ID3, C4.5, CART三种决策树实现

实验内容

- **算法原理**
  - 处理数据

    采用python语言编写，用二维列表train_list存储读入的数据，用字典feature_dic存储各个特征的数据，key为特征的下标（0-5），value为特征的所有数据，用字典feature_distinct_value存储各个特征的域，key为特征的下标，value为特征的所有可能取值。

  - ID3

    输入为当前的数据集下标列表train_set，选取的特征feature；首先求出train_set的经验熵，求熵过程为：假设一个数据集m，先求出m中每一种取值的频数fre，可得每种取值在m中的比例分数，再根据求熵公式得出熵；根据特征feature的取值种数x，将train_set划分为x个子数据集sub_data，求出每个sub_data的熵，进而按照公式求出条件熵，最后求得信息增益=经验熵-条件熵，可得当前特征的信息增益。

  - C4.5

    首先按照ID3的方法求得当前特征feature的信息增益，然后求feature的熵（不是经验熵，经验熵是针对label求的），因此，信息增益/feature的熵为信息增益率。

  - CART

    输入为当前的数据集下标列表train_set，选取的特征feature；根据特征feature的取值种数x，将train_set划分为x个子数据集sub_data，得出每个sub_data在train_set的比例，然后针对每个sub_data，求出feature每个取值对应的gini系数，最后，求出label在特征feature下的gini系数。

  - 决策树建立

    声明一个类Decision_tree，所有生成决策树和分类的函数都在这个类中。递归调用建树函数，输入为train_set（当前数据集的下标列表），feature_set（当前特征的下标列表），由于需要不断根据选取的特征划分数据集，如果直接对存储数据的train_list直接划分，一是比较麻烦，而是会影响其他分支的向下划分，因此所有分支共用一个train_list，而划分的是数据集和特征的下标列表，在建树函数中，检查剩余特征，如果为空，直接返回label中的众数取值，否则继续建树，以ID3为例，计算每个特征的信息增益，取最高的特征，对train_set、feature_set进行划分，采用字典存储树的结构，存储当前特征、数据集，并且指向划分的分支，对划分的分支递归调用建树函数，输入sub_train_set、sub_feature_set，最后返回根节点。

  - 分类查询

输入每一行数据集m，递归调用查询函数，如果当前节点不是字典，说明是叶节点，直接返回该值，否则查询当前节点的feature，选择根据数据集m在该特征的取值选择下一分支进行查询，直到到达叶节点，返回结果。

- **伪代码**
  - ID3

```
input: train_set,feature  #当前数据集下标列表（特征集加标签值），选择的特征
output: information_gain #返回信息增益

    label_fre <- get_label_fre(train_set) #统计标签各种取值的频数
    empirical_entropy <- get_entropy(label_fre) #求经验熵

    for items in feature_value: #特征的所有取值集合
        sub_data <- split_data(train_set,feature,items) #求出各个取值的在数据集的下标集合，为下面求条件熵作准备
        item_fre <- get_label_fre(sub_data) #统计子数据集的标签频数
        item_entropy <- get_entropy(item_fre) #求熵
        condition_entropy += (sub_size / total_size) * item_entropy  #求条件熵

    information_gain <- empirical_entropy - condition_entropy #求信息增益
    return information_gain
```

  - C4.5

```
#求信息增益过程同ID3
#下面求特征的熵（展示求频数、求熵过程，ID3中也用到这两个过程）
input: train_set, feature, inf_gain  #当前数据集下标列表（特征集加标签值），选择的特征，信息增益
output: inf_gain_rate #返回信息增益率

    #统计特征中各个取值的频数
    for i in range(train_set_size):
        feature_value <- train_list[train_set[i]][feature] #train_list为数据集，二维结构存储
        feature_fre[feature_value] += 1

    #求熵
    for key,value in feature_fre.items():
        tmp <- float(value) / feature_size #每种取值的比例
        entropy += (-1) * tmp * math.log(tmp,2) #求熵公式

    inf_gain_rate <- inf_gain / entropy #求信息增益率
    return inf_gain_rate
```

