احسان اسپندار – ۹۹۴۴۲۰۱۱ – توضیحات پروژه درخت تصمیم – هوش مصنوعی – دکتر عبدی

تجزیه و تحلیل دیتاست

به منظور عملکرد صحیح الگوریتم ابتدا باید داده ها تمیز می شدند که در راستای آن اقدامات زیر انجام شد :

- ستون های contact,pdays,poutcome,campaign,previous شامل مقادیر زیادی داده ی مفقودی بودند اما سایر ستون های تاپل های متناظر آنان دارای اطلاعات درست بودند که از آنجا نتیجه گرفتم به منظور کم شدن پیچیدگی درخت این ستون ها را از داده اصلی حذف کنم.
- داده های ستون age گاه دارای مقادیر منفی و گاه دارای مقادیر نامعقول مانند ۹۹۹ بودند که تاپل های مربوط به آن ها را حذف کردم.
- ستون id به دلیل داشتن مقادیر یکتا در محاسبه ی info gain درخت تداخل ایجاد میکرد و باعث میشد درخت به سمت تعداد شاخه های بی شمار متمایل شود پس این ستون نیز حذف شد.
- در نهایت به دلیل تعداد بسیار زیاد متغیر های هدف با مقدار no که حدود ۹۰ درصد متغیر هدف را تشکیل میداد برخی از سطر های جدول را بصورت رندوم حذف کردم که درصد no و yes به ترتیب تقریبا به ۶۶ و ۳۳ درصد برسد.
- داده های پیوسته نیز به ۵ بازه ی گسسته تبدیل شدند.

کد مربوط به این قسمت :

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split

# Load data
col_names =
['id','age','job','material','education','default','balance','housing','loan','co
ntact','day','month','duration','campaign','pdays','previous','poutcome','y']
data = pd.read_csv("data.csv", skiprows=1, header=None, names=col_names)
```

```
# Data preprocessing
data = data.drop(['contact', 'pdays', 'poutcome'], axis=1)
data = data[data['age'] > 0]
data = data.drop(['previous', 'id'], axis=1)

# Balance the dataset
label_counts = data['y'].value_counts()
num_yes = label_counts['yes']
num_no = label_counts['no']

if num_no > 2 * num_yes:
    no_samples_to_keep = 2 * num_yes
    no_indices_to_keep = data[data['y'] == 'no'].sample(n=no_samples_to_keep,
random_state=42).index
    data_balanced = data.loc[no_indices_to_keep.union(data[data['y'] ==
'yes'].index)]
else:
    data_balanced = data

# Shuffle the dataset
data_balanced =
data_balanced.sample(frac=1,random_state=42).reset_index(drop=True)

continuous_columns = ['age', 'balance', 'day', 'duration']
num_bins = 5
categorical_columns = ['c_age', 'c_balance', 'c_day', 'c_duration']

for i, col in enumerate(continuous_columns):
    bins = pd.qcut(X[col], num_bins, labels=False, duplicates='drop')
    X[categorical_columns[i]] = pd.Categorical(bins, categories=range(num_bins))
X.drop(continuous_columns, axis=1, inplace=True)
```

پیاده سازی الگوریتم درخت تصمیم:

بعد از پاکسازی داده ها با توجه به تعداد زیاده داده و وقت گیر بودن آن، تصمیم گرفتم به جای الگوریتم ID3 که برای درخت های با شاخه های غیر دودویی نیز کاربرد دارد از الگوریتم CART استفاده کنم که در آن

درخت بصورت دودویی ساخته می شود و در هر مرحله مقادیر متغیر مورد نظر بر اساس مقدار متغیر که اطلاعات بیشتری بدست میدهد سنجیده میشوند و به دو دسته تقسیم میشوند.

در این الگوریتم کلاس ها و توابع زیر برای کاربرد ذکر شده پیاده سازی شدند :

- کلاس Node : به منظور ذخیره اطلاعات گره های درخت
- کلاس DecisionTree : به منظور ذخیره ساختار یک درخت تصمیم و پیاده سازی متد های مورد نیاز در آن از جمله :
  - __init__ : که کنستراکتور کلاس است و ویژگی های اولیه را به محض فراخوانی کلاس به آن اضافه میکند.
  - Build : به منظور fit کردن درخت با استفاده از داده های موجود.
  - SplitChoice : به منظور انتخاب فیچر مناسب و ترشهولد مناسب برای شکستن گره به گره های چپ و راست بر اساس معیار مناسب.
  - split : به منظور شکستن گره به گره های زیرین چپ و راست.
  - InfoGain : به منظور ارزیابی فیچر های در دسترس برای دستیابی به اطلاعات بیشتر بعد از شکسته شدن گره فعلی بر اساس مقدار آنتروپی یا جینی.
  - Entropy,Gini : برای محاسبه مقادیر آنتروپی و جینی به منظور استفاده از آنها در انتخاب تقسیم بندی.
  - Leaf : برای انجام عمل رای گیری در برگ ها.
  - ShowTree : به منظور پرینت کردن و نمایش انتزاعی درخت تصمیم ایجاد شده.
  - Fit : به منظور مرج کردن متغیر های هدف و غیر هدف سپس فراخوانی Build .
  - Predict : به منظور فراخوانی تابع MakePrediction به ازای همه سطر های داده های تست.
  - MakePrediction : به منظور پیمایش درخت ساخته شده با استفاده از سطر از داده تست.

کد مربوطه :

