基于决策树算法分类葡萄酒种类

1 问题描述

葡萄酒是一种广受人们欢迎的含酒精饮品，按照其制作的原材料与制作工艺的不同，可以简单的将其区分为红葡萄酒与白葡萄酒两类。由于葡萄酒的不同风味的不同，不同类别的葡萄酒也适用于不同的场合之中。同时，葡萄酒的特有的风味来源于其中含有的各类化学物质与其物理的属性，而制作原材料的不同选择决定了葡萄酒的理化性质。从而，通过分析葡萄酒中所含有的不同分为物质的含量与其物理特性，我们能够区分葡萄酒的种类，从产品的理化参数，反推其制作的原材料的种类。

因此，在本文中，基于两个葡萄牙北部的葡萄酒质量数据集，我们利用决策树算法，根据不同种类葡萄酒之间的理化特点，对葡萄酒类型进行分类，并检验不同特征选择的条件下模型的预测准确率。

数据集链接：https://archive.ics.uci.edu/ml/datasets/Wine+Quality (链接到外部网站。)

2 算法策略

2.1 信息熵、信息增益与信息增益率

2.1.1 信息熵

在信息论中，熵被用来度量信息量，熵越大，所含的有用信息越多，其不确定性就越大；而熵越小，有用信息越少，确定性越大。在决策树中，用熵来表示数据集的复杂度，如果某个数据集中只有一个类别，其确定性最高，熵为0；反之，熵越大，越不确定，表示数据集中的分类越多样。熵的计算公式如下所示：

$$\text{Entropy } = -\sum_{i=1}^{n} p\left(x_i\right) * \log_2 p\left(x_i\right)$$

其中，$p(x_i)$为类别x_i出现的概率，n为类别的数量。

对于随机变量(X, Y)，对于随机变量Y在随机变量X给定的条件下，产生的**条件熵**为：

$$E(Y \mid X)=\sum_{i=1}^{n} p_i E\left(Y \mid X=x_i\right)$$

其中，$p_i=P\left(X=x_i\right), \quad i=1,2, \cdots, n.$

2.1.2 信息增益

信息增益表示在已知特征X的条件下，类别Y的信息不确定度减少的程度。即对特征A对训练集D的信息增益定义为集合D的熵$E(D)$与给定A条件下D的条件熵$E(D \mid A)$之差。信息增益公式如下所示：

$$Gain(D, A)=E(D)-E(D \mid A)$$

假设训练集D中存在有K个类，标记为C_i，$i=1,2, \ldots k$，$|C_i|$，为类别C_i中样本的个数。故集合D的熵$E(D)$表示为：

$$E(D)=-\sum_{i=1}^{k} \frac{|C_i|}{|D|} \log_2 \frac{|C_i|}{|D|}$$

设特征A存在有n个不同的值，将集合D划分为n个子集$D_1, D_2, \cdots, D_n$，$|D_i|$表示为其样本数量。记D_i中属于类别C_j的集合为D_{ij}，$|D_{ij}|$表示为其样本数量。故给定A条件下D的条件熵$E(D \mid A)$为：

$$E(D \mid A)=\sum_{i=1}^{n} \frac{|D_i|}{|D|} E\left(D_i\right)=-\sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{J=1}^{K} \frac{|D_{ij}|}{|D_i|} \log_2 \frac{|D_{ij}|}{|D_i|}$$

故信息增益为：

$$Gain(D, A)=-\sum_{i=1}^{k} \frac{|C_i|}{|D|} \log_2 \frac{|C_i|}{|D|}+\sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{J=1}^{K} \frac{|D_{ij}|}{|D_i|} \log_2 \frac{|D_{ij}|}{|D_i|}$$

