中山大学数据科学与计算机学院
移动信息工程专业—人工智能
本科生实验报告

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	1506	专业（方向）	移动信息工程（互联网）
学号	15352116	姓名	洪子淇

1 实验题目

1. 理解 ID3，C4.5，GINI 决策指标的不同以及相关含义；
2. 实现 ID3，C4.5，CART 三种决策树；
3. 在实验基础上对决策树进行优化。

2 实验内容

2.1 算法原理

2.1.1 决策树

定义

决策树是一种类似于流程图的树结构。一般包括根节点，内部节点以及叶子节点。叶子节点对应于决策结果，其他每个节点对应每个属性下的测试，节点下的分支对应测试的输出。测试样本序列从上往下对应减少上层节点的属性，比如根节点包括了全部属性样本集合，根节点的下一层的每一个子节点的测试序列将不会存在根节点的属性。

当决策树建好之后，根节点到每个叶子节点对应了一个判定测试序列，当输入未给定结果的测试样本，会沿着决策树的判定测试序列找到叶子结点，并且返回测试结果。

信息熵

香农将统计物理学的熵引申到信道通信中，得到信息熵公式为

$$S(p_1,p_2,...,p_3)=-k\sum_{i=1}^{N}p_i log_2 p_i \quad k>0,\ 0\le p_i\le 1$$

当且仅当所有的信息出现的概率都相同时，信息熵取得最大值 $k\log_2 N$. 证明如下（拉格朗日乘数法）：

$$\varphi(\boldsymbol{p})=\sum_{i=1}^{N}p_i-1=0$$

$$f(\boldsymbol{p})=-k\sum_{i=1}^{N}p_i log_2 p_i$$

$$F(\boldsymbol{p})=f(\boldsymbol{p})-\lambda\varphi(\boldsymbol{p})$$

$$
\begin{aligned}
&\text{令 } \frac{\partial F(\boldsymbol{p})}{\partial p_i} = -k(log_2 p_i + \frac{1}{\ln 2}) + \lambda = 0 \\
&\therefore p_i = p_j = \frac{1}{N} \quad \forall i, j \in \{1, 2, \cdots, N\} \\
&\therefore \boldsymbol{p}^* = \left(\frac{1}{N}, \cdots, \frac{1}{N}\right) \text{是极值点} \\
&\text{令 } \boldsymbol{p_0} = (1, 0, \cdots, 0)\text{，有 } f(\boldsymbol{p}^*) = k \log_2 N > f(\boldsymbol{p_0}) = 0 \\
&\therefore \boldsymbol{p}^* \text{是极大值点} \\
&\text{又 } \frac{\partial^2 f(\boldsymbol{p})}{\partial p_i \partial p_j} = 0 \quad (i \neq j), \qquad \frac{\partial^2 f(\boldsymbol{p})}{\partial p_i^2} < 0 \\
&\therefore f(\boldsymbol{p}) \text{是凹函数 (Concave Function)，} \boldsymbol{p}^* \text{是最大值点}
\end{aligned}
$$

当 $p_i = 1, p_j = 0, i \neq j, i, j \in \{1, 2, \cdots, N\}$ 时，信息熵取得最小值。证明如下：

设 $p_a + p_b = c$，保持其他 $p_j (p_j \neq p_a, p_j \neq p_b)$ 不变。

令 $g(p_a) = p_a \log_2 p_a + p_b \log_2 p_b = p_a \log_2 p_a + (c - p_a) \log_2 (c - p_a)$. 则：

$$
\begin{aligned}
&\frac{\mathrm{d}g(p_a)}{\mathrm{d}p_a} = \log_2 p_a + \frac{1}{\ln 2} - \log_2 (c - p_a) - \frac{1}{\ln 2} = \log_2 \frac{p_a}{c - p_a} \\
&S(\boldsymbol{p}) = -kg(p_a, p_b) + C
\end{aligned}
$$

当 $p_a \in \left(0, \frac{c}{2}\right)$ 时，$g(p_a)$ 单调递减，$S(\boldsymbol{p})$ 单调递增；

当 $p_a \in \left(\frac{c}{2}, c\right)$ 时，$g(p_a)$ 单调递增，$S(\boldsymbol{p})$ 单调递减。

$\therefore g(p_a)_{\min} = g(0) = g(c)$

所以 $\boldsymbol{p_0} = (1, 0, \cdots, 0)$ 是某一个最小值点，因为对任意一个 $\boldsymbol{q}$，$\boldsymbol{q}$ 有大于一个非零分量，都可以通过固定 $\boldsymbol{p_0}$ 其他 $N - 2$ 个分量不变，贪心地（每次最优）调整其中两个分量的取值，重复这样操作若干次得到 $\boldsymbol{q}$，而每次操作后，$S(\boldsymbol{p})$ 的取值都会增大。

在二维信息系统中，显然当两个信息出现的概率 p_i 相等时，此时系统信息纯度最低，判别系统信息最困难。综上，信息熵用于度量样本信息的纯度。当信息熵的值越小，表示了样本集合的纯度越高。

