یاسمن توکلی: 98522175

پروژه درخت تصمیم

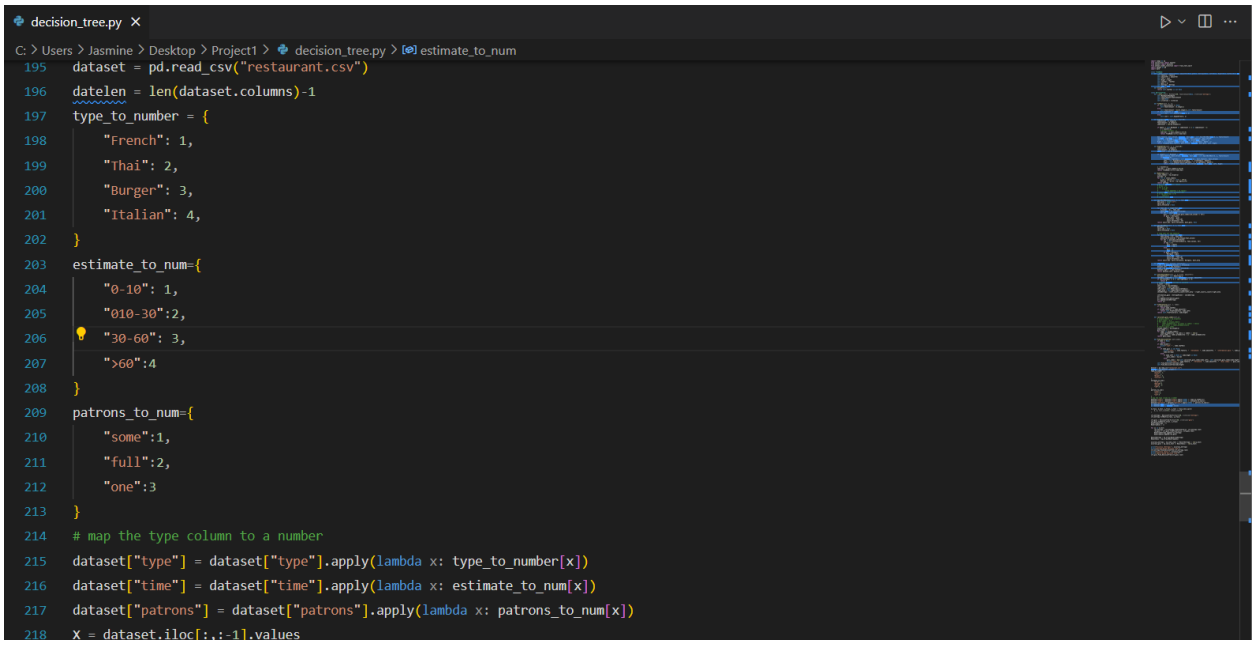
در این داک به توضیح دو روش تقریبا مشابه که درخت تصمیم را پیاده سازی شده میپردازم.

کد اول:

در مرحله اول داده های خود را به دو دسته train و test تقسیم میکنیم. ورودی به صورت (x,y) است که x برداری از مجموعه ویژگی ها و y مقدار خروجی کد به صورت 0 یا 1 است.

ما داده های خود را به عدد مپ میکنیم تا بتوانیم آن ها را در فرمول آنتروپی و جینی قرار دهیم.

خروجی نهایی برنامه، یک درخت تصمیم است که از root چاپ میشود. همچنین برگ بودن یا نبودن آن همراه با آنتروپی و information gain آن در خروجی چاپ میشود.



توضیح کد:

ابتدا کتابخانه های مورد استفاده مثل panda و numpy را import میکنیم. سپس کلاس های مربوط به درخت ها را پیاده میکنیم که مقدارهایی همچون مقدار درصورت برگ بودن، نام feature و entropy و gini دارند. در Decision Tree، تصمیم نهایی برمبنای ریشه درخت، ماکزیمم عمق و تعداد ویژگی های ذخیره شده است.

```python
from collections import Counter
from sklearn import datasets
from sklearn.model_selection import train_test_split
import pandas as pd
import math

class TreeNode:
    def __init__(self, feature=None, amountThr=None,g=None, Entropy=None, Left=None, Right=None,leafVal=None,gini=None):
        self.feature = feature
        self.amountThr = amountThr
        self.Left = Left
        self.Right = Right
        self.leafVal = leafVal
        self.gain = g
        self.Entropy = Entropy
        self.gini = gini
    def IsLeaf(self):
        return self.leafVal is not None
```

```python
class DecisionTree:
    def __init__(self, MaxDepth=50, featureCount=None, criterion='entropy'):
        self.MaxDepth=MaxDepth
        self.featureCount=featureCount
        self.root=None
        self.criterion = criterion
```

