机器学习第三次实验报告——决策树

April 6, 2018

1 问题描述

本次实验的目标为用决策树算法实现分类问题。对于离散值采用 IC3 算法构造决策树，对于连续值采用 CART 算法构造决策树，并在测试数据上测试，计算准确率。

2 解决方案

2.1 原理介绍

决策树算法主要应用于分类问题，其中决策树由节点 (node) 和有向边 (directed edge) 组成。节点的类型有两种：内部节点和叶子节点。其中，内部节点表示一个特征，而叶子节点表示一个分类。当构建好一棵决策树时，就可以根据样本的特征从决策树的根节点下降到叶子节点，决定样本的类别。

构造决策树的关键问题在于如何选择特征，优先选择的特征应该具有最大的区分度，即依据此特征划分样本后，应该使同一类别的样本尽可能属于同一类别。

为了衡量特征的区分度，首先引入了信息熵的概念。信息熵主要用来衡量样本的混乱程度。例如，随机变量 X 的信息熵可以表示为

$$H(X) = \sum_{i=1}^{n} p_i * logp_i$$

其中 n 代表 X 的 n 种不同的离散取值。而 p_i 代表了 X 取值为 i 的概率，log 为以 2 为底的对数。这个值越大，代表信息的不确定性越大。

除此之外，引入了条件熵的概念。同样对于样本 X 和特征 Y，X 对于 Y 的条件熵可以表示为：

$$H(X|Y) = \sum_{j=1}^{k} p(y_j) * H(X|y_i)$$

即对于特征 Y 的不同取值，计算划分后的各个子数据集的信息熵乘以该特征值出现的概率。

这样 $H(X) - H(H|Y)$ 的意义就是以 Y 特征划分数据集后，数据集混乱程度的减少程度，称之为信息增益。一个特征的信息增益越大，代表它减少信息混乱程度的能力越强，越有区分度。

当一个特征对于数据集没有区分度时，即无论该特征取什么值，样本的类别都是相同的，这时可以算出此特征的信息增益为 0。同样的，当一个特征对于不同类别的样本的特征值都是相同的时，这个特征也是没有意义的，这时的信息增益也是 0。

构造决策树时，应该优先选择信息增益大的特征作为决策依据，将样本划分为不同类别。

2.2 具体实现

算法的总体目标是构造一棵树用来做决策。此处将树的结构定义为：树的键是用来划分的特征的各个特征值，值分两种情况。若经过该特征划分后的子数据集均属于一种类别，则值为样本的类别标签; 若子数据集属于不同类别，则递归继续划分特征，构造子树。

其中划分特征时，若没有剩余特征可供划分或者有剩余特征但是选取的特征的信息增益太小 (此处阈值设为 0，即该特征无意义)，就选择出现次数最多的标签作为数据集的标签。

算法的基本框架为一个递归函数 decision_tree($dataset, labels$), 该函数的返回值或者是一个字典变量，或者是一个类别标签，分别对应上述有无特征可供划分的情况。该函数的定义如下:

```
#define a recursive function to build the decision tree, each call for the
#function should return either a number, which
#represents the label of the data with the feature, or a dictionary, whose
#key means the unique value of the feature within
#the dataset and the value is another dictionary
def decision_tree(dataset,labels):
    #first check if the labels in the dataset are the same
    if np.unique(labels).shape[0]==1:
        #return the label
        return labels[0]
```

```
    else:
        #find_feature returns the index of the feature, which starts from zerox
        maxinfo_fea = find_feature(dataset,labels)
        if maxinfo_fea<0:
            #if the information gain is too little or there are no
            #features left to split, return the label
            #which appeared most
            num= 0
            most_label="1"
            for lab in set(labels):
                curr_num= np.sum(labels==lab)
                if num<curr_num:
                    num=curr_num
                    most_label = lab
            return most_label
        #otherwise split the dataset with according to the
        #selected feature and build the sub tree
        else:
            #append the feature to the global variable
            tree_fea.append(maxinfo_fea)
            #recursive build the decision tree
            #initialize a dictionary first
            child_tree={}
            #extract the values of the dataset on the feature
            vals = dataset[:,maxinfo_fea]
            #print (vals)
            uni_vals = np.unique(vals)
            #for each unique value of the feature, rebuild the
            #dataset and the labels to
            #build the decision tree recursively
            child_tree[0]=maxinfo_fea
            for val in uni_vals:
                ##'where' returns the indiceof an array
                #which satisfy the condition
                sub_index = (np.where(vals==val))[0]
                sub_set= dataset[sub_index,:]
                sub_labels = labels[sub_index]
                child_tree[val]= decision_tree(sub_set,sub_labels)
            tree_fea.remove(maxinfo_fea)
            return child_tree
```

