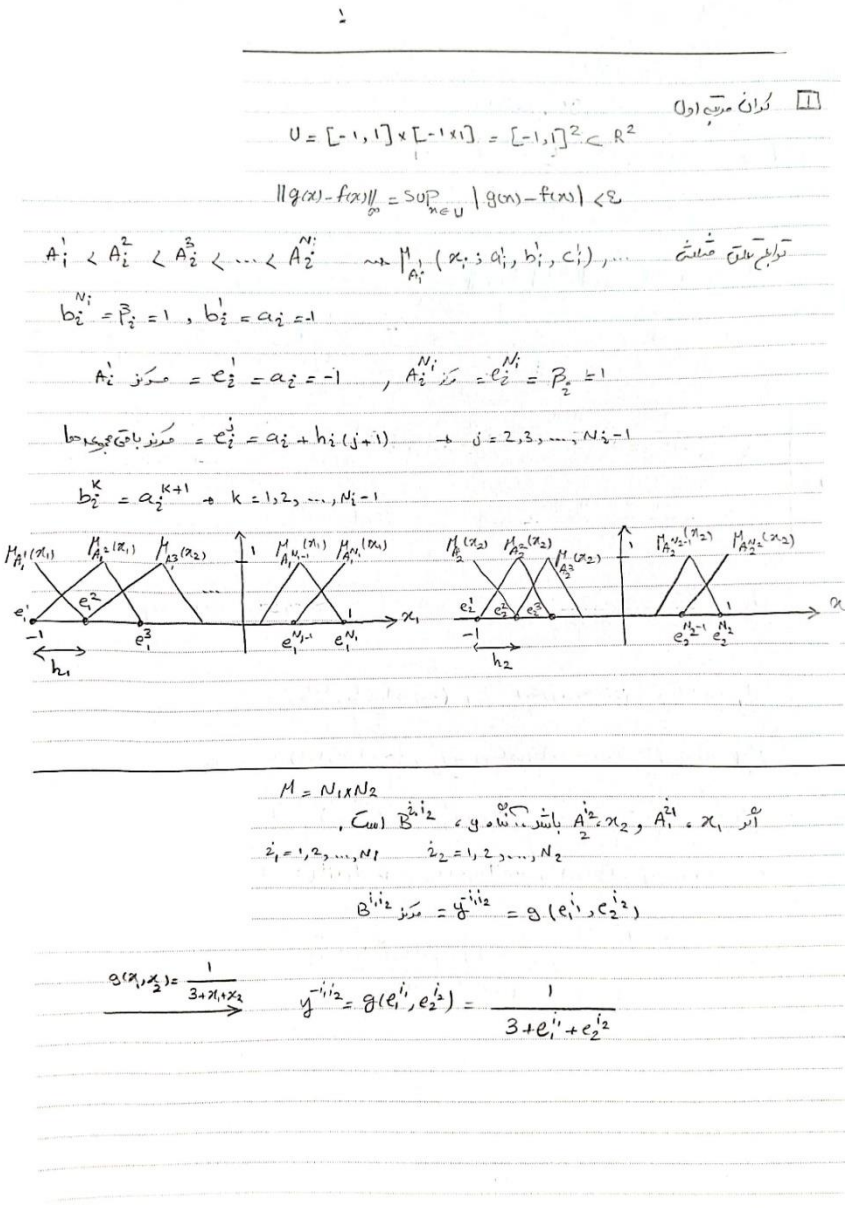
به نام خداوند بخشنده مهربان

درس مبانی سیستم‌های هوشمند

گزارش مینی پروژه سوم

معصومه شریف‌تبار عزیزی – ۹۹۲۷۶۱۳

۱.



