

电子科技大学信息与软件工程学院

# 实 验 报 告

学 号 2018091618008

姓 名 袁昊男

(实验) 课程名称 机器学习技术与应用

理论教师 黄 俊

实验教师 杨 珊

# 电子科技大学

## 实验报告

学生姓名：袁昊男      学号：2018091618008      指导教师：杨珊

实验地点：在线实验      实验时间：2020.05.06

一、实验室名称：信息与软件工程学院实验中心

二、实验名称：预测隐形眼镜类型的决策树模型

三、实验学时：4 学时

四、实验原理：

决策树算法是一种逼近离散函数值的方法。它是一种典型的分类方法，首先对数据进行处理，利用归纳算法生成可读的规则和决策树，然后使用决策对新数据进行分析。本质上决策树是通过一系列规则对数据进行分类的过程。决策树方法的基本思想是：利用训练集数据自动地构造决策树，然后根据这个决策树对任意实例进行判定。其中决策树（Decision Tree）是一种简单但是广泛使用的分类器。通过训练数据构建决策树，可以高效的对未知的数据进行分类。决策数有两大优点：（1）决策树模型可读性好，具有描述性，有助于人工分析；（2）效率高，决策树只需要一次构建，反复使用，每一次预测的最大计算次数不超过决策树的深度。

决策树算法构造决策树来发现数据中蕴涵的分类规则。如何构造精度高、规模小的决策树是决策树算法的核心内容。决策树构造可以分两步进行。第一步，决策树的生成：由训练样本集生成决策树的过程。一般情况下，训练样本数据集是根据实际需要有历史的、有一定综合程度的，用于数据分析处理的数据集。第二步，决策树的剪枝：决策树的剪枝是对上一阶段生成的决策树进行检验、校正和修下的过程，主要是用新的样本数据集（称为测试数据集）中的数据校验决策树生成过程中产生的初步规则，将那些影响预测准确性的分枝剪除。

决策树方法最早产生于上世纪 60 年代，到 70 年代末。由 J Ross Quinlan 提出了 ID3 算法，此算法的目的在于减少树的深度。但是忽略了叶子数目的研究。C4.5 算法在 ID3 算法的基础上进行了改进，对于预测变量的缺值处理、剪枝技术、派生规则等方面作了较大改进，既适合于分类问题，又适合于回归问题。

ID3 算法最早是由罗斯昆（J. Ross Quinlan）于 1975 年在悉尼大学提出的一种分类预测算法，算法的核心是“信息熵”。ID3 算法通过计算每个属性的信息增益，认为信息增益高的是好属性，每次划分选取信息增益最高的属性为划分标准，重复这个过程，直至生成一个能完美分类训练样例的决策树。

在 ID3 算法中，决策节点属性的选择运用了信息论中的熵概念作为启发式函数。

在这种属性选择方法中，选择具有最大信息增益（information gain）的属性作为当前划分节点。通过这种方式选择的节点属性可以保证决策树具有最小的分支数量，使得到的决策树冗余最小。

## 五、实验目的：

- 1、了解决策树算法原理。
- 2、掌握 ID3 决策树生成算法、决策树剪枝。
- 3、编程实现 ID3 算法。

## 六、实验内容：

- 1、收集数据，提供文本文件。
- 2、准备数据，解析 Tab 键分隔的数据行。
- 3、分析数据，快速检查数据，确保正确地解析数据内容。
- 4、训练算法，实现 ID3 决策树算法。
- 5、测试算法，编写测试函数验证决策树可以正确分类给定的数据实例。
- 6、存储树的数据结构。

## 七、实验器材（设备、元器件）：

- 1、PC 电脑。
- 2、Python。

## 八、实验步骤：

### 1、计算香农熵

```
1. # tree.py
2.
3. def cal_shannon_ent(dataset):
4.     # 计算 data_set 的大小
5.     data_len = len(dataset)
6.     # 为所有可能的分类创建字典
7.     label_counts = {}
8.     for feat_vec in dataset:
```

```

9.         current_label = feat_vec[-1]
10.        if current_label not in label_counts.keys():
11.            label_counts[current_label] = 0
12.            label_counts[current_label] += 1
13.        # 计算香农熵, 并返回
14.        shannon_ent = 0.0
15.        for key in label_counts:
16.            # 计算 label_counts[key] 的概率
17.            prob = float(label_counts[key]) / data_len
18.            # 香农熵(信息熵, 信息期望值)
19.            shannon_ent -= prob * log(prob, 2)
20.        return shannon_ent