函数参数为待处理的数据集 $dataset$ 以及对应的标签 $labels$, 函数内部首先判断数据集中的样本是否对应于同一标签，若是直接返回标签数。否则调用 find_feature($dataset, labels$) 找到信息增益最大的特征，若该函数返回值为-1，表示没有特征剩余或者剩余特征的信息增益小于阈值，此时寻找出现次数最多的标签返回。若可以找到用来划分的特征，首先将该特征加入到全局数组 $maxinfo\_fea$ 中，然后遍历所有特征值递归调用函数构造子树。遍历所有特征后，返回当前树。

其中用到的函数 find_feature($dataset, labels$) 的定义为

```
#returns the index of the feature which brings the maximum
#information gain
def find_feature(dataset,labels):
    max_fea = -1
    #first calculate the entropy of the information without any feature
    base_info = sub_infogain(labels)
    #the initial calue also means the threshold
    max_gain =0
    for i in range(feature_num):
        #if the feature has been chosen,continue
        curr_info= 0
        if i in tree_fea:
            continue
        else:
            vals = dataset[:,i]
            uni_vals = np.unique(vals)
            #for each unique feature value, accumulate the information gain
            for val in uni_vals:
                sub_index = np.where(vals==val)
```

```
                sub_set= dataset[sub_index,:]
                sub_labels = labels[sub_index]
                sub_num = len(sub_index[0])
                prob = sub_num/vals.shape[0]
                #the function 'sub_infogain' returns the base entropy of information of the subset
                curr_info = prob*sub_infogain(sub_labels)
            if base_info-curr_info>max_gain:
                max_fea = i
                max_gain = curr_info-base_info
    return max_fea
```

函数中的 max_gain 表示信息增益的最大值，其初始值即代表阈值。然后遍历所有特征，全局变量 $tree\_fea$ 存储已经用作决策依据的特征，对于没有用过的特征，对于其每个特征值，计算划分出的子数据集的信息熵与该特征值出现概率的乘积并累加起来，作为条件熵。然后计算其信息增益，与当前最大增益比较并更新最大增益，最后返回具有最大增益的特征。

其中函数 sub_infogain(sub_labels) 用来计算信息熵，其定义为：

```
def sub_infogain(labels):
    unique_labels = np.unique(labels)
    res = 0
    total_num= labels.shape[0]
    for unique_label in unique_labels:
        sub_num =np.sum((labels==unique_label))
        sub_prob= sub_num/total_num
        inc = sub_prob*np.log2(sub_prob)
        res+=inc
    return -res
```

定义如上函数后，准备好测试集数据即可调用递归函数 decision_tree($dataset, labels$) 用来构造决策树。

2.3 实验结果

实验构造的决策树为:

```
模糊:       否
清晰:
    沉闷:           是
    浊响:           是
    清脆:           否
稍糊:
    沉闷:           否
    浊响:
        稍蜷:
            乌黑:               是
            青绿:               否
```

导入测试集数据对决策树进行测试，测试函数为 test_tree($test\_dataset, test\_labels$)，该函数定义如下：

```
def test_tree(test_dataset,test_labels):
    corr_num=0
    for index,data in enumerate(test_dataset):
        curr_tree= my_tree
        while(1):
            i= curr_tree[0]
            my_fea= data[i]
            curr_tree = curr_tree[my_fea]
            if curr_tree ==" 是"or curr_tree==" 否":
                ans= curr_tree
                fact= test_labels[index]
                if ans==fact:
                    corr_num+=1
                break
    return corr_num/test_dataset.shape[0]
```

