《数据挖掘》实验报告（一）

专　　业　_____信息管理与信息系统_____

年　　级　____2016_____

学　　号　_____2016214288_____

学生姓名　_____曾德勤_____

指导老师　_____刘向_____

华中师范大学信息管理学院

I　实验目的和意义

数据挖掘设计实验的目的是培养学生具有初步的 python 程序设计、编程、调试的能力。通过实验使学生进一步熟悉并掌握 python 程序的调试运行环境、程序设计过程、程序的基本结构以及程序设计的基本方法。通过实验，使学生将程序设计的理论知识与实践相结合，为学生学习其他计算机编程语言打下基础。

II　实验内容

实验一　　决策树

【实验意义】

本章构造的决策树算法能够读取数据集合，构建的决策树是为了理解数据中所蕴含的知识信息，因此决策树可以使用不熟悉的数据集合，并从中提取出一系列规则。

【实验要求及步骤】

使用决策树预测隐形眼镜类型，可以帮助人们判断需要佩戴的镜片类型。

（1）收集数据：提供文本文件【lenses.txt】

（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，确保正确的解析数据内容，使用 createPlot()函数绘制最终的树形图。

（4）训练算法：使用代码参考中的 createTree() 函数。

（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）使用算法：存储树的数据结构，以便下次使用时无需重新构造树。

【数据说明】lenses.txt 的每一行包含 5 个数据，前四个分别为患者的属性 age、prescript、astigmatic、tearRate，第五个为隐形眼镜类型包括硬材质、软材质以及不适合佩戴隐形眼镜。

具体实验内容

（1）收集数据：提供文本文件【lenses.txt】

（2）准备数据：解析 tab 键分隔的数据行。

```
# coding=UTF-8
import trees
import treePlotter
fr = open('lenses.txt')
dataset = []
#labels '年龄', '处方', '散光', '眼镜材质'
labels = ['age', 'prescript', 'astigmatic', 'tearRate']
for line in fr.readlines():
    d = line.strip().split('\t')
    dataset.append(d)
fr.close()

print dataset
print '\n'
```

引入课件中的 tree.py 和 treePlotter.py，一行一行读取“lenses.txt”的内容，并忽略回车，用 tab 键解析分离数据。打印出数据集。

```
Press ENTER or type command to continue
[['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'], ['pre', 'myope', 'no', 'normal', 'soft'], ['pre', 'myope', 'yes', 'reduced', 'no lenses'], ['pre', 'myope', 'yes', 'normal', 'hard'], ['pre', 'hyper', 'no', 'reduced', 'no lenses'], ['pre', 'hyper', 'no', 'normal', 'soft'], ['pre', 'hyper', 'yes', 'reduced', 'no lenses'], ['pre', 'hyper', 'yes', 'normal', 'no lenses'], ['presbyopic', 'myope', 'no', 'reduced', 'no lenses'], ['presbyopic', 'myope', 'no', 'normal', 'no lenses'], ['presbyopic', 'myope', 'yes', 'reduced', 'no lenses'], ['presbyopic', 'myope', 'yes', 'normal', 'hard'], ['presbyopic', 'hyper', 'no', 'reduced', 'no lenses'], ['presbyopic', 'hyper', 'no', 'normal', 'soft'], ['presbyopic', 'hyper', 'yes', 'reduced', 'no lenses'], ['presbyopic', 'hyper', 'yes', 'normal', 'no lenses']]
```