ID3 决策指标

采用了信息熵的概念，首先是计算数据值 D 的经验熵，以及特征 A 对数据集 D 的条件熵，经验熵和条件熵的差值作为信息增益，选择了信息增益最大的特征作为决策点。根据信息增益的定义，当条件熵越小，即分裂后的子节点越纯，分类效果越好，选择分类效果越好的特征作为分裂特征。

数据集 D 经验熵：

$$H(D) = -\sum_{d \in D} p(d) \log p(d)$$

特征 A 对数据集 D 的条件熵

$$H(D|A) = \sum_{a \in A} p(a) H(D|A = a)$$

信息增益

$$g(D, A) = H(D) - H(D|A)$$

C4.5 决策指标

对于 ID3 算法来说，当某一个特征样本值多的时候，会导致极端深度为 2 的树的出现，举个例子：对训练样本编号（1~N），并把编号也作为一个分类依据的话，那么编号这个属性就会产生 N 个子节点，每

个子节点对应了该编号样本的结果，这些子节点的纯度已经是最大的了（条件熵为 0），也就是信息增益最大，那么决策树就会选择该属性作为根节点，并且划分出 N 个叶子节点，显然这样的决策树不具备泛化能力。ID3 偏好选择多子节点的特征属性，为了减少这种偏好带来的不利影响，C4.5 决策树将每个特征条件下的信息增益除以该特征的熵，对于某个特征的子节点越多的时候，该特征的熵也会也会越大，改善了 ID3 对于可取值数目多的偏好带来的可能不利影响。所以根据信息增益的比例（增益率）作为决策树分裂依据。

数据集 D 关于特征 A 的值的熵

$$SplitInfo(D,A) = -\sum_{j=1} v\frac{|D_j|}{|D|}\log(\frac{|D_j|}{|D|})$$

信息增益率

$$gRatio(D,A) = \frac{g(D,A)}{SplitInfo(D,A)}$$

启发式的 c4.5 决策指标

由于增益率准则对可取值数目较少的属性有所偏好，启发式的 c4.5 分裂依据是：先从候选特征中找出信息增益高于平均水平的特征，再从中选择增益率最高的。本次实验中评判决策时当剩余特征大于 3 的时候，才会求出每个特征的信息增益并且截断最后三个将剩下的特征的信息增益除以该特征对应的信息熵（即信息增益率），将增益率最大的特征最为分裂依据。当剩余特征小于等于 3 的时候就没有必要截取信息增益较大的属性了，直接计算每个特征下的增益率，将最大的增益率最大的作为分裂依据。

CART 决策指标

cart 决策树与 GINI 系数有关。此前提到信息熵的概念，公式为

$$S(p_1,p_2,...,p_3) = -k\sum_{i=1}^{N} p_i log_2 p_i$$

而基尼系数的公式就是熵 (和信息熵差 $\ln 2$ 倍) 在 $p_i = 1$ 时的一阶泰勒展开

$$p\ln p = p\ln(1+p-1) \sim p(p-1)$$

所以根据信息熵的定义，熵的值越大表示样本纯度越低。所以选择 gini 系数最小的特征作为决策点。

计算特征 A 的条件下，数据集 D 的 Gini 系数

$$gini(D,A) = \sum_{j=1}^{v} p(A_j)\cdot gini(D_j|A=A_j)$$

$$\text{其中：} gini(D_j|A=A_j) = \sum_{i=1}^{n} p_i(1-p_i) = 1-\sum_{i=1}^{n} p_i^2$$

2.1.2 K-折交叉验证

K-折交叉验证的目的是为了评价决策树的泛化能力，进行决策树的选择。优点在于：验证集的样本正负比例比较平均，样本选择比较随机，比较符合现实中较为理想的验证模型。

实现：将样本集合划分为 1 和-1 两类，将两类样本集合随机分成 K 份，然后再将两类样本的 K 份进行交叉组合得出 K 份，其中 K-1 作为训练集，1 份作为验证集，依次轮换训练集和验证集 K 次，评价指标选择信息论中的 Accuracy,recall,percision,F1。

2.1.3 后剪枝

后剪枝主要作用是提高决策树的泛化能力。

实现：后剪枝验证集和训练集：同样将原本的训练集折成 10 份（折取方式参考 K-折交叉验证），选取 3 份作为验证集，7 份作为训练集。（当验证集太小的时候回发生过拟合现象，所以选取了三份作为验证标准）。后剪枝原理：自底向上地对于某个内部节点进行考察，如果它的子节点都是叶子节点的话，考虑将这个内部节点剪枝作为叶子节点，决策结果是叶子节点中 y 值的众数。评判剪枝前后两个树的准确率，如果剪枝后的树的准确率有所提高或不变，那么就进行剪枝，否则就不进行剪枝。

