# 数据挖掘实验报告

## 1.实验环境

编程语言：实验程序使用Python编写

运行环境：Windows，Python3.5

数据集：Iris，Wine，Seeds

## 2.实验内容

### 2.1 决策树算法实现

1. 数据结构：

数据集存储结构：

Transactions =

[

[1.0,2.0,3.0,….], //属性值列表，最后一项为此项数据的类别标签

[1.3,2.3,3.3,….],

……

]

可选属性列表：

Attr\_list = [0,1,2,3,….] //长度为单个数据记录属性值个数

决策树结构：

Decision\_tree =

{

“index,value”:

{

“low”:{…},

“high”:{…}

}

}

Index为划分属性的下标，value为划分属性的阈值，“low”表示第index个属性值小于等于value的取值，“high”表示第index个属性值大于value的取值， 叶子节点格式为{“final”：“class”}

1. 决策树生长算法实现：

If 没有可用属性：

停止划分，当前节点为叶子节点，计算节点的类别为出现最多的类别，存入决策树

返回

根据信息增益、GINI指数或错误率得到最佳划分属性，并得到划分后的两个子集

If 有一个子集为空集：

将当前节点作为叶子节点，计算节点类别，存入决策树

返回

将使用的属性从可用属性列表中删除

If 子集中的所有记录类别相同：

停止划分，当前节点为叶子节点，计算节点类别，存入决策树

返回

else：

对子集进行递归，成长决策树

结束

1. 最佳划分属性选择算法实现：

遍历所有的可用属性：

遍历所有属性值：

将数据集划分为小于等于属性值的集合和大于属性值的集合

计算得到划分的信息增益、GINI指数或错误率

更新最小的GINI指数、错误率或最大信息增益的属性和属性值

返回最优的划分属性，属性值和两个子集

结束

1. 主要函数说明：

CalcProbs() //计算每种类别出现的概率

CalcEntropy() //计算信息熵

CalcGini() //计算GINI指数

CalcErrorRate() //计算错误率

SeletBestSplit() //选择最优的划分属性

SplitDataSet() //根据划分阈值将数据集划分为两个子集

GrowTree() //决策树生长算法函数

IsAllinOneClass() //判断数据集的类别是否一致

SelectClassLabel() //计算出现最多的类别标签作为叶子节点的类别标签

1. 实验结果：

属性划分方式为信息增益，无剪枝策略，Iris数据集生成的决策树，如图2-1所示：

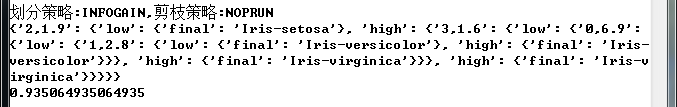


图2-1 Iris数据集的决策树

属性划分方式为Gini指数，无剪枝策略，Wine数据集生成的决策树，如图2-2所示：

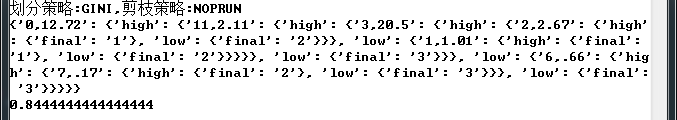


图2-2 Wine数据集决策树

属性划分方式为错误率，无剪枝策略，Seeds数据集生成的决策树，如图2-3所示：

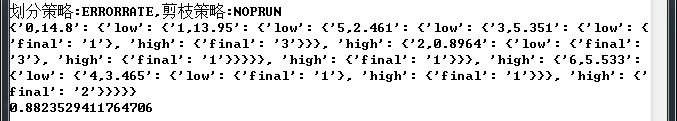


图2-3 Seeds数据集的决策树

### 2.2 决策树验证方法

1. HoldOut验证思想：

将原始数据集随机分为两个数据集，一个最为训练集，另一个作为验证集。使用训练集训练出决策树，并用验证集验证决策树的正确率。

1. 10折交叉验证思想：

将数据集平分为10份，轮流将其中9份作为训练集得到决策树，另外1份作为验证集，计算每次验证的正确率，最后计算平均值作为决策树的正确率。

1. Boostrap验证思想：

从数据集中可放回地随机抽取N条记录作为训练集，训练得到决策树，使用整个数据集来验证决策树得到正确率，迭代多干次

1. 正确率验证算法：

遍历验证集的所有数据：

key = 根节点的key

tree = decision\_tree[key]