2.1.3 信息增益率

信息增益率为公式表达为：

$$\text{Gain\_ ratio}(D, A)=\frac{\text{Gain}(D, A)}{\text{IV}(A)}$$

对于给定的特征A，定义其固有值为数据集D按照特征A的i特征值划分产生的不确定性。固有值公式表达为：

$$IV(A)=-\sum_{v=1}^{n} \frac{|D^n|}{|D|} \log_2 \frac{|D^n|}{|D|}$$

一般的，特征的值的种类越多，固有值越大，说明该属性的的不确定性也越大。一般的，增益率可以随着特征熵的增大而减小该特征的重要性。故该准则对特征值种类少的特征有所偏好。

2.2 数据集中的连续变量值处理

对于连续值采用二分法的方式来划分连续特征数据。对于特征A的n个不同的特征值，将其升序排列得到$\left\{A^1, A^2, \ldots, A^n\right\}$，选择相邻特征值的中位数$\frac{A^i+A^{i+1}}{2}$作为划分标准$t$，能够得到包含$n-1$个划分点的集合$T_A=\left\{\frac{A^i+A^{i+1}}{2} \mid 1 \leq i \leq n-1\right\}$。

从集合T_A中选择划分点后，根据是否$< t$能够将特征A的特征值划分为两类，从而可以计算在每一个划分点下的信息增益率。最后，根据信息增益率的大小选择最优划分点，作为当前节点最优的划分特征。计算公式如下所示：

$$\operatorname{Gain}(D, A)=\max _{t \in T_a} \operatorname{Gain}(D, A, t)=\max _{t \in T_A}\left[\operatorname{Ent}(D)-\sum_{\lambda \in\{1,0\}} \frac{\left|D_t^\lambda\right|}{|D|} \operatorname{Ent}\left(D_t^\lambda\right)\right]$$

2.3 决策树算法伪代码

选择信息增益率作为划分属性标准，决策树构建的伪代码如下所示：

```
# 决策树伪代码
Input: 数据集 data
       特征集 label

Do: Function build_tree(data, label):
            产生节点node
            if data中样本属于同一类别i：
                将node标记为i类叶的节点；
                return
            end if
            if label为空 or data在label上的取值相同：
                将node标记为i类叶的节点；
                标记其类别为最多的类；
                return
            end if
            通过计算信息增益率，从data中选择最优的属性A*；
            for(A*_i in A* )do:
                对node生成分支；D_i为选择A*_i时生成的样本子集；
                if D_i为空：
                    标记分支节点为叶节点；
                    标记其类别为最多的类；
                    return
                else:
                    build_tree(D_i, label\{A*})
                end if
            end for

Output:tree(root=node)
```

2.4 错误率降低剪枝法(REP)

REP方法是一种后剪枝方法。在该方法中，数据被分为训练集与验证集：训练集用来生成决策树，验证集用来评估这个决策树的精度。伪代码实现如下：

```
Input: tree

Do:Function rep(tree):
        选定tree的一个叶节点root;
        del root为根的子树;
        使得root为叶节点;
        赋予root相关的训练数据的最常类别
        if acc(tree_before)<=acc(tree_after):
                del root
        end if

Output: tree
```