2.1.4 随机森林

随机森林表示了没有关联或者关联度较少的决策树的集合。使用随机森林去预测每个样本的结果，假如有 K 棵树就会得到 K 个结果，对结果采取众数就会综合各个决策树的优点，减少每一棵树的误差。所以在训练上有较好的效果，实现起来也比较简单

实现：根据 K-折交叉验证中将原始训练样本随机生成 K 份，采取一份作为验证集。而随机森林里面具有原始的 c4.5 决策树，以及启发式的 c4.5 的决策树和经过后剪枝的决策树组成。而前面两个都是通过 9-折交叉验证中选取准确值和 F1 值都高于 0.5 的决策树。而后剪枝的决策树是通过构建 N 棵后剪枝的决策树之后再进行过滤，过滤标准也是准确值和 F1 值都高于 0.5。

2.1.5 离散化

由于 id3 决策树对于连续型的特征泛化能力不好，将数据聚类离散化会提高决策树的泛化能力。

2.2 伪代码

Algorithm 1 Importance

function entropy(samples, attribute)
    P = $\{\frac{\|g\|}{\sum\|g\|} \mid g \in samples\backslash attribute\}$ ▷ groupby: $S\backslash A = \{\cdots, \{s|s.A = a_i, s \in S\}, \cdots\}$
    hd = $-\sum_{p\in P} p \log_2 p$
    return hd
end function

function id3(samples, attribute)
    Pa = $\{\frac{\|gba\|}{\sum\|gba\|} \mid gba \in samples\backslash attribute\}$
    Hda = {entropy(gba) | $gba \in samples\backslash attribute$}
    id_3 = entropy(samples) - $\mathrm{Pa}^{\mathbf{T}}\cdot$Hda
    return id_3
end function

function c4_5(samples, attribute)
    return id3(samples, attribute) / entropy(samples, attribute)
end function

function gini(samples)
    P = $\{\frac{\|g\|}{\sum\|g\|} \mid g \in samples\backslash\{1, -1\}\}$
    return 1 - $\sum_{p\in P} p^2$
end function

function cart(samples, attribute)
    Pa = $\{\frac{\|gba\|}{\sum\|gba\|} \mid gba \in samples\backslash attributes\}$
    Gini = {gini(gba) | $gba \in samples\backslash attributes$}
    return - $\mathrm{Pa}^{\mathbf{T}}\cdot$Gini
end function

Algorithm 2 Decision Tree

```
function DecisionTreeLearning(samples, attribute, parentSamples)
    if samples are empty then
        return {'leaf':True, 'y':PluralityValue(parentSamples)}
    end if
    if all samples have the same classification then
        return {'leaf':True, 'y': classification }
    end if
    if attributes are empty then
        return {'leaf':True, 'y': PluralityValue(samples) }
    end if
    A = arg max_{a∈attributes} importance(samples, a)        ▷ IMPORTANCE can be id3, c4_5, cart
    tree = {'leaf':True, 'attr':A, 'child':[]}
    for each value v of A do
        exa = {e.A = v | e ∈ samples}
        tree.child[v] = DecisionTreeLearning(exa, attributes - A, samples)
    end for
    return tree
end function
```

```
function Heuristic_c4_5_Tree(samples, attribute, parentSamples)
    if samples are empty then
        return {'leaf':True, 'y':PluralityValue(parentSamples)}
    end if
    if all samples have the same classification then
        return {'leaf':True, 'y': classification }
    end if
    if attributes are empty then
        return {'leaf':True, 'y': PluralityValue(samples) }
    end if
    if length of attributes > 3 then
        Pid3 = {a:id3(samples, a) | a∈sttributes}
        sort Pid3 descendingly by Pid3.value, drop the last 3 elements
        A = arg max_{a∈Pid3.key} Pid3[a] / entropy(samples,a)
    elseA = arg max_{a∈attributes} c4_5(samples, a)
    end if
    tree = {'leaf':True, 'attr':A, 'child':[]}
    for each value v of A do
        exa = {e.A = v | e ∈ samples}
        tree.child[v] = Heuristic_c4_5_Tree(exa, attributes - A, samples)
    end for
    return tree
end function
```

Algorithm 3 Prune tree

```
function PruneTree(train, attributes)
    _train, _valid = train                    ▷ divide train into train set and validation set
    dcTree = DecisionTreeLearning(_train, attributes)
    PTree = PostPrune(dcTree, dcTree, _valid)
    eva = evaluateTree(PTree, _valid)
    return PTree, eva
end function
```

Algorithm 4 Prune tree

```
1:  function PostPrune(tree, subtree, valid)
2:      if subtree is a leafnode then
3:          return true
4:      end if
5:      flag = True
6:      for node in subtree do
7:          flag &= PostPrune(tree, node, valid)
8:      end for
9:      if flag then                          ▷ all leave nodes means it may be pruned
10:         e1 = evaluate(tree, valid)
11:         make the subtree as leafnode
12:         e1 = evaluate(tree, valid)
13:         if e2 <= e1 then                                          ▷ bad prune
14:             recover the subtree
15:             return false                          ▷ means it is not pruned
16:         else                                                      ▷ good prune
17:             return true                 ▷ means the subtree become a leafnode
18:         end if
19:     end if
20:     return False
21: end function
```