其中 i 为当前字数索引的特征，my_fea 为样本数据的特征值，当前子树在该特征值上的值为“是”或“否”时停止递归。

该函数在测试集上的输出结果为：

```
['浅白' '蜷缩' '浊响' '清晰']:  预测值为 是，真实值为 是
['乌黑' '稍蜷' '沉闷' '清晰']:  预测值为 是，真实值为 是
['乌黑' '蜷缩' '沉闷' '清晰']:  预测值为 是，真实值为 是
['青绿' '蜷缩' '沉闷' '稍糊']:  预测值为 否，真实值为 是
['浅白' '蜷缩' '浊响' '清晰']:  预测值为 是，真实值为 是
['青绿' '稍蜷' '浊响' '清晰']:  预测值为 是，真实值为 否
['乌黑' '稍蜷' '浊响' '稍糊']:  预测值为 是，真实值为 否
['青绿' '稍蜷' '浊响' '模糊']:  预测值为 否，真实值为 否
['乌黑' '稍蜷' '沉闷' '稍糊']:  预测值为 否，真实值为 否
['青绿' '硬挺' '清脆' '模糊']:  预测值为 否，真实值为 否
准确率为0.7
```

3 实验改进

IC3 算法仅适用于数据离散分布的情况，而为了处理连续分布的情况，采用 CART 算法构造决策树。

3.1 算法简介

CART 决策树是一棵二叉树，树仍分为内部节点和叶子节点，叶子节点对应该数据集的类别标签，而内部节点对应一种数据集的划分方式。由于 CART 算法产生的是二叉树，每个内部节点将数据集划分为两部分。对于离散型特征值，每个特征值将样本数据集划分具有此特征值的数据和其他数据; 对于连续型特征值 f_i，将样本的 n 个不同的连续特征值由小到大排列后取 $n-1$ 个中间值作为划分数据集的切分点，对于每个中间值 a, 将数据集划分为 $f_i <= a$ 和 $f_i > a$ 两部分。

为了选择最优的切分特征以及切分点，引入了“基尼指数”。随机变量 X 的基尼指数可以表示为:

$$Gini(X) = \sum_{i=1}^{n}\sum_{i'\neq 1} p_i * p_i' = 1 - \sum_{i=1}^{|n|}(p_k)^2$$

Gini(X) 反映了从数据集 X 的纯度，Gini(X) 越小，数据集 D 的纯度越高。

类似于条件熵，数据集 X 在条件 Y 下的基尼指数为

$$Gini(X|Y) = \sum_{j=1}^{k} p(y_j) * Gini(X|y_i)$$

于是在选择划分属性和划分点时，选择使划分后基尼指数最小的属性作为最优划分属性。同时需要注意的是，与离散属性不同，若当前节点划分属性为连续属性，该属性还可以作为其后代节点的划分属性。

3.2 具体实现

算法的实现流程与 IC3 算法类似，递归构造一棵用来决策的树。树仍由字典表示，每个字典有四个键 0,1,$name$,$value$。$name$,$value$ 对应的值分别为特征名和用来划分的特征值，而 0 和 1 对应的值分别为满足特征值和不满足特征值的样本集对应的子树。

算法的基本框架为递归函数 cart_tree($dataset, labels$), 其具体实现为

```python
def cart_tree(dataset,labels):
     if np.unique(labels).shape[0]==1:
        #if all samples in the dataset fall into one label,return the label directly
        return labels[0]
     else:
        #'findBestFeature'  return the feature, the partition point and the divided datasets
        #with the minimum Gini index.
        #If the Gini index's decrease is under a certain threshold, which means there is no
        #need to split the dataset, return the label which appeared most
        fea_name,fea_val,sub_set1,sub_set2,sub_label1,sub_label2 = findBestFeature(dataset,labels)
        if fea_name==-1:
            num= 0
            most_label="1"
            for lab in set(labels):
                curr_num= np.sum(labels==lab)
                if num<curr_num:
                    num=curr_num
                    most_label = lab
            return most_label
        else:
            #otherwise call itself reursively to build the subtree
            res={}
```

```
            res['name']=fea_name
            res['val']=fea_val
            res[0]=cart_tree(sub_set1,sub_label1)
            res[1]=cart_tree(sub_set2,sub_label2)
            return res
```