توابع مربوط به ساختن درخت نیز در Decision Tree پیاده شده اند.

Expand Tree:

X.shape مقدار i وj یک تابع دو بعدی را نشان میدهد. که در این جا به ترتیب تعداد ردیف ها یا sampleهایی که موجود است را x.shape[0] و x.shape[1] تعداد featureها را نشان میدهد.

تابع را به صورت بازگشتی صدا میزنیم و نیاز است شرط اتمام بازگشتی بودن را مشخص کنیم. سه شرط زیر را برای اینکه گره دیگر تقسیم نشود و ادامه پیدا نکند یا به برگ تبدیل شود، داریم:

1. به مقدار max depth برسیم و دیگر قادر نباشیم درخت را از این بزرگتر کنیم(که البته در اینجا بعید است.)
2. تعداد lableهای آن یک باشد.(یعنی همه خروجی ها یا 0 باشند یا 1)
3. تعداد سمپل ها کمتر از 2 باشد.

```
def ExpandTree(self, X, y, depth=0):
    sampleCount = X.shape[0]
    featureCount = X.shape[1]
    LableCount = len(np.unique(y))

    if (depth<=self.MaxDepth or LableCount!=1 or 2<sampleCount):
        best_feature, best_thresh, Bestgain, best_entp = self.SplitWithBest(X, y, featureCount)
        if(Bestgain>0):
            leftIdxs, rightIdxs = self.Splitnode(X[:, best_feature], best_thresh)
            Left = self.ExpandTree(X[leftIdxs, :], y[leftIdxs], depth+1)
            Right = self.ExpandTree(X[rightIdxs, :], y[rightIdxs], depth+1)
            return TreeNode(best_feature, best_thresh,Bestgain, best_entp, Left, Right)

    c = Counter(y)
    leaf_Val = c.most_common(2)[0][0]
    return TreeNode(leafVal=leaf_Val)
```

به هرگره باید یک لیبل اختصاص داده شود. اگر فقط یک سمپل داشته باشیم که لیبل گره همان است. اگر به ماکزیمم عمق هم رسیده باشیم و تعدادی 0 و 1 داشته باشیم، باید بین آن ها ماکزیمم گرفته شود و آن را به عنوان لیبل انتخاب میکنیم. مقدار لیبل با تابع most common تعیین میشود. در اخر تابع را به صورت بازگشتی برای فرزندان گره صدا خواهیم زد.

Entropy:

چون برای ما در آنتروپی بسیار اهمیت دارد که کدام ویژگی اول انتخاب شود، معمولا ویژگی های مهمتر را باید در گره های اولیه بررسی کنیم. برای این امر، از تابع آنتروپی استفاده میکنیم. این تابع، پیاده سازی دستی فرمول آنتروپی است که در مرحله اول تعداد yes و noهای خروجی را تعیین میکنیم و سپس با فرمول آنتروپی را محاسبه میکنیم.

```
def MyEntropy(self, y):
    class_labels = np.unique(y)
    entropy = 0
    for cls in class_labels:
        p_cls = len(y[y == cls]) / len(y)
        entropy += -p_cls * np.log2(p_cls)
    return entropy
```