Algorithm 5 K-fold cross validation

```
1:  function GroupValidation(train, k=10)
2:      Divide train in two groups according the label(1, -1) and shuffle each group
3:      Divide into k parts on each group
4:      Cross add each parts on each group
5:      return [K parts]
6:  end function
1:  function CrossValidaton(train, attributes, treefunc)
2:      subforest = []
3:      k parts = GroupValidation(train)
4:      for  loop k times do
5:          validation = ith part                                         ▷ 1 part
6:          train = remaining parts                                       ▷ k-1 part
7:          tree = TreeFunc(train, attributes)
8:                                  ▷ treefunc can be decisionTreeLearning or heuristic_c4_5_Tree
9:          if EvaluateTree(Tree, validation) > threshold then
10:             subforest += tree
11:         end if
12:     end for
13:     return subforest
14: end function
```

2.3 关键代码

分裂决策指标：

```
def entropy(samples, attr='y'):
    gb = samples.groupby(attr).groups.values() #group by 'y' initially
    p = np.array(list(map(len, gb)))
    p = p / sum(p)
    i = (p > 1e-6)
    p[i] = p[i] * np.log2(p[i]) # calculate the entropy when p != 0
    return -sum(p)
def id3(samples, attr):
    gba = samples.groupby(attr).groups.values() #group by attribute
    pa = np.array(list(map(len, gba)))
    pa = pa / sum(pa) # get the probability of eigenvalue
    hda = map(lambda i:entropy(samples.loc[i]), gba)
    hda = np.array(list(hda)) #get the eigenvalue sequence of attribution
    return entropy(samples) - sum(hda * pa) # GDA = HD - HDA
def c4_5(samples, attr):
    splitInfo = entropy(samples, attr)
    return id3(samples, attr) / splitInfo if splitInfo != 0 else 0 # gRatioDA = GDA / SplitInfoDA
def gini(samples):
    gb = samples.groupby('y').groups.values()
    p = np.array(list(map(len, gb)))
    p = p / sum(p)
    return 1 - sum(p * p)
def cart(samples, attr):
    gba = samples.groupby(attr).groups.values()
    p = np.array(list(map(len, gba)))
    p = p / sum(p)
    g = map(lambda i:gini(samples.loc[i]), gba)
    g = np.array(list(g))
    return -sum(p * g)
```

普通决策树（id3, c4.5, gini）：

```
def decisionTreeLearning(samples, attributes, parent_samples=None,importance=c4_5):
    if len(samples)==0: #samples is empty
        return {'leaf':True, 'y':pluralityVal(parent_samples)}
    gb_y = samples.groupby('y')
    if len(gb_y) == 1: #all samples have the same classification
        return {'leaf':True, 'y':list(gb_y.groups.keys())[0]}
    attr = attributes.keys()
    if len(attr) == 0: #attributes is empty
        return {'leaf':True, 'y':pluralityVal(samples)}
    # importance = id3 #choose the determination strategy
    im = list(map(lambda a:importance(samples,a), attr)) #evaluation of each atrribution in strategy
    a = reduce(lambda x,y: x if x[0]>=y[0] else y, zip(im,attr))[1] #determine the next attribution
    ag = range(attributes.pop(a)) #the range of the next attribution
    tree = {'leaf':False, 'attr':a, 'child':[]}
    for v in ag:
        exa = samples[samples[a]==v]
        tree['child'] += [decisionTreeLearning(exa, attributes, samples,importance)]
    return tree
```

启发式的 c4.5 决策树：

边界条件和普通的决策树一致，只是在判别上先判断特征的 id3，挑选 id3 平均水平较高的再计算 c4.5，得到最大的 c4.5 对应的特征作为分裂特征。

```
def heuristic_c4_5Tree(samples, attributes, parent_samples=None,importance=None):
    if len(samples)==0: #samples is empty
        return {'leaf':True, 'y':pluralityVal(parent_samples)}
    gb_y = samples.groupby('y')
    if len(gb_y) == 1: #all samples have the same classification
        return {'leaf':True, 'y':list(gb_y.groups.keys())[0]}
    attr = attributes.keys()
    if len(attr) == 0: #attributes is empty
        return {'leaf':True, 'y':pluralityVal(samples)}
    preid3 = list(map(lambda a:id3(samples,a), attributes))
    if len(attributes) > 3:
        li = sorted(zip(preid3, attributes), key = lambda x:x[0], reverse=True)
        res = zip(map(lambda x: x[0]/entropy(samples,x[1]), li[:-3]), map(lambda x: x[1], li[:-3]))
        # a = sorted(res, key=lambda x:x[0], reverse=True)
    else:
        li = list(map(lambda a:c4_5(samples,a), attr)) #evaluation of each atrribution in strategy
        res = zip(li,attr)
    a = reduce(lambda x,y: x if x[0]>y[0] else y, res)[1] #determine the next attribution
    ag = range(attributes.pop(a)) #the range of the next attribution
    tree = {'leaf':False, 'attr':a, 'child':[]}
    for v in ag:
        exa = samples[samples[a]==v]
        tree['child'] += [heuristic_c4_5Tree(exa, attributes, samples)]
    return tree
```