首先判断是否所有类别标签都相同，若相同直接返回样本标签。否则调用函数 findBestFeature($dataset, labels$) 寻找最优特征，若其对基尼指数的减少值小于等于 0，则直接返回出现次数最多的标签类别。否则递归调用函数来构造子树，然后返回当前树。

用来寻找最优特征的函数 findBestFeature($dataset, labels$) 定义为:

```
def findBestFeature(dataset,labels):
    fea_name=-1
    fea_val=0
    sub_set1=np.array([])
    sub_label1=np.array([])
    sub_set2=np.array([])
    sub_label2=np.array([])
    best_gini =CalGini(labels)
    for i in range(feature_num):
        vals = dataset[:,i]
        uni_val= np.unique(vals)
        if uni_val.shape[0]==1:
            continue
        else:
            #if the feature is discrete
            if i in dis_fea:
                uni_val2=uni_val
            else:
                uni_val2=[]
                uni_val=uni_val.astype(np.float64)
                #calculate n-1 partition point for continuous feature
                for index,con_val in enumerate(uni_val):
                    if index==uni_val.shape[0]-1:
                        continue
                    else:
                        var1 = uni_val[index+1]
                        var2 = (con_val)
                        uni_val2.append((var1+var2)/2)
            for val in uni_val2:
                #for discrete the feature, choose the sample with the feature value
                if i in dis_fea:
                    sub_index1 = (np.where(vals==val))[0]
                else:
                #for continuous feature, choose the sample with the feature value below the partition p
                    vals=vals.astype(np.float64)
                    sub_index1 = (np.where(vals<=val))[0]
                sub_set1_temp= dataset[sub_index1,:]
                sub_label1_temp= labels[sub_index1]
                sub_index2= np.array(list(set(range(dataset.shape[0]))-set(sub_index1)))
                sub_set2_temp= dataset[sub_index2,:]
                sub_label2_temp = labels[sub_index2]
                temp_gini=len(sub_index1)/dataset.shape[0]*CalGini(sub_label1_temp)+len(sub_index2)/dat
                #when the temp gini index is the minimum, update the gini index
                if temp_gini<best_gini:
                    fea_name=i
                    fea_val=val
                    best_gini=temp_gini
                    sub_set1=sub_set1_temp
                    sub_set2=sub_set2_temp
                    sub_label1=sub_label1_temp
                    sub_label2=sub_label2_temp
    return fea_name,fea_val,sub_set1,sub_set2,sub_label1,sub_label2
```

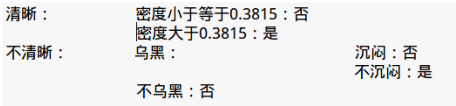
首先计算数据集未划分时的基尼指数，基尼指数增长的阈值为 0，遍历数据集的所有特征，对于连续型的特征，计算该特征上的所有划分点。再依据特征点将数据集划分为两部分，分别计算各部分的基尼指数乘以该

部分出现的概率并累加得到数据集在该特征上的基尼指数，若基尼指数的增长大于阈值，返回对应的特征，划分点以及划分后的子数据集。其中计算子数据集基尼指数的函数 CalGini($labels$) 定义为

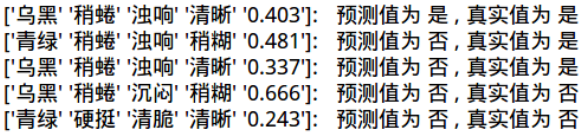
```
def CalGini(labels):
    unique_labels = np.unique(labels)
    res = 0
    total_num= labels.shape[0]
    for unique_label in unique_labels:
        sub_num =np.sum((labels==unique_label))
        sub_prob= sub_num/total_num
        inc = sub_prob*sub_prob
        res+=inc
    return 1-res
```

3.3 实验结果

CART 算法构造出的决策树为



测试集上的实验结果为



4 实验总结

本次实验基于 CART 算法和 IC3 算法构造决策树，分别依据信息熵的增益和基尼指数判断划分数据集的特征和划分点，选择字典存储决策树，递归调用函数，最后在测试集上进行预测。