山东大学计算机科学与技术学院

机器学习与模式识别课程实验报告

学号：201600181073	姓名： 唐超	班级： 智能16
实验题目：Decision Tree		
实验学时：2	实验日期:2018.11.15	

实验目的：

将葡萄酒特征属性及其分类数据集分为训练集和测试集，在训练集上构建决策树，并在测试集上对样本进行分类，用交叉验证的方法多次划分数据集并训练测试，最后求得平均准确率，用以决策树模型在该数据集上分类的效果。

硬件环境：

DELL 台式机

软件环境：

Python3.6

实验步骤与内容：

1. **下载、解压并读取数据集 ex6Data.csv,由于数据集内的每个数据都是以字符串类型读取的，在进行后续处理时需要将其转化为 float.**
2. **划分数据集**

   由于要进行 10-fold 交叉验证，分别令 r = 0,1,…,9，将每组数据的序号除以 10 的余数为 r 的作为验证集，剩余数据作为训练集。
3. **在训练集上构建决策树**

   构建决策树的函数为 createTree，其最终构建的决策树以字典形式返回，它的两个输入参数为数据集 dataSet 和特征属性 features，用递归的方法构建。

   (1) 递归构建树的终止条件

   - 如果 dataSet 中所有样本都属于一个类别时，停止往下构建，返回此类别
   - 如果特征属性在之前的构建过程中已全部用完，即 features 为空，停止往下构建，返回 dataSet 中占多数的类别

   (2) 如果不满足终止条件，选择最优划分特征及划分点

   选择最优划分特征及划分点的函数为 chooseBestFeature，其输入参数为数据集和特征属性的索引 features_num，返回最优特征属性的索引和划分点的值。

   采用信息增益率作为评价指标，对不同特征属性分别进行以下计算：

   - 按此特征属性值从小到大对数据集进行排序
   - 找到相邻两个类别不同的样本，以他们这一特征属性值的平均值作为划分点，依次计算在该划分点下的信息熵、信息增益、内在信息（intrinsic information），最后用信息增益除以内在信息得到信息增益率
   - 在遍历每个特征属性及其可能划分点的过程中记录最大信息增益率，最后返回最大信息增益率对应的特征属性索引和划分点

   (3) 划分子树

   根据上一步得到的最优划分特征和划分点，将目前的数据集 dataSet 划分特征

属性值小于和大于划分点两类，对于每一类数据集递归调用 createTree 函数构建子树，此处传入参数 features 需要删去目前的最优划分特征（子树不再使用）。

4. 在测试集上评价决策时的预测效果

分类函数为 classify，其参数为一组样本数据、构建好的决策树以及特征属性表，返回预测类别 0 或 1，同样采用递归的方法，对测试集中的每组数据按照决策树依次进行预测，最后对比预测类别与实际类别，将相一致的个数除以数据集的大小得到准确率，将 10 次划分并测试的准确率求平均得到最后的模型准确率。

10 次划分的准确率分别如下：0.7735，0.8082，0.7796，0.7714，0.8000，0.8122，0.7980，0.7980，0.7935，0.8078. 最终准确率为 0.7942.

