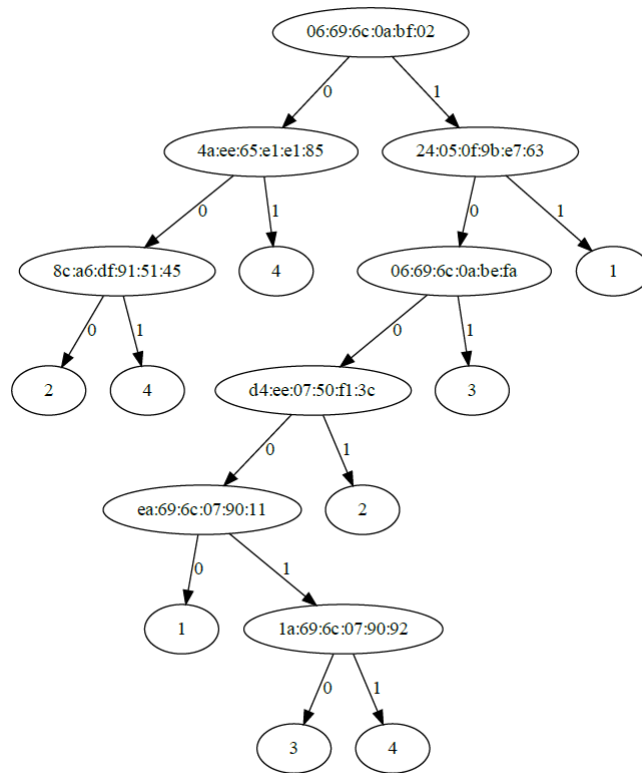
- 分类器生成函数：

```
def classify(inputTree, featLabels, testVec):    # 决策树，特征属性，测试数据
    ...
    return classLabel # 测试结果
```

- 代码细节：

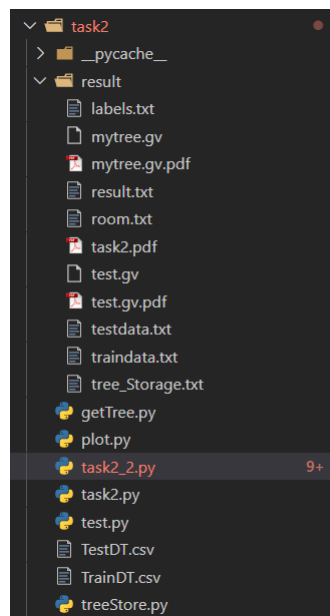
- (1) 采用递归的思路递归构建分类器，对输入的数据，按照决策树的结构递归“行走”得出相对应叶子节点内容为结果
- (2) 递归部分：classLabel = classify(valueOfFeat, featLabels, testVec)

- 决策树测试函数：



- 图片分析：上图为任务二生成的决策树，可以看到与图1中控制台打印的决策树保持一致，决策树构建无误

结果截图四



- 图片分析：上图为任务二的文件目录

三、总结体会

程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 软件工程量较大，建议采用模块化设计，同时不同的内容可以分为不同的文件，优先设计函数头，确定参数及返回值后具体设计逻辑
切记不要一个文件写完，那样导致调试修改复杂，不利于后续修改。

TASK3: IMDB数据集电影评测分类（二分类问题）

一、软件设计

1. 设计目标

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 目标分析

- 对给出的训练集进行数据处理，one-hot编码，10000维
- 再使用sklearn库生成决策树

自己写的决策树处理这么大的数据时间太长

- 对测试集的数据进行测试，得出测试集的预测标签结果

2. 算法设计分析

所用项目环境 IDE: VS code Python版本: python3.7.5

① 决策树算法

TASK1中详细描述了决策树算法的实现，TASK3的数据量太大，自己编写的决策树处理时间高达几个小时，改用sklearn库

[跳转到task决策树算法](#)

