决策树算法

18231179 张兴鹏

一、实验目的

1.理解为什么我们使用基于熵的测度来构造决策树。
2.了解在构建决策树的过程中如何使用信息增益来选择属性。
3.理解为什么我们有时需要修剪树以及我们如何修剪树。
4.理解决策树与一组规则的等价性
5.了解软决策树的概念，以及为什么它们是经典决策树的重要扩展。

二、实验原理

1.信息熵

度量样本集合纯度中，我们最常用的指标是信息熵(information entropy)。

$$Ent(D) = -\sum_{k=1}^{N}(p_k log_2 p_k)$$

信息熵用于衡量样本集合 D 中，N 类样本的纯度。信息熵值越小，说明纯度越高（想象集合中只有一类样本，则 Ent(D)=0）。

信息增益，是根据样本的类别进行计算。假定属性 a 有 V 个可能的取值，则信息增益计算公式如下：

$$Gain(D,a) = Ent(D) - \sum_{v=1}^{V}(\frac{|D^v|}{|D|}Ent(D^v))$$

2.决策树剪枝问题

决策树的剪枝有两种思路：预剪枝（Pre-Pruning）和后剪枝（Post-Pruning）
预剪枝（Pre-Pruning）

在构造决策树的同时进行剪枝。所有决策树的构建方法，都是在无法进一步降低熵的情况下才会停止创建分支的过程，为了避免过拟合，可以设定一个阈值，熵减小的数量小于这个阈值，即使还可以继续降低熵，也停止继续创建分支。但是这种方法实际中的效果并不好。

后剪枝（Post-Pruning）

决策树构造完成后进行剪枝。剪枝的过程是对拥有同样父节点的一组节点进行检查，判断如果将其合并，熵的增加量是否小于某一阈值。如果确实小，则这

一组节点可以合并一个节点，其中包含了所有可能的结果。后剪枝是目前最普遍的做法。
后剪枝的剪枝过程是删除一些子树，然后用其叶子节点代替，这个叶子节点所标识的类别通过大多数原则(majority class criterion)确定。所谓大多数原则，是指剪枝过程中, 将一些子树删除而用叶节点代替,这个叶节点所标识的类别用这棵子树中大多数训练样本所属的类别来标识,所标识的类 称为 majority class。

三、实验代码与结果分析

1.一种实现方法

1.1 获得属性的每个值的熵：

```
def getEntropy(counter):
    res = 0
    denominator = np.sum(counter)
    if denominator == 0:
        return 0
    for value in counter:
        if value == 0:
            continue
        res += value / denominator * math.log2(value / denominator if value > 0 and
                                               denominator > 0 else 1)
    return -res
```

1.2 对每个划分点计算熵，选择信息增益最大的特征为划分点：

```
for i in range(len(feature)):
        leng=i+1
        remain = length - leng
        d1=getEntropy(feature[i][1])
        d2=getEntropy(feature[i][2])
        remain=length-leng
        entropy=(leng/length)*d1+(remain/length)*d2
        if entropy<minEntropy:
            minEntropy=entropy
            razor=feature[i][0]
    return razor
```

1.3 寻找最大索引：

```
def findMaxIndex(dataSet):
    maxIndex = 0
    maxValue = -1
    for index, value in enumerate(dataSet):
        if value > maxValue:
            maxIndex = index
```

```
            maxValue = value
    return maxIndex
```