```

**说明：**函数作用是计算给定数据集的香农熵；dataset 为给定的数据集；函数返回值为香农熵。首先计算数据集的大小，并为所有可能的分类初始化一个字典。遍历 data\_set 数据集，每行的最后一个元素为 label 标签，如果 label 不在字典中，则创建 key，并赋值为 0；否则加 1。香农信息熵的计算：设  $X$  是一个取值离散的随机变量，其概率分布为  $P(X = x_i) = p_i$ ， $i = 1, 2, \dots, n$ ，则随机变量  $X$  的信息熵定义为  $H(X) = -\sum_{i=1}^n p_i \log_2 p_i$ 。函数返回数据集的香农熵（信息熵）。

## 2、划分数据集

```

1. # tree.py
2.
3. def split_dataset(dataset, axis, value):
4.     ret_data_set = []
5.     # 遍历数据集
6.     for feat_vec in dataset:
7.         # 如果在数据集里找到给定特征的值，则除了该特征的值没有，剩下的都有
8.         if feat_vec[axis] == value:
9.             # 将 0~axis-1 存入列表中
10.            reduce_feat_vec = feat_vec[:axis]
11.            # 将 axis+1~最后一个存入列表中
12.            reduce_feat_vec.extend(feat_vec[axis+1:])
13.            ret_data_set.append(reduce_feat_vec)
14.    return ret_data_set

```

**说明：**函数作用是按照给定特征 axis 划分数数据集；dataset 为给定的数据集；axis 为划分数数据集的特征；value 为需要划分的特征值；函数返回值为给定特征 axis 的值。因为在函数内部对列表对象的修改，将会影响该列表对象的整个生存周期，所以新建一个列表对象；然后遍历数据集，注意区别 append 和 extend 的区别：append 是加入列表对象，extend 是加入列表元素。

## 3、决定最好的数据集划分

```

1. # tree.py
2.
3. def choose_best_feature_split(dataset):
4.     # 数据的最后一列是当前示例的类别标签，不包括在特征中
5.     num_features = len(dataset[0]) - 1

```

```

6.     base_ent = cal_shannon_ent(dataset)
7.     # 初始化信息增益和特征
8.     best_info_gain = 0.0
9.     best_feature = -1
10.    # 遍历数据集中的所有特征
11.    for i in range(num_features):
12.        # 将数据集中所有第 i 个特征值写入新 list 中
13.        feat_list = [example[i] for example in dataset]
14.        # 剔除掉重复的特征值
15.        unique_vals = set(feat_list)
16.        new_ent = 0.0
17.        # 遍历当前特征中的所有唯一属性值
18.        for value in unique_vals:
19.            # 对每一个特征划分一次数据集
20.            sub_dataset = split_dataset(dataset, i, value)
21.
22.            prob = len(sub_dataset) / float(len(dataset))
23.            new_ent += prob * cal_shannon_ent(sub_dataset)
24.
25.        info_gain = base_ent - new_ent
26.        # 比较所有特征中的信息增益，并返回最好特征划分的索引值
27.        if info_gain > best_info_gain:
28.            best_info_gain = info_gain
29.            best_feature = i
30.    return best_feature

```

**说明：**函数作用是选择最好的数据集划分方式；dataset 为给定的数据集；函数返回值为最好的特征。遍历数据集中的所有特征，计算整个数据集的原始香农熵，用于与划分完之后的数据集计算的熵值进行比较。对每一个特征划分一次数据集后，计算特征值 value 对应子集占数据集的比例  $\text{prob} = |A_j|/|D|$ ，然后对所有唯一特征值得到的熵求和。信息增益是熵的减少或者是数据无序度的减少，信息增益  $H(D, A)$  等于原始数据集的信息熵  $H(D)$  减去特征  $A$  对数据集进行划分后信息熵  $H(D/A)$ ，其中  $H(D/A) = \sum(|A_j|/|D| \times H(A_j))$ ， $j$  属于  $A$  的  $k$  种取值， $|A_j|/|D|$  表示特征  $A$  第  $j$  种取值的样本数占有所有取值样本总数的比例， $|D|$  表示数据集的样本总数。

#### 4、多数表决器

```

1.  # tree.py
2.
3.  def majority_cnt(classlist):
4.      class_count = {}
5.      for vote in classlist:
6.          if vote not in class_count.keys():
7.              class_count[vote] = 0
8.              class_count[vote] += 1
9.      sorted_class_count = sorted(class_count.items(),
10.                                  key=operator.itemgetter(1), reverse=True)
11.      return sorted_class_count[0][0]