- sklearn决策树算法:

```
def tree_func(traindata, label_data):
    clf = tree.DecisionTreeClassifier(max_depth=10)
    clf = clf.fit(list(traindata), label_data)
    dot_data = StringIO()
    tree.export_graphviz(clf, out_file=dot_data, class_names=clf.classes_,
                        filled=True, rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    graph.write_pdf("task3/result/task3_tree.pdf")
    print('决策树生成完毕')
    txtsave('task3/result/test_result.txt', clf.predict(list(traindata)))
    print('测试集结果预测完毕')
```

- 代码细节:

- (1) 使用tree.DecisionTreeClassifier构造决策树，使用clf.fit()导入训练数据及标签，使用tree.export_graphviz绘制决策树
- (2) txtsave()函数实现将预测标签列表储存为txt文本

②数据处理

===== >> 数据处理 <<=====

算法介绍：将训练集的数据进行处理，按照采集时间的不同生成不同矢量

$f_i = [BSSID_1:RSS_1, BSSID_2:RSS_2, \dots, RoomLabel]$

- 训练集数据处理算法:

```
def dataget(file_path1, file_path2):
    train_data = txtread(file_path1)    # 读取训练集数据
    label_data = txtread(file_path2)    # 读取训练集标签
    ...
    return traindata, label_data        # 处理后训练数据，训练集标签
```

- 代码细节:

- (1) 将traindata初始化为25000*10000的二维列表，二维列表每一个元素，将读取的训练集数据每一行出现的数字作为索引，将对应位置设为1，其余初始化为0，使训练集数据每一行都形成10000个1，0序列
- (2) txtread()函数实现读取txt文件，并将其转化为二维的列表存储

③文件操作

===== >> 文件操作 <<=====

算法介绍：txt文件的读取以及保存函数

- txt读取算法:

```
def txtread(filepath):
    with open(filepath, 'r') as fr:
        file_data = [inst.strip().replace('\n', '').split(' ') for inst in fr.readlines()]
        # print(file_data)
    print(filepath, '文件读取成功')
    return file_data
```

- 代码细节:

(1) 使用python文件操作, 注意.strip()去除每一行的开始结尾; .split()将数据分片成为列表

- txt保存算法:

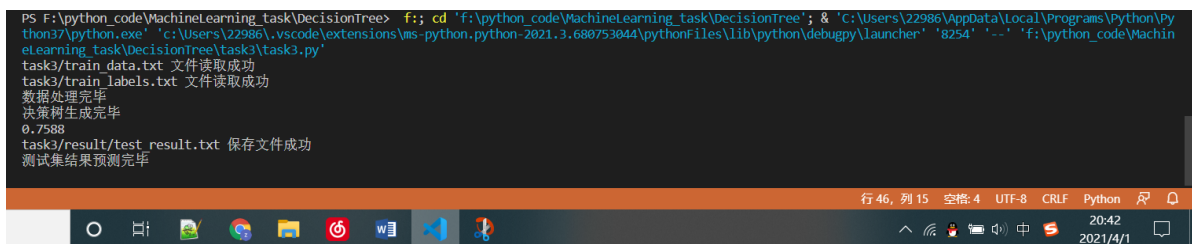
```
def txtsave(filename, data): # filename为写入txt文件的路径, data为要写入数据列表.
    file = open(filename, 'w')
    for i in range(len(data)):
        s = str(data[i]).replace('[', '').replace(']', '').replace(',', '\t') # 去除[]
        s = s.replace("'", '').replace('.', '').replace(' ', '\t') + '\n' # 去除单引号, 逗号, 每行末尾追加换行符
        file.write(s)
    file.close()
    print(filename, "保存文件成功")
```

- 代码细节:

(1) 使用python文件操作, 注意.replace()将特定字符替换, 同时行末写入换行符

二、运行结果

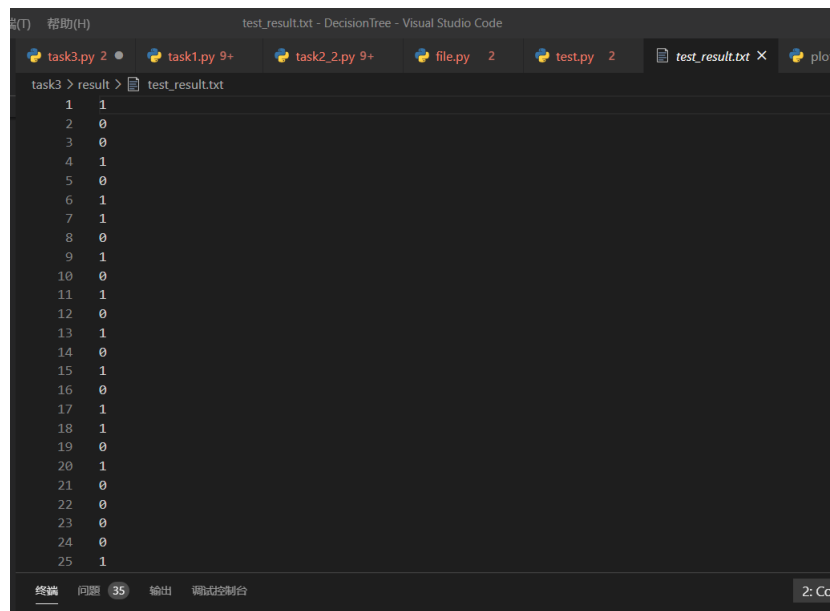
结果截图1



```
PS F:\python_code\MachineLearning_task\DecisionTree> f:: cd 'f:\python_code\MachineLearning_task\DecisionTree'; & 'C:\Users\22986\AppData\Local\Programs\Python\Python37\python.exe' 'c:\Users\22986\.vscode\extensions\ms-python.python-2021.3.680753044\pythonFiles\lib\python\debugpy\launcher' '8254' '--' 'f:\python_code\MachineLearning_task\DecisionTree\task3\task3.py'
task3/train data.txt 文件读取成功
task3/train labels.txt 文件读取成功
数据处理完毕
决策树生成完毕
0.7588
task3/result/test_result.txt 保存文件成功
测试集结果预测完毕
```

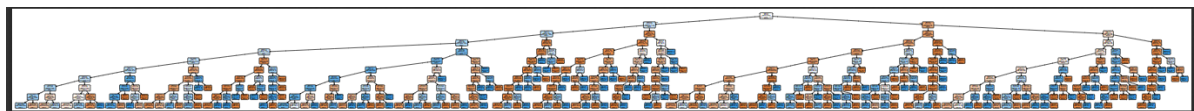
- 图片分析: 上图运行结果, 控制台打印信息显示程勋一切运行正常, 预测结果标签保存为txt文件成功

结果截图二



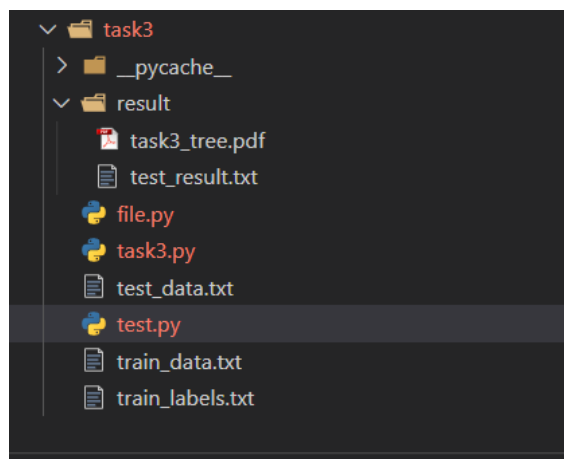
- 图片分析：上图为预测结果的标签存储文件，与训练集的标签一致

结果截图三



- 图片分析：上图为任务二生成的决策树，由于特征较多，决策树比较复杂，大概图片如上，程序运行无误

结果截图四



- 图片分析：上图为任务三的文件目录

三、总结体会

程序设计心得:

1. 本次算法过程中，遇到程序错误在正常不过，知错能改，善莫大焉，一个完整的工程背后一定是一次的debug，自身犯错最多的地方在于差错处理，可以多考虑极端情况，多进行程序的调试，对每一种特殊情况进行处理，避免程序运行的自行结束。通过这次软件课设，自身养成了不惧困难，不退缩，提高思维能力
2. 调用sklearn，重点在于对于该库的了解，可以多看看官方文档，以及找一些实战案例学习使用