  - CART

```
input: train_set,feature  #当前数据集下标列表（特征集加标签值），选择的特征
```

```
output: gini 返回gini指数

    for items in feature_value: #特征的所有取值集合
        sub_data <- split_data(train_set,feature,items) #求出各个取值的在数据集的下标集合，为下面求条件熵作准备
        item_fre <- get_label_fre(sub_data) #统计子数据集的标签频数

        condition_gini <- 1 #当前字数据集的gini指数
        for key,value in label_fre.items():
            condition_gini -=  (value / size) *(value / size)   #计算当前子数据集的gini指数

        gini += (sub_data_size / train_size) *  condition_gini #计算gini指数

    return gini
```

- **关键代码**
  - 处理数据：用feature_dic存储各个特征的所有数据，feature_distinct_value存储各个特征的所有不重复取值（域）。

```
# 遍历每一行输入的数据
for i in range(train_size):
    for j in range(6):
        # 加入数据到对应的特征
        feature_dic[j].append(train_list[i][j])
        if train_list[i][j] not in feature_distinct_value[j]:
            # 加入不同的数据到对应的特征
            feature_distinct_value[j].append(train_list[i][j])
```

  - 建树

    首先判断是否需要结束建树，如果已经选完特征，返回label众数，如果当前数据集只有一种label，直接返回label值。

```
# 求出label的频数
label_fre = self.get_label_fre(train_set)
#如果特征已经选完，则返回label的众数为叶子结果
if not feature_set:
    label_value = max(list(label_fre.items()) ,key = lambda x : x[1] )[0]
    return label_value
#如果当前数据集中只有单一的label值，说明无须划分，返回结果
if len(label_fre) == 1:
    label_value = list(label_fre.keys())
    return label_value[0]
```

以ID3为例，求各个特征的信息增益，选取信息增益最大的特征。

```
information_gain = {}
feature_size = len(feature_set)
train_size = len(train_set)
#求各个特征的信息增益
for i in range(feature_size):
    information_gain[feature_set[i]] = self.information_gain(train_set,feature_set[i])

    #求出信息增益最大的特征
    feature = max ( list(information_gain.items()) , key = lambda x : x[1] )[0]
    #如果信息增益小于0，说明划分没有意义，直接返回label众数结果
    if information_gain[feature] < self.threshold:
        label_value = max(list(label_fre.items()), key=lambda x: x[1])[0]
    return label_value
```

创建字典节点，划分子数据集、特征集。

```
#创建节点
new_node = {}
#存储特征名字
new_node['feature'] = feature
#存储数据集名字
new_node['dataset'] = train_set
sub_data_list = []
sub_data_name = []
data_size = len(self.feature_distinct_value[feature])
#依次根据feature取值划分
for i in range(data_size):
    value = self.feature_distinct_value[feature][i]
    # 存储子数据集名字
    sub_data_name.append(value)
    #划分子数据集
    sub_data = self.split_dataset(train_set,feature,value)
    if len(sub_data) > 0 :
        sub_data_list.append(sub_data)

sub_data_size = len(sub_data_list)
#子特征集下标列表
sub_feature_set = feature_set[:]
sub_feature_set.remove(feature)
```

对每个划分的子数据集递归调用建树函数，最后返回根节点。

```
#对每个子数据集，递归调用建树函数，接受结果作为子节点
for k in range(sub_data_size):
    new_node[sub_data_name[k]] = self.build_ID3_tree(sub_data_list[k],sub_feature_set)
    #返回根节点
    return new_node
```

- ID3

先求出经验熵。

```
empirical_entropy = self.get_entropy(label_fre,size)
```

根据feature不同取值，划分子数据集。

```
#根据特征的不同取值，划分多个子数据集
for i in range(data_size):
    #划分子数据集，是下标集合
    sub_data = self.split_dataset(train_set,feature,self.feature_distinct_value[feature]
[i])
    #如果子数据集存在，就加入到子数据集集合中
    if len(sub_data) > 0:
        sub_data_list.append(sub_data)
```

对每个子数据集，计算对应的熵，累加到条件熵

```
#对每个子数据集进行计算
for i in range(size2):
    size3 = len(sub_data_list[i])
    #得出各种取值的频数
    local_fre = self.get_label_fre(sub_data_list[i])
    #求熵
    local_entropy = self.get_entropy(local_fre,size3)
    #加到条件熵中
    condition_entropy += (float(size3)/size) * local_entropy
```

最后求出信息增益。

```
information_gain = empirical_entropy - condition_entropy
```

- C4.5

  求信息增益过程同ID3.