```

**说明：**函数的作用是多数表决器；classlist 为输入类标签列表；函数返回值为出现最多次数的标签名称（key）。首先遍历所有的标签列表，如果不存在相应的标签，则扩展字典，并赋值为 0，否则加 1；然后将字典中的键值（value）进行从大到小的排序。

## 5、创建树

```
1. # tree.py
2.
3. def create_tree(dataset, labels):
4.     # 标签列表
5.     classlist = [example[-1] for example in dataset]
6.     if classlist.count(classlist[0]) == len(classlist):
7.         return classlist[0]
8.     if len(dataset[0]) == 1:
9.         return majority_cnt(classlist)
10.
11.     best_feat = choose_best_feature_split(dataset)
12.     best_feat_label = labels[best_feat]
13.     # 字典变量 my_tree 存储了树的所有信息
14.     my_tree = {best_feat_label: {}}
15.     del (labels[best_feat])
16.     feat_values = [example[best_feat] for example in dataset]
17.     unique_vals = set(feat_values)
18.     # 遍历当前特征包含的所有属性值
19.     for value in unique_vals:
20.         # 复制类别标签
21.         sub_labels = labels[:]
22.         # 递归调用函数 create_tree()
23.         my_tree[best_feat_label][value] = create_tree(split_dataset(dataset, best_feat, value), sub_labels)
24.     return my_tree
```

**说明：**函数的作用是使用 ID3 算法创建树；dataset 为给定的数据集；labels 为给定的标签列表。如果类别完全相同则停止继续划分，遍历完所有特征时返回出现次数最多的标签名称；然后划分数据集，获得最好的特征与特征标签并删除最好的特征标签。

## 6、分类函数

```
1. # tree.py
2.
3. def classify(input_tree, feat_labels, test_vec):
4.     first_str = list(input_tree.keys())[0]
5.     second_dict = input_tree[first_str]
6.     # 将标签字符串转换为索引
7.     feat_index = feat_labels.index(first_str)
8.     class_label = {}
9.     # 递归遍历整棵树
10.    for key in second_dict.keys():
11.        if test_vec[feat_index] == key:
12.            if type(second_dict[key]).__name__ == 'dict':
13.                class_label = classify(second_dict[key], feat_labels, test_vec)
```

```

14.         else:
15.             class_label = second_dict[key]
16.         return class_label

```

**说明：**函数的作用是使用创建好的决策树对数据进行分类；`input_tree` 为训练好的决策树；`feat_labels` 为特征标签集；`test_vec` 为测试数据向量。使用递归的方法遍历整棵树，依次深入到树的最底层即为最后的分类结果。

## 7、树的存储与读取

```

1. # tree.py
2.
3. def store_tree(input_tree, filename):
4.     import pickle
5.     # 以二进制的方式写入文件中
6.     fw = open(filename, 'wb')
7.     pickle.dump(input_tree, fw)
8.     fw.close()
9.
10.
11. def grab_tree(filename):
12.     import pickle
13.     fr = open(filename, 'rb')
14.     return pickle.load(fr)

```

**说明：**`store_tree` 函数的作用是使用 `pickle` 以二进制的方式存储树的数据结构到 `txt` 文件中；`input_tree` 为已经训练好的决策树；`filename` 为输出文件路径及文件名。`grab_tree` 函数作用是从文件中读取训练完成的树的数据结构；`filename` 为读入文件路径及文件名。

## 8、main 函数

```

1. # tree.py
2.
3. if __name__ == '__main__':
4.     fr = open('lenses.txt')
5.     lenses = [inst.strip().split('\t') for inst in fr.readlines()]
6.     lenses_labels = ['age', 'prescript', 'astigmatic', 'tear-
Rate']
7.     lenses_tree = create_tree(lenses, lenses_labels)
8.     labels = ['age', 'prescript', 'astigmatic', 'tearRate']
9.     predict = classify(lenses_tree, labels, ['pre', 'my-
ope', 'no', 'normal'])
10.    print(predict)
11.    store_tree(lenses_tree, "classifierStorage.txt")
12.    treePlotter.create_plot(lenses_tree)

```

**说明：**首先读入训练数据 `lenses.txt`，并解析由 Tab 符分隔的数据行。为数据建立标签，数据共 4 个属性：`age`（年龄）、`prescript`（视力缺陷）、`astigmatic`（是否散光）、`tearRate`（泪腺分泌情况）；调用 `create_tree` 函数根据以上数据训练决策树模型，并使用测试数据进行测试