反复进行上面的操作，从底向上的处理结点，删去训练集中的虚假特征，降低模型过拟合的风险，提高模型的分类准确度与精度。

3 代码实现

3.1 数据集导入与训练集、测试集划分

利用$pandas$库中的文件操作函数，读入数据集。之后，以80%训练集、20%测试集的形式随机从读取的数据集中随机划分，得到实验中使用的训练集与测试集。函数参数get_feature为选择的特征参数。

```
# 导入数据集并拼接为一个data_frame，颜色标签：red=1，white=0，并随即生成训练集与测试集
def load_data(df1, df2, get_feature):
    df_red = pd.read_csv(df1, sep=";")
    df_red.columns = ["fixed acidity", "volatile acidity", "citric acid",
"residual sugar", "chlorides",
                      "free sulfur dioxide", "total sulfur dioxide", "density",
"pH", "sulphates", "alcohol", "quality"]
    df_red["type"] = 1
    df_red = df_red[get_feature]

    df_white = pd.read_csv(df2, sep=";")
    df_white.columns = ["fixed acidity", "volatile acidity", "citric acid",
"residual sugar", "chlorides",
                        "free sulfur dioxide", "total sulfur dioxide",
"density", "pH", "sulphates", "alcohol",
                        "quality"]
    df_white["type"] = 0
    df_white = df_white[get_feature]

    df_red = np.array(df_red)
    df_white = np.array(df_white)

    df_red_train = df_red[:int(df_red.shape[0] * 0.8), :]
    df_red_test = df_red[int(df_red.shape[0] * 0.8) + 1:-1, :]
    df_white_train = df_white[:int(df_white.shape[0] * 0.8), :]
    df_white_test = df_white[int(df_white.shape[0] * 0.8) + 1:-1, :]

    train_set = np.concatenate((df_red_train, df_white_train))
    test_set = np.concatenate((df_red_test, df_white_test))
    train_set = train_set.tolist()
    test_set = test_set.tolist()
    value_list = get_feature[:-1]
```

```
    return train_set, test_set, value_list
```

3.2 熵的计算

根据2.1.1中的公式，实现计算数据集信息熵的函数。如下所示：

```
# 定义信息熵函数
def entropy(data):
    label_list = {}
    for value in data:
        if value[-1] in label_list:
            label_list[value[-1]] += 1
        else:
            label_list[value[-1]] = 1
    info_en = 0
    for label in label_list:
        info_en -= label_list[label] / len(data) * math.log2((label_list[label]
/ len(data)))
    return info_en
```

3.3 决策树实现

根据2.1.3中的信息增益率的公式推导、2.2中数据集的连续变量值处理方法与2.3中的决策树构建的伪代码，将上述过程封装为decision_tree类。代码实现如下所示：

```
class decision_tree:
    def __init__(self, data, label):
        self.data = data
        self.label = label
        self.tree = {}

    # 选定特征划分数据集。参数：axis:划分的特征；value：特征值；data_relationship:value与
两侧的关系，1">=" or 0 "<"
    def split_data(self, data, axis, value, data_relationship):
        new_data = []
        for item in data:
            if data_relationship == 1:
                if item[axis] >= value:
                    temp = item[:axis]
                    temp.extend(item[axis + 1:])
                    new_data.append(temp)
            if data_relationship == 0:
                if item[axis] < value:
                    temp = item[:axis]
                    temp.extend(item[axis + 1:])
                    new_data.append(temp)
        return new_data

    # 根据使用连续变量处理方法计算特征的信息增益率，选择最优的划分特征与划分值
    def find_best_feature(self, data, label):
        base_entropy = entropy(data)
        base_gain_rate = 0.0
        best_feature_id = -1
        # split_dict用于存放连续性变量最佳划分点的具体值
        split_dict = {}
        for i in range(len(data[0]) - 1):
```

```
            feature_value = [item[i] for item in data]
            best_split = -1
            sorted_feature_value = sorted(feature_value)
            split_list = []
            for j in range(len(feature_value) - 1):
                split_list.append((sorted_feature_value[j] +
sorted_feature_value[j + 1]) / 2.0)
            # 便利划分点，计算信息熵
            # 计算信息增益率 = 信息增益 / 该划分方式的分裂信息
            for j in range(len(split_list)):
                temp_entropy = 0.0
                value = split_list[j]
                l_data = self.split_data(data, i, value, 0)
                s_data = self.split_data(data, i, value, 1)
                p_l = len(l_data) / float(len(data))
                p_s = len(s_data) / float(len(data))
                if p_l != 0 and p_s != 0:
                    temp_entropy += p_l * entropy(l_data) + p_s *
entropy(s_data)
                    # 计算该划分方式的分裂信息
                    infor_split = -p_l * math.log(p_l, 2) - p_s * math.log(p_s,
2)
                    gain_rate = float(base_entropy - temp_entropy) / infor_split
                    if gain_rate > base_gain_rate:
                        base_gain_rate = gain_rate
                        best_split = j
                        best_feature_id = i
            split_dict[label[i]] = split_list[best_split]
        best_feature_value = split_dict[label[best_feature_id]]
        return best_feature_id, best_feature_value

    # 计算类别的出现次数，并返回出现最多的类别标签
    def label_max(self, label):
        label_item = dict([(label.count(i), i) for i in label])
        return label_item[max(label_item.keys())]

    def build_tree(self, temp_data, temp_label):
        cate_list = [item[-1] for item in temp_data]
        # 如果分类类别完全相同，则结束
        if cate_list.count(cate_list[0]) == len(cate_list):
            return cate_list[0]
        # 遍历完所有特征，返回出现最多的类别标签
        if len(temp_data[0]) == 1:
            return self.label_max(cate_list)
        best_id, best_value = self.find_best_feature(temp_data, temp_label)
        if best_id == -1:
            return self.label_max(cate_list)
        # 对于连续性特征，不删除特征并构建大于/小于两条子树
        best_feature_label = temp_label[best_id] + "<" + str(best_value)
        tree = {best_feature_label: {}}
        sub_label = temp_label[:]
        value_sub_left = "True"
        tree[best_feature_label][value_sub_left] =
self.build_tree(self.split_data(temp_data, best_id, best_value, 0),
                                                                    sub_label)
        value_sub_right = "False"
        tree[best_feature_label][value_sub_right] =
self.build_tree(self.split_data(temp_data, best_id, best_value, 1),
```

```
                                                                sub_label)
        return tree

    def train(self):
        label = copy.deepcopy(self.label)
        self.tree = self.build_tree(self.data, label)
        return self.tree
```

