Week 1 - Decision Tree

决策树的基本思想决策树是一种基本的分类与回归方法，它可以看作if-then规则的集合，也可以认为是定义在特征空间与类空间上的条件概率分布。将决策树转换成if-then规则的过程如下：由决策树的根节点到叶节点的每一条路径构建一条规则； 路径内部结点的特征对应规则的条件； 叶节点的类对应规则的结论.
决策树的路径具有一个重要的性质：互斥且完备,即每一个样本均被且只能被一条路径所覆盖。
决策树学习算法主要由三部分构成：特征选择，决策树生成，决策树的剪枝。

0.熵Entropy

> **Def** 表示随机变量的不确定性程度
> **Equ** 经验熵H(X), 经验条件熵 H(Y|X)的计算
> $$H(Y|X) = H(X,Y) - H(X)$$
> $$= -\sum_{x,y} P(X,Y)logP(X,Y) + \sum_x P(X)logP(X)$$
> $$= -\sum_{x,y} P(X,Y)logP(X,Y) + \sum_x(\sum_y P(X,Y))logP(X)$$
> $$= -\sum_{x,y} P(X,Y)logP(X,Y + \sum_{x,y} P(X,Y)logP(X)$$
> $$= -\sum_{x,y} log\frac{P(X,Y)}{P(X)}$$
> $$= -\sum_{x,y} logP(Y|X)$$
> $$= -\sum_x \sum_y P(X)P(Y|X)logP(Y|X)$$
>
> > $$= -\sum_x P(X) \sum_y P(Y|X)logP(Y|X)$$
>
> $$= \sum_x P(X)H(Y|X = x_i)$$

> **Ann** 2为底的对数

```
def uniquecounts(rows):
    results = {}
    for row in rows:
        #计数结果在最后一列
        r = row[len(row)-1]
        if r not in results:results[r] = 0
        results[r]+=1
    return results
```

```
def entropy(rows):
    from math import log
    log2 = lambda x:log(x)/log(2)
    results = uniquecounts(rows)
    #开始计算熵的值
    ent = 0.0
    for r in results.keys():
        p = float(results[r])/len(rows)
        ent = ent - p*log2(p)
    return ent
```

1. 特征选择

1.1 信息增益

> **Def** 本质是训练数据集中的类与特征的互信息。得知特征X的信息而使得类Y的信息的不确定性减少的程度。

> **Equ** 特征A有 $(a_1, a_2, ..., a_n)$, 数据集 D, 信息增益 g, 条件熵 H, 分类个数 K
> $$g(D, A) = H(D) - H(D|A)$$
> $$= -\sum_{k=1}^{K} P(C_k)logP(C_k) + \sum_{i=1}^{n} P(A_i)\sum_{k=1}^{K} P(D_k|A_i)logP(D_k|A_i)$$
> $$= -\sum_{k=1}^{K} \frac{|C_k|}{|D|}log\frac{|C_k|}{|D|} + \sum_{i=1}^{n} \frac{|D_i|}{|D|}\sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|}log\frac{|D_{ik}|}{|D_i|}$$

1.2 *信息增益比*

> **Def** 是信息增益的校正: 目的是解决信息增益往往会偏向于选择取值较多的特征的情况。

> **Equ** $g_R(D, A) = \frac{g(D,A)}{H_A(D)}$
> $$= \frac{g(D,A)}{-\sum_{i=1}^{n} \frac{|D_i|}{|D|}log\frac{|D_i|}{|D|}}$$

1.3 Gini coefficient基尼指数

> **Def** 个人理解是数据集的样本集合的确定性程度，基尼指数数值越大，样本集合的不确定性也就越大。

> **Equ** $Gini(D) = \sum_{i \neq j} PiPj$
> $$\because binaryTree$$
> $$= \sum_{k=1}^{K} P_k(1 - P_k))$$

> $= 1 - \sum_{k=1}^{K}(P_k)^2$
> $= 1 - \sum_{k=1}^{K}(\frac{|C_k|}{|D|})^2$

> 样本集合D根据特征值A是否取某一个可能的值a，分成了集合D_1和D_2
> **Equ** $Gini(D, A) = \frac{|D_1|}{|D|}Gini(D1) + \frac{|D_2|}{|D|}Gini(D2)$

2.决策树生成

> **Def**
> $ID3$ ---- 计算依据：信息增益
> $C4.5$ ---- 计算依据：信息增益比
> 计算过程：
> $1)$ calculate each gain , choose the maximum gain.
> $2)$ divide recursively until (subtree in same class) or (gain $\leq$ threshold)