```
class Node():
    def __init__(self, feature_index=None, threshold=None, left=None, right=None,
info_gain=None, value=None):
```

```
        self.feature_index = feature_index
        self.threshold = threshold
        self.left = left
        self.right = right
        self.info_gain = info_gain
        self.value = value

class DecisionTree():
    def __init__(self, min_samples_split=2, max_depth=2):
        self.root = None
        self.min_samples_split = min_samples_split
        self.max_depth = max_depth

    def Build(self, dataset, curr_depth=0):
        X, Y = dataset[:, :-1], dataset[:, -1]
        num_samples, num_features = np.shape(X)

        if num_samples >= self.min_samples_split and curr_depth <=
self.max_depth:
            best_split = self.SplitChoice(dataset, num_samples, num_features)
            if best_split["info_gain"] > 0:
                left_subtree = self.Build(best_split["dataset_left"], curr_depth
+ 1)
                right_subtree = self.Build(best_split["dataset_right"],
curr_depth + 1)
                return Node(best_split["feature_index"], best_split["threshold"],
                            left_subtree, right_subtree, best_split["info_gain"])

        leaf_value = self.Leaf(Y)
        return Node(value=leaf_value)

    def SplitChoice(self, dataset, num_samples, num_features):
        best_split = {}
        max_info_gain = -float("inf")

        for feature_index in range(num_features):
            feature_values = dataset[:, feature_index]
            possible_thresholds = np.unique(feature_values)
            for threshold in possible_thresholds:
                dataset_left, dataset_right = self.split(dataset, feature_index,
threshold)
                if len(dataset_left) > 0 and len(dataset_right) > 0:
                    y, left_y, right_y = dataset[:, -1], dataset_left[:, -1],
dataset_right[:, -1]
                    curr_info_gain = self.InfoGain(y, left_y, right_y, "gini")
```

```
                    if curr_info_gain > max_info_gain:
                        best_split["feature_index"] = feature_index
                        best_split["threshold"] = threshold
                        best_split["dataset_left"] = dataset_left
                        best_split["dataset_right"] = dataset_right
                        best_split["info_gain"] = curr_info_gain
                        max_info_gain = curr_info_gain

        return best_split

    def split(self, dataset, feature_index, threshold):
        dataset_left = np.array([row for row in dataset if row[feature_index] <=
threshold])
        dataset_right = np.array([row for row in dataset if row[feature_index] >
threshold])
        return dataset_left, dataset_right

    def InfoGain(self, parent, l_child, r_child, mode="entropy"):
        weight_l = len(l_child) / len(parent)
        weight_r = len(r_child) / len(parent)
        if mode == "gini":
            gain = self.Gini(parent) - (weight_l * self.Gini(l_child) + weight_r
* self.Gini(r_child))
        else:
            gain = self.entropy(parent) - (weight_l * self.entropy(l_child) +
weight_r * self.entropy(r_child))
        return gain

    def entropy(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

    def Gini(self, y):
        class_labels = np.unique(y)
        gini = 0
        for cls in class_labels:
            p_cls = len(y[y == cls]) / len(y)
            gini += p_cls**2
        return 1 - gini

    def Leaf(self, Y):
```

```
        Y = list(Y)
        return max(Y, key=Y.count)

    def ShowTree(self, tree=None, indent="-"):
        if not tree:
            tree = self.root

        if tree.value is not None:
            print(tree.value)
        else:
            print("X_" + str(tree.feature_index), "=?", tree.threshold)
            print("%sleft:" % (indent), end="")
            self.ShowTree(tree.left, indent + indent)
            print("%sright:" % (indent), end="")
            self.ShowTree(tree.right, indent + indent)

    def Fit(self, X, Y):
        dataset = np.concatenate((X, Y), axis=1)
        self.root = self.Build(dataset)

    def Predict(self, X):
        return [self.MakePrediction(x, self.root) for x in X]

    def MakePrediction(self, x, tree):
        if tree.value is not None: return tree.value
        feature_val = x[tree.feature_index]
        if feature_val <= tree.threshold:
            return self.MakePrediction(x, tree.left)
        else:
            return self.MakePrediction(x, tree.right)
```

آموزش مدل

در این مرحله ابتدا داده ی کلی را به دو قسمت train و test و هرکدام را به دو قسمت X و Y تبدیل میکنیم. سپس با استفاده از داده ی train که ۸۰ درصد داده ی مارا تشکیل می دهد درخت را با ایجاد کلاس DecisoinTree و فراخوانی متد Fit می سازیم.

کد مربوطه :

