HW1 Solutions

Machine learning | winter 1400 | Dr.abdi

Teacher Assistant:

Zahra Dehghani

Student name : **Amin Fathi**

Student id : **400722102**

توضیح کد :

در این پروژه تنها از دو کتابخانه numpy و pandas استفاده می کنیم .

```
import numpy as np
import pandas as pd
```

بعد از امپورت کردن کتابخانه ها ، شروع به پیش پردازش داده مربوط به دیابت می کنیم .

ده نمونه اول را مشاهده می کنید در شکل زیر :

```
diabetes =  pd.read_csv("./diabetes.csv")
diabetes.head(10)
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0
8	2	197	70	45	543	30.5	0.158	53	1
9	8	125	96	0	0	0.0	0.232	54	1

داده ما به صورت پیوسته است و به همین جهت نیاز است که از گسسته سازی کنیم ، برای این کار از کتابخانه pandas استفاده کرده و با استفاده از pd.cut اقدام به این کار می کنیم .

برای این کار به شکل زیر عمل میکنیم

```
category_Glucose = ['0_Glucos_50', '50_Glucose_100', '100_Glucose_150', '150_Glucose_200' , '200_Glucose_250']
category_BloodPressure = ['0_BloodPressure_25','25_BloodPressure_50','50_BloodPressure_75','75_BloodPressure_100','100_BloodPressure_125','125_BloodPressure_150']
category_Pregnancies = ['0_Pregnancies_5','5_Pregnancies_10','10_Pregnancies_15','15_Pregnancies_20']
category_SkinThickness = ['0_SkinThickness_20','20_SkinThickness_40','40_SkinThickness_60','60_SkinThickness_80','80_SkinThickness_100','100_SkinThickness_120']
category_DiabetesPedigreeFunction = ['0_DiabetesPedigreeFunction_0.25','0.25_DiabetesPedigreeFunction_0.5','0.5_DiabetesPedigreeFunction_0.75','0.75_DiabetesPedigreeFunction_1','1_DiabetesPedigreeFunction_1.25',
category_Age= ['0_Age_20','20_Age_40','40_Age_60','60_Age_80','80_Age_100']
category_BMI= ['0_BMI_30','30_BMI_45','45_BMI_60','60_BMI_80']
category_Insulin= ['0_Insulin_50','50_Insulin_100','100_Insulin_150','150_Insulin_200','200_Insulin_250','250_Insulin_300','300_Insulin_350','350_Insulin_400','400_Insulin_500','500_Insulin_600','600_Insulin_700
diabetes['Glucose']=pd.cut(x=diabetes['Glucose'], bins=[-1,50,100,150,200,250],labels=category_Glucose)
diabetes['BloodPressure']=pd.cut(x=diabetes['BloodPressure'], bins=[-1,25,50,75,100,125,150] , labels=category_BloodPressure)
diabetes['SkinThickness']=pd.cut(x=diabetes['SkinThickness'], bins=[-1,20,40,60,80,100,120] , labels= category_SkinThickness)
diabetes['Age']=pd.cut(x=diabetes['Age'], bins=[-1,20,40,60,80,100] , labels= category_Age)
diabetes['Pregnancies']=pd.cut(x=diabetes['Pregnancies'], bins=[-1,5,10,15,20] , labels= category_Pregnancies)
diabetes['BMI']=pd.cut(x=diabetes['BMI'], bins=[-1,30,45,60,80] , labels= category_BMI)
diabetes['Insulin']=pd.cut(x=diabetes['Insulin'], bins=[-1,50,100,150,200,250,300,350,400,500,600,700,900] , labels= category_Insulin)