K-折分组：

```
def groupValidation(train, k=10): #get the validation by separating the train
    gb_id = train.groupby('y').groups.values() #divided in two groups (1,-1)
    L = []
    for g in gb_id:
        n = len(g)
        m = math.ceil(n/k) #divide in k parts
        id_gbid = list(range(n))
        random.shuffle(id_gbid) #scatter in group 1(-1)
        L += [list(map(lambda y:list(map(lambda x:g[x], id_gbid[y:y+m])), map(lambda i:i*m,range(k))))]
```

K-折交叉验证选择较好的树：

```
def crossValidation(train, attributes, treefunc=decisionTreeLearning, choose=c4_5):
    gval = groupValidation(train)
    subforest = []
    global_var[choose] = []   #reserve the check result and use into evaluating the model
    for i in range(len(gval)): #len(gval)=k
        cp_gval = copy.deepcopy(gval)
        _valid = train.loc[cp_gval.pop(i)] #take out validation i
        _std = list(_valid['y'])
        _valid.drop('y', axis=1)
        _train = train.loc[reduce(operator.add, cp_gval)] # 1 for validation, 9 for train
        Tree = treefunc(_train, copy.deepcopy(attributes), importance=choose)
        _res = list(map(lambda x:decide(Tree, x[1]), _valid.iterrows()))
        cv_res = check(_res, _std)
        global_var[choose] += [cv_res]
        if cv_res[0] > 0.5 and cv_res[3] >0.5: # evaluate standard
            subforest += [Tree]
        # print(cv_res[-1])
    return subforest
```

后剪枝：

```
def postPrune(tree, subtree, valid):
    if subtree['leaf']:
        return True
    flag = True
    for node in subtree['next']:
        flag &= postPrune(tree, node, valid) #if prune last time, it can prune this time
    if flag: #prune if raise accuracy else not prune
        prev_score = evaluatePrune(tree, valid)
        y = map(lambda n: n['y'], subtree['next'])
        subtree['leaf'] = True
        _attr = subtree.pop('attr')
        _next = subtree.pop('next')
        subtree['y'] = Counter(y).most_common()[0][0]
        cur_score = evaluatePrune(tree, valid)
        if cur_score < prev_score: #recover the tree
            subtree['leaf'] = False
            subtree.pop('y')
            subtree['attr'] = _attr
            subtree['next'] = _next
            return False
        else:
            return True
    return False
```

后剪枝的验证集采取 K 折的 3 个部分，训练集采取 k-3 个部分，来进行评测一棵后剪枝树的效果主要是，如果验证集过小可能会导致过拟合的现象出现。

```
def pruneTree(train, attributes):
    gval = groupValidation(train)
    _valid = train.loc[reduce(operator.add, gval[:3])] #3 for validation, 7 for train
    _std = list(_valid['y'])
    # _valid.drop('y', axis=1)
    _train = train.loc[reduce(operator.add, gval[3:])]
    dcTree = decisionTreeLearning(_train, copy.deepcopy(attributes))
    postPrune(dcTree, dcTree, _valid)
    _res = list(map(lambda x:decide(dcTree, x[1]), _valid.iterrows()))
    cv_res = check(_res, _std)
    # print(cv_res)
    return (dcTree,cv_res)
```

离散化：

对特征 1 进行离散化

```
def discretization(data):
    col = data[0]
    col[col<=30] = 0
    col[(col>30)&(col<=40)] = 1
    col[col>40] = 2
    data[0] = col
```

2.4 创新点及优化

1. 实现了连续数据的离散化
2. 实现了后剪枝
3. 实现了启发式的 c4.5 决策树
4. 通过 K-折交叉验证选择较好的树（里面有原始的 c4.5 决策树，启发式的 c4.5 决策树，以及采取 3:7(验证集: 训练集) 的比例进行后剪枝的 c4.5 决策）构成随机森林得到决策结果。（为了排版好看我写在上面了，这里只是简单说明一下）

3 实验结果及分析

3.1 实验结果展示示例

小数据集评测：

数据	长鼻子（x）	大耳朵（y）	是否大象（1 or -1）
A1	1	1	1
A2	0	1	-1
A3	1	0	-1
A4	0	0	-1

ID3:

$HD = -\frac{1}{4}\log\frac{1}{4} - \frac{3}{4}\log\frac{1}{4} = 0.8112781$

$H(D|A='x') = \frac{1}{2}(-\frac{1}{2}\log\frac{1}{2} - (\frac{1}{2})\log\frac{1}{2}) + \frac{1}{2}(0-0) = 0.5$

$H(D|A='y') = \frac{1}{2}(-\frac{1}{2}\log\frac{1}{2} - (\frac{1}{2})\log\frac{1}{2} + \frac{1}{2}(0-0) = 0.5$