解析得到根节点划分的属性和属性值

While key != ‘final’:

If 训练集属性值>=划分属性值:

更新key = tree[‘low’]的key

tree = tree[‘low’][key]

else:

更新key = tree[‘high]的key

tree = tree[‘high][key]

/\*此时tree为最终的叶子节点的类别标签\*/

If tree == 数据集的实际类别标签

正确记录数+1

return 正确记录数/验证集总数

结束

1. 主要函数说明：

HoldOutMethod() //HoldOut验证

CrossValidation() //10折交叉验证

BootstrapMethod() //bootstrap验证

ValidTransaction() //验证一条数据记录的正确性

ValidTree() //计算决策树的正确率

1. 实验结果

划分方式为信息增益，无剪枝策略，Iris数据集采用HoldOut验证的正确率如图2-4所示，数据集被随机划分为训练集和验证集，图中显示了训练集生成的决策树和验证集验证的正确率：

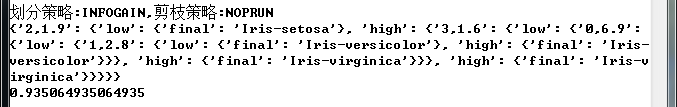


图2-4 Iris数据集HoldOut验证的正确率

划分方式为信息增益，无剪枝策略， Iris数据集采用10折交叉验证的正确率如图2-5和2-6所示，数据集被划分为10份，迭代10次，每次选择1份作为验证集，剩下的作为训练集，图中显示了10折验证法的每次迭代生成的决策树和累计的正确率：

