实验 Decision Tree

姓名：李昀哲　学号：20123101

1 题目和数据：

题目：Decision Tree

学习并实现决策树算法----以 ID3 算法为例;
基于决策树算法预测隐形眼镜类型;

数据描述:

实验数据包括学习实现决策树算法的测试数据海洋动物分类和隐形眼镜数据集，数据集概况如图 1 所示，其中，海洋动物数据集的 feature 共两种，分别是“不浮出水面是否可以生存”以及“是否有脚蹼”，只有是或否两种可能，因此可以采用 0，1 编码；而隐形眼镜数据集的 4 类 feature：age、prescript、astigmatic、tearRate 中 age 分为三类，其余均为两类，因此编码方式要做出一定变化。

id	不浮出水面是否可以生存	是否有脚蹼	属于鱼类
1	是	是	是
2	是	是	是
3	是	否	否
4	否	是	否
5	否	是	否

	0	1	2	3	4
0	young	myope	no	reduced	no lenses
1	young	myope	no	normal	soft
2	young	myope	yes	reduced	no lenses
3	young	myope	yes	normal	hard
4	young	hyper	no	reduced	no lenses
5	young	hyper	no	normal	soft

图 1 Decision Tree 实验数据

2 算法：

决策树（Decision Tree）是在已知各种情况发生概率的基础上，通过构成决策树来求取净现值的期望值大于等于零的概率，评价项目风险，判断其可行性的决策分析方法，是直观运用概率分析的一种图解法。由于这种决策分支画成图形很像一棵树的枝干，故称决策树。在机器学习中，决策树是一个预测模型，他代表的是对象属性与对象值之间的一种映射关系。常用于分类任务中。

其核心评估思想是计算分类前后系统的混乱程度，常用信息熵来计算，计算公式如下，

$$\mathrm{H}(X)=\sum P(x_i)\log P(x_i)$$

由此衍生出信息增益等评估标准。常见的几类决策树是 ID3，C4.5，CART。优势在于：决策树易于理解和实现，人们在在学习过程中不需要使用者了解很多的背景知识；对于决策树，数据的准备往往是简单或者是不必要的，而且能够同时处理数据型和常规型属性，在相对短的时间内能够对大型数据源做出可行且效果良好的结果；易于通过静态测试来对模型进行评测，可以测定模型可信度；如果给定一个观察的模型，那么根据所产生的决策树很容易推出相应的逻辑表达式。其缺点也较为明显：对连续性的字段比较难预测；对有时间顺序的数据，需要很多预处理的工作；当类别太多时，错误可能就会增加的比较快。

3 代码及结果

数据集构建

下述为海洋动物分类数据集的构建，隐形眼镜数据集直接采用 pandas 库中 ***read_table*** 方法读入 txt 文本文件；

```
def createDataSet():
    """
    函数说明：创建数据集
    returns:
        dataSet - 数据集
        labels - 分类属性
    """
    dataSet = [[1, 1, 'yes'],
               [1, 1, 'yes'],
               [1, 0, 'no'],
               [0, 1, 'no'],
               [0, 1, 'no']]
    labels = ['no surfacing','flippers']
    return dataSet, labels
```

计算信息熵

```
import math
def calcShannonEnt(dataSet):
    """
    函数说明:计算给定数据集的香农熵
    parameters:
        dataSet - 数据集
    returns:
        shannonEnt - 香农熵
    """
    num_total_samples = len(dataSet)
    labelcnt = {}
    for  sample in  dataSet:
        current_label = sample[-1]
        if current_label not in labelcnt.keys():
            labelcnt[current_label] = 0
        labelcnt[current_label] += 1

    shannoEntropy = 0
    for key in labelcnt:
        prob = float(labelcnt[key])/num_total_samples
        shannoEntropy += -prob*math.log(prob, 2)
    return shannoEntropy
```

信息熵的计算主要依据香农公式；

划分数据集

每一步决策树节点的选择过后都会对数据集进行一次切分，对切分后的数据集进行后续迭代

```
def splitDataSet(dataSet, axis, value):
    """
    函数说明：按照给定特征划分数据集
    parameters:
        dataSet - 待划分的数据集
        axis - 划分数据集的特征（第axis个特征）
        value - 特征值
    returns:
        retDataSet - 划分后的数据集
    """
    retData = []
    for sample in dataSet:
        if sample[axis] == value:
            # 删去某一列
            reduced_sample = sample[:axis]
            reduced_sample.extend(sample[axis+1:])
            retData.append(reduced_sample)
    return retData
```