۱

• موتور استنتاج ضرب، فازی ساز یکتایی، غیرفازی ساز میانگین مراکز

تعداد قواعد $N_1 \times N_2$ $\quad g(e_1^{i_1}, e_2^{i_2}) = \bar{y}^{i_1 i_2}$

$$f(x) = \frac{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \bar{y}^{i_1 i_2} \left[\mu_{A_1^{i_1}}(x_1) \, \mu_{A_2^{i_2}}(x_2) \right]}{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \left[\mu_{A_1^{i_1}}(x_1) \, \mu_{A_2^{i_2}}(x_2) \right]}$$

$$\| g(x) - f(x) \|_\infty \leq \left\| \frac{\partial g}{\partial x_1} \right\|_\infty h_1 + \left\| \frac{\partial g}{\partial x_2} \right\| h_2 < \varepsilon$$

$h_1 = h_2 = h$ فرض $\Rightarrow \quad h < \dfrac{\varepsilon}{\left\| \frac{\partial g}{\partial x_1} \right\|_\infty + \left\| \frac{\partial g}{\partial x_2} \right\|_\infty} \overset{(*)}{=} \dfrac{0.1}{1+1} = 0.05 \rightsquigarrow \boxed{h = 0.04}$

$$(*) \begin{cases} \dfrac{\partial g}{\partial x_1} = \dfrac{-1}{(3+x_1+x_2)^2} \xrightarrow{\| \|_\infty} = \left| \dfrac{-1}{(3-1-1)^2} \right| = 1 \\ \dfrac{\partial g}{\partial x_2} = \dfrac{-1}{(3+x_1+x_2)^2} \xrightarrow{\| \|_\infty} = \left| \dfrac{-1}{(3-1-1)^2} \right| = 1 \end{cases} \qquad \max \longrightarrow (x_1, x_2) = (-1, -1)$$

$$h = \frac{\beta - \alpha}{n} = \frac{1-(-1)}{n} = 0.04 \rightarrow n = 50 \quad \Rightarrow N_1 = N_2 = n+1 = \boxed{51}$$

تعداد توابع تعلق
(تعداد مجموعه‌های فازی A^m)

$$\mu_{A_1}(x) = \mu_{A_1}(x; a_1, b_1, c_1) = \mu_{A_1}(x; -1, -1, -1+h)$$

$$\mu_{A_{51}}(x) = \mu_{A_{51}}(x; a_{51}, b_{51}, c_{51}) = \mu_{A_{51}}(x; 1-h, 1, 1)$$

$$\mu_{A_j}(x) = \mu_{A_j}(x; a_j, b_j, c_j) = \mu_{A_j}(x; e^{j-1}, e^j, e^{j+1}) \quad ; j = 2, \ldots, 50$$

$$: e^j = a + h[j-1] = -1 + 0.04(j-1)$$

تعداد قواعد $N_1 \times N_2 = 51 \times 51 = 2601$

$$f(x) = \frac{\sum_{i_1=1}^{51} \sum_{i_2=1}^{51} g(e_1^{i_1}, e_2^{i_2}) \left[\mu_{A_1^{i_1}}(x_1) \, \mu_{A_2^{i_2}}(x_2) \right]}{\sum_{i_1=1}^{51} \sum_{i_2=1}^{51} \left[\mu_{A_1^{i_1}}(x_1) \, \mu_{A_2^{i_2}}(x_2) \right]}$$

• کران مرتبه دوم

$$\|g(x)-f(x)\|_\infty \leq \frac{1}{8}\left[\left\|\frac{\partial^2 g}{\partial x_1^2}\right\|_\infty h_1^2 + \left\|\frac{\partial^2 g}{\partial x_2^2}\right\| h_2^2\right] < \varepsilon$$

$$\xrightarrow{h_1=h_2=h} h^2 < \frac{8\varepsilon}{\left\|\frac{\partial^2 g}{\partial x_1^2}\right\|_\infty + \left\|\frac{\partial^2 g}{\partial x_2^2}\right\|_\infty} \rightarrow h < \sqrt{\frac{8\varepsilon}{\left\|\frac{\partial^2 g}{\partial x_1^2}\right\|_\infty + \left\|\frac{\partial^2 g}{\partial x_2^2}\right\|_\infty}} = \sqrt{\frac{2\times 0.1}{2+2}} = 0.4472$$

$$\rightarrow \boxed{h = 0.25} = \frac{\beta-\alpha}{n} = \frac{1-(-1)}{n} \rightarrow n = 8 \Rightarrow N_1 = N_2 = n+1 = 9$$