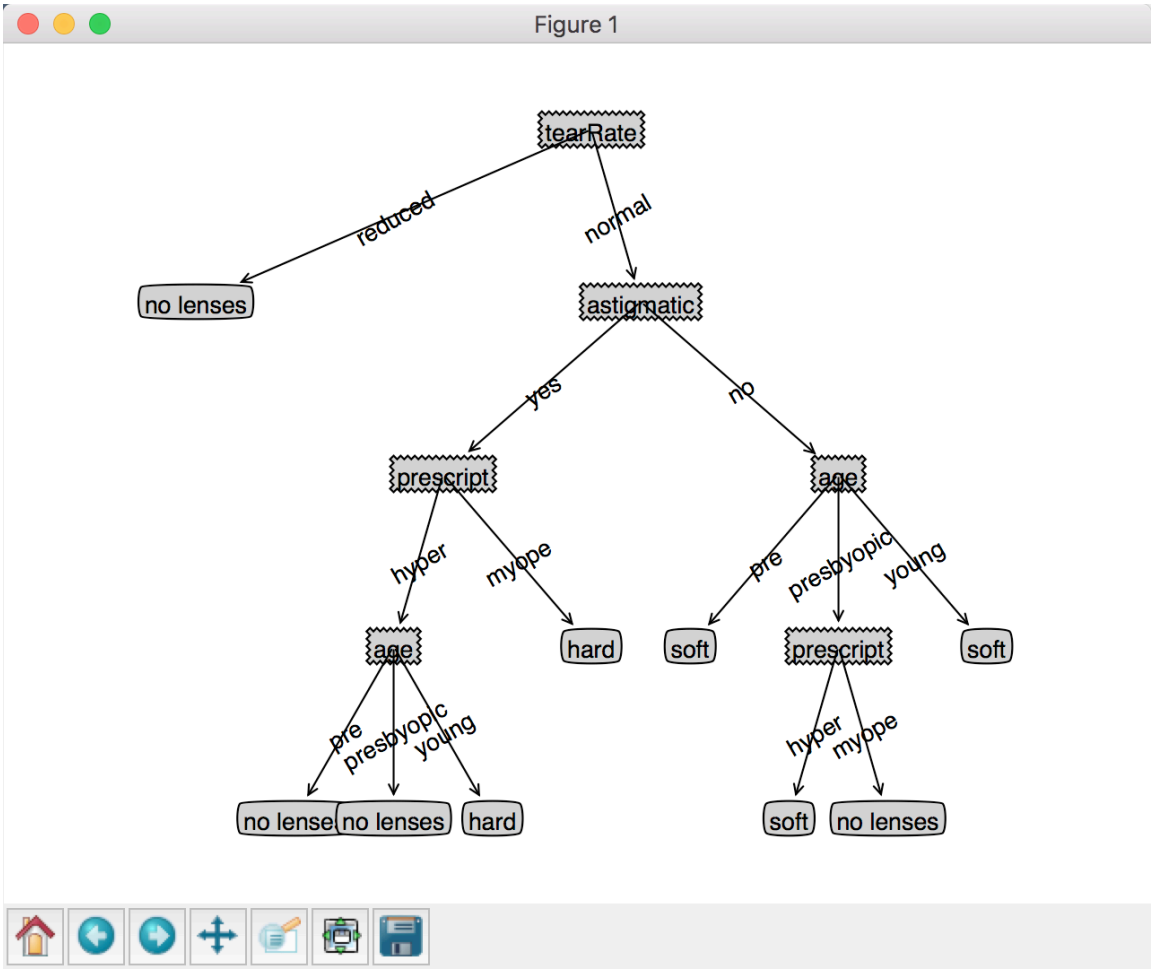
（3）分析数据：快速检查数据，确保正确的解析数据内容，使用 createPlot()函数绘制最终的树形图。

（4）训练算法：使用代码参考中的 createTree()函数。

```
print '数据集类的香农熵：'
print  trees.calcShannonEnt(dataset)
print '\n'

bestFeatureColumn = trees.chooseBestFeatureToSplit(dataset)
print '数据集最佳分类的属性是：'
print labels[bestFeatureColumn]
print '\n'

print '决策树：'
Tree = trees.createTree(dataset, labels)
print Tree
firstFeature = Tree.keys()[0]
print firstFeature
firstFeatureValues = Tree[firstFeature].keys()
print firstFeatureValues
print '\n'

treePlotter.createPlot(Tree)
```

分析数据：引入 tree.py 中的 calcShannonEnt()函数，计算 dataset 的香农熵，利用 chooseBetterFeatureToSplit()计算当前数据集最佳分类属性，createTree()算出数据集的决策树。引入 treePlotter.py，利用 createPlot()函数绘制最终的树形图。

```
数据集类的香农熵：
1.32608752536


数据集最佳分类的属性是：
tearRate


决策树：
{'tearRate': {'reduced': 'no lenses', 'normal': {'astigmatic': {'yes': {'prescript': {'hyper': {'age': {'p
re': 'no lenses', 'presbyopic': 'no lenses', 'young': 'hard'}}, 'myope': 'hard'}}, 'no': {'age': {'pre': '
soft', 'presbyopic': {'prescript': {'hyper': 'soft', 'myope': 'no lenses'}}, 'young': 'soft'}}}}}}
tearRate
['reduced', 'normal']
```



（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。

```
36 testVec = ['pre', 'myope', 'yes', 'normal']
37 print '测试数据'
38 print testVec
39 labels.append('tearRate')
40 print '匹配过程：'
41 result = trees.classify(Tree, labels, testVec)
42 print '匹配结果：'
43 print result
44 print '\n'
```

定义 testVec 测试数据，调用 tree.py 内的 classify()函数，算得测决策树为测试数据分得的类别。

```
测试数据
['pre', 'myope', 'yes', 'normal']
匹配过程：
('+++', 'tearRate', 'xxx', {'reduced': 'no lenses', 'normal': {'astigmatic': {'yes': {'prescript': {'hyper
': {'age': {'pre': 'no lenses', 'presbyopic': 'no lenses', 'young': 'hard'}}, 'myope': 'hard'}}, 'no': {'a
ge': {'pre': 'soft', 'presbyopic': {'prescript': {'hyper': 'soft', 'myope': 'no lenses'}}, 'young': 'soft'
}}}}}, '---', 'normal', '>>>', {'astigmatic': {'yes': {'prescript': {'hyper': {'age': {'pre': 'no lenses',
 'presbyopic': 'no lenses', 'young': 'hard'}}, 'myope': 'hard'}}, 'no': {'age': {'pre': 'soft', 'presbyopi
c': {'prescript': {'hyper': 'soft', 'myope': 'no lenses'}}, 'young': 'soft'}}}})


('+++', 'astigmatic', 'xxx', {'yes': {'prescript': {'hyper': {'age': {'pre': 'no lenses', 'presbyopic': 'n
o lenses', 'young': 'hard'}}, 'myope': 'hard'}}, 'no': {'age': {'pre': 'soft', 'presbyopic': {'prescript':
 {'hyper': 'soft', 'myope': 'no lenses'}}, 'young': 'soft'}}}, '---', 'yes', '>>>', {'prescript': {'hyper'
: {'age': {'pre': 'no lenses', 'presbyopic': 'no lenses', 'young': 'hard'}}, 'myope': 'hard'}})


('+++', 'prescript', 'xxx', {'hyper': {'age': {'pre': 'no lenses', 'presbyopic': 'no lenses', 'young': 'ha
rd'}}, 'myope': 'hard'}, '---', 'myope', '>>>', 'hard')


匹配结果：
hard
```