$G(D|A='x') = G(D|A='y') = 0.8112781 - 0.5 = 0.3112781$

C4.5

$G(D|A='x') = G(D|A='y') = 0.8112781 - 0.5 = 0.3112781$

$SplitInfo(D|A='x') = SplitInfo(D|A='y') = -\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2}$

$gRation(D|A='x') = gRation(D|A='y')$

cart

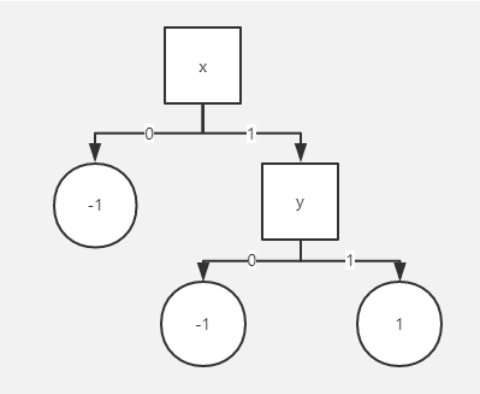
$gini(D|A='x') = \frac{|D_x=0|}{|D|}gini(D_x=0) + \frac{|D_x=1|}{|D|}gini(D_x=1) = \frac{1}{2}(1-\left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2) + \frac{1}{2}(1-1) = 0.25$

$gini(D|A='x') = \frac{|D_y=0|}{|D|}gini(D_y=0) + \frac{|D_y=1|}{|D|}gini(D_y=1) = \frac{1}{2}(1-\left(\frac{1}{2}\right)^2 - \left(\frac{1}{2}\right)^2) + \frac{1}{2}(1-1) = 0.25$

$gini(D|A='x') = gini(D|A='y')$

当两个的特征的信息增益（信息增益率/gini 系数）相同，规定取排在前面的那一个，那么第一个根节点为 x；根据特征 x 的特征值分为两个样本集合（不包括特征 x）：$\{A2, A4|x=0\}$，$\{A1, A3|x=1\}$，而集合 $\{A2, A4\}$ 的 label 值相同，所以作为一个叶子节点属性为 0。而集合 $\{A1, A3\}$ 只有一个特征 y，所以再次根据特征 y 划分为两个集合 $\{A1|y=1\}$，$\{A3|y=0\}$，同理，这两个集合都作为叶子节点，并且分别返回对应样本集合的 label 值。

这三种决策树得到的结果都是一样的：



代码测试结果：

```
id3 :
{'leaf': False, 'attr': 'x1', 'child': [{'leaf': True, 'y': -1}, {'leaf': False, 'attr': 'x2', 'child': [{'leaf': True, 'y': -1}, {'leaf': True, 'y': 1}]}]}
c4.5 :
{'leaf': False, 'attr': 'x1', 'child': [{'leaf': True, 'y': -1}, {'leaf': False, 'attr': 'x2', 'child': [{'leaf': True, 'y': -1}, {'leaf': True, 'y': 1}]}]}
cart :
{'leaf': False, 'attr': 'x1', 'child': [{'leaf': True, 'y': -1}, {'leaf': False, 'attr': 'x2', 'child': [{'leaf': True, 'y': -1}, {'leaf': True, 'y': 1}]}]}
```

3.2 评测指标展示即分析

没有任何优化的原始决策树

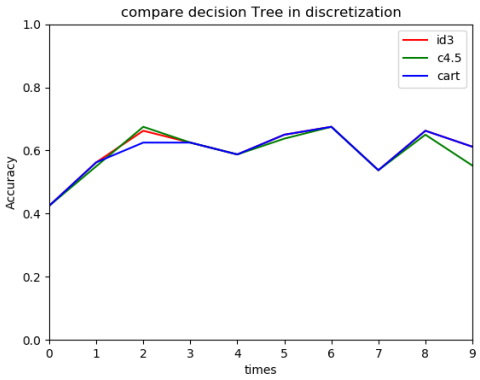
使用 9 折交叉验证的方法（每次取 1 份作为验证集，剩余 9 份作为训练集循环训练 10 次），根据不同的验证值标得到评测结果如下：

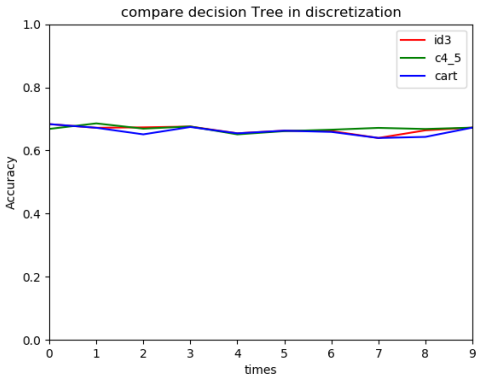
[这里没有进行变量控制，所以这个评测系统是不理想的。放在这里只是为了与较为理想的评测系统做比较]