در مرحله بعد، ابتدا آنتروپی را برای والد حساب میکنیم و سپس گره split میشود. سپس آنتروپی برای فرزندان گره نیز محاسبه میشود و دراخر یک آنتروپی وزن دار برای تمام فرزندان به دست می آوریم. با کم کردن این دو مقدار از یکدیگر، مقدار information gain را می یابیم. در اخر نیز مقدار آنتروپی به دست آمده از اطلاعات را برمیگردانیم:

```
def InformationGain(self, y, X_column, amountThr):
    EntropyParent = self.MyEntropy(y)
    leftIdxs, rightIdxs = self.Splitnode(X_column, amountThr)
    if len(leftIdxs) == 0 or len(rightIdxs) == 0:
        return 0
    # weighted avgerage entropy of children
    y_count = len(y)
    left_count = len(leftIdxs)
    right_count = len(rightIdxs)
    left_entp = self.MyEntropy(y[leftIdxs])
    right_entp = self.MyEntropy(y[rightIdxs])
    childEntropy = (left_count/y_count)*left_entp + (right_count/y_count)*right_entp

    information_gain = EntropyParent - childEntropy
    arr = []
    arr.append(information_gain)
    arr.append(childEntropy)
    return arr
```

SplitWithBest and SplitNode:

این دو تابع براساس بهترین ویژگی، گره را split میکند. برای این منظور، ستون های ما که همان ویژگی های ما هستند را بررسی میکنیم و بهترین gain را پیدا میکنیم. با تابع SplitNode فرزندان چپ و راست گره را براساس

مقدار threshold تعیین میکنیم. تابع argwhere ایندکس عناصری را میدهد که شرط داده شده را رعایت میکنند و در آخر آن ها را برمیگرداند.

```
def SplitWithBest(self, X, y, feat_idxs):
    Bestgain = -1
    split_idx = None
    split_threshold = None

    # loop over all the features
    for feat_idx in range(feat_idxs):
        feat_values = X[:, feat_idx]
        possible_thresholds = np.unique(feat_values)
        for thr in possible_thresholds:
            tmp = self.InformationGain(y, feat_values, thr)
            if tmp != 0:
                gain = tmp[0]
                entp = tmp[1]
            else:
                gain = 0
                entp = 0
            if gain > Bestgain:
                Bestgain = gain
                best_entp = entp
                split_idx = feat_idx
                split_threshold = thr
    return split_idx, split_threshold, Bestgain, best_entp
```

```
def Splitnode(self, dataset, threshold):
    l_idx = np.argwhere(dataset <= threshold)
    dataset_left = l_idx.flatten()
    r_idx = np.argwhere(dataset > threshold)
    dataset_right = r_idx.flatten()
    return dataset_left, dataset_right
```

FindDecisionTree and TraverseTree:

در تابع اول درخت را با یک ورودی به اسم x، از root شروع کرده و آن را طی میکنیم. این تابع را به طور بازگشتی روی فرزندان هر گره نیز صدا میزنیم. شرط اتمام بازگشتی بودن هر گره، برگ بودن آن است. اگر گره برگ باشد، باید مقدار آن را return کینم. تابع دیگر برای پرینت کردن درخت در کنسول است. در این تابع مجددا از ریشه درخت شروع میکنیم و گره ها را بررسی میکنیم و اگر برگ نبود و به انتهای درخت نیز نرسیدیم، تابع را به صورت بازگشتی مجددا روی فرزندان صدا میزنیم. برای هر گره مقدار threshold و ویژگی گره و آنتروپی آن را چاپ میکنیم. اگر هم به برگ رسیدیم فقط مقدار value آن را چاپ میکنیم. شرط اتمام نیز رسیدن به نود none است که یعنی درخت را کامل طی کردیم و زمان خارج شدن از تابع است.

```
def TreeTraverse(self, x, node):
    if node.IsLeaf():
        return node.leafVal
    if x[node.feature] <= node.amountThr:
        return self.TreeTraverse(x, node.Left)
    return self.TreeTraverse(x, node.Right)
```

