决策树算法（Decision tree）

更新时间：2017-10-02 14:57:31

1、简介

分类决策树模型是一种描述对实例进行分类的树形结构。决策树由结点和有向边组成。结点有两种类型：内部节点和叶节点，内部节点表示一个特征或属性，叶节点表示一个类。分类的时候，从根节点开始，当前节点设为根节点，当前节点必定是一种特征，根据实例的该特征的取值，向下移动，直到到达叶节点，将实例分到叶节点对应的类中。决策树的属性结构其实对应着一个规则集合：由决策树的根节点到叶节点的每条路径构成的规则组成；路径上的内部特征对应着if条件，叶节点对应着then结论。决策树和规则集合是等效的，都具有一个重要的性质：互斥且完备。也就是说任何实例都被且仅被一条路径或规则覆盖。决策树还是给定特征条件下类的条件概率分布的一种退化表示（非等效，个人理解）。该条件分布定义在特征空间的划分上，特征空间被花费为互不相交的单元，每个单元定义一个类的概率分布就构成了一个条件概率分布。决策树的每条路径对应于划分中的一个单元。给定实例的特征X，一定落入某个划分，决策树选取该划分里最大概率的类作为结果输出。

2、模型

决策树学习算法包含特征选择、决策树的生成与剪枝过程。决策树的学习算法一般是递归地选择最优特征，并用最优特征对数据集进行分割。由于决策树表示条件概率分布，所以高度不同的决策树对应不同复杂度的概率模型。最优决策树的生成是个NP问题，能实现的生成算法都是局部最优的，剪枝则是既定决策树下的全局最优。

A特征选择：

如何判断一个特征的分类能力呢？有以下两种方法：1、信息增益；2、信息增益比。

全站首页

信息增益：

对于一个可能有n种取值的随机变量：$P(X=x_i)=p_i,\quad i=1,2,\cdots,n$，其熵为$H(X)=-\sum_{i=1}^{n}p_i\log p_i$，且$0\leqslant H(p)\leqslant\log n$。

设有随机变量(X,Y)，其联合分布为：$P(X=x_i,Y=y_j)=p_{ij},\quad i=1,2,\cdots,n;\quad j=1,2,\cdots,m$，条件熵H(Y|X)表示在已知随机变量X的条件下随机变量Y的不确定性，定义为在X给定的条件下，Y的条件概率分布对X的数学期望：$H(Y|X)=\sum_{i=1}^{n}p_iH(Y|X=x_i)$，$p_i=P(X=x_i),\quad i=1,2,\cdots,n$。当上述定义式中的概率由数据估计（比如上一章提到的极大似然估计）得到时，所对应的熵和条件熵分别称为经验熵和经验条件熵。则有了相关定义：

定义 5.2（信息增益） 特征 A 对训练数据集 D 的信息增益 $g(D,A)$，定义为集合 D 的经验熵 $H(D)$ 与特征 A 给定条件下 D 的经验条件熵 $H(D|A)$ 之差，即

$$g(D,A)=H(D)-H(D|A) \tag{5.6}$$

决策树学习中的信息增益等价于训练数据集中类与特征的互信息。具体计算流程如下：

设训练数据集为 D，$|D|$ 表示其样本容量，即样本个数．设有 K 个类 C_k，$k=1,2,\cdots,K$，$|C_k|$ 为属于类 C_k 的样本个数，$\sum_{k=1}^{K}|C_k|=|D|$．设特征 A 有 n 个不同的取值 $\{a_1,a_2,\cdots,a_n\}$，根据特征 A 的取值将 D 划分为 n 个子集 $D_1,D_2,\cdots,D_n$，$|D_i|$ 为 D_i 的样本个数，$\sum_{i=1}^{n}|D_i|=|D|$．记子集 D_i 中属于类 C_k 的样本的集合为 D_{ik}，即 $D_{ik}=D_i\cap C_k$，$|D_{ik}|$ 为 D_{ik} 的样本个数．于是信息增益的算法如下：

算法 5.1（信息增益的算法）

输入：训练数据集 D 和特征 A；

最近活动

i半搜

20 Mysql综合案例

1212双12活动盛大开启，5折优惠惊喜不断

该类最新

1 CV：image caption(SCA-CNN Spatial and Ch

2 机器学习基本知识

3 感知器原理

4 CV：image caption(What Value Do Explicit Hi

5 线性回归(liner regression)相关算法

6 CV：image caption(Show and Tell: A Neural I

7 KNN（k-Nearest Neighbor）算法

8 CV：image caption(Show, Attend and Tell: Ne

9 决策树算法（Decision tree）

10 朴素贝叶斯算法（Naive Bayesian）

11 CV：image caption(Deep Captioning With M

12 逻辑回归（logistic regression）

13 最大熵模型（maximum entropy）

14 支持向量机（SVM：support vector machin

15 CV：image caption(A Hierarchical Approach

16 CV：image caption(Deep Visual-Semantic Al

17 神经网络（NN）

18 卷积神经网络（CNN）

19 循环神经网络（RNN）

20 Aprior算法、FP Growth算法

输出：特征 A 对训练数据集 D 的信息增益 $g(D,A)$.