结论分析与体会：

通过这次实际构建决策树的实验，我加深了对决策树的理解，对 python 语言也更加熟悉了。

附录：程序源代码

```
import csv
from math import log
import json

# Enter You Name Here
myname = "Tang"


'''求信息熵'''
def calShannonEnt(dataset):
    m = len(dataset)
    lableCount = {}
    '''计数'''
    for data in dataset:
        currentLabel = data[-1]
        if currentLabel not in lableCount.keys():
            lableCount[currentLabel] = 0
        lableCount[currentLabel] += 1
    '''遍历字典求和'''
    entropy = 0
    for label in lableCount:
        p = float(lableCount[label]) / m
        entropy -= p * log(p, 2)
    return entropy
```

```
'''第 i 个特征根据取值 value 划分子数据集'''
def splitdataset0(dataset, axis, value):
    subSet = []
    for data in dataset:
        if (data[axis] <= value):
            subSet.append(data)
    return subSet    # subSet 是一个列表，其每一个元素为一组测试数据


def splitdataset1(dataset, axis, value):
    subSet = []
    for data in dataset:
        if (data[axis] > value):
            subSet.append(data)
    return subSet


'''遍历数据集求最优特征和划分点'''
def chooseBestFeature(dataSet, features_num):    #  是否还应有一个剩余 feature 的参数
    origin_ent = calShannonEnt(dataSet)
    bestFeatureAndValue = [0, 0.0]
    #print("features_num:", features_num, "len(dataSet):", len(dataSet))
    for i in features_num:
        #  按第 i 个特征将 dataset 排序
        #print("i:", i)
        feature_sort = []
        for j in range(len(dataSet)):
            #print("j:", j)
            #print("len(dataSet[j]):", len(dataSet[j]))
            feature_sort.append([dataSet[j][i], dataSet[j][-1]])
        feature_sort.sort()
        #print(feature_sort)
        for k in range(len(dataSet) - 1):
            if feature_sort[k][1] != feature_sort[k + 1][1]:
                split_value = (feature_sort[k][0] + feature_sort[k + 1][0]) / 2
                m = float(k + 1) / len(dataSet)
                #  计算该 split 下的信息熵、信息增益
                info=m*calShannonEnt(feature_sort[:k])+(1-m)*
                   calShannonEnt(feature_sort[k + 1:-1])
                gain = origin_ent - info
                #  该划分下的内在信息
                instrinsic_info = -m * log(m, 2) - (1 - m) * log((1 - m), 2)
                #  信息增益率
                igr = gain / instrinsic_info
```

```
                    #print("split_value:", split_value,"igr:",igr)
                    if igr > bestFeatureAndValue[1]:
                        bestFeatureAndValue[0] = i
                        bestFeatureAndValue[1] = split_value
    return bestFeatureAndValue


'''计数并返回最多类别'''
def majorityCnt(classList):
    classCount = {}
    for class_ in classList:
        if class_ not in classCount.keys():
            classCount[class_] = 0
        classCount[class_] += 1
    if classCount[0.0] > classCount[1.0]:
        return 0.0
    else:
        return 1.0


'''向下递归创建树  '''
def createTree(dataSet, features):    # feaLabel 是下方继续建树时剩余的 feature 的 index
    classList = [example[-1] for example in dataSet]    #  每组数据的实际标签
    #print("classList:",classList)
    '''判断是否属于 2 个终止类型'''
    '''1  全属一个类'''
    if len(classList) == classList.count(classList[0]):
        return classList[0]
    '''2  特征属性已经用完'''
    if len(features) == 0:
        majorClass = majorityCnt(classList)
        return majorClass
    '''继续划分'''
    features_num = [i[1] for i in features]
    temp = chooseBestFeature(dataSet, features_num)
    #print("temp:", temp)
    best_feature = temp[0]    #  最优划分特征的下标号
    for i in features:
        if i[1] == best_feature:
            best_feaLabel = i[0]
            features.remove(i)        #  特征属性中删去最优特征
            break
    deci_tree = {}    #  子树的根的 key 是此次划分的最优特征名，value 是再往下递归划分的
子树（也是字典）
    value = temp[1]
```

```
    value_str = str(value)
    subLabel = features[:]
    subset0 = splitdataset0(dataSet, best_feature, value)
    deci_tree[best_feaLabel+"<"+value_str] = createTree(subset0, subLabel)   # key 为 0 的 value 是 split 左边数据继续向下的决策树
    # print(deci_tree)
    subset1 = splitdataset1(dataSet, best_feature, value)
    deci_tree[best_feaLabel+">"+value_str] = createTree(subset1, subLabel)
    # print(deci_tree)
    return deci_tree


def classify(test_instance, features,decision_tree):   #  分类
    #print(features)
    if isinstance(decision_tree, float):
        result = decision_tree
    else:
        first_str0 = list(decision_tree.keys())[0]
        first_str1 = list(decision_tree.keys())[1]
        first_str_label = first_str0[:first_str0.index('<')]     #[:n]  不包括索引 n 的内容
        second_dict0 = decision_tree[first_str0]
        second_dict1 = decision_tree[first_str1]
        features_label = [i[0] for i in features]
        testvalue = float(test_instance[features_label.index(first_str_label)])
        value = float(first_str0[first_str0.index('<')+1:])
        if testvalue < value:
            result = classify(test_instance, features, second_dict0)
        else:
            result = classify(test_instance, features, second_dict1)
    return result


def run_decision_tree():
    # Load data set
    with open("C:\\Users\\19843\\Desktop\\ML exp\\ex6Data.csv") as f:
        next(f, None)
        data = [tuple(line) for line in csv.reader(f, delimiter=",")]
    print("Number of records: %d" % len(data))

    final_accuracy = 0
    for r in range(10):
        # Split training/test sets
        # You need to modify the following code for cross validation.
        K = 10
        training_set = [x for i, x in enumerate(data) if i % K != r]
```

```
        test_set = [x for i, x in enumerate(data) if i % K == r]

        trainList = []
        testList = []
        for sample in training_set:
            temp = []
            for i in range(0, len(sample)):
                temp.append(float(sample[i]))
            trainList.append(temp)

        for sample in test_set:
            temp = []
            for i in range(0, len(sample)):
                temp.append(float(sample[i]))
            testList.append(temp)

        features = [["fixed acidity", 0], ["volatile acidity", 1], ["citric acid", 2],
                    ["residual sugar", 3], ["chlorides", 4], ["free sulfur dioxide", 5], ["total
sulfur dioxide", 6],
                    ["density", 7], ["pH", 8], ["sulphates", 9], ["alcohol", 10]]
        decision_tree = createTree(trainList, features)
        features = [["fixed acidity", 0], ["volatile acidity", 1], ["citric acid", 2],
                    ["residual sugar", 3], ["chlorides", 4], ["free sulfur dioxide", 5], ["total
sulfur dioxide", 6],
                    ["density", 7], ["pH", 8], ["sulphates", 9], ["alcohol", 10]]
        print(decision_tree)

        #  存储树
        jsObj = json.dumps(decision_tree)
        fileObject                    =                    open('C:\\Users\\19843\\Desktop\\ML
exp\\decisionTree'+str(r)+'.json', 'w')
        fileObject.write(jsObj)
        fileObject.close()

        # Classify the test set using the tree we just constructed
        results = []
        for instance in test_set:
            result = classify(instance[:-1], features, decision_tree)
            results.append(result == float(instance[-1]))
        print(results)
        # Accuracy
        accuracy = float(results.count(True)) / float(len(results))
        print("accuracy: %.4f" % accuracy)
        final_accuracy +=accuracy
    final_accuracy = final_accuracy/10
```

```
        print(final_accuracy)

        # Writing results to a file (DO NOT CHANGE)
        f = open(myname + "result.txt", "w")
        f.write("accuracy: %.4f" % accuracy)
        f.close()


if __name__ == "__main__":
        run_decision_tree()
```