(['pre','myope','no','normal'], 预期分类结果应为 soft)。最后保存训练好的决策树模型，并使用 matplotlib 工具将决策树可视化地绘制出来。

## 9、使用 Matplotlib 库可视化决策树

```
1. # draw.py
2.
3. import matplotlib.pyplot as plt
4.
5. decisionNode = dict(boxstyle="square", fc="0.8")
6. leafNode = dict(boxstyle="square", fc="0.8")
7. arrow_args = dict(arrowstyle="<-")
8.
9. def plot_node(node_txt, center_pt, parent_pt, node_type):
10.     create_plot.ax1.annotate(node_txt, xy=parent_pt, xy-
11.     coords="axes fraction",
12.     xytext=center_pt, textcoords="axes fraction", va="center",
13.     ha="center", bbox=node_type, arrowprops=arrow_args)
14.
15. def get_num_leafs(my_tree):
16.     num_leafs = 0
17.     # 第一个判断结点
18.     first_str = list(my_tree.keys())[0]
19.     second_dict = my_tree[first_str]
20.     # 遍历判断结点的左右分支的 keys
21.     for key in second_dict.keys():
22.         if type(second_dict[key]).__name__ == 'dict':
23.             num_leafs += get_num_leafs(second_dict[key])
24.         else:
25.             num_leafs += 1
26.     return num_leafs
27.
28.
29. def get_tree_depth(my_tree):
30.     # 初始化深度
31.     max_depth = 0
32.     # 第一个判断结点
33.     first_str = list(my_tree.keys())[0]
34.     second_dict = my_tree[first_str]
35.     for key in second_dict.keys():
36.         if type(second_dict[key]).__name__ == 'dict':
37.             this_depth = 1 + get_tree_depth(second_dict[key])
38.         else:
39.             this_depth = 1
40.         if this_depth > max_depth:
41.             max_depth = this_depth
42.     return max_depth
43.
44.
45. def retrive_tree(i):
46.     list_of_trees = [{'no surfacing': {0: 'no', 1: {'flippers':
47.     {0: 'no', 1: 'yes'}}}},
48.     {'no surfacing': {0: 'no', 1: {'flippers':
```



```

49.         {0: {'head': {0: 'no', 1: 'yes'}}, 1: 'no'}}}
50.     ]
51.     return list_of_trees[i]
52.
53.
54. def plot_mid_text(cntr_pt, parant_pt, txt_string):
55.     x_mid = (parant_pt[0] - cntr_pt[0])/2.0 + cntr_pt[0]
56.     y_mid = (parant_pt[1] - cntr_pt[1])/2.0 + cntr_pt[1]
57.     # 在中间位置处添加文本标签信息
58.     create_plot.ax1.text(x_mid, y_mid, txt_string)
59.
60.
61. def plot_tree(my_tree, parant_pt, node_txt):
62.     # 计算叶节点
63.     num_leafs = get_num_leafs(my_tree)
64.     depth = get_tree_depth(my_tree)
65.     first_str = list(my_tree.keys())[0]
66.     cntr_pt = (plot_tree.xoff + (1.0 + float(num_leafs))/2.0/plot
        _tree.totalW, plot_tree.yoff)
67.     # 计算父节点和子节点的中间位置, 并在此处添加文本标签信息
68.     plot_mid_text(cntr_pt, parant_pt, node_txt)
69.     plot_node(first_str, cntr_pt, parant_pt, decisionNode)
70.     # 第一个判断节点的 value
71.     second_dict = my_tree[first_str]
72.     plot_tree.yoff = plot_tree.yoff - 1.0/plot_tree.totalD
73.     # 遍历 second_dict.keys(), 绘制子节点(可以是叶子结点, 或判断节
    点)
74.     for key in second_dict.keys():
75.         if type(second_dict[key]).__name__ == 'dict':
76.             plot_tree(second_dict[key], cntr_pt, str(key))
77.         else:
78.             plot_tree.xoff = plot_tree.xoff + 1.0/plot_tree.to-
            talW
79.             plot_node(sec-
            ond_dict[key], (plot_tree.xoff, plot_tree.yoff),
80.                 cntr_pt, leafNode)
81.             plot_mid_text((plot_tree.xoff, plot_tree.yoff), cntr_
            pt, str(key))
82.
83.     plot_tree.yoff = plot_tree.yoff + 1.0/plot_tree.totalD
84.
85.
86. def create_plot(in_tree):
87.     # 创建窗口
88.     fig = plt.figure(1, facecolor='white')
89.     fig.clf()
90.     axprops = dict(xticks=[], yticks=[])
91.     create_plot.ax1 = plt.subplot(111, frameon=False, **ax-
        props)
92.     # 全局变量 plot_tree.totalW 存储决策树的宽度
93.     # 全局变量 plot_tree.totalD 存储决策树的高度
94.     # 全局变量 plot_tree.xoff 存储决策树节点的 x 坐标
95.     # 全局变量 plot_tree.yoff 存储决策树节点的 y 坐标
96.     plot_tree.totalW = float(get_num_leafs(in_tree))
97.     plot_tree.totalD = float(get_tree_depth(in_tree))
98.     plot_tree.xoff = -0.5/plot_tree.totalW
99.     plot_tree.yoff = 1.0