1.4 递归生成树：

```
def tree(featureSet, dataSet, counterSet):
    if (counterSet[0] == 0 and counterSet[1] == 0 and counterSet[2] != 0):
        return iris.target_names[2]
    if (counterSet[0] != 0 and counterSet[1] == 0 and counterSet[2] == 0):
        return iris.target_names[0]
    if (counterSet[0] == 0 and counterSet[1] != 0 and counterSet[2] == 0):
        return iris.target_names[1]
    if len(featureSet) == 0:
        return iris.target_names[findMaxIndex(counterSet)]
    if len(dataSet) == 0:
        return []
    res = sys.maxsize
    final = 0
    for feature in featureSet:
        i = razors[feature]
        set1 = []
        set2 = []
        counter1 = [0, 0, 0]
        counter2 = [0, 0, 0]
        for data in dataSet:
            index = int(data[-1])

            if data[feature] < i:
                set1.append(data)
                counter1[index] = counter1[index] + 1
            elif data[feature] >= i:
                set2.append(data)
                counter2[index] = counter2[index] + 1
        #计算属性的熵
        a = (len(set1) * getEntropy(counter1) + len(set2) * getEntropy(counter2)) / len(dataSet)
        #获得熵最小的属性作为树节点(即信息增益最大的属性)
        if a < res:
            res = a
            final = feature
    featureSet.remove(final)
    child = [0, 0, 0]
    child[0] = final
    #递归生成其他树节点
    child[1] = tree(featureSet, set1, counter1)
    child[2] = tree(featureSet, set2, counter2)
```

```
return child
```

2.另一种实现方法

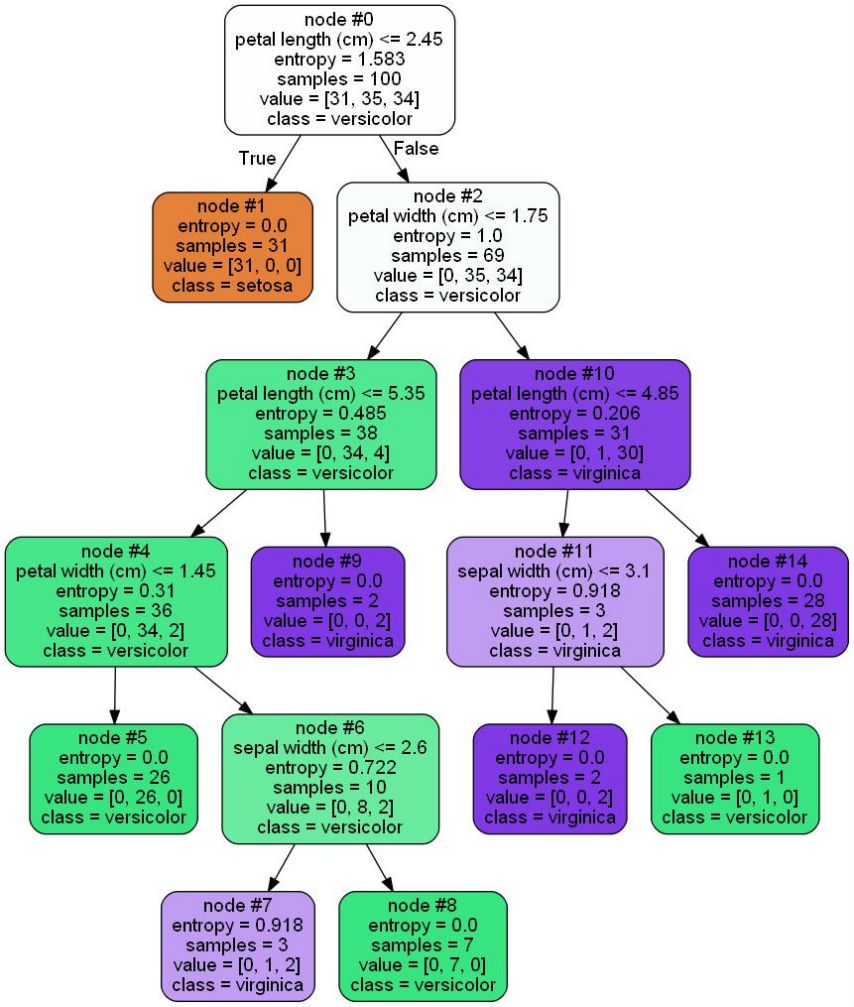
我调用了 sklearn 库实现了整个决策树算法对鸢尾花数据集的分类。实验代码如下。

```
import pydotplus
from sklearn import datasets # 导入方法类
from sklearn.tree import export_graphviz
from sklearn.tree import DecisionTreeClassifier #此时是分类树
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from IPython.display import Image, display

#加载数据，得到特征数据与分类数据
iris = datasets.load_iris()
iris_feature = iris.data
iris_target = iris.target
#划分训练集与测试集
feature_train, feature_test, target_train, target_test = train_test_split(iris_feature,
                                                    iris_target,test_size=0.33, random_state=42)
target_train
#训练模型
dt_model = DecisionTreeClassifier(criterion='entropy',max_depth=5,splitter='best')
# 选择熵做损失函数，采用“最佳”分裂策略，控制最大深度为 5 防止过拟合
dt_model.fit(feature_train,target_train)
# 使用训练集训练模型
predict_results = dt_model.predict(feature_test)
# 使用模型对测试集进行预测

print('predict_results:', predict_results)
print('target_test:', target_test)
print('精确度=',accuracy_score(target_test,predict_results))
#可视化
#out_file=None 直接把数据赋给 image，不产生中间文件.dot
#filled=Ture 添加颜色，rounded 增加边框圆角
image = export_graphviz(dt_model, out_file=None,feature_names=iris.feature_names,
                        class_names=iris.target_names,filled=True,node_ids=True,rounded=True)
graph = pydotplus.graph_from_dot_data(image)
display(Image(graph.create_png()))
```