diabetes['DiabetesPedigreeFunction']=pd.cut(x=diabetes['DiabetesPedigreeFunction'], bins=[-1,0.25,0.5,0.75,1,1.25,1.5,1.75,2,2.25,2.5,2.75,3] , labels=category_DiabetesPedigreeFunction)
```

در واقع لیبل های گسسته ای که قصد داریم به جای لیبل های پیوسته فعلی قرار دهیم را ابتدا تعریف میکنیم در لیست هایی ، سپس ستون های مد نظر در دیتابیس را به قسمت هایی برابر با هم و متناظر با لیبل های جدید تقسیم میکنیم برای جا افتادن مطلب ، فرایند را به طور مثال برای ویژگی سن پیاده کنیم :

داده ما برای ۷۶۸ نفر تهیه شده است و سن آن ها در بازه ۲۱ تا ۸۱ سالگی دارد برای گسسته سازی ۵ بازه به طول مساوی در نظر میگیریم به صورت زیر :

ستون سمت چپ لیبل ها در حالت گسسته است و ستون سمت راست بازه مربوط به هر لیبل ، به طور مثال اگر سن یک بیمار برابر ۲۸ سال باشد ، از این به بعد به جای اینکه در ستون سن برچسپ ۲۸ برای او ثبت شده باشد ، برچسپ 20_Age_40 درج خواهد شد ، توجه شود که انتها باز ها بسته می باشند و مثلا سن ۲۰ سال جزو دسته اول به حساب می آید نه جزو دسته دوم.

0_Age_20	[0 , 20]
20_Age_40	(20 , 40]
40_Age_60	(40 , 60]
60_Age_80	(60 , 80]
80_Age_110	(80 , 100]

به همین ترتیب بقیه ویژگی ها را هم گسسته میکنیم ، ۱۰ داده اول این بار به شکل زیر خواهند بود که میتوانید آن را با تصویر حالت قبل از گسسته سازی که بالاتر درج شد مقایسه کنید .

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	5_Pregnancies_10	100_Glucose_150	50_BloodPressure_75	20_SkinThickness_40	0_Insulin_50	30_BMI_45	0.5_DiabetesPedigreeFunction_0.75	40_Age_60	1
1	0_Pregnancies_5	50_Glucose_100	50_BloodPressure_75	20_SkinThickness_40	0_Insulin_50	0_BMI_30	0.25_DiabetesPedigreeFunction_0.5	20_Age_40	0
2	5_Pregnancies_10	150_Glucose_200	50_BloodPressure_75	0_SkinThickness_20	0_Insulin_50	0_BMI_30	0.5_DiabetesPedigreeFunction_0.75	20_Age_40	1
3	0_Pregnancies_5	50_Glucose_100	50_BloodPressure_75	20_SkinThickness_40	50_Insulin_100	0_BMI_30	0_DiabetesPedigreeFunction_0.25	20_Age_40	0
4	0_Pregnancies_5	100_Glucose_150	25_BloodPressure_50	20_SkinThickness_40	150_Insulin_200	30_BMI_45	2.25_DiabetesPedigreeFunction_2.5	20_Age_40	1
5	0_Pregnancies_5	100_Glucose_150	50_BloodPressure_75	0_SkinThickness_20	0_Insulin_50	0_BMI_30	0_DiabetesPedigreeFunction_0.25	20_Age_40	0
6	0_Pregnancies_5	50_Glucose_100	25_BloodPressure_50	20_SkinThickness_40	50_Insulin_100	30_BMI_45	0_DiabetesPedigreeFunction_0.25	20_Age_40	1
7	5_Pregnancies_10	100_Glucose_150	0_BloodPressure_25	0_SkinThickness_20	0_Insulin_50	30_BMI_45	0_DiabetesPedigreeFunction_0.25	20_Age_40	0
8	0_Pregnancies_5	150_Glucose_200	50_BloodPressure_75	40_SkinThickness_60	500_Insulin_600	30_BMI_45	0_DiabetesPedigreeFunction_0.25	40_Age_60	1
9	5_Pregnancies_10	100_Glucose_150	75_BloodPressure_100	0_SkinThickness_20	0_Insulin_50	0_BMI_30	0_DiabetesPedigreeFunction_0.25	40_Age_60	1

در نهایت برای استفاده از داده گسسته شده ، آن را به حالت numpy array تبدیل می کنیم.