选择最佳特征

根据核心思想，比较分类前后信息熵的变化，选择信息熵分类后最小的作为决策树当前节点；

```
def chooseBestFeatureToSplit(dataSet):
    """
    函数说明：选择最优特征
    parameters:
        dataSet - 数据集
    returns:
        bestFeature - 信息增益最大的(最优)特征的索引值
    """
    # 首先要知道有几个待选特征
    num_features = len(dataSet[0]) - 1

    # 再先算历史数据的熵值
    base_entropy = calcShannonEnt(dataSet)
    bestInfoGain = 0
    bestFeature = -1

    # 再算以每一个特征作为决策点的熵值
    for feature_index in range(num_features):
        featureList = [sample[feature_index] for sample in dataSet]
        unique_value = set(featureList)
        newEntropy = 0
        # 选择一个特征后将每个岔的熵值加权求和
        for val in unique_value:
            subDataSet = splitDataSet(dataSet, feature_index, val)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob*calcShannonEnt(subDataSet)
        infoGain = base_entropy - newEntropy
        if (infoGain > bestInfoGain):
            bestInfoGain = infoGain
            bestFeature = feature_index


    return bestFeature
```

叶子节点分类结果

如叶子节点并未得到确定的分类值，将以类别众数作为分类结果

```
import operator
import numpy as np
def majorcnt(classList):
    """
    终止条件：当特征全部筛选完成后，返回当前list中出现最多的类别
    """
    classcount = {}
    for vote in classList:
        if vote not in classcount.keys():
            classcount[vote]=0
        classcount[vote] += 1
    major_count = sorted(classcount.items(), key=operator.itemgetter(1), reverse=True)
    return major_count[0][0]
```

构建树

以递归为主要思想，每一轮选择当前最佳特征，选完后看分为了几个分岔，切分数据集和标签集（去掉已经完成的分类），再对每个分岔进行递归。终止条件为某分支下所有实例都相同或所有特征都已分完

```
def createTree(dataSet, labels, featureLabels):
    """
    函数说明:创建决策树
    Parameters:
        dataSet - 训练数据集
        labels - 分类属性标签
    Returns:
        myTree - 决策树
    """

    # 判断该分支下所有实例是否都相同,全相同就return
    classList = [sample[-1] for sample in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]

    # 特征全部筛选完了
    if len(dataSet[0]) == 1:
        return majorcnt(classList)

    # 选择当前最优特征
    bestFeature_index = chooseBestFeatureToSplit(dataSet)
    bestFeatue = labels[bestFeature_index]
    featureLabels.append(bestFeatue)
    myTree = {bestFeatue:{}} # 构建一个字典树模型
    # 删去当前这一列特征
    del labels[bestFeature_index]

    # 选完特征看分几岔
    feature_values = [sample[bestFeature_index] for sample in dataSet]
    unique_values = set(feature_values) # 变为一个集合，得到唯一的多个值
    # 分岔
    for value in unique_values:
        sublabels = labels[:]
        # 每个岔中再建树
        myTree[bestFeatue][value] = createTree(splitDataSet(dataSet, bestFeature_index,
    return myTree
```

分类

根据构建的决策树和实际数据进行分类

```
def classify(inputTree, featLabels, testVec):
    """
    函数说明:使用决策树分类
    Parameters:
        inputTree - 已经生成的决策树
        featLabels - 存储选择的最优特征标签
        testVec - 测试数据列表，顺序对应最优特征标签
    Returns:
        classLabel - 分类结果
    """
    classLabel = 'None'
    firstStr = next(iter(inputTree))     # 获取决策树结点
    secondDict = inputTree[firstStr]     # 下一个字典
    featIndex = featLabels.index(firstStr) #树根代表的属性，所在属性标签的位置，即第几个
    for key in secondDict.keys():
        if testVec[featIndex] == key:
            if type(secondDict[key]).__name__ == 'dict':
                classLabel = classify(secondDict[key], featLabels, testVec)
            else:
                classLabel = secondDict[key]
    return classLabel
```