  求出特征的各种取值的频数，为求熵作准备。

```
#求出特征中各种取值的频数
def get_split_info_fre(self,train_set,feature):
    label_fre = {}
    size = len(train_set)
    #遍历数据集
    for i in range(size):
        #如果key不存在，创建key，value加1
        if self.train_list[train_set[i]][feature] not in label_fre.keys():
            label_fre[self.train_list[train_set[i]][feature]] = 1
        else:
            label_fre[self.train_list[train_set[i]][feature]] += 1
    return label_fre
```

求特征的熵。

```
#计算熵
def get_entropy(self,data_fre,size):
    entropy = 0.0
    for key,value in data_fre.items():  #key为取值名字，value为取值频数
        tmp = float(value) / size  #比例
        entropy += (-1) * tmp * math.log(tmp,2) #求熵公式
    return entropy
```

最后求出信息增益率。

```
information_gain[feature_set[i]] = information_gain[feature_set[i]] / split_encropy
```

- CART

  根据feature不同取值，划分子数据集，求出

```
#根据feature的取值划分子数据集，对每个子数据集求gini
for i in range(data_size):
    #划分子数据集
    sub_data = self.split_dataset(train_set,feature,self.feature_distinct_value[feature]
[i])
```

对每个子数据集，求出其中各个取值的频数，进而利用求gini指数公式求出条件指数，并且按照总的求gini公式累加到gini。

```
    if len(sub_data) > 0:
        size = len(sub_data)
        #求各个取值的频数
        label_fre = self.get_label_fre(sub_data)
        #求该条件下的gini指数
        condition_gini = self.get_condition_gini(label_fre,size)
        #求gini指数公式
        gini += (float(size)/train_size) * condition_gini
```

get_condition_gini如下：

```
def get_condition_gini(self,label_fre,size):
    condition_gini = 1.0
    for key,value in label_fre.items():
        #求gini公式
        condition_gini -= (float(value) / size)*(float(value) / size)
    return condition_gini
```

- **创新点&优化**
  - 划分数据集不直接划分存储数据的列表，而是划分下标集合，所有节点公用一个存储数据的结构，节省存储空间。
  - 预剪枝
    - 提前设立一个阈值，判断当前节点的数据集数，如果小于该阈值，则停止划分，将该节点作为叶子节点。
    - 判断当前节点划分的性能与不划分的性能比较，如果不划分性能更好，则将当前节点作为叶子，返回label众数值。

实验结果及分析

- **实验结果展示**
  - 按顺序比例划分不同的训练集和验证集结果

比例（训练集：验证集）	ID3	C4.5	CART
9:1	0.9567901234567902	0.9567901234567902	0.9444444444444444
8:2	0.9537037037037037	0.9444444444444444	0.9537037037037037
7:3	0.9506172839506173	0.948559670781893	0.9506172839506173
6:4	0.9428129829984544	0.9520865533230294	0.9443585780525502
5:5	0.9468479604449939	0.9456118665018541	0.9468479604449939
4:6	0.933058702368692	0.933058702368692	0.933058702368692
3:7	0.9090106007067138	0.9125441696113075	0.9090106007067138
2:8	0.8879443585780525	0.8887171561051005	0.8879443585780525
1:9	0.8921703296703297	0.8942307692307693	0.8921703296703297