```
data_diabetes = diabetes.to_numpy(dtype='str')
```

درخت تصمیم

اولین تابعی که اینجا مورد استفاده قرار گرفته تابع زیر است ، هدف آن پیدا کردن مقادیر متفاوت فیچر ها در یک مجموعه داده است ، به طور مثال مقدار گلوسورول در کل داده ها پس از گسسته سازی از ۴ حالت ({'0_Glucos_50', '100_Glucose_150', '150_Glucose_200', '50_Glucose_100}) خارج نمی باشد

```
def unique(rows, col):
    #finding the unique values of attributes & reutruns dic
    return set([row[col] for row in rows])
```

```
unique(data_diabetes , 1)
```

```
{'0_Glucos_50', '100_Glucose_150', '150_Glucose_200', '50_Glucose_100'}
```

تابع دیگری که در ادامه معرفی می کنیم class counts است ، وظیفه این تابع شمارش و دسته بندی خروجی ها در قابل یک دیکشنری در دیتایی که به عنوان ورودی میگیرید است ، به طور مثال در کل داده ها ۵۰۰ نمونه لیبل خروجی ۰ دارند و ۲۶۸ داده لیبل خروجی ۱ . از این تابع در ادامه برای یافتن اکثریت لیبل در برگ ها یا برای دریافتن یک دست بودن نوع لیبل ها در شاخه استفاده خواهد شد

```
def class_counts(rows):
    counts = {}  # for saving the count of labels in dataset
    for row in rows:
        label = row[-1]
        if label not in counts:
            counts[label] = 0
        counts[label] += 1
    return counts
```

```
class_counts(data_diabetes)
```

```
{'0': 500, '1': 268}
```

تابع بعدی is numeric است ، این تابع وظیفه بررسی عدد بودن ورودی اش را به عهده دارد و اگر عدد باشد (چه int و چه float) خروجی True بر میگرداند و در غیر این صورت False ، این تابع در ادامه برای بررسی عدد بودن یا غیر عدد بودن مقدار ویژگی ها به کار خواهد رفت .

```
def is_numeric(value):
    # if datavalue is numeric return True , Else return False
    return isinstance(value, int) or isinstance(value, float)
```

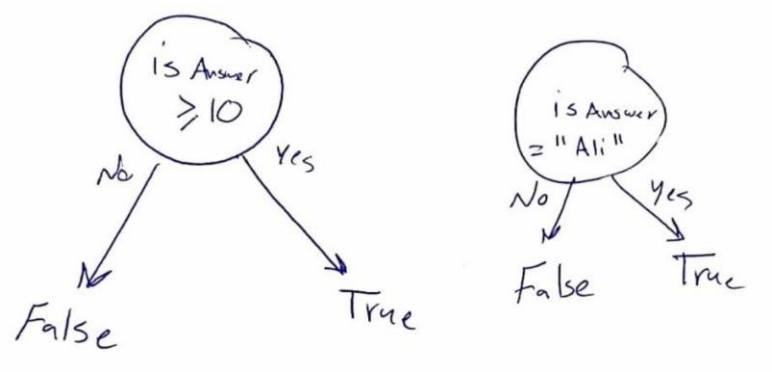
در ادامه به بررسی کلاس question می پردازیم :

در این کلاس ، ورودی column برای اشاره به اینکه سوال ما در مورد چه ویژگی است و value برای اشاره اینکه کدام مقدار از مقادیر برچسپ های ویژگی مد نظر را قرار است مورد سنجش قرار دهیم به کار میرود ، header هم برای استفاده در نمایش گرافیکی به کار خواهد رفت و به صورت زیر است

```
header = ["Pregnancies", "Glucose", "BloodPressure" , "SkinThickness" , "Insulin" , "BMI" , "DiabetesPedigreeFunction" , "Age" ]
```

```
class Question:
    # for matching question with values and showing it

    def __init__(self, column, value , header):
        self.column = column
        self.value = value
        self.header = header

    def match(self, example):
        # Compare the feature value in an example to the
        # feature value in this question.
        # if match , returns true
        #else returns flase
        val = example[self.column]
        if is_numeric(val):
            return val >= self.value
        else:
            return val == self.value