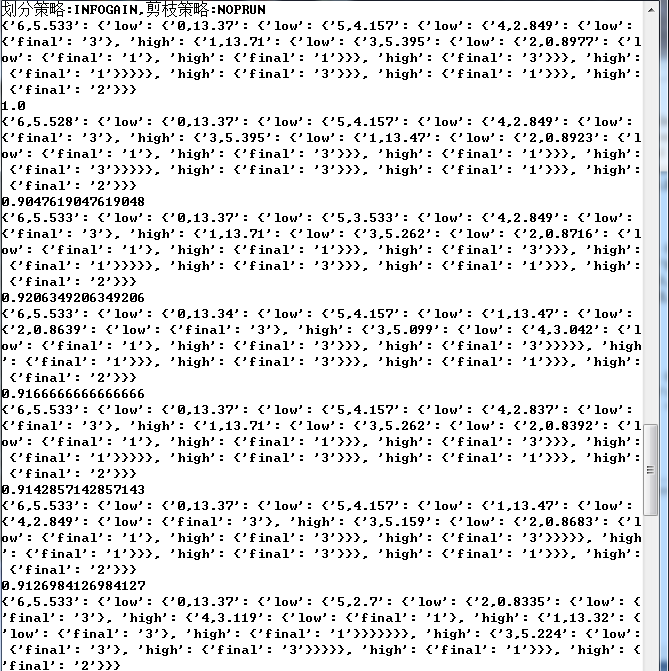


图2-5 Iris数据集10折交叉验证的正确率（1）

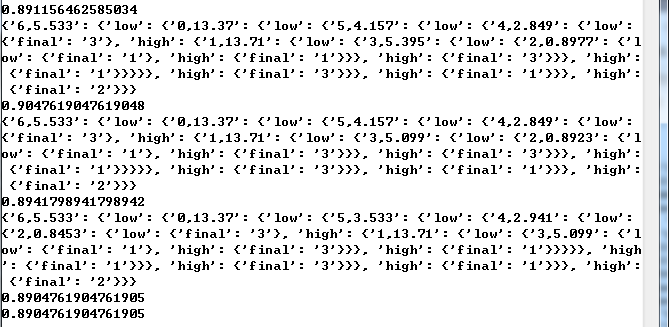


图2-6 Iris数据集10折交叉验证的正确率（2）

### 2.3 预剪枝和后剪枝

1. 预剪枝思想：

预剪枝是在决策树生成过程中增加额外的终止条件，如对数据子集的大小做出约束，当划分后的一个数据子集小于阈值，则停止划分。

1. 预剪枝实现：

预剪枝是在决策树生成时增加约束，本次实验我增加了对划分后的数据子集大小的约束，即判断划分后的子集的大小是否有一个小于3，若是则停止此次划分，否则就继续划分。

1. 后剪枝思想：

后剪枝是在决策树生成后，对过拟合的决策树进行优化，已达到简化决策树的目的，本实验采用的是Reduced-Error Pruning。

1. 后剪枝实现：

本实验采用的是Reduced-Error Pruning，大致实现如下：

遍历决策树的所有非叶子节点：

将以该节点为根的子树替换为叶子节点

将此分支出现最多的类别标签作为叶子节点的类别标签

计算修改后的决策树的正确率

If 新树的正确率 >= 旧树的正确率：

删除原决策树中的以此节点为根的子树，用叶子节点代替

1. 主要函数说明：

PostPrun() //对生成的决策树进行后剪枝算法

1. 实验结果

对Iris数据集不进行剪枝，预剪枝和后剪枝的对比如图2-7所示，可以发现预剪枝和后剪枝的可能提高决策树的正确率，后剪枝可以通过简化决策树来提高决策树的正确率，一定程度上解决过拟合问题。

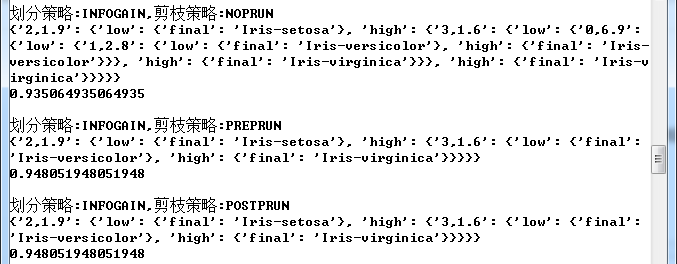


图2-7 Iris剪枝结果对比

图2-8展示3种剪枝策略对Wine数据集的影响，可以看到剪枝后的决策树更加简化，正确率也相对较高。

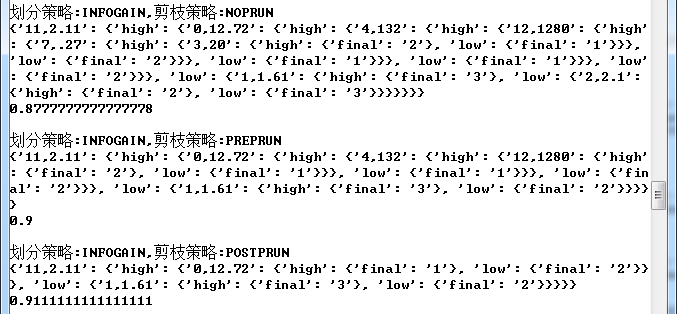


图2-8 Wine剪枝结果对比

1. 存在问题

经过多次实验发现，预剪枝有时反而会降低决策树的正确率。这是因为有时选择的控制数据子集大小的阈值不合适，决策树忽略了数据集某些正确的性质造成的，如图2-9所示。

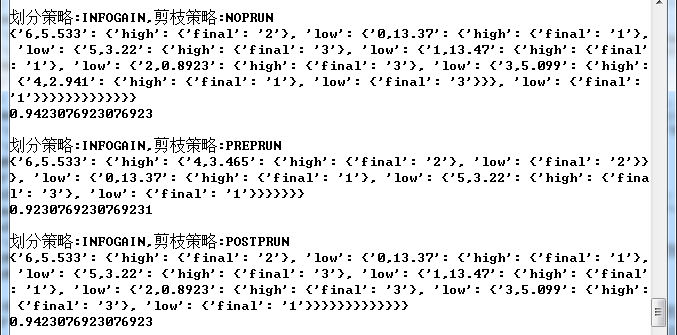


图2-9 预剪枝降低决策树精度问题

### 2.4 AdaBoost算法

1. AdaBoost算法思想：

初始化数据集的每条记录的权值为1/N，可重复的选取N个数据集作为训练集，训练得到一棵弱决策树，将整个数据集作为验证集验证决策树，得到带权值的错误率，根据错误率更新每条数据记录的权值，验证正确的权值降低，验证错误的权值升高。迭代T次，生成带权重的T个决策树，预测结果为每个决策树结果的加权和。

1. AdaBoost算法实现

由于时间原因，AdaBoosr算法的实现调用的是sklearn库的AdaBoostClassifier类，该类较好地实现了AdaBoost算法，需要传入训练集的属性列表和类别标签列表，返回预测结果类别标签列表，实现代码如下：

def AdaBoostClassifier(self):

FeatureSet = [example[0:-1] for example in self.transactions]