（1）计算数据集 D 的经验熵 $H(D)$

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$$

（2）计算特征 A 对数据集 D 的经验条件熵 $H(D|A)$

$$H(D|A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$$

（3）计算信息增益

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

信息增益比：

信息增益的值是相对于训练数据集而言的，当H(D)大的时候，信息增益值往往会偏大，这样对H(D)小的特征不公平。改进的方法是信息增益比：

定义 5.3（信息增益比） 特征 A 对训练数据集 D 的信息增益比 $g_R(D,A)$ 定义为其信息增益 $g(D,A)$ 与训练数据集 D 的经验熵 $H(D)$ 之比：

$$g_R(D,A)=\frac{g(D,A)}{H(D)} \tag{5.10}$$

B生成：

ID3算法：

从根节点开始，计算所有可能的特征的信息增益，选择信息增益最大的特征作为当前节点的特征，由特征的不同取值建立空白子节点，对空白子节点递归调用此方法，直到所有特征的信息增益小于阈值或者没有特征可选为止。

C4.5算法：

全站首页

C4.5算法与ID3相似，但是在选择的时候使用的是信息增益比，形式化地描述如下：

5.3.2 C4.5 的生成算法

C4.5 算法与 ID3 算法相似，C4.5 算法对 ID3 算法进行了改进．C4.5 在生成的过程中，用信息增益比来选择特征．

算法 5.3（C4.5 的生成算法）

输入：训练数据集 D，特征集 A，阈值 ε；

输出：决策树 T.

（1）如果 D 中所有实例属于同一类 C_k，则置 T 为单结点树，并将 C_k 作为该结点的类，返回 T；

（2）如果 $A=\varnothing$，则置 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该结点的类，返回 T；

（3）否则，按式(5.10)计算 A 中各特征对 D 的信息增益比，选择信息增益比最大的特征 A_g；

（4）如果 A_g 的信息增益比小于阈值 ε，则置 T 为单结点树，并将 D 中实例数最大的类 C_k 作为该结点的类，返回 T；

（5）否则，对 A_g 的每一可能值 a_i，依 $A_g=a_i$ 将 D 分割为子集若干非空 D_i，将 D_i 中实例数最大的类作为标记，构建子结点，由结点及其子结点构成树 T，返回 T；

（6）对结点 i，以 D_i 为训练集，以 $A-\{A_g\}$ 为特征集，递归地调用步(1)～步(5)，得到子树 T_i，返回 T_i. ■

C剪枝：

决策树很容易发生过拟合，过拟合的原因在于学习的时候过多地考虑如何提高对训练数据的正确分类，从而构建出过于复杂的决策树。解决这个问题的办法就是简化已生成的决策树，也就是剪枝。决策树的剪枝往往通过极小化决策树整体的损失函数或代价函数来实现。设决策树T的叶节点有|T|个，t是某个叶节点，t有Nt个样本点，其中归入k类的样本点有Ntk个，Ht(T)为叶节点t上的经验熵，α≥0为参数，则损失函数可以定义为：

$$C_\alpha(T)=\sum_{t=1}^{|T|}N_tH_t(T)+\alpha|T|$$

其中经验熵Ht(T)为：

$$H_t(T) = -\sum_k \frac{N_{tk}}{N_t} \log \frac{N_{tk}}{N_t}$$

表示叶节点t所代表的类别的不确定性。损失函数对它求和表示所有被导向该叶节点的样本点所带来的不确定的和的和。我没有多打一个“的和”，第二个是针对叶节点t说的。

在损失函数中，将右边第一项记作：

$$C(T) = \sum_{t=1}^{|T|} N_t H_t(T) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t}$$

则损失函数可以简单记作：

$$C_\alpha(T) = C(T) + \alpha |T|$$

C(T)表示模型对训练数据的预测误差，即模型与训练数据的拟合程度，|T|表示模型复杂度，参数α≥0控制两者之间的影响，α越大，模型越简单，α=0表示不考虑复杂度。