    def __repr__(self):
        condition = "=="
        if is_numeric(self.value):
            condition = ">="
        return "Is %s %s %s?" % (
            self.header[self.column], condition, str(self.value))
```

تابع match در این کلاس وظیفه چک کردن ویژگی داده با مقدار مد نظر صورت سوال را دارد ، ویژگی های داده از دو حالت خارج نیستند ، یا به صورت رشته هستند یا مقدار عددی دارند خروجی تابع در هر دو حالت از دو قسم خارج نیست یا True یا False است . به طور مثال اگر ویژگی مد نظر عددی باشد ، و سوال این شکل باشد که " آیا ویژگی از ۱۰ بزگتر است ؟ " خروجی در صورتی که مقدار ویژگی از ۱۰ بیشتر باشد برابر True و در غیر این صورت False است . یا حالتی را فرض کنید که ویژگی مورد بحث عددی نیست و میخاهیم بدانیم ایا ویژگی مورد نظر (مثلا اسم) برابر "ali" است یا خیر ، اگر دقیقا ali بود که True بر میگردانیم و در غیر این صورت False ، این خروجی True و False در ادامه برای دو دسته کردن داده در گره والد به کار خواهد رفت



تابع __repr__ در این کلاس برای نمایش صورت سوال در نمایش حالت گرافیکی درخت به کار خواهد رفت .

حال به سراغ تابع بعدی می رویم ، تابع devide می رویم ، این تابع داده ورودی در گره والد را به همراه اطلاعات سوالی که قرار است بر اساس آن ها گره والد را به دو دسته true و false دسته بندی کنیم به عنوان ورودی تحویل میگیرد و به ازای هر کدام از داده های ورودی چک میکند که ایا شرط های سوال گره والد را رعایت میکنند یا نه ، اگر رعایت کنند که در لیست true_rows ذخیره می کنیم و در غیر این صورت در لیست false_rows ذخیره خواهیم کرد و در نهایت این دو لیست را برای ادامه کار خروجی می دهد . (در واقع برای هر گره والد با توجه به سوال مد نظر در آن گره والد ، داده را به دو نود که یکی نماینده دسته ارضا کننده شرط سوال و دیگری نماینده دسته ارضا نکننده شرط سوال خواهد بود تبدیل کرده و خروجی میدهد)

```
def devide(rows, question):
    """for partitioning the dataset
    """
    true_rows, false_rows = [], []
    for row in rows:
        if question.match(row):
            true_rows.append(row)
        else:
            false_rows.append(row)
    return true_rows, false_rows
```

```python
def entropy(s):
    res = 0
    val, counts = np.unique(s, return_counts=True)
    freqs = counts.astype('float')/len(s)
    for p in freqs:
        if p != 0.0:
            res -= p * np.log2(p)
    return res
```

آنتروپی را مطابق اسلاید های درس به دست می آوریم .

برای به دست آوردن information gain هم ابتدا وزن هر دو نود فرزند را حساب کرده و سپس وزن هر کدام را در آنتروپی هر یک ضرب کرده و حاصل جمع را از آنتروپی گره والد کم میکنیم و به عنوان خروجی تحویل میدهیم .

```python
def info_gain( parent, l_child, r_child):
        w_left = len(l_child) / len(parent)
        w_right = len(r_child) / len(parent)
        gain = entropy(parent) - (w_left*entropy(l_child) + w_right*entropy(r_child))
        return gain
```

در ادامه به توضیح تابع find_best_split می پردازیم :

این تابع برای داده در هر گره ، بهترین سوال یا بهتر است بگوییم بهترین ویژگی برای طبقه بندی را پیدا میکند ، به این صورت که به ازای فیچر ها متفاوت و مقادیر منحصر به فرد در هر فیچر مقدار information gain را حساب میکند و مقدار منحصر به فردِ فیچری که بیشترین information gain را داشته باشد را به عنوا فیچر مناسب برای طبقه بندی آن گره به عنوان خروجی بر میگرداند .

```python
def find_best_split(rows):

    max_gain = 0
    best_question = None
    features_number = len(rows[0]) - 1  # number of feautures
    header = ["Pregnancies", "Glucose", "BloodPressure" , "SkinThickness" , "Insulin" , "BMI" , "DiabetesPedigreeFunction" , "Age" ]
    for col in range(features_number):  # for each feature
        values = set([row[col] for row in rows])  # unique values in each feature
        for val in values:  # for each value of that feature
            question = Question(col, val, header)

            #split dataset with question
            true_rows, false_rows = devide(rows, question)

            # if does not split skip the value of feature
            if len(true_rows) == 0 or len(false_rows) == 0:
                continue

            # Calculate the information gain from this value of feauture and check it with max_gain
            gain = info_gain(rows , true_rows, false_rows)
            if gain > max_gain:
                max_gain, best_question = gain, question

    return max_gain, best_question
```

لازم به ذکر است شرطی در تابع مذکور هست که چک میکند که آیا فیچری که مورد استفاده قرار دادیم قادر به طبقه بندی هست یا خیر همه داده های گره والد را به یکی از گره های فرزند انتقال داده و فرزند دیگر داده ای ندارد ، که در این صورت طبقه بندی زاید و اضافی است و از این فیچر صرف نظر میکنیم .