3.4 模型输出与性能检验

在训练完决策树模型后，检验模型的输出与模型分类准确性。

```
# 测试模型，返回分类结果
def classify(result_tree, label, test):
    root = list(result_tree.keys())[0]
    symbol_id = str(root).find("<")
    root_label = str(root)[:symbol_id]
    next_dict = result_tree[root]
    feature_id = label.index(root_label)
    class_label = None
    for key in next_dict.keys():
        temp_value = float(str(root)[symbol_id + 1:])
        if test[feature_id] < temp_value:
            if type(next_dict["True"]).__name__ == 'dict':
                class_label = classify(next_dict["True"], label, test)
            else:
                class_label = next_dict["True"]
        else:
            if type(next_dict["False"]).__name__ == 'dict':
                class_label = classify(next_dict["False"], label, test)
            else:
                class_label = next_dict["False"]
    return class_label

# 测试决策树正确率
def testing(tree, label, test):
    error = 0.0
    for k in range(len(test)):
        temp = classify(tree, label, test[k])
        if int(temp) != test[k][-1]:
            error += 1
    return float(error)
```

3.5 错误率降低剪枝法(REP)

根据2.4中的推导与伪代码，实现REP后剪枝法。

```
# 测试节点正确率
def testing_major(major, test):
    error = 0.0
    for i in range(len(test)):
        if int(major) != test[i][-1]:
            error += 1
    return float(error)
```

```
# 为提升模型的泛化能力，进行后剪枝操作
def post_cut_leaf(tree, train, test, label):
    root = list(tree.keys())[0]
    next_dict = tree[root]
    class_list = [item[-1] for item in train]
    feature_key = re.compile("(.+<)").search(root).group()[:-1]
    feature_value = float(re.compile("(<.+)").search(root).group()[1:])
    label_id = label.index(feature_key)
    temp_label = copy.deepcopy(label)
    for key in next_dict.keys():
        if type(next_dict[key]).__name__ == "dict":
            if key == 'True':
                sub_data = st_data(train, label_id, feature_value, '0')
                sub_test = st_data(test, label_id, feature_value, '0')
            else:
                sub_data = st_data(train, label_id, feature_value, '1')
                sub_test = st_data(test, label_id, feature_value, '1')
        if len(sub_test) > 0:
            tree[root][key] = post_cut_leaf(next_dict[key], sub_data, sub_test,
copy.deepcopy(label))
    if testing(tree, temp_label, test) <= testing_major(max(class_list,
key=class_list.count), test):
        print(tree)
        return tree
    return max(class_list, key=class_list.count)
```