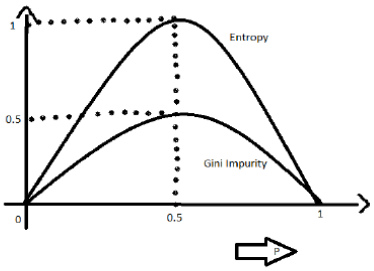
```
def Find_DecisionTree (self,node):
    if node == None:
        return
    if node.IsLeaf():
        print("-leaf: " , node.leafVal)
    else:
        if node.gain is not None:
            print("-feat: ", node.feature, "  -threshold: ", node.amountThr, "  -information gain: ", node.gain, "  -entropy: ",
            node.Entropy)
        else:
            if node.Left is None or node.Right is None:
                gini_index = np.nan
            else:
                gini_index = max(self.calculate_gini_index(node.Left), self.calculate_gini_index(node.Right))
                print("-feat: ", node.feature, "  -threshold: ", node.amountThr, "  -Gini index: ", gini_index)
    self.Find_DecisionTree(node.Left)
    self.Find_DecisionTree(node.Right)
```

Gini Index:

هدف جینی ایندکس همانند انتروپی، کاهش ناخالصی ها از گره های ریشه به گره های تصمیم است. حداقل مقدار شاخص جینی 0 است و زمانی اتفاق می افتد که گره خالص باشد. پس این گره دوباره تقسیم نخواهد شد.

تفاوت میان شاخص جینی و آنتروپی:

دوتفاوت اصلی میان آنتروپی و جینیی ایندکس در این است که شاخص آنتروپی در مقادیر0-1 است و این در جینی ایندکس در بازه 0-0.5 است. از نظر محسباتی فرمول آنتروپی پیچیده تر است و محاسبه شاخص جینی آسانتر و سریعتر است.



$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

در برنامه من مشابه تمام توابع بالا برای آنتروپی، برای شاخص جینی نیز درست شده است. تفاوت در محاسبه مقدار این شاخص است. البته متاسفانه کد بنده در قسمت جینی ایندکس به درستی کار نمیکند و از دقت پایینی برخوردار است.

در قسمت های اخر برنامه، فایل دارای اطلاعات خوانده شده و به هر کدام از استرینگ ها به یک عدد مپ میشوند تا بتوان از آن ها در فرمول های جینی و آنتروپی استفاده کرد.

```
type_to_number = {
    "French": 1,
    "Thai": 2,
    "Burger": 3,
    "Italian": 4,
}
estimate_to_num={
    "0-10": 1,
    "010-30":2,
    "30-60": 3,
    ">60":4
}
patrons_to_num={
    "some":1,
    "full":2,
    "one":3
}
# map the type column to a number
dataset["type"] = dataset["type"].apply(lambda x: type_to_number[x])
dataset["time"] = dataset["time"].apply(lambda x: estimate_to_num[x])
dataset["patrons"] = dataset["patrons"].apply(lambda x: patrons_to_num[x])
X = dataset.iloc[:,:-1].values
y = dataset.iloc[:, datelen].values
```

در نهایت توابع توضیح داده شده صدا میشوند و دقت هرکدام از روش ها چاپ میشود. بدیهی است که با تعداد train بیشتر، نتیجه test نیز دقیق تر خواهد بود.

نتیجه نهایی برنامه:

```
Accuracy (Entropy): 0.8
Entropy-based Decision Tree:
-feat:  5   -threshold:  1   -information gain:  0.5216406363433185   -entropy:  0.46358749969093305
-leaf:  1
-feat:  0   -threshold:  3   -information gain:  0.31127812445913283   -entropy:  0.5
-feat:  0   -threshold:  1   -information gain:  1.0   -entropy:  0.0
-leaf:  0
-leaf:  1
-leaf:  0
Accuracy (Gini): 0.4

Gini-based Decision Tree:
-feat:  0   -threshold:  0   -information gain:  1.0   -entropy:  None
-leaf:  1
-feat:  0   -threshold:  1   -information gain:  1.0   -entropy:  None
-leaf:  0
-feat:  0   -threshold:  3   -information gain:  1.0   -entropy:  None
-leaf:  1
-feat:  0   -threshold:  5   -information gain:  1.0   -entropy:  None
-leaf:  1
-feat:  0   -threshold:  7   -information gain:  1.0   -entropy:  None
-leaf:  1
-leaf:  0
```

روش دوم:

در این روش که در ارائه مفصل توضیح تر توضیح داده خواهد شد، تمامی موارد مشابه است تنها تفاوت در پیاده سازی برخی توابع است. البته این کد جزو نمونه اولیه بوده و به طور کامل کار نمیکند.