```
if len(true_rows) == 0 or len(false_rows) == 0:
    continue
```

کلاس های Leaf و Decision_Node برای ذخیره سازی اطلاعات مربوط به درخت به کار میروند ، در کلاس Leaf اطلاعات مربوط به برگ ها ذخیره میشود (به صورت دیکشنری که کلید های آن ۰ و ۱ هستند و ارزش کلید ها هم تعداد آن ۰ و ۱ هاست و در واقع نشان می دهد در آن برگ چند داده با برچسپ ۰ داریم و چند داده با برچسپ ۱)

در کلاس Decision_Node اطلاعات مربوط به گره ها را نگه می داریم ، اطلاعاتی شامل اینکه چه سوالی در آن گره پرسیده شده و فرزدانش بر اثر آن سوال کیستند ! از این دو کلاس در ادامه استفاده خواهد شد .

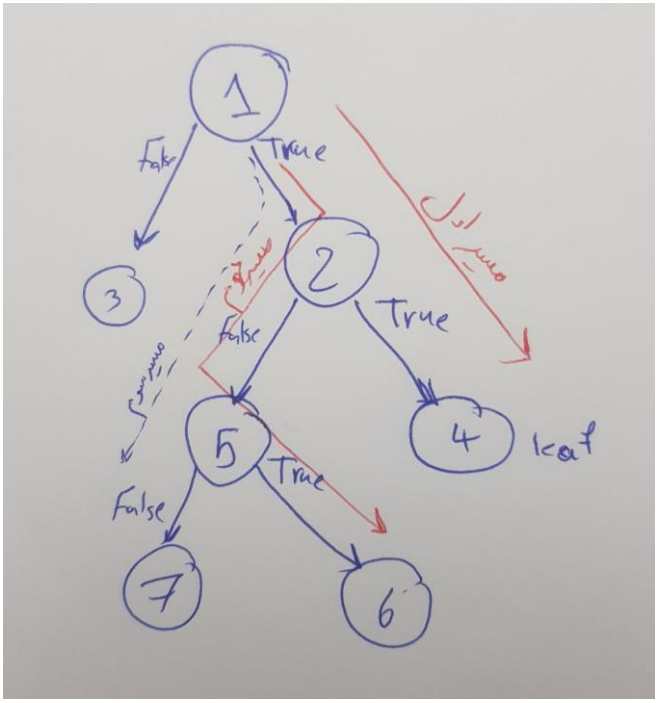
```
class Leaf:
    # saves the count and final results in leaf ( dect )
    def __init__(self, rows):
        self.predictions = class_counts(rows)

class Decision_Node:
    #A Decision Node asks a question.
    # This holds a reference to the question, and to the two child nodes.
    def __init__(self,question,true_branch,false_branch):
        self.question = question
        self.true_branch = true_branch
        self.false_branch = false_branch
```

در ادامه به توضیح تابع build tree می پردازیم ، همانطور که از نامش پیداست وظیفه آن ساخت درخت با استفاده از دو درخت قبلی است . در این تابع ابتدا بهترین فیچر و مقدار منحصر به فرد فیچر برای تهیه پرسش در گره والد به کمک تابع find_best_split برای مجموعه از داده ها در گره مد نظر پیدا میشود ، اگر information gain آن صفر بود که به معنای یکدست بودن داده یا نبود ویژگی برای دسته بندی داده است ، آن گره به عنوان یک برگ به کتابخانه Leaf ارسال می شود تا مقادیر خروجی داده در برگ در یک دیکشنری ثبت شود ، در غیر این صورت (اگر داده یکدست نباشد یا با مشکل نبود ویژگی نداشته باشیم) داده ها را به دو دسته تقسیم کرده و اطلاعات گره والد را در کتباخانه Decision_Node ذخیره کرده و همچنین با توجه به صدا زدن تابع در درون تابع برای داده ای که true بود به صورت حریصانه این عمل را تا آخرین شاخه سمت true تکرار میکنیم و اطلاعات گره ها را ثبت می کنیم ، با توجه به اینکه تابع ساخت درخت را برای دسته false ها هم

صدا کرده ایم ، بعد از اتمام دسته true ها و از آخر به اول اطلاعات مربوط به شاخه ها و برگ های false را هم با استفاده از آن دو کتابخانه ثبت میکنیم.

