实验二 RL 报告

江辉--171250506

2020-4-14

南京大学

软件学院

目 录

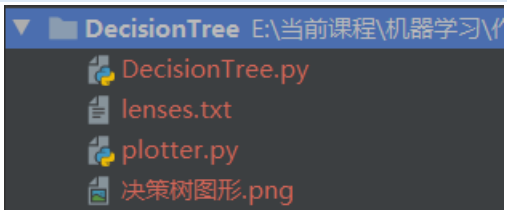
一、 实验内容描述

本程序用 python 编写，手写 ID3 算法的实现，完成决策树实验。给定隐形眼镜小量数据集，构造决策树预测患者佩戴隐形眼镜类型，并通过 Matplotlib 绘制树图形

1、项目目录描述

本项目推荐使用 pycharm 打开

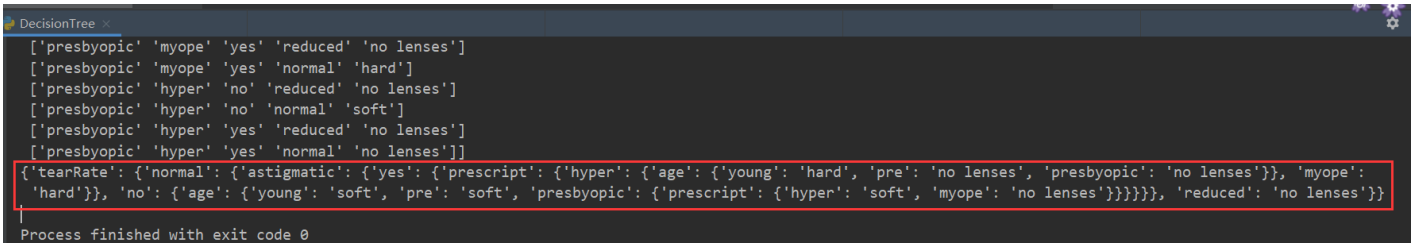
DecisionTree.py 里写了 main 函数和决策树实现的代码，**plotter.py** 是画树图形的代码



二、 实验结果和树图形

1、生成的决策树

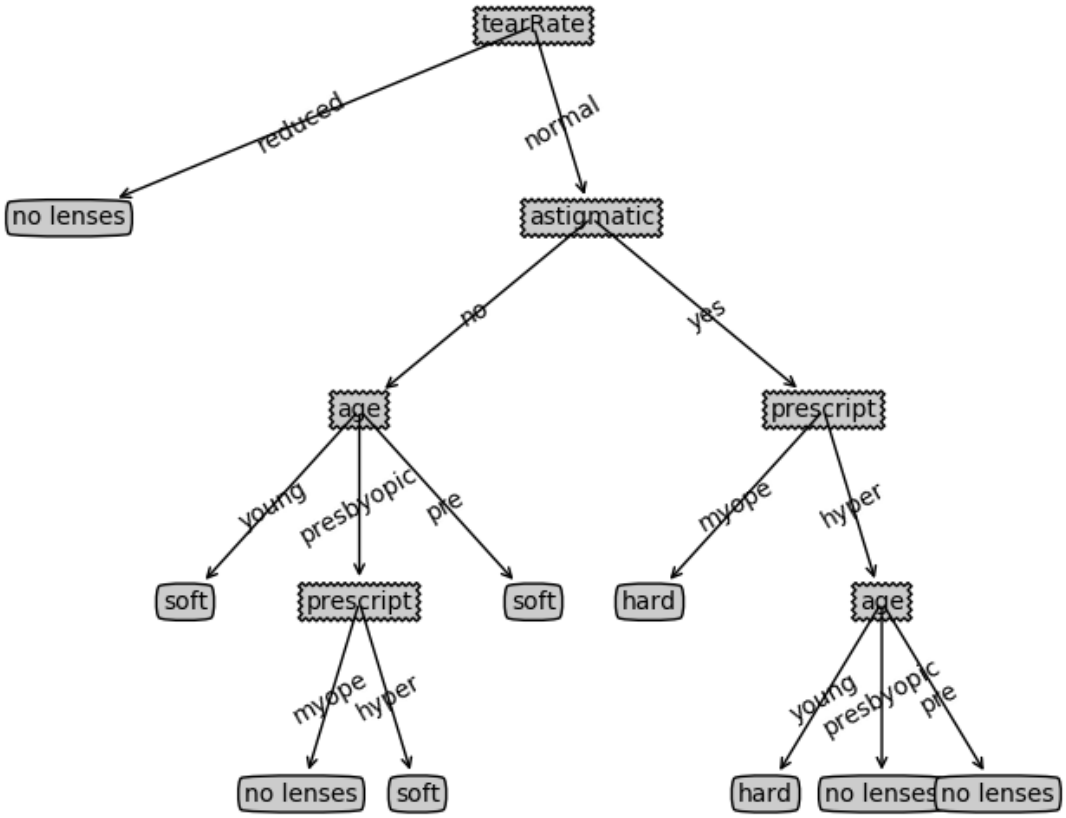
以下是控制台输出的结果，我也用文本编辑器转换为了用 json 格式查看的结果




```
{
    "tearRate": {
        "reduced": "no lenses',
        "normal": {
            "astigmatic": {
                "no": {
                    "age": {
                        "presbyopic": {
                            "prescript": {
                                "hyper": "soft",
                                "myope": "no lenses"
                            }
                        },
                        "young": "soft",
                        "pre": "soft"
                    }
                },
                "yes": {
                    "prescript": {
                        "hyper": {
                            "age": {
                                "presbyopic": "no lenses",
                                "young": "hard",
                                "pre": "no lenses"
                            }
                        },
                        "myope": "hard"
                    }
                }
            }
        }
    }
}
```