```

```

100.     plot_tree(in_tree, (0.5, 1.0), '') # 决策树的起点:
        (0.5,1.0)
101.     # 调用"绘制带箭头的注解"函数
102.     # plot_node('decisionNodes', (0.5, 0.1), (0.1, 0.5), deci-
        sionNode)
103.     # plot_node('leafNodes', (0.8, 0.1), (0.3, 0.8), leafNode)
104.     plt.show()

```

## 10、使用 sklearn 库创建决策树

```

1. # tree.py
2.
3. from sklearn import tree
4.
5. if __name__ == '__main__':
6.     fr = open('lenses.txt')
7.     lenses = [inst.strip().split('\t') for inst in fr.readlines()]
8.     print(lenses)
9.     lensesLabels = ['age', 'prescript', 'astigmatic', 'tearRate']
10.    clf = tree.DecisionTreeClassifier()
11.    lenses = clf.fit(lenses, lensesLabels)

```

说明：直接读入 txt 数据会报错，这是因为因为 fit()函数不能接收 string 类型的数据。通过打印的信息可以看到 txt 中数据都是 string 类型的，因此在使用 fit()函数之前，我们需要对数据集进行编码，将其序列化。

## 11、使用 Graphviz 库可视化决策树

```

1. # draw.py
2.
3. from sklearn.preprocessing import LabelEncoder, OneHotEncoder
4. from sklearn.externals.six import StringIO
5. from sklearn import tree
6. import pandas as pd
7. import numpy as np
8. import pydotplus
9.
10. if __name__ == '__main__':
11.     with open('lenses.txt', 'r') as fr: # 加载文件
12.         lenses = [inst.strip().split('\t') for inst in fr.read-
            lines()]
13.         lenses_target = [] # 提取每组数据的类别，保存在列表里
14.         for each in lenses:
15.             lenses_target.append(each[-1])
16.         print(lenses_target)
17.
18.         lensesLabels = ['age', 'prescript', 'astigmatic', 'tear-
            Rate']
19.         lenses_list = [] # 保存 lenses 数据的临时列表
20.         lenses_dict = {} # 保存 lenses 数据的字典，用于生成 pandas
21.         for each_label in lensesLabels: # 提取信息，生成字典
22.             for each in lenses:
23.                 lenses_list.append(each[lensesLabels.index(each_la-
                    bel)])
24.                 lenses_dict[each_label] = lenses_list
25.                 lenses_list = []
26.         # print(lenses_dict) # 打印字典信息