> $CART$ ---- Gini coefficient
> $1)$ choose the min $Gini(D, A_i)$ as the optimal segmentation point
> $2)$ divide recursively until (each feature traversed) or (subtree in same class)

```
class decisionnode:
    def __init__(self,col = -1,value = None, results = None, tb = None,fb = None
        self.col = col   # col是待检验的判断条件所对应的列索引值
        self.value = value # value对应于为了使结果为True，当前列必须匹配的值
        self.results = results #保存的是针对当前分支的结果，它是一个字典
        self.tb = tb ## desision node,对应于结果为true时，树上相对于当前节点的子树上的节
        self.fb = fb ## desision node,对应于结果为true时，树上相对于当前节点的子树上的节



# 基尼不纯度


# 随机放置的数据项出现于错误分类中的概率
def giniimpurity(rows):
    total = len(rows)
    counts = uniquecounts(rows)
    imp =0
    for k1 in counts:
        p1 = float(counts[k1])/total
        for k2 in counts: # 这个循环是否可以用（1-p1）替换？
            if k1 == k2: continue
            p2 = float(counts[k2])/total
```

```
            imp+=p1*p2
    return imp


# 改进giniimpurity


def giniimpurity_2(rows):
    total = len(rows)
    counts = uniquecounts(rows)
    imp = 0
    for k1 in counts.keys():
        p1 = float(counts[k1])/total
        imp+= p1*(1-p1)
    return imp




#在某一列上对数据集进行拆分。可应用于数值型或因子型变量
def divideset(rows,column,value):
    #定义一个函数，判断当前数据行属于第一组还是第二组
    split_function = None
    if isinstance(value,int) or isinstance(value,float):
        split_function = lambda row:row[column] >= value
    else:
        split_function = lambda row:row[column]==value
    # 将数据集拆分成两个集合，并返回
    set1 = [row for row in rows if split_function(row)]
    set2 = [row for row in rows if not split_function(row)]
    return(set1,set2)


# 以递归方式构造树

def buildtree(rows,scoref = entropy):
    if len(rows)==0 : return decisionnode()
    current_score = scoref(rows)

    # 定义一些变量以记录最佳拆分条件
    best_gain = 0.0
    best_criteria = None
    best_sets = None

    column_count = len(rows[0]) - 1
    for col in range(0,column_count):
        #在当前列中生成一个由不同值构成的序列
        column_values = {}
```