2、决策树图形

'以下是用 plotter.py 画出的树图形



三、决策树算法原理阐述

决策树的目的是构建一棵树，从根节点出发，每一个非叶节点都是一个属性判断分支，直至叶节点确定分类，即在样本的多维属性中不断选择具有区分度的属性进行判断，逐步将样本精确至分类结果的算法。其中决策树的生成包含以下几个主要步骤

1、特征属性选择：从目前有的特征中选一个作为当前节点的分支选项，每个选项会达到下一层的子树或叶节点

一般认为能越好地区分数据集的特征属性越应该优先进行判断，如 ID3 和 C4.5 以信息增益和信息增益率来选择能最大化地区分样本的特征，而 CART 中以类似信息熵的悉尼指数来简化计算，本质上都是为了选择最能提现类别不同的特征

2、递归生成完整的决策树：对每个非叶节点递归进行步骤 1 中的操作，直至没有可再选择分支的特征

3、可选的剪枝：决策树构建完后容易过拟合，所以需要限制分支的数量，有些算法会在生成决策子树的时候进行筛选，即预剪枝，有些会在在最后会对生成的决策树进行合并或裁剪，即后剪枝，目的都是增强泛化能力，本次实验中我采用 ID3 算法，没有做剪枝操作

四、决策树部分核心代码

1、ID3 算法类

下图是我实现的 ID3 算法类，__init__中可以看到就是存了数据集合属性集

calculate_tree 是递归生成决策树的函数

cal_entropy 是熵的计算函数

max_count_label 是在没有可划分属性时选择占比最大的类别的函数

filter_horizontal 是选择特征分支时将数据集根据属性分割成不同的子数据集

delete_col 是选完一个特征后将其从数据集中移除

col_best_header_col 是选择最佳特征

```
class ID3Tree:
    def __init__(self, dataset, headers):
        """
        :param dataset: 数据集
        :param headers: 属性列名
        """
        self.dataset = dataset
        self.headers = headers

    def cal_entropy(self, dataset):...

    def max_count_label(self):...

    def filter_horizontal(self, col, feature):...

    def delete_col(self, col):...

    def cal_best_header_col(self):...

    def calculate_tree(self):...
```

下面我对关键的函数进行简单描述

2、递归生成决策树流程代码

下面这个方法是递归主体，先判断是否是叶节点，是叶节点就返回标签，不是叶节点就选择最佳特征，然后对特征内每个属性构建子数据集并递归构造子树

```
def calculate_tree(self):
    """
    计算ID3决策树
    :return: map形式的决策树
    """
    env = self.cal_entropy(self.dataset)
    # 熵等于0意味着所有样本都是同一类型的样本
    if env == 0:
        return self.dataset[0][-1]
    # 没有属性列可分时返回最多的类型标签
    if len(self.headers) == 0:
        return self.max_count_label()
    # 获取最大熵增益列
    best_header_col = self.cal_best_header_col()
    best_header = self.headers[best_header_col]
    # 教程里的树结构是这样的嵌套字典
    tree = {best_header: {}}
    features = set(self.dataset[:, best_header_col])
    for feature in features:
        # 创建子数据集
        sub_dataset = self.filter_horizontal(best_header_col, feature)
        sub_dataset = np.delete(sub_dataset, best_header_col, axis=1)
        sub_headers = self.headers.copy()
        del sub_headers[best_header_col]
        subID3 = ID3Tree(sub_dataset, sub_headers)
        # 递归构建决策树
        tree[best_header][feature] = subID3.calculate_tree()
    return tree
```

3、最佳特征选择函数

先计算当前根节点的熵，再对每个特征计算条件熵，选出信息增益最大的特征返回特征列号

```python
def cal_best_header_col(self):
    """
    选择最佳属性列
    :return: 最佳属性列号
    """
    col_len = len(self.headers) - 1
    # 计算根节点的信息熵
    root_env = self.cal_entropy(self.dataset)
    max_info_gain = 0.0
    best_header = None
    # 对每一列计算信息熵
    for i in range(col_len):
        features = set(self.dataset[:, i])
        entropy = 0.0
        for feature in features:
            sub_data = self.filter_horizontal(i, feature)
            probability = len(sub_data) / float(len(self.dataset))
            entropy += probability * self.cal_entropy(sub_data)
        info_gain = root_env - entropy
        # 选择信息增益最大的列
        if info_gain > max_info_gain:
            max_info_gain = info_gain
            best_header = i
    return best_header
```

4、熵的计算

对每一类样本进行统计并根据熵的公式计算熵

```python
def cal_entropy(self, dataset):
    """
    计算指定数据集的熵
    :param dataset: 数据集，最后一列是类别标签
    :return: 熵
    """
    # 数据集总项数
    data_len = float(len(dataset))
    # 标签计数对象初始化
    label_counts = {}
    for data in dataset:
        # 最后一列是分类标签
        label = data[-1]
        # 为每一类数据计数
        if label not in label_counts.keys():
            label_counts[label] = 0
        label_counts[label] += 1
    entropy = 0.0
    for key in label_counts.keys():
        prop = label_counts[key] / data_len
        entropy -= prop * log(prop, 2)
    return entropy
```