评测系统模型：本次实验除了对离散化以及随机森林的评测之外，其他评测都采用了以下两种评测系统模型。

1. 进行了验证集固定的 9 折交叉验证，横坐标为 0~9，表示 10 份分组中的第 i 份验证集的坐标
2. 对 9 折交叉验证（每次 9 折交叉验证中同样固定了验证集）外套循环验证，横坐标为 0~9，表示外套循环中第 j 次循环的坐标。

（评测模型 1 更加关注单个验证集的效果，而模型 2 综合了不同验证集的效果。）





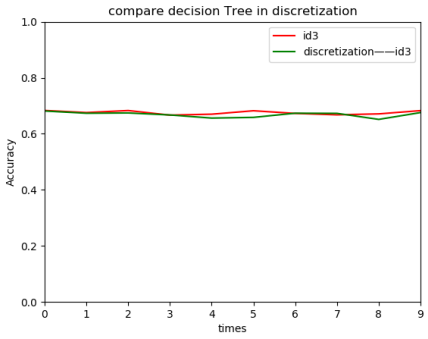
简单对特征 1 进行离散化

离散化评测：

这里也有两种评测方案

1. 随机分组之后再对属性进行离散化，这样子就可以将验证集作为不变量，验证离散化与否的效果。
2. 一开始就将输入的训练集进行离散化之后再对离散化的训练集划分为验证集和训练集，由于结构不同，所以第二种不能将验证集作为不变量，可以在 9 折的交叉验证的循环验证外层再嵌套一层循环，在内层循环求出 9 次交叉验证的平均效果，来对离散化进行评测。

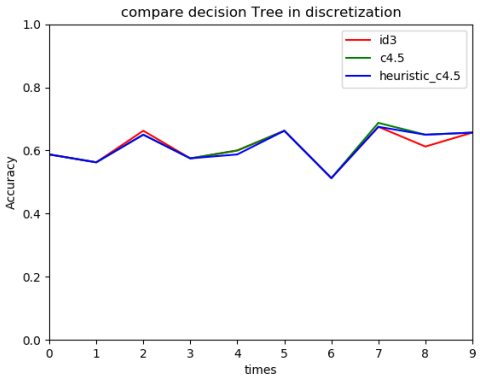
由于分组采取的是原来训练集的存储下标，实现第一种方法比较麻烦，本次实验采取的是第二种方案，结果如下（本次只针对 id3 的方法进行评测）：

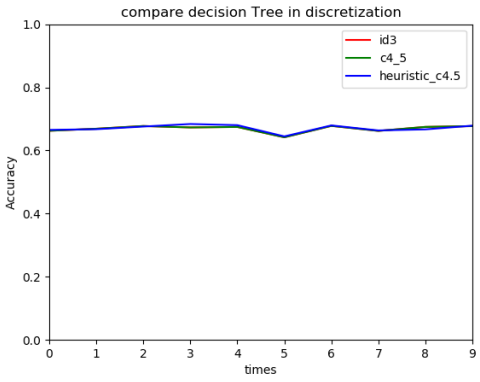


根据上图分析可得，对于本次的训练数据，离散化的效果不明显。

启发式的 c4.5Tree

根据上面提到的验证模型，此处进行了 id3，c4.5，以及启发式的 c4.5 的评测，左图采用了验证模型 1，右图采用了验证模型 2

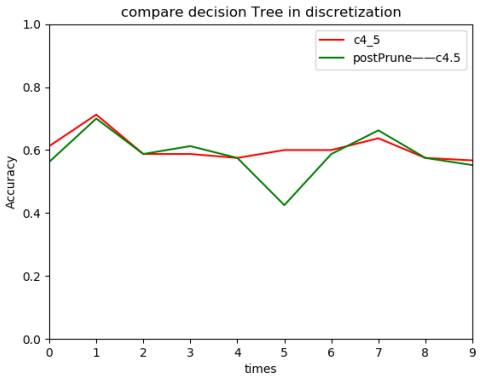


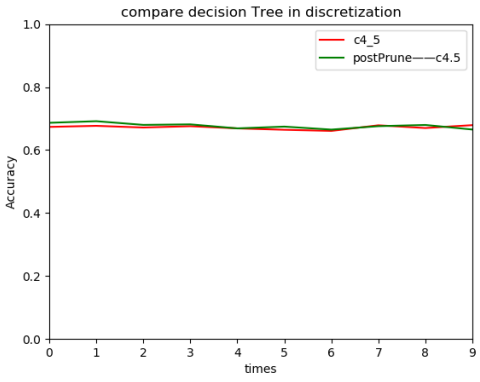


通过观测两种评测系统模型下的曲线，id3，c4.5 以及启发式的 c4.5 在这个训练系统中的效果差不多。

后剪枝评测

本次实验中只评测了 c4.5 决策树的后剪枝和 c4.5 决策树的效果，左图采用了验证模型 1，右图采用了验证模型 2：





根据左图分析，对于单一的验证集考虑，后剪枝可能会导致准确率下降的问题。根据右图分析，考虑多个验证集的验证的效果综合，后剪枝后的决策树和没有进行剪枝的决策树的准确率差不多，可是根据理论分析，后剪枝的决策树在泛化能力上面更优。