```

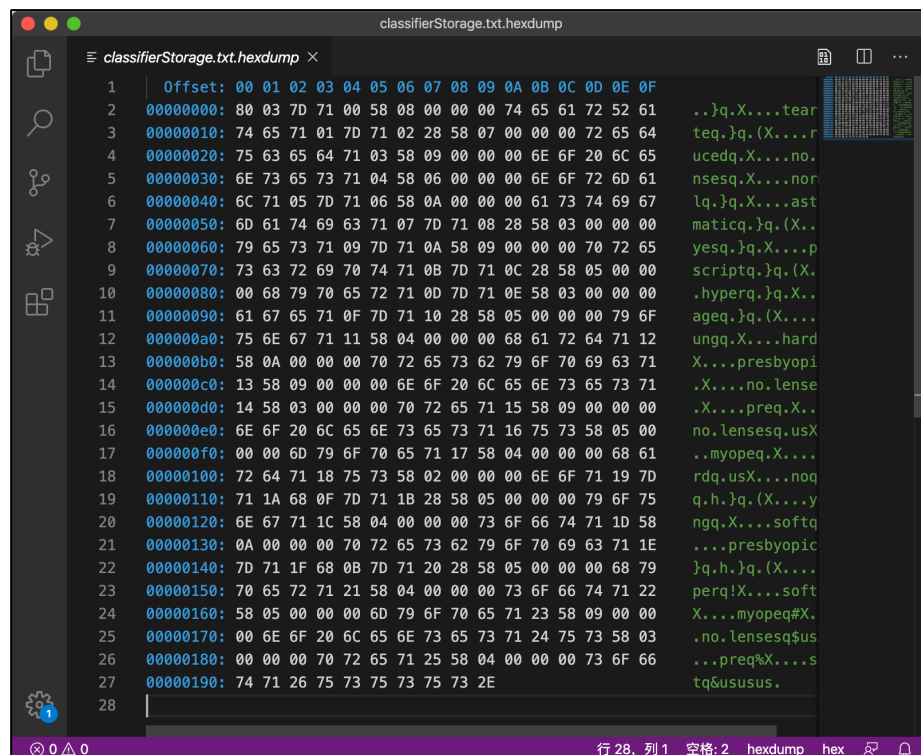
```

27.     lenses_pd = pd.DataFrame(lenses_dict)    # 生成 pandas.DataFrame
28.     # print(lenses_pd)    # 打印 pandas.DataFrame
29.     le = LabelEncoder()    # 创建 LabelEncoder()对象，用于序列化
30.     for col in lenses_pd.columns:    # 序列化
31.         lenses_pd[col] = le.fit_transform(lenses_pd[col])
32.     # print(lenses_pd) # 打印编码信息
33.
34.     clf = tree.DecisionTreeClassifier(max_depth = 4)    # 创建 DecisionTreeClassifier()类
35.     clf = clf.fit(lenses_pd.values.tolist(), lenses_target)    # 使用数据，构建决策树
36.     dot_data = StringIO()
37.     tree.export_graphviz(clf, out_file = dot_data,    # 绘制决策树
38.                           feature_names = lenses_pd.keys(),
39.                           class_names = clf.classes_,
40.                           filled=True, rounded=True,
41.                           special_characters=True)
42.     graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
43.     graph.write_pdf("tree.pdf")