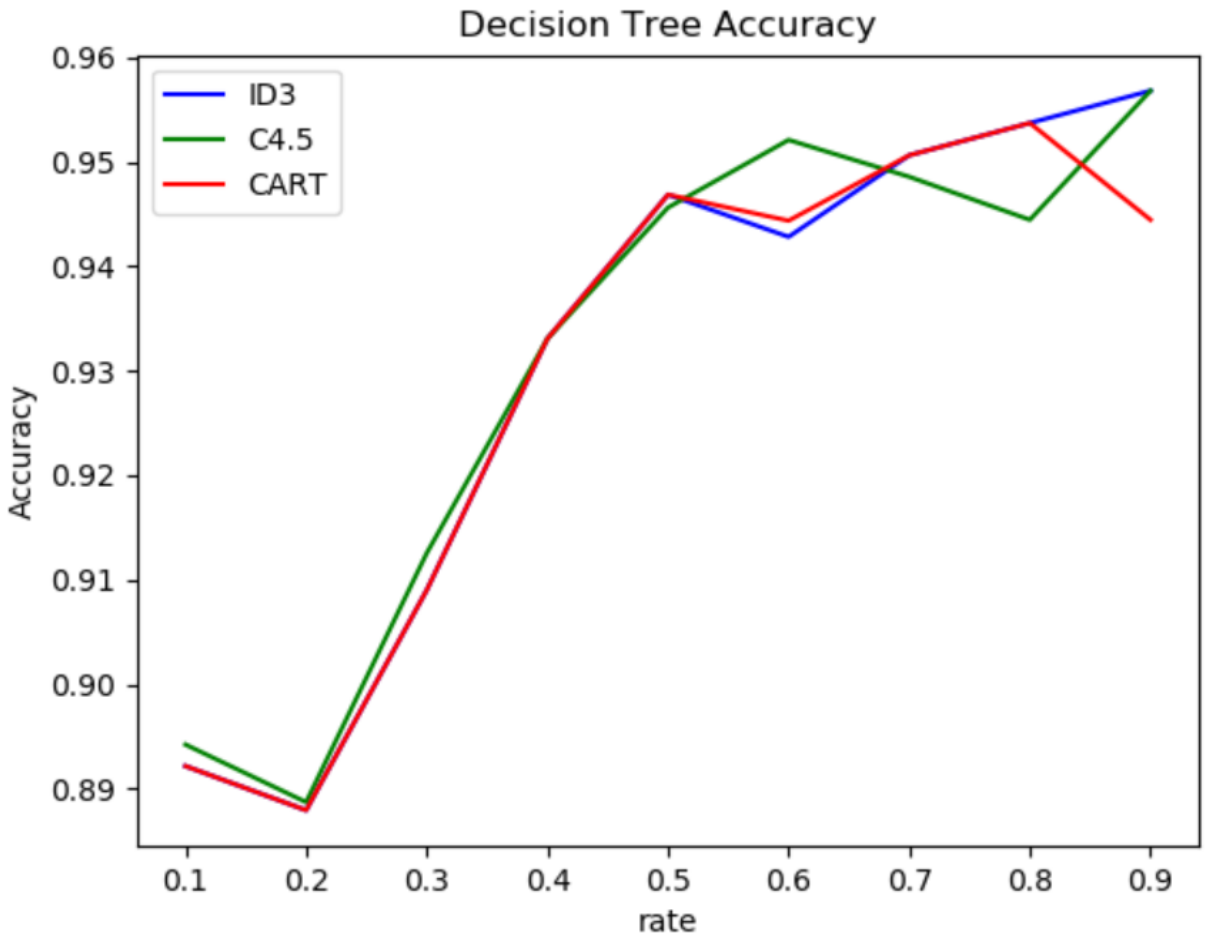
- 按10折交叉验证，将数据集分成10份，9份用于训练，1份用于验证，分别取其中一份为验证集，并测试准确率，最后取平均值，得出准确率。