随机森林参数评测

根据本次实验的各种方案比较可以看出，优化方案没有在准确率上有很大的提高，所以针对每一种树构成了随机森林，随机森林的优点在于综合了各种树的效果，减低了每棵树带来的误差。随机森林里面可以有多种树，在每一种树都是通过了过滤才放进随机森林里面，**过滤指标**采取了准确值和 F1，当准确值和 F1 的值都高于 0.4 的时候，放进随机森林。

根据理论上，c4.5 决策指标会有较好的效果，本实验中的随机森林由三种树构成，分别是原始的 c4.5 决策树，启发式的 c4.5 的决策树，以及进行后剪枝的 c4.5 的决策树，前面两种树通过 9 折交叉验证经过过滤得到；后剪枝的时候为了防止过拟合现象，对后剪枝的树的评价采用了 9 折分组中的 3 组作为验证集进行评测，最后经过过滤得到。

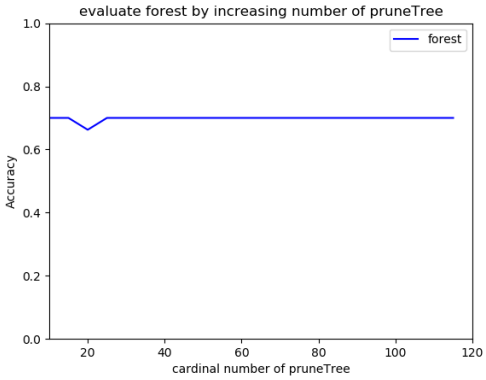
测评方案 1：原始的 c4.5 决策树以及启发式的 c4.5 决策树进行了一次循环 9 折交叉验证的挑选（即从 10 次验证中挑选验证效果较好的决策树得到，每种决策树的数目 ≤10），以后剪枝的决策树的基数作为横坐标，通过固定一份随机的验证集，对随机森林进行评测。

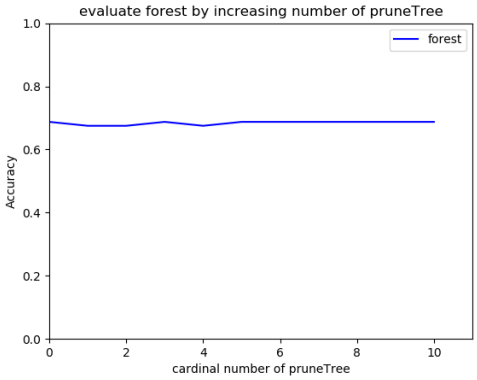
基数的选择，以 0 为起点，根据对后剪枝的决策树进行评价的标准，以 $C_1 0^3 = 120$ 作为终点，间隔 10。

测评方案 2：得到树的方式参考方案一，对森林中不同的树的比例进行评测。

$[(2,2,0),(2,0,2),(0,2,2),(2,2,2),(1,1,4),(1,4,1),(4,1,1),(1,4,4),(4,1,4),(4,4,1),(4,4,4)]$

结果如下，左图表示方案一，右图表示方案二：





使用随机森林之后，预测结果很稳定，预测效果也比较高。

4 思考题

1. 决策树有哪些避免过拟合的方法？

   (a) 对连续性的特征进行离散化，这是因为在 id3 决策树中，连续性的特征一般较多子节点，id3 会偏好子节点多的特征。好比本次训练集的第一列属性，采取离散化，会提高决策树的泛化能力，减少过拟合现象。

   (b) 后剪枝，后剪枝作用就是为了提高决策树的泛化能力，从另一方面也可以认为他降低了决策树的过拟合。

   (c) 采取 k-折交叉验证，K-折得到的验证集比较正确地代表现实中的事件数据。根据比较理想的验证集去训练决策树，可以在一定程度上减低决策树对于训练样本的拟合现象。

2. C4.5 相比于 ID3 的优点是什么？

   这个问题在算法原理分析过。对于 ID3，它偏好于取值样本数较多的特征，为了减低这种偏好可能带来的不利影响，所以对信息增益乘上了该特征的信息熵，样本数较多的时候，信息增益越大，对应该特征的信息熵也越大，从而减低了这种偏好带来可能带来的不利影响。

3. 如何用决策树来判断特征的重要性？

   决策树的构建就是通过 id3，c4.5 或者 gini 系数等决策指标来进行分裂指标的。而这些指标的含义就是，该特征的信息比重大，纯度较高。所以当决策树生成之后，特征的重要性随着树的深度增加而降低。

References

[1] C4.5 算法 https://m.aliyun.com/yunqi/articles/37904

[2] 周志华，机器学习第四章，清华大学出版社

[3] Stuart J.Russel, Peter Norvig , Artificial Intelligence A Modern Approach Chapter4, Prentice Hall