剪枝，就是当α确定时，选择损失函数最小的模型。子树越大C(T)越小，但是α|T|越大，损失函数反映的是两者的平衡。决策树的生成过程只考虑了信息增益或信息增益比，只考虑更好地拟合训练数据，而剪枝过程则考虑了减小复杂度。前者是局部学习，后者是整体学习。

3、CART算法

分类与回归树（CART）模型同样由特征选取、树的生成和剪枝组成，既可以用于分类也可以用于回归。CART假设决策树是二叉树，内部节点特征的取值为是和否，对应一个实例的特征是否是这样的。决策树递归地二分每个特征，将输入空间划分为有限个单元。

CART生成

全站首页

全站首页

递归地构建二叉决策树的过程。对回归树用平方误差最小化准则，对分类树用基尼指数最小化准则，进行特征选取，生成二叉树。

回归树与分类树在数据集上的不同就是数据集的输出部分不是类别，而是连续变量。假设输入空间已经被分为M个单元$R_1, R_2, \cdots, R_M$，分别对应输出值cm，于是回归树模型可以表示为：

1、回归树

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

回归树的预测误差：

$$\sum_{x_i \in R_m} (y_i - f(x_i))^2$$

那么输出值就是使上面误差最小的值，也就是均值：

$$\hat{c}_m = \text{ave}(y_i \mid x_i \in R_m)$$

难点在于怎么划分，一种启发式的方法（其实就是暴力搜索吧）：

遍历所有输入变量，选择第j个变量和它的值s作为切分变量和切分点，将空间分为两个区域：

$$R_1(j,s) = \{x \mid x^{(j)} \leqslant s\} \quad 和 \quad R_2(j,s) = \{x \mid x^{(j)} > s\}$$

然后计算两个区域的平方误差，求和，极小化这个和，具体的，就是：

$$\min_{j,s}\left[\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2\right]$$

全站首页

当j最优化的时候，就可以将切分点最优化：

$$\hat{c}_1 = \text{ave}(y_i \mid x_i \in R_1(j,s)) \quad 和 \quad \hat{c}_2 = \text{ave}(y_i \mid x_i \in R_2(j,s))$$

递归调用此过程，这种回归树通常称为最小二乘回归树。

2、分类树

与回归树算法流程类似，只不过选择的是最优切分特征和最优切分点，并采用基尼指数衡量。基尼指数定义：

$$\text{Gini}(p) = \sum_{k=1}^{K} p_k(1-p_k) = 1 - \sum_{k=1}^{K} p_k^2$$

对于给定数据集D，其基尼指数是：

$$\text{Gini}(D) = 1 - \sum_{k=1}^{K} \left(\frac{|C_k|}{|D|} \right)^2$$

Ck是属于第k类的样本子集，K是类的个数。Gini(D)反应的是D的不确定性（与熵类似），分区的目标就是降低不确定性。

D根据特征A是否取某一个可能值a而分为D1和D2两部分：

$$D_1 = \{(x,y) \in D \mid A(x) = a\}, \quad D_2 = D - D_1$$

则在特征A的条件下，D的基尼指数是：

$$\text{Gini}(D,A) = \frac{|D_1|}{|D|}\text{Gini}(D_1) + \frac{|D_2|}{|D|}\text{Gini}(D_2)$$

有了上述知识储备，可以给出CART生成算法的伪码：

设节点的当前数据集为D，对D中每一个特征A，对齐每个值a根据D中样本点是否满足A==a分为两部分，计算基尼指数。对所有基尼指数选择最小的，对应的特征和切分点作为最优特征和最优切分点，生成两个子节点，将对应的两个分区分配过去，然后对两个子节点递归。

全站首页

CART剪枝

在上面介绍的损失函数中，当α固定时，一定存在使得损失函数最小的子树，记为复杂度=Tα，α偏大Tα就偏小。设对α递增的序列，对应的最优子树序列为Tn，子树序列第一棵包含第二棵，依次类推。

从T0开始剪枝，对它内部的任意节点t，只有t这一个节点的子树的损失函数是：$C_\alpha(t)=C(t)+\alpha$，以t为根节点的子树的损失函数是：$C_\alpha(T_t)=C(T_t)+\alpha|T_t|$