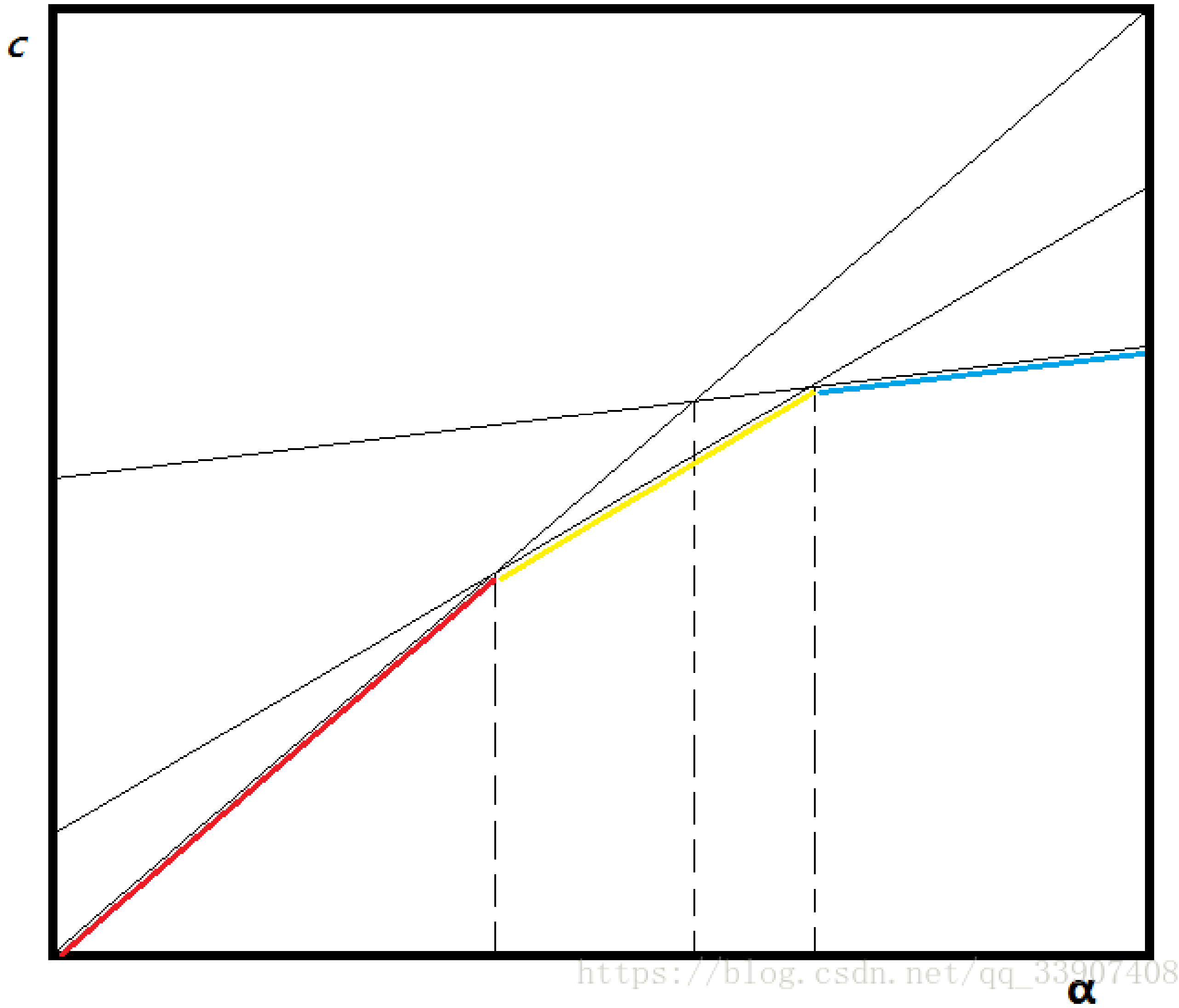
```
        for row in rows:
            column_values[row[col]] = 1 # 初始化
        #根据这一列中的每个值，尝试对数据集进行拆分
        for value in column_values.keys():
            (set1,set2) = divideset(rows,col,value)

            # 信息增益
            p = float(len(set1))/len(rows)
            gain = current_score - p*scoref(set1) - (1-p)*scoref(set2)
            if gain>best_gain and len(set1)>0 and len(set2)>0:
                best_gain = gain
                best_criteria = (col,value)
                best_sets = (set1,set2)

    #创建子分支
    if best_gain>0:
        trueBranch = buildtree(best_sets[0])  #递归调用
        falseBranch = buildtree(best_sets[1])
        return decisionnode(col = best_criteria[0],value = best_criteria[1],
                            tb = trueBranch,fb = falseBranch)
    else:
        return decisionnode(results = uniquecounts(rows))

# 决策树的显示
def printtree(tree,indent = ''):
    # 是否是叶节点
    if tree.results!=None:
        print str(tree.results)
    else:
        # 打印判断条件
        print str(tree.col)+":"+str(tree.value)+"? "
        #打印分支
        print indent+"T->",
        printtree(tree.tb,indent+"  ")
        print indent+"F->",
        printtree(tree.fb,indent+"  ")


# 对新的观测数据进行分类


def classify(observation,tree):
    if tree.results!= None:
        return tree.results
    else:
        v = observation[tree.col]
        branch = None
```

```
        if isinstance(v,int) or isinstance(v,float):
            if v>= tree.value: branch = tree.tb
            else: branch = tree.fb
        else:
            if v==tree.value : branch = tree.tb
            else: branch = tree.fb
        return classify(observation,branch)
```

3.剪枝Pruning

> **Def** 减轻过拟合程度
> **损失函数**$C_\alpha(T)$ a real number intuitively representing some “cost” associated with the event.

> **Equ** 决策树叶节点数量 $|T|$, 分类数量 N_t, 超参数 α
> $C_\alpha(T)=\sum_{t=1}^{|T|}N_tH_t(T)+\alpha\left|T\right|$
> $=-\sum_{t=1}^{|T|}\sum_{k=1}^{K}N_{tk}log\frac{N_{tk}}{N_t}+\alpha\left|T\right|$
> 如果损失函数减小说明是需要剪枝的



横坐标被划分成很多小[ai, aj]区间，每一个区间对应唯一一颗子树。选择标准是上图中的带颜色的子树，以保证最小的损失函数。

> $CART\ Pruning$
> 和之前正常剪枝不同的是没有人为给定超参数α
> 从损失函数的角度看, α 越小，就越偏向于生成叶子越多的树，反之亦然.
> 剪枝发生的时机是二者具有相同的损失函数，节点更少更可取。
> $C_\alpha(t) = C(t) + \alpha$
> $C_\alpha(T_t) = C(T_t) + \alpha\,|T|$
> $\because C_\alpha(t) = C_\alpha(T_t)$
> $\therefore \alpha = \frac{C(t) - C(T_t)}{|T| - 1}$
> 过程：
> 1) 对于剪掉每一个内部节点都会对应一棵树，能计算出该树为最佳的时候对应的 α
> 2) 获取到 α序列，和相应的子树序列
> 3) 交叉验证: 验证过程同正常决策树。

```
on the way
```