  由表可得平均准确率 C4.5 > ID3 > CART

验证集序号	ID3	C4.5	CART
1	0.9440993788819876	0.9440993788819876	0.9440993788819876
2	0.9192546583850931	0.9440993788819876	0.9192546583850931
3	0.968944099378882	0.968944099378882	0.968944099378882
4	0.9565217391304348	0.9503105590062112	0.9565217391304348
5	0.9503105590062112	0.9503105590062112	0.9503105590062112
6	0.968944099378882	0.968944099378882	0.968944099378882
7	0.9192546583850931	0.9130434782608695	0.9130434782608695
8	0.968944099378882	0.968944099378882	0.968944099378882
9	0.9627329192546584	0.9627329192546584	0.9627329192546584
10	0.9565217391304348	0.9565217391304348	0.9565217391304348
average	0.9515527950310559	0.9527950310559008	0.9509316770186336

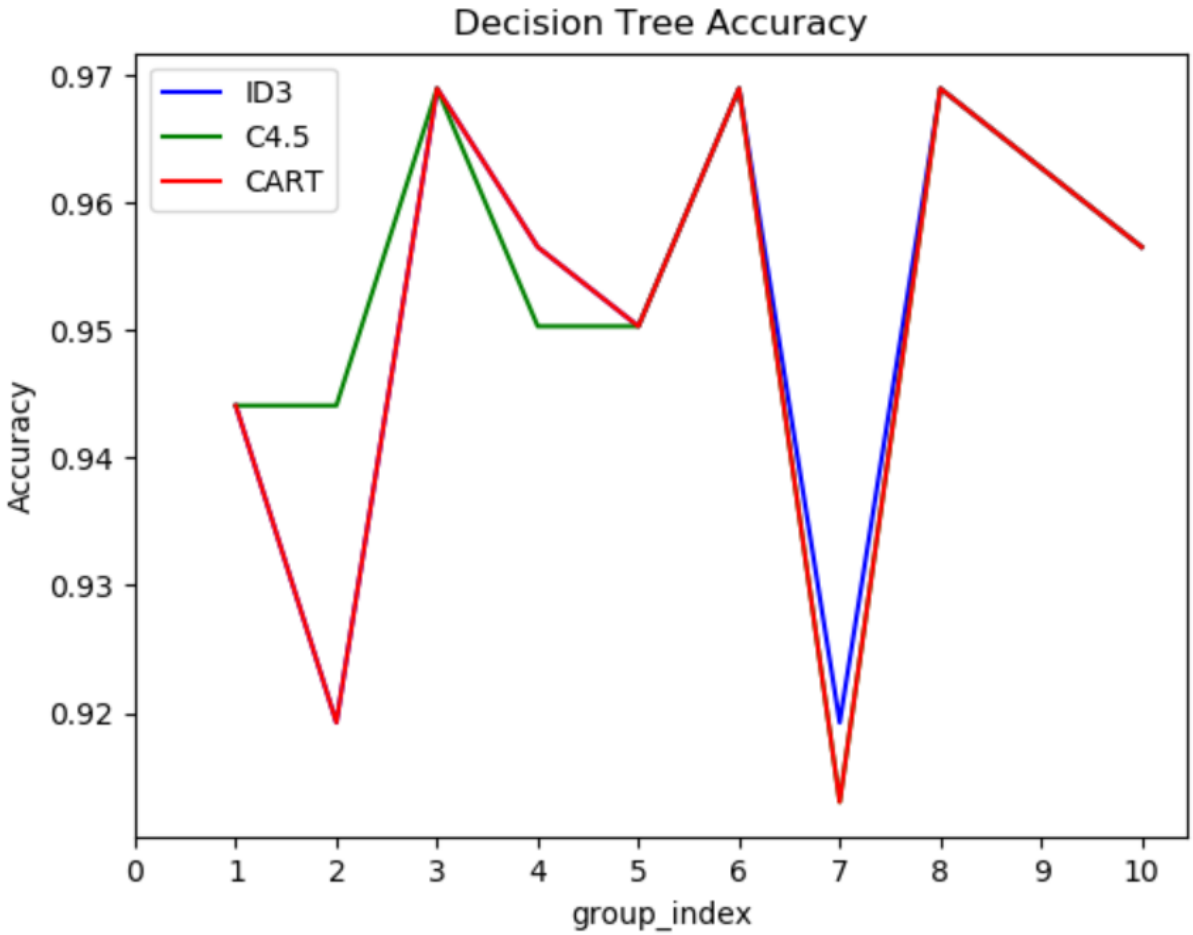
- **评测指标展示**

- 按顺序比例划分不同的训练集和验证集准确率分析



由图可知，训练集越多，决策树的模型训练得越充分，泛化性能越好，预测的效果越好；同时训练集比例rate在0.1-0.5时，三种决策树的准确率接近，ID3在rate=0.6有下降趋势，但随着rate继续升高，准确率上升，在rate=0.9最高；C4.5在rate为0.6-0.8范围内下降，在rate=0.9时上升，达到最高点；而CART在rate=0.8时达到最高，之后下降。ID3、C4.5最高准确率比CART高。

- 按10折交叉验证准确率分析



根据分析，CART的准确率变化起伏与ID3几乎一致，只有验证集取第8组时，两者有差异，同时变化起伏较大，相比之下C4.5准确率变化起伏较小，除了group_index在1-5时，其余情况准确率与ID3吻合，总体上说，C4.5准确率比ID3好，ID3准确率比CART好。

拓展思考