Label = [example[-1] for example in self.transactions]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(FeatureSet, Label, random\_state=1)

clf = AdaBoostClassifier(n\_estimators=100)

clf = clf.fit(X\_train, y\_train)

pre\_labels = clf.predict(X\_test)

其中，FeatureSet存储的是数据集的属性列表，Label存储数据集的所有标签，X\_train，X\_test，y\_train，y\_test分别为交叉验证生成的训练集属性列表，验证集属性列表，训练集实际结果列表，验证集实际结果列表，pre\_labels为预测结果列表

1. 实验结果：

图2-10展示Iris数据集使用AdaBoost验证的正确率为97.4%：

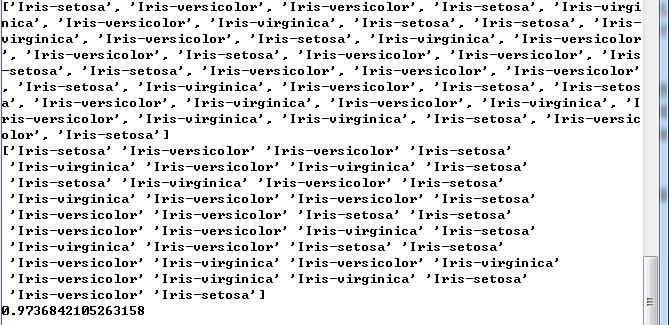


图2-10 Iris数据集采用AdaBoost验证

图2-11展示Iris数据集用HoldOut验证的正确率为93.5%：

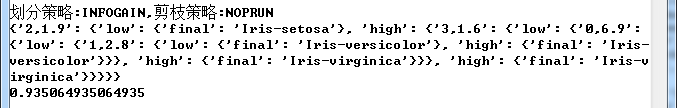


图2-11 Iris数据集采用HoldOut验证

图2-12展示Iris数据集用10折交叉检验验证的正确率为89.0%：



图2-12 Iris数据集采用10折交叉检验验证

可以看出AdaBoost采用多个分类器提升的验证方法确实提高了分类器的精度。

### 2.5 K-means算法

1. 算法思想：

K-means算法是将数据集分为K个簇，每个簇有一个质心，满足簇内的点到簇的质心的距离小于到别的质心的距离。

1. 算法实现：

随机选择k个质心

将数据集中的所有点归到离自己最近的质心所在的簇

old\_val = -1

new\_val = 聚类划分的SSE

while |new\_val – old\_val| >= 1： /\*两次聚类划分的SSE只差小于1，停止划分\*/

重新计算质心

根据新的质心重新生成簇

Old\_val = new\_val

New\_val = 新聚类的SSE

1. 主要函数说明：

k\_means() //实现K-means算法

grow\_clusters() //将所有的数据放入相应的簇中

get\_dist() //计算点到质心的欧式距离

clusterof() //根据质心决定点属于哪个簇

get\_var() //计算簇的SSE

get\_means() //计算当前簇的质心

1. 实验结果：

图2-13—2-17分别展示K=1—5的情况下，Iris数据集SSE的变化情况，从图中可以看出，K=1—5时，最终的SSE分别为：291,128,98,85,78。

图2-18展示了Iris数据集SSE随K值变化的曲线。可以看出K值越大，即划分的簇越多，聚类的SSE就越小，当K>3之后SSE变化明显变缓，K=3是SSE变化的拐点，因此对于Iris数据集最佳的K值为3.