```

## 九、实验数据及结果分析

### 1、决策树存储结果

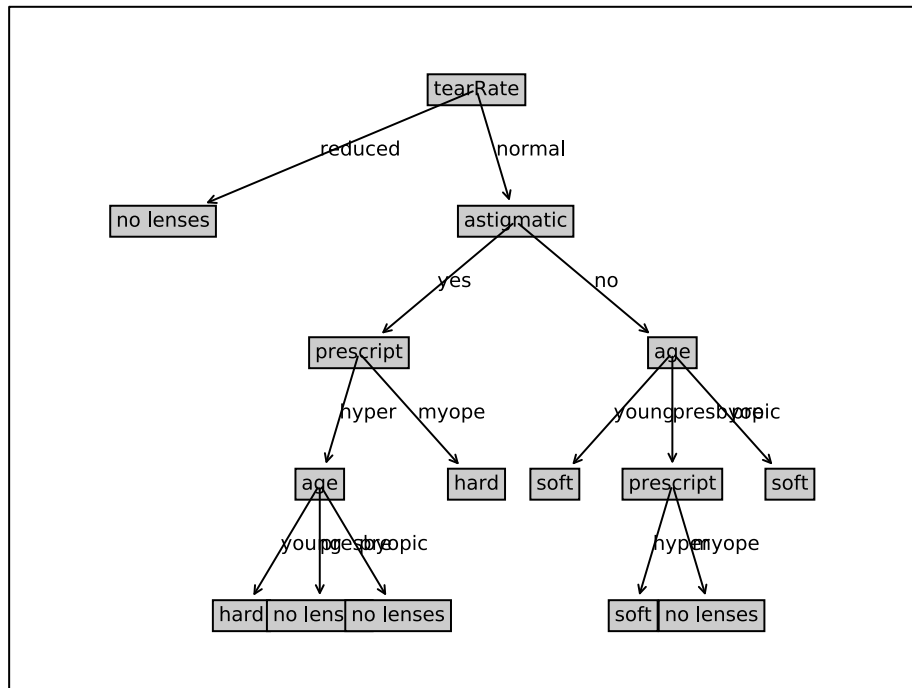


```

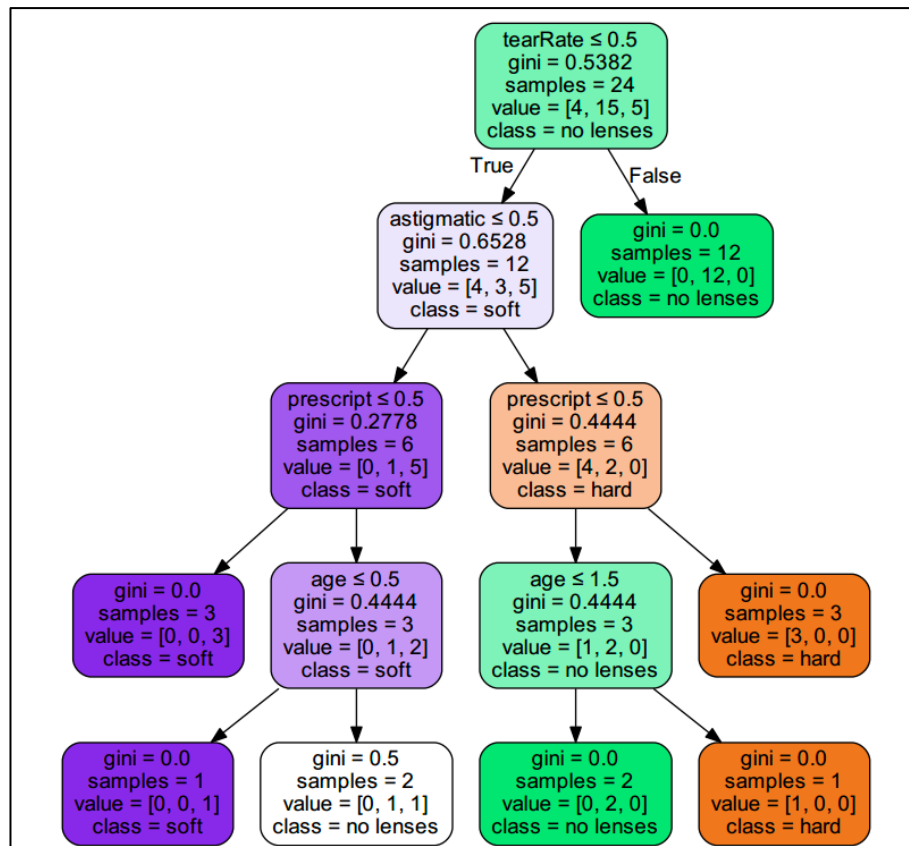
1  Offset: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
2  00000000: 80 03 7D 71 00 58 08 00 00 00 74 65 61 72 52 61  ..}q.X....tear
3  00000010: 74 65 71 01 7D 71 02 28 58 07 00 00 00 72 65 64  teq.}q.(X....r
4  00000020: 75 63 65 64 71 03 58 09 00 00 00 6E 6F 20 6C 65  ucedq.X....no.
5  00000030: 6E 73 65 73 71 04 58 06 00 00 00 6E 6F 72 6D 61  nsesq.X....nor
6  00000040: 6C 71 05 7D 71 06 58 0A 00 00 00 61 73 74 69 67  lq.}q.X....ast
7  00000050: 6D 61 74 69 63 71 07 7D 71 08 28 58 03 00 00 00  maticq.}q.(X..
8  00000060: 79 65 73 71 09 7D 71 0A 58 09 00 00 00 70 72 65  yesq.}q.X....p
9  00000070: 73 63 72 69 70 74 71 0B 7D 71 0C 28 58 05 00 00  scriptq.}q.(X.
10 00000080: 00 68 79 70 65 72 71 0D 7D 71 0E 58 03 00 00 00  .hyperq.}q.X..
11 00000090: 61 67 65 71 0F 7D 71 10 28 58 05 00 00 00 79 6F  ageq.}q.(X....
12 000000a0: 75 6E 67 71 11 58 04 00 00 00 68 61 72 64 71 12  ungq.X....hard
13 000000b0: 58 0A 00 00 00 70 72 65 73 62 79 6F 70 69 63 71  X....presbyopi
14 000000c0: 13 58 09 00 00 00 6E 6F 20 6C 65 6E 73 65 73 71  .X....no. lense
15 000000d0: 14 58 03 00 00 00 70 72 65 71 15 58 09 00 00 00  .X....preg.X..
16 000000e0: 6E 6F 20 6C 65 6E 73 65 73 71 16 75 73 58 05 00  no. lensesq.usX
17 000000f0: 00 00 6D 79 6F 70 65 71 17 58 04 00 00 00 68 61  ..myopeq.X....
18 00000100: 72 64 71 18 75 73 58 02 00 00 00 6E 6F 71 19 7D  rdq.usX....noq
19 00000110: 71 1A 68 0F 7D 71 18 28 58 05 00 00 00 79 6F 75  q.h.}q.(X....y
20 00000120: 6E 67 71 1C 58 04 00 00 00 73 6F 66 74 71 1D 58  ngq.X....softq
21 00000130: 0A 00 00 00 70 72 65 73 62 79 6F 70 69 63 71 1E  ....presbyopic
22 00000140: 7D 71 1F 68 0B 7D 71 20 28 58 05 00 00 00 68 79  }q.h.}q.(X....
23 00000150: 70 65 72 71 21 58 04 00 00 00 73 6F 66 74 71 22  perq!X....soft
24 00000160: 58 05 00 00 00 6D 79 6F 70 65 71 23 58 09 00 00  X....myopeq#X.
25 00000170: 00 6E 6F 20 6C 65 6E 73 65 73 71 24 75 73 58 03  .no. lensesq$us
26 00000180: 00 00 00 70 72 65 71 25 58 04 00 00 00 73 6F 66  ...preg%X....s
27 00000190: 74 71 26 75 73 75 73 75 73 2E  tq&ususus.
28