1. 决策树有哪些避免过拟合的方法?
    - 前剪枝：在建树过程中，对当前结点首先进行判断，将划分后的性能与没有划分的相比，如果泛化性能没有提高，则不划分，并且将结点作为叶结点。
    - 后剪枝：建树完毕后，自下向上遍历非叶结点，判断将当前的子树替代为叶结点是否带来性能提升，如果有，则将子树替代为叶结点。
    - 合理选择训练数据，有针对性选择典型的训练集训练模型。
    - 确保构建树的过程正确。
    - 设定一个高度，规定建树达到该高度就停止。
    - 设定一个阈值，判断到达每个结点的个数是否小于阈值，小于则停止建树，设为叶结点。
2. C4.5相比于ID3的优点是什么，C4.5又可能有什么缺点?
    - ID3倾向于熵减少程度最大的特征划分数据，是贪心策略，因此会倾向于选择取值比较多的特征，一旦取值过多，则按这个特征划分就没有太大意义，同时ID3只能处理离散型属性；
    - C4.5改进的优点：

        - 而C4.5引入信息增益比，可以避免倾向于取值比较多的特征，
        - 同时可以处理连续型特征，准确率高。
        - 可以在树构造过程中进行剪枝。
        - 处理不完整数据。
    - 缺点：
        - 算法相对于ID3比较低效，在建树过程中，需要不断对数据集进行处理，开销比较大，导致算法比较低效。
        - 能够处理的数据集有限，只能适合于驻留在内存的数据集，训练集过大，内存不能存储时，无法继续运行。

3. 如何用决策树来进行特征选择（判断特征的重要性）？
    - ID3：利用信息增益衡量特征的重要程度，信息增益越大，特征越重要，信息增益是通过经验熵 - 条件熵，信息熵是用于衡量变量不确定性，熵越小，不确定性越小，通过对特征的划分，计算出的条件熵越小，证明不确定性越小，在经验熵指定的情况下，从而信息增益越大，表示通过对该特征的划分，能够获得的增加的信息量更多，能够更加确定分类。
    - CART：利用信息增益率衡量特征重要性，信息增益率越高，特征越重要由于ID3倾向于取值比较多的特征，因为特征取值多，那么划分的分类就越多，从而求得的条件熵更低，信息增益更高，数据集确定性更高。为了避免这种情况，将求得的信息增益 / 所选特征的熵求得信息增益率，避免趋向取值极端情况。
    - CART：利用基尼系数确定特征重要性，基尼系数越小，特征越重要。基尼系数与熵比较相似的，都是描述样本的纯度，在该算法中，基尼系数通过计算所选的数据集中每一种取值的概率*它被分错的概率得到，基尼系数越大，不确定性越高。熵与基尼系数相比，熵达到峰值过程慢点，因此熵的惩罚力度大一点。