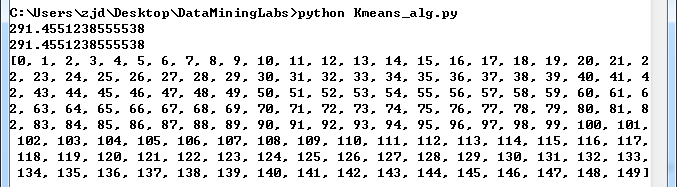


图2-13 Iris数据集K=1时聚类结果

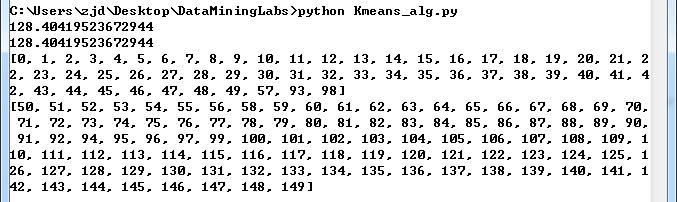


图2-14 Iris数据集K=2时聚类结果

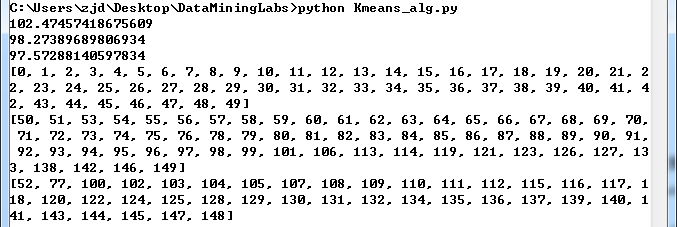


图2-15 Iris数据集K=3时聚类结果

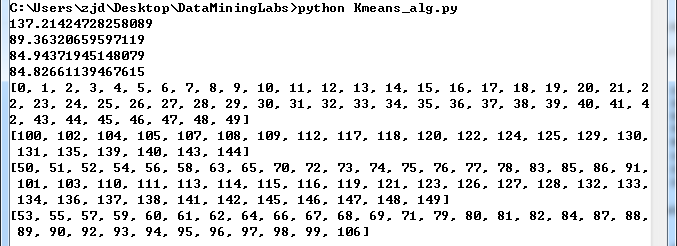


图2-16 Iris数据集K=4时聚类结果

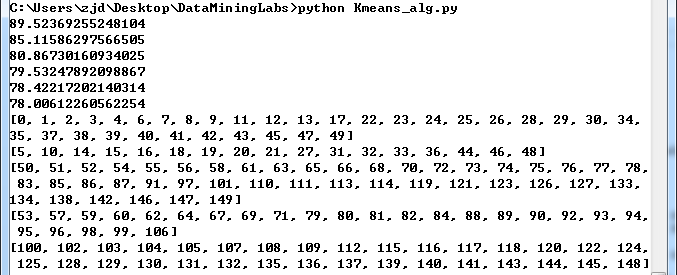


图2-17 Iris数据集K=5时聚类结果

图2-18 Iris数据集SSE随K值变化曲线

对于Wine数据集K=1—5的SSE分别为：46380,23755,16555,13086,10727。图2-19展示了Wine数据集SSE随K值变化的曲线。可以看出K>3后SSE变化趋势明显变缓，K=3为SSE变化的拐点，因此对于Wine数据集最佳的K值为3.

图2-19 Wine数据集SSE随K值变化曲线

对于Seeds数据集K=1—5的SSE分别为：690,422,314,295,256。图2-20展示了Seeds数据集SSE随K值变化的曲线。可以看出K>3后SSE变化趋势明显变缓，K=3为SSE变化的拐点，因此对于Seeds数据集最佳的K值为3.

图2-20 Seeds数据集SSE随K值变化曲线

图2-21和图2-22展示了K=3时两次运行程序对Iris数据集进行聚类的结果，由于初始质心的选择是随机的，因此虽然两次实验的K都是3，但是不同的初始质心选择导致最终的SSE存在差异，一般来说，初始质心之间相距越远，划分得到的聚类越好：

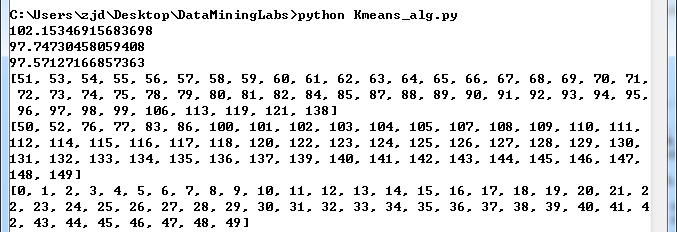


图2-21 K=3时Iris数据集聚类第一次运行结果

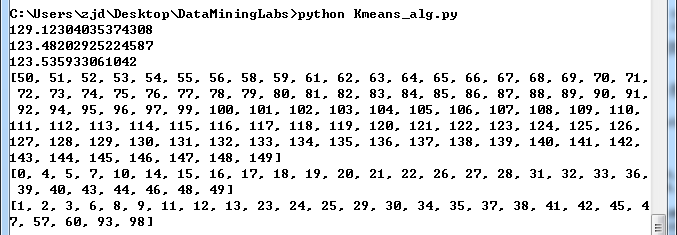


图2-22 K=3时Iris数据集聚类第二次运行结果