测试主函数

海洋动物分类

```
if __name__ == '__main__':
    testVec1 = [1,0]
    testVec2 = [1,1]
    # 使用决策树对testVec1和testVec2分类
    dataset, labels = createDataSet()
    featureLabels = []
    myTree = createTree(dataset, labels, featureLabels)
    print(myTree)
    result = classify(myTree, featureLabels, testVec1)
    print(result)
```

```
{'no surfacing': {0: 'no', 1: {'flippers': {0: 'no', 1: 'yes'}}}}
no
```

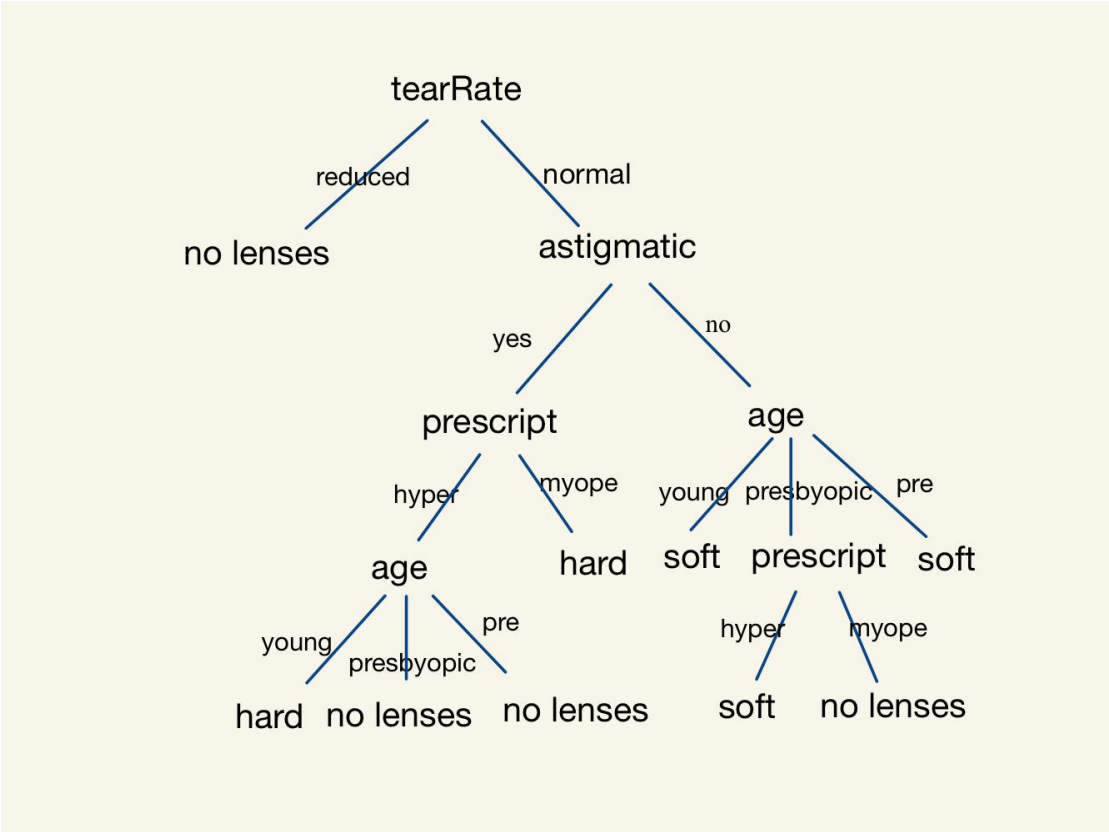
隐形眼镜

```
import pandas as pd

label = ["age", "prescript", "astigmatic", "tearRate", "class"]
data = pd.read_table("./decision_tree_glass/lenses.txt", header=None)
data = list(np.array(data))
for i in range(len(data)):
    data[i] = list(data[i])
featureLabels = []
myTree = createTree(data, label, featureLabels)
print(myTree)
```

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

树形图如下图所示：


tearRate
reduced
normal
no lenses
astigmatic
yes
no
prescript
age
hyper
myope
young
presbyopic
pre
age
hard
soft
prescript
soft
young
presbyopic
pre
hyper
myope
hard
no lenses
no lenses
soft
no lenses