```
def build_tree(rows):

    # print('************row time **************')
    # print(rows)
    # print('**********question time **********')
    # Try partitioing the dataset on each of the unique attribute,
    # calculate the information gain,
    # and return the question that produces the highest gain.

    gain, question = find_best_split(rows)

    #info gain = 0 => it is Leaf !
    if gain == 0:
        return Leaf(rows)

    #else it is not leaf and we are in node
    true_rows, false_rows = devide(rows, question)

    # print(question)
    # print('type true rows' ,type(true_rows) )
    # print('-------')
    # print('type false_rows ' ,type(false_rows) )
    # print('******************split done build the true*************************')



    #  build the true branch of node .
    true_branch = build_tree(true_rows)


    #  build the false branch.
    false_branch = build_tree(false_rows)

    # Return a Question node and store it in decision node class
    return Decision_Node(question, true_branch, false_branch)
```

در واقع ترتیب فراخوانی توابع باعث شده است تا الگوریتم درخت حریصانه بازگشتی باشد ، به طور مثال در درخت زیر ابتدا اطلاعات مربوط به مسیر ۱ که همگی شاخه های true هستند ثبت میشود سپس مسیر دو که فقط در مرحله یکی مانده به آخر false هست و به همین ترتیب ...



تابع classify وظیفه یافتن تعلق داده به کدام برگ را دارد ، یعنی مشخص میکند که داده مد نظر ما در کدام برگ از برگ های درخت قرار خواهد گرفت ، ابتدا بخش دوم تابع اجرا میشود و داده در ساختار درخت پیمایش میشود تا if اول شامل حالش شود به یک برگ رسیده باشد که در این صورت دیکشنری برچسپ خروجی های آن برگ به عنوان خروجی اطلاق می شود .

```
def classify(row, node):

    # return the dict of node if we reach node
    if isinstance(node, Leaf):
        return node.predictions

    # trace the tree with example
    if node.question.match(row):
        return classify(row, node.true_branch)
    else:
        return classify(row, node.false_branch)
```

در تابع leaf_detection هم مشخص می شود که در هر برگ کدام برچسپ خروجی بیشترین مقدار را دارد و آن به عنوان برچسپ (به صورت رشته نه عدد) به عنوان خروجی آن برگ و برچسپ آن برگ معرفی می شود .

```
def leaf_detection(counts):
    #detect the lable of leaf
    res = 0
    for lbl in counts.keys():
      if int(counts[lbl]) > int(res) :
          res = lbl
    return str(res)
```

با اجرای درخت مذکور بر روی داده رستوران (فایل داده رستوران در پیوست آمده است) مشاهده می شود که تمامی داده ها را به درستی طبقه بندی کرده است

```
count10 = 0
count20 = 0


for row in data3:

    if row[-1] == leaf_detection(classify(row, my_tree10)):
      count10 = count10 + 1
    count20 = count20 + 1
print(count10 , count20)
print('accuracy :' , float(count10/ count20) * 100 , '%')
```