4 模型表现

在使用决策树对葡萄酒数据集进行分类的任务中，我们选择了固定酸度，挥发性酸度，柠檬酸，残糖，氯化物，游离二氧化硫，总二氧化硫，密度，pH，硫酸盐含量与酒精含量这11种理化参数作为作为可选的特征数据，以红、白两种葡萄酒类型作为分类标准，以（0，1）作为分类标签，希望通过以上的理化特征，使用构建完成的决策树实现葡萄酒种类的区分。

4.1 决策树的实现

在实验检验的过程中，我们选择了固定酸度，挥发性酸度，柠檬酸，残糖这四种理化参数作为实验参数，放入决策树模型进行训练分类，得到决策树结果（以字典结构输出），如下所示：

```
# 选择固定酸度，挥发性酸度，柠檬酸，残糖这四种理化参数作为实验参数，放入决策树模型后的输出结果
tree = {'volatile acidity<0.4875': {'True': {'fixed acidity<9.850000000000001':
{'True': {'fixed acidity<0.005': {'True': {'fixed acidity<4.300000000000001':
{'True': 1.0, 'False': 0.0}}, 'False': {'fixed acidity<4.05': {'True': 0.0,
'False': 0.0}}}}, 'False': {'fixed acidity<0.8200000000000001': {'True': {'fixed
acidity<1.25': {'True': 0.0, 'False': 1.0}}, 'False': 0.0}}}}, 'False': {'citric
acid<8.149999999999999': {'True': {'fixed acidity<5.45': {'True': {'fixed
acidity<0.165': {'True': 0.0, 'False': 1.0}}, 'False': {'fixed acidity<0.525':
{'True': 1.0, 'False': 1.0}}}}, 'False': {'fixed acidity<9.9': {'True': {'fixed
acidity<0.13': {'True': 0.0, 'False': 0.0}}, 'False': 1.0}}}}}}
```

4.2 决策树准确率

为确定参数数量与决策树模型准确率之间的关系，考虑到模型的运行时间，我们分别选择了三至八个可选的理化参数，放入决策树模型运行。在模型训练完成之后，输入测试集进行检验，将得到的参数数量与模型准确率之间的关系总结成下表，并绘制关系曲线。

表一 参数数量与决策树模型准确率之间的关系

参数数量	3	4	5	6	7	8
模型准确率	0.756	0.890	0.878	0.891	0.907	0.947

如上表与上图所示，可以得到以下的结论 **(1)** 决策树模型的准确性会随着参数数量的增加而增加。同时，在测试集的检验中准确率呈现递增的趋势，在测试集与训练集上检验的表现没有明显差异，表明模型在可选的参数范围内没有出现过拟合的情况。 **(2)** 在可选的参数范围内，当选择8个模型参数后，决策树模型的准确率为**0.947**，对比二分类模型0.5的准确率基线，可以看出决策树模型能够有效的通过可选的理化参数，实现葡萄酒的分类任务。