（6）使用算法：存储树的数据结构，以便下次使用时无需重新构造树。

```
46 # 把树存在磁盘中
47 print '将树存放磁盘...'
48 trees.storeTree(Tree, 'myTree.txt')
49 print '\n'
50
51 # 从磁盘中取出树
52 print '再从磁盘中读取树： '
53 print trees.grabTree('myTree.txt')
54
```

```
测试数据
['pre', 'myope', 'yes', 'normal']
匹配过程：
('+++', 'tearRate', 'xxx', {'reduced': 'no lenses', 'normal': {'astigmatic': {'yes': {'prescript': {'hyper
': {'age': {'pre': 'no lenses', 'presbyopic': 'no lenses', 'young': 'hard'}}, 'myope': 'hard'}}, 'no': {'a
ge': {'pre': 'soft', 'presbyopic': {'prescript': {'hyper': 'soft', 'myope': 'no lenses'}}, 'young': 'soft'
}}}}}, '---', 'normal', '>>>', {'astigmatic': {'yes': {'prescript': {'hyper': {'age': {'pre': 'no lenses',
 'presbyopic': 'no lenses', 'young': 'hard'}}, 'myope': 'hard'}}, 'no': {'age': {'pre': 'soft', 'presbyopi
c': {'prescript': {'hyper': 'soft', 'myope': 'no lenses'}}, 'young': 'soft'}}}})


('+++', 'astigmatic', 'xxx', {'yes': {'prescript': {'hyper': {'age': {'pre': 'no lenses', 'presbyopic': 'n
o lenses', 'young': 'hard'}}, 'myope': 'hard'}}, 'no': {'age': {'pre': 'soft', 'presbyopic': {'prescript':
 {'hyper': 'soft', 'myope': 'no lenses'}}, 'young': 'soft'}}}, '---', 'yes', '>>>', {'prescript': {'hyper'
: {'age': {'pre': 'no lenses', 'presbyopic': 'no lenses', 'young': 'hard'}}, 'myope': 'hard'}})


('+++', 'prescript', 'xxx', {'hyper': {'age': {'pre': 'no lenses', 'presbyopic': 'no lenses', 'young': 'ha
rd'}}, 'myope': 'hard'}, '---', 'myope', '>>>', 'hard')


匹配结果：
hard
```

调用 tree.py 内的 storeTree()和 grabTree()，将树的数据结构存储在当前工作目录的 myTree.txt 文件中，再从磁盘中取出验证下一次取出无需构造树。

```
将树存放磁盘...


再从磁盘中读取树：
{'tearRate': {'reduced': 'no lenses', 'normal': {'astigmatic': {'yes': {'prescript': {'hyper': {'age': {'p
re': 'no lenses', 'presbyopic': 'no lenses', 'young': 'hard'}}, 'myope': 'hard'}}, 'no': {'age': {'pre': '
soft', 'presbyopic': {'prescript': {'hyper': 'soft', 'myope': 'no lenses'}}, 'young': 'soft'}}}}}}
python task.py  0.87s user 0.45s system 1% cpu 1:36.40 total
Press ENTER or type command to continue
```

实习小结	这次实验，基本上读懂了 **tree.py** 和 **treePlotter.py** 中的函数。 同时加深了我对决策树建立的主要思路的印象：通过计算香农熵和增益找出当前数据集的最佳分类属性，作为当前子树的根结点，并将该数据集根据最佳分类属性的不同的值进行分组，引出这个根结点的子树，再对每个子树和对应的数据集进行上述的递归操作，直到子数据集的类别数为 **1** 或者数据集的属性只有一列，当属性只有一列的时候，根节点取数量最多的那个类别。 计算决策树的函数中，有一行是删除 **labels** 的 **bestFeature**，这个 **labels** 是引用类型，在函数中 **del** 掉某个值，外面的 **labels** 就会跟着发生变化，所以我为了方便在主函数中重新修改了 **labels**，但是这里建议在函数初始的时候就深复制一个 **labels**，这样 **del** 就不会影响外面的 **labels** 了。 **Python2.7** 解决中文编码问题：在每个源代码开头添加：**# coding=UTF-8**