پروژه درخت تصمیم

استاد : دکتر عبدی

مریم جعفری 99521181

توضیحات :

برای ساختن درخت از node استفاده می کنیم هر node دارای ویژگی های زیر می باشد :

Parent
Value
Chosen_attribute
Lable
Entropy
Info_Gain

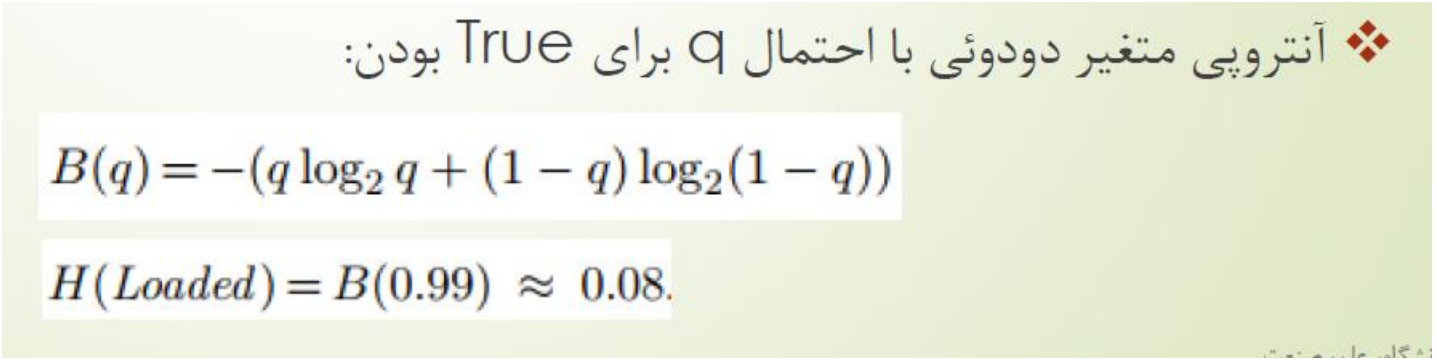
```
class Node:
    def __init__(self, parent,attributes_list,value,datas):
        self.parent = parent
        self.value = value
        self.remain_attribute_list = attributes_list
        self.examples = datas
        self.chosen_attribute = None
        self.lable= ''                          # for leaf
        self.entropy = Entropy(self.examples)
        self.info_Gain =None
```

توضیحات :
این تابع pluarity Value می باشد که در آن اینکه چه مقداری بیشتر است و تعداد آن چقدر است (زیرا دیتا های ما 0 و 1 هستند برای همین با استفاده از آن می توانیم انتروپی را نیز محاسبه کنیم)

```
def Pluarity_Value(examples):
    value = examples.iloc[:,-1].value_counts().nlargest(1).index[0]
    count = examples.iloc[:,-1].value_counts().nlargest(1).tolist()[0]
    return value,count
```

توضیحات :

این تابع entropy می باشد که با توجه به توضیحات اسلاید درسی زده شده است




```
def Entropy(examples):
    value,count = Pluarity_Value(examples)
    count_all = examples.shape[0]
    q= count/count_all
    if q ==1 or q ==0:
        return 0
    else:
        return -(q*math.log2(q)+(1-q)*math.log2(1-q))
```

توضیحات :
این تابع Gini Index می باشد که در اینترنت سرچ کردم و با این فرمول زدم

```
def Gini_Index(examples):
    value,count = Pluarity_Value(examples)
    count_all = examples.shape[0]
    q= count/count_all
    if q ==1 or q ==0:
        return 0
    else:
        return 1-(q*q+(1-q)*(1-q))
```

$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right)$$

```
def Reminder(examples,attribute):
    count_all = examples.shape[0] #number of rows(data)
    unique_values = examples[attribute].unique()
    sum = 0
    for value in unique_values:
        group = examples.loc[examples[attribute] == value]
        count_group = group.shape[0]
        group_Entropy = Entropy(group)
        sum+= group_Entropy*count_group/count_all
    return sum
```

توضیحات:
در این تابع برای همه ی ویژگی های داده شده ی باقی مانده reminder را محاسبه می کنیم و ویژگی ای که کمترین Reminder را داشته باشد به عنوان ویژگی انتخابی برمی گرداند

```
def choose_Attribute(examples,list_attr):
    chosen = ''
    min_reminder=1000
    for attr in list_attr:
        rem = Reminder(examples,attr)
        if rem<min_reminder:
            min_reminder = rem
            chosen=attr
    return chosen
```

```
node.info_Gain = node.entropy -Reminder(node.examples,attr)
```

```
node_list = []
def decision_Tree(node):
    # if node.examples = null then node.lable =pluarity_value(node.parent)
    if node.examples.empty==True:
        node.lable,x = Pluarity_Value(node.parent.examples)
    # if node.attributes == null then node.lable = pluarity_value(node)
    elif len(node.remain_attribute_list)== 0 or node.remain_attribute_list ==None:
        node.lable,x = Pluarity_Value(node.examples)
    # if poluarityvalue(node).count = all => node.lable = poluarityvalue(node)
    elif Pluarity_Value(node.examples)[1] ==node.examples.shape[0]:
        node.lable,x = Pluarity_Value(node.examples)
    else :
        attr = choose_Attribute(node.examples,node.remain_attribute_list)
        node.chosen_attribute = attr
        node.info_Gain = node.entropy -Reminder(node.examples,attr)
        new_attributes = node.remain_attribute_list
        new_attributes.remove(attr)
        unique_values = node.examples[attr].unique()
        for value in unique_values:
            group = node.examples.loc[node.examples[attr] == value]
            t = Node(node,new_attributes,value=value,datas=group)
            node_list.append(t)
            decision_Tree(t)
```

تست کردن

```
def test_row_by_tree(row,node):
    tree_lable=''
    if node.lable=='':
        children = [n for n in node_list if n.parent == node]
        is_find = False
        for child in children:
            v =  row[node.chosen_attribute].tolist()[0]
            if str(child.value) ==str(v):
                is_find=True
                return test_row_by_tree(row,child)
        if is_find==False:
            v,c= Pluarity_Value(df)
            if str(v) == str(row["survived"].tolist()[0]):
                return True
            else:
                return False
    else:
        tree_lable = node.lable
        if str(node.lable) == str(row["survived"].tolist()[0]):
            return True
        else:
            return False
```

گسسته سازی داده پیوسته

```
def SplitDataToGroups(attribute,min_value,max_value,numberofBaze,examples):
    new_Values = []
    new_attr = str(attribute)+" group"
    len_baze = (max_value-min_value)//numberofBaze
    for i in range(0,examples.shape[0]):
        row = examples.iloc[[i]]
        if row[attribute].tolist()[0] < min_value:
            new_Values.append("group -1")
        elif row[attribute].tolist()[0] >= max_value:
            new_Values.append("group "+str(numberofBaze))
        else :
            i = (row[attribute].tolist()[0] - min_value)//len_baze
            new_Values.append("group "+str(i))
    examples.insert(2,new_attr,new_Values,True)
    attributes.remove(attribute)
    print(examples)
```

ایده برای افزایش دقت تبدیل بازه ها :

ایده ای که داشتم این بود که در ابتدا دیتاها را سورت کنیم سپس آنهایی که پشت سر هم مقادیر خروجی یکسانی داشتند را یک گروه می کنیم سپس گروه ها را حرص می کنیم بر اساس فاکتورهای مختلف