4.3 剪枝结果

在4.2中分析完决策树模型准确率与参数的关系后，发现在可选的模型范围内，模型准确率随着参数数量的增加而增加，表明模型理论上没有出现过拟合的情况，故推测是否剪枝对目前决策树的影响不大。在剪枝检验的过程中，我们选择五个可选参数进行实验，检验REP剪枝方法对于决策树模型的影响。

```
# 剪枝前决策树模型输出结果
tree_before_prune = {'chlorides<0.0645': {'True': {'volatile
acidity<1.0074999999999998': {'True': {'fixed acidity<10.75': {'True': {'fixed
acidity<0.035': {'True': {'fixed acidity<1.25': {'True': 0.0, 'False': 1.0}},
'False': {'fixed acidity<2.625': {'True': 0.0, 'False': 0.0}}}}, 'False':
{'fixed acidity<0.41000000000000003': {'True': 0.0, 'False': {'fixed
acidity<1.35': {'True': 0.0, 'False': 1.0}}}}}}, 'False': 1.0}}, 'False':
{'residual sugar<9.25': {'True': {'volatile acidity<0.175': {'True': {'fixed
acidity<8.7': {'True': 0.0, 'False': 1.0}}, 'False': {'volatile acidity<0.8':
{'True': {'fixed acidity<6.45': {'True': 0.0, 'False': 1.0}}, 'False': {'fixed
acidity<8.2': {'True': 0.0, 'False': 1.0}}}}}}, 'False': {'fixed acidity<9.25':
{'True': {'volatile acidity<0.16': {'True': {'fixed acidity<0.5825': {'True':
0.0, 'False': 1.0}}, 'False': {'fixed acidity<0.28500000000000003': {'True':
0.0, 'False': 0.0}}}}, 'False': 1.0}}}}}}
```

```
# 剪枝后决策树模型输出结果
tree_after_prune = {'chlorides<0.0645': {'True': {'volatile
acidity<1.0074999999999998': {'True': {'fixed acidity<10.75': {'True': {'fixed
acidity<0.035': {'True': {'fixed acidity<1.25': {'True': 0.0, 'False': 1.0}},
'False': {'fixed acidity<2.625': {'True': 0.0, 'False': 0.0}}}}, 'False':
{'fixed acidity<0.41000000000000003': {'True': 0.0, 'False': {'fixed
acidity<1.35': {'True': 0.0, 'False': 1.0}}}}}}, 'False': 1.0}}, 'False':
{'residual sugar<9.25': {'True': {'volatile acidity<0.175': {'True': {'fixed
acidity<8.7': {'True': 0.0, 'False': 1.0}}, 'False': {'volatile acidity<0.8':
{'True': {'fixed acidity<6.45': {'True': 0.0, 'False': 1.0}}, 'False': {'fixed
acidity<8.2': {'True': 0.0, 'False': 1.0}}}}}}, 'False': {'fixed acidity<9.25':
{'True': {'volatile acidity<0.16': {'True': {'fixed acidity<0.5825': {'True':
0.0, 'False': 1.0}}, 'False': {'fixed acidity<0.28500000000000003': {'True':
0.0, 'False': 0.0}}}}, 'False': 1.0}}}}}}
```

由上述结果可知，当选择五个可选参数进行实验后，发现是否进行剪枝对决策树输出结果的没有影响。这一结果也与实验前的预期相符合：由于葡萄酒数据集的数据量较大，而模型的参数量较小，过拟合的风险较小。通过4.2中的实验结果可知，在可选的参数范围内，决策树模型的准确性会随着参数数量的增加而增加，模型在已有的参数条件下没有出现过拟合的情况，故是否剪枝对当前任务下的决策树没有影响。

5 总结与展望

在本文中，以两个葡萄酒理化性质数据集为基础，我们利用决策树模型，通过对葡萄酒理化性质的特征参数的运算，对葡萄酒的类型进行分类。通过对模型的实验，可以得出以下结论：

- 决策树模型的准确性会随着参数数量的增加而增加。同时，在测试集的检验中准确率呈现递增的趋势，在测试集与训练集上检验的表现没有明显差异，表明模型在可选的参数范围内没有出现过拟合的情况。
- 在可选的参数范围内，当选择8个模型参数后，决策树模型的准确率为**0.947**，对比二分类模型0.5的准确率基线，可以看出决策树模型能够有效的通过可选的理化参数，实现葡萄酒的分类任务。
- 模型在已有的参数条件下没有出现过拟合的情况，故是否剪枝对当前任务下的决策树没有影响。

参考文献

[1]P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis.
Modeling wine preferences by data mining from physicochemical properties. In Decision Support Systems, Elsevier, 47(4):547-553, 2009.