```
X = data_balanced.iloc[:, :-1]
Y = data_balanced.iloc[:, -1].values.reshape(-1, 1)

X_train, X_test, Y_train, Y_test = train_test_split(X.values, Y, test_size=.2,
random_state=41)

DT = DecisionTree(min_samples_split=3, max_depth=3)
DT.Fit(X_train, Y_train)
```

ارزیابی مدل :

با توجه به معیار MSE یا میانگین مربعات خطا متوجه شدم که خطای پیشبینی در مدل پیاده شده بر روی داده های تست ۲۱٪ و دقت آن ۷۹٪ میباشد. هم چنین برای داده های آموزش این معیار دارای دقت ۸۰٪ است.

کد و تصاویر مربوطه :

```
def mean_squared_error(y_true, y_pred):
    """Compute the Mean Squared Error (MSE) between true and predicted values."""
    # Ensure the lengths of y_true and y_pred are the same
    if len(y_true) != len(y_pred):
        raise ValueError("Lengths of y_true and y_pred must be the same.")

    Y_test_df = pd.DataFrame(y_true, columns=['y'])
    Y_pred_df = pd.DataFrame(y_pred, columns=['y'])

    # Define a dictionary to map labels
    label_map = {"yes": 1, "no": 0}

    # Apply mapping to the labels using map function
    Y_test_df['y'] = Y_test_df['y'].map(label_map)
    Y_pred_df['y'] = Y_pred_df['y'].map(label_map)

    # Compute the squared differences between true and predicted values
```

```
    squared_errors = [(true - pred) ** 2 for true, pred in zip(Y_test_df['y'],
Y_pred_df['y'])]

    # Compute the mean of squared errors
    mse = sum(squared_errors) / len(y_true)

    return mse

mse = mean_squared_error(Y_test, Y_pred)
print("Mean Squared Error (MSE):", mse)
print('Accuracy  :',1-mse)
```

```
Mean Squared Error (MSE): 0.20901639344262296
Accuracy  : 0.790983606557377
```

نمایش درخت تصمیم

با توجه به اینکه درخت تصمیم را بدون استفاده از کتابخانه های سایکیت لرن و ... ساختیم استفاده کردن از کتابخانه برای نمایش آن کار مشکلی بود به همین دلیل آن را با استفاده از print و ترمینال نمایش دادم.

کد و تصویر مربوطه :

```
def ShowTree(self, tree=None, indent="-"):
        if not tree:
            tree = self.root

        if tree.value is not None:
            print(tree.value)
        else:
            print("X_" + str(tree.feature_index), "=?", tree.threshold)
            print("%sleft:" % (indent), end="")
            self.ShowTree(tree.left, indent + indent)
            print("%sright:" % (indent), end="")
            self.ShowTree(tree.right, indent + indent)

DT.ShowTree()
```

```
X_11 =? 3
-left:X_11 =? 1
--left:X_6 =? nov
----left:X_11 =? 0
--------left:no
--------right:no
----right:X_11 =? 0
--------left:no
--------right:yes
--right:X_4 =? no
----left:X_9 =? 0
--------left:no
--------right:yes
----right:X_6 =? nov
--------left:no
--------right:yes
-right:X_6 =? nov
--left:X_1 =? married
----left:X_6 =? dec
--------left:yes
--------right:yes
----right:X_4 =? no
--------left:yes
--------right:yes
--right:X_10 =? 0
----left:X_8 =? 3
--------left:yes
--------right:yes
----right:X_9 =? 2
--------left:yes
--------right:yes
```

در این نمایش متغیر های هدف با ایندکس آن ها در دیتاست نهایی نمایش داده شده اند. همچنین حداکثر عمق درخت ۳ و حدقل نمونه لازم برای تقسیم درخت ۳ میباشد.

تحلیل و بررسی نهایی

در پیاده سازی الگوریتم با استفاده از هر دو معیار gini و Entropy به ساخت درخت تصمیم پرداخته شد و در خطای نهایی تاثیر چندانی نداشت (کمتر از ۰.۰۱ درصد).

همچنین درخت تصمیم بر روی داده های آموزشی نیز تنها از ۱٪ دصت بیشتری برخوردار بود بنا براین میتوان گفت که درخت بیش برازش یا overfit نیز نداشته است و دقت قابل قبولی برخوردار است.
از آنجا که داده ها از کیفیت کافی برخوردار نبودند و برای از بین بردن اریب مجبور شدیم سطر و ستون هایی را

حذف کنیم بنظر من استفاده از داده مناسب تر به افزایش دقت ما منجر خواهد شد. همچنین میتوان مقادیر مختلف عمق درخت و مینیمم سمپل اسپلیت را نیز برای افزایش دقت امتحان کرد.

** توجه: برای اجرای برنامه کافی است فایل زیپ را استخراج کرده و فایل پایتون را بعد از نصب کردن پکیج های مربوطه اجرا کرد. برای تغییر ملاک gini و entropy کافی است پارامتر داخل تابع InfoGain را متناظر با این دو پارامتر عوض کنید.**