```
12 12
accuracy : 100.0 %
```

همانطور که در بالا مشاهده میشود ، دقت را بر اساس حاصل تقسیم تعداد داده درست پیشبینی شده با کمک درخت train داده شده بر تعداد کل داده ها به دست می آوریم ، لازم به ذکر است در داده رستوران ، از همه نمونه ها برای آموزش درخت استفاده کردیم و از همه داده ها هم برای test کردن درخت استفاده کردیم که حاصل دقت 100 درصدی بوده

حال به سراغ داده دیابت می رویم ، در داده دیابت ، دادگان را ۴ بار استفاده کردیم ، یک بار به صورتی که ۶۵ درصد دادگان را به عنوان داده آموزشی و مابقی دادگان برای تست کردن درخت ، یک بار ۸۰ درصد دادگان را به عنوان داده آموزشی و مابقی

دادگان برای تست کردن درخت آموزش دیده ، یک بار هم این نسبت ۹۰ به ۱۰ درصد است و در نهایت هم داده آموزشی و داده تست هر دو یکی هستند و هر دو کل داده ما هستند (بخش no split) که اطلاعات این ۴ قسمت در زیر آمده است :

type	No split	0.1 split	0.2 split	0.35 split
Accuracy	91.4 %	70.1%	66.9 %	68.7 %

چالش ها

مشاهده می شود با کاهش سهم دادگان آموزش ، درخت ضعیف تری train می کنیم ، همچنین چالش دیگری که با آن رو به رو بودیم لزوم شافل کردن داده هاست که میتواند درصد ها را کمی جا به جا کند ، همچنین چنانچه هم که تمام دادگان را برای آموزش درخت استفاده کنیم ، در نهایت هم باز خطا خواهیم داشت که به این معنی است که ویژگی هایی که ما داریم برای دسته بندی کامل (همانند حالت دادگان رستوران) کافی نیستند و شاید بهتر است نوع دیگری از گسسته سازی را انجام دهیم یا ویژگی های دیگری هم اضافه کنیم .

از چالش های دیگری که در این دو سوال با آن مواجه بودیم ، توانایی پیش بینی و trace کردن داده جدید در درختی که train شده است بود ، که با مشاهده نمونه کد های اماده در اینترنت توانستم ساختار درخت را مجددا به گونه ای طراحی کنم که بتوان به وسیله ان predict کرد (و نه فقط رسم درخت) ، از چالش های دیگر بنده این بود که چه بازه هایی برای گسسته سازی مناسب تر است ، مثلا در مورد بارداری ، مشخصا در مورد مردان بارداری ممکن نیست و نمیتوان دسته بارداری را به دسته هایی با طول برابر طبقه بندی کرد (مثلا مردی که باردار نبوده ! با زنی که یک بار باردار بوده را نمیتوان در یک دسته طبقه بندی کرد ! از لحاظ پزشکی)

نتایج چاپ درخت هم در کد ها آورده شده که با استفاده از دیتا های ذخیره شده در دو کلاس گره ها و برگ ها اقدام به چاپ کرده و به صورت زیر است (به دلیل طولانی بودن آن را اینجا نیاوردیم و در کد قابل مشاهده است ولی بخش از آن را در زیر آورده ایم)

```
Is SkinThickness == 0_SkinThickness_20?
=> True:
  Is BMI == 0_BMI_30?
  => True:
    Is Glucose == 50_Glucose_100?
    => True:
      Is Insulin == 0_Insulin_50?
      => True:
        Is DiabetesPedigreeFunction == 0.25_DiabetesPedigreeFunction_0.5?
        => True:
          Is BloodPressure == 50_BloodPressure_75?
          => True:
            Predict set is  {'0': 12}
          => False:
            Is Pregnancies == 0_Pregnancies_5?
            => True:
              Is BloodPressure == 75_BloodPressure_100?
              => True:
                Predict set is  {'0': 1}
              => False:
                Predict set is  {'0': 4}
            => False:
              Predict set is  {'1': 1}
        => False:
          Is Pregnancies == 0_Pregnancies_5?
          => True:
            Is DiabetesPedigreeFunction == 0_DiabetesPedigreeFunction_0.25?
            => True:
              Is BloodPressure == 50_BloodPressure_75?
              => True:
                Predict set is  {'0': 5}
              => False:
                Is BloodPressure == 0_BloodPressure_25?
                => True:
                  Predict set is  {'0': 3}
                => False:
                  Is BloodPressure == 25_BloodPressure_50?
                  => True:
                    Predict set is  {'0': 1}
```

در هر گره سوال مطرح میشود و چاپ می شود ، در هر برگ هم نوع و تعداد برچسپ خروجی نشان داده میشود.

در مورد بخش سوم متاسفانه به دلیل کمبود وقت قادر به اجرای آن نبودم .