当α充分小，肯定有 $C_\alpha(T_t)<C_\alpha(t)$，这个不等式的意思是复杂模型在复杂度影响力小的情况下损失函数更小。当α增大到某一点，这个不等式的符号会反过来。只要 $\alpha=\frac{C(t)-C(T_t)}{|T_t|-1}$，损失函数值就相同，但是t更小啊，所以t更可取，于是把Tt剪枝掉。为此，对每一个t，计算 $g(t)=\frac{C(t)-C(T_t)}{|T_t|-1}$，表示损失函数的减少程度，从T中剪枝掉g(t)最小的Tt，取新的α=g(t)，直到根节点。这样就得到了一个子树序列，对此序列，应用独立的验证数据集交叉验证，选取最优子树，剪枝完毕。

4、总结

决策树算法是机器学习中一个大类算法，这些理论是基础，后面的随机森林，GBDT等等算法都是这些算法的提升。需要着重理解。

全站首页

5、实现

此处，我们给出ID3算法和CART算法的一些实现，如下所示：

ID3：

```
from math import log
import operator

def createDataSet():
    dataSet = [[1, 1, 'yes'],
               [1, 1, 'yes'],
               [1, 0, 'no'],
               [0, 1, 'no'],
               [0, 1, 'no']]
    labels = ['no surfacing','flippers']
    #change to discrete values
    return dataSet, labels

def calcShannonEnt(dataSet):
    numEntries = len(dataSet)
    labelCounts = {}
    for featVec in dataSet: #the the number of unique elements and their occurance
        currentLabel = featVec[-1]
        if currentLabel not in labelCounts.keys(): labelCounts[currentLabel] = 0
        labelCounts[currentLabel] += 1
    shannonEnt = 0.0
    for key in labelCounts:
```

```
        prob = float(labelCounts[key])/numEntries
        shannonEnt -= prob * log(prob,2) #log base 2
    return shannonEnt

def splitDataSet(dataSet, axis, value):
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]     #chop out axis used for splitting
            reducedFeatVec.extend(featVec[axis+1:])
            retDataSet.append(reducedFeatVec)
    return retDataSet

def chooseBestFeatureToSplit(dataSet):
    numFeatures = len(dataSet[0]) - 1      #the last column is used for the labels
    baseEntropy = calcShannonEnt(dataSet)
    bestInfoGain = 0.0; bestFeature = -1
    for i in range(numFeatures):        #iterate over all the features
        featList = [example[i] for example in dataSet]#create a list of all the examples of this featı
        uniqueVals = set(featList)       #get a set of unique values
        newEntropy = 0.0
        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value)
            prob = len(subDataSet)/float(len(dataSet))
            newEntropy += prob * calcShannonEnt(subDataSet)
        infoGain = baseEntropy - newEntropy     #calculate the info gain; ie reduction in entropy
        if (infoGain > bestInfoGain):       #compare this to the best gain so far
```

全站首页

```
            bestInfoGain = infoGain         #if better than current best, set to best
            bestFeature = i
    return bestFeature                  #returns an integer
```

全站首页

```
def majorityCnt(classList):
    classCount={}
    for vote in classList:
        if vote not in classCount.keys(): classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.iteritems(), key=operator.itemgetter(1), reverse=True
    return sortedClassCount[0][0]

def createTree(dataSet,labels):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]#stop splitting when all of the classes are equal
    if len(dataSet[0]) == 1: #stop splitting when there are no more features in dataSet
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatLabel = labels[bestFeat]
    myTree = {bestFeatLabel:{}}
    del(labels[bestFeat])
    featValues = [example[bestFeat] for example in dataSet]
    uniqueVals = set(featValues)
    for value in uniqueVals:
        subLabels = labels[:]       #copy all of labels, so trees don't mess up existing labels
        myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFeat, value),subLabe
```

```
    return myTree

def classify(inputTree,featLabels,testVec):
    firstStr = inputTree.keys()[0]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstStr)
    key = testVec[featIndex]
    valueOfFeat = secondDict[key]
    if isinstance(valueOfFeat, dict):
        classLabel = classify(valueOfFeat, featLabels, testVec)
    else: classLabel = valueOfFeat
    return classLabel

def storeTree(inputTree,filename):
    import pickle
    fw = open(filename,'w')
    pickle.dump(inputTree,fw)
    fw.close()

def grabTree(filename):
    import pickle
    fr = open(filename)
    return pickle.load(fr)
```

CART：

此文链接：http://makaidong.com/taojake-ML/991_908173.html

转载请注明出处：决策树算法（Decision tree）

全站首页

来源：马开东云搜索（微信/QQ：420434200,微信公众号：makaidong-com）

欢迎分享本文，转载请保留出处！

【原文阅读】:http://www.cnblogs.com/taojake-ML/p/6117469.html

[1] [2]

全站首页

最近活动

1212双12活动盛大开启，5折优惠惊喜不断