```

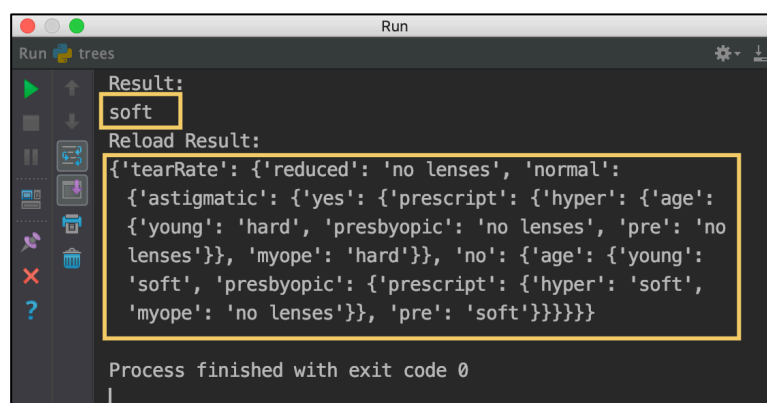
## 2、决策树可视化结果（Matplotlib）



## 3、决策树可视化结果（Graphviz）



#### 4、决策树测试及重新加载结果



说明：测试['pre','myope','no','normal']，预期分类结果应为 soft，分类结果正确。

## 十、实验结论

本实验通过应用决策树模型，结合 matplotlib 库完成了预测隐形眼镜的决策树模型及其可视化。得到的决策树模型可以将其数据结构固化为 txt 文件供以后使用，该模型可以准确预测测试数据样本，通过可视化图形可以更直观地了解到决策树的数据结构。本实验还可以直接使用 sklearn 库中的 DecisionTreeClassifier 构建决策树，配合 Graphviz 对其进行可视化。

## 十一、总结及心得体会

思考题：分析 ID3 算法的优略和局限性，如何修改能够使模型更加健壮。

解答：

ID3 算法是以信息论为基础，以信息熵和信息增益度为衡量标准，从而实现数据的归纳分类。ID3 算法计算每个属性的信息增益，并选取具有最高增益的属性作为给定的测试属性。C4.5 算法核心思想是 ID3 算法，是 ID3 算法的改进，改进方面有四方面，第一就是用信息增益率来选择属性，克服了用信息增益选择属性时偏向选择取值多的属性的不足。第二就是在树构造过程中进行剪枝。第三就是能处理非离散的数据。第四就是能处理不完整的数据。

C4.5 算法的优点是产生的分类规则易于理解，准确率较高。而缺点就是在构造树的过程中，需要对数据集进行多次的顺序扫描和排序，因而导致算法的低效。同时 C4.5 只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时程序无法运行。

## 十二、对本实验过程及方法、手段的改进建议

实验部分课堂讲授部分较简单, 学生需要花费更多的课下时间来学习相关的算法知识并实现。还可以直接使用 `sklearn.tree` 模块, 该模块中提供了决策树模型, 用于解决分类问题和回归问题。使用 `DecisionTreeClassifier` 和 `export_graphviz`, 前者用于决策树构建, 后者用于决策树可视化。调参时注意将样本数据读入时, 需先将其序列化。

**报告评分:**

**指导教师签字:**