实验结果：

```
predict_results: [1 0 2 1 1 0 1 2 2 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2
1
 0 0 0 2 1 1 0 0 1 1 2 1 2]
target_test: [1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0 0 0 1 0 0 2 1
 0 0 0 2 1 1 0 0 1 2 2 1 2]
精确度= 0.96
```

画出决策树：



3.连续值的离散化

由于连续属性的可取值数目不再有限，因此，不能直接根据连续属性的可取值来对节点进行划分。此时，我们可以使用连续属性离散化的技术将连续属性转换为离散的。

最简单的策略是使用二分法对连续属性进行处理，即把区间[ai,ai+1)的中位点作为候选划分点。然后我们就可像离散属性值一样来考察这些划分点，选取最优的划分点进行样本集合的划分。

四、决策树与朴素贝叶斯模型

在许许多多的分类模型中，决策树模型和朴素贝叶斯模型（NBC）应用最为广泛。决策树模型通过数据集来构造一棵决策树，为未知样本产生一个分类。决策树便于使用，高效；容易构造出易于解释的规则；可以较好地扩展到大型数据

库中；可以对有许多属性的数据集构造决策树。同时，决策树模型也有一些缺点，比如过度拟合问题，对缺失数据处理困难，忽略数据集中属性之间的相关性等。

朴素贝叶斯模型发源于古典数学理论，所需估计的参数很少，对缺失数据不太敏感，算法较为简单。在属性相关性较小时，NBC 模型的性能优于决策树模型。在属性个数较多或者属性之间相关性较大时，NBC 模型的分类效率低于决策树模型。

五、软决策树

针对泛化能力强大的深度神经网络（DNN）无法解释其具体决策的问题，深度学习殿堂级人物 Geoffrey Hinton 等人发表论文提出软决策树这一概念。相较于从训练数据中直接学习的决策树，软决策树的泛化能力更强；并且通过层级决策模型把 DNN 所习得的知识表达出来，具体决策解释容易很多。这最终缓解了泛化能力与可解释性之间的张力。

六、总结

在本次大作业中，我学习了决策树的分类方法，同时了解了 sklearn 库的使用方法，构造出了精确度为 96%的决策树，实现了对 iris 数据集的分类，同时我还了解了朴素贝叶斯以及软决策树的方法，让我对决策树等分类方法的概念与原理有了更加深刻的了解。