$$\mu_{A^1}(x) = \mu_{A^1}(x; a_1, b_1, c_1) = \mu_{A^1}(x; -1, -1, -1+h)$$

$$\mu_{A^9}(x) = \mu_{A^9}(x; a_9, b_9, c_9) = \mu_{A^9}(x; 1-h, 1, 1)$$

$$\mu_{A^j}(x) = \mu_{A^j}(x; a_j, b_j, c_j) = \mu_{A^j}(x; e^{j-1}, e^{j}, e^{j+1}) \quad ; j = 2, \dots, 8$$

$$e^j = \alpha + h[j-1] = -1 + 0.25(j-1) \qquad N_1 \times N_2 = 81$$

$$f(x) = \frac{\sum_{i_1=1}^{9} \sum_{i_2=1}^{9} g(e_1^{i_1}, e_2^{i_2}) \left[\mu_{A_1^{i_1}}(x_1)\mu_{A_2^{i_2}}(x_2)\right]}{\sum_{i_1=1}^{9} \sum_{i_2=1}^{9} \left[\mu_{A_1^{i_1}}(x_1)\mu_{A_2^{i_2}}(x_2)\right]}$$

با کران دوم محاسبات خیلی کمتر می شود.

برنامه نمایش سیستم فازی با پایتون برای کران مرتبه اول

trimf (x, abc) : تابعی برای تعریف یک تابع سه‌گانه فازی مثلثی.

این تابع مثلث فازی با استفاده از پارامترهای ورودی x و abc محدوده مشخصی از مقادیر x را به شکل یک مثلث در بازه مشخص شده ایجاد می‌کند. در این تابع، ابتدا فاصله نسبی x از ابتدای و یا انتهای محدوده مثلث محاسبه شده و سپس مقدار فازی بر اساس این فاصله محاسبه می‌شود. اگر x در خارج محدوده مثلث باشد، مقدار فازی صفر خواهد بود.

آرایه‌های g_bar و e_i2, e_i1 برای ذخیره مقادیر مربوط به توابع فازی هستند.

با استفاده از حلقه‌های تو در تو، توابع فازی mu_A_x1 و mu_A_x2 برای هر نقطه از فضای ورودی محاسبه می‌شوند.

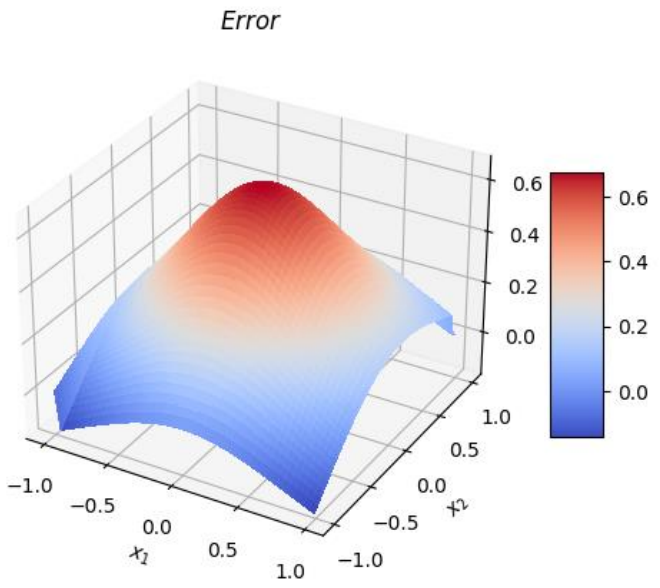
مقادیر g_bar برای هر نقطه محاسبه می‌شوند و در نهایت، تابع f_x با تقسیم جمع‌آوری شده مقادیر num بر تقسیم جمع‌آوری شده مقادیر den محاسبه می‌شود.

توابع g_x و f_x ترسیم شده و خطای محاسبه شده (E) به عنوان نمودار سه‌بعدی نشان داده می‌شود. (نمودار سه‌بعدی نشان می‌دهد که چقدر توابع g_x و f_x از یکدیگر اختلاف دارند.)

```
import time
start_time = time.time()
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import warnings
warnings.filterwarnings('ignore')

alpha = -1
beta = 1
h = 0.04
N = 51

x1 = np.arange(alpha, beta, 0.01)
x2 = np.arange(alpha, beta, 0.01)
x1, x2 = np.meshgrid(x1, x2)

g_bar = np.zeros((N*N, 1))
e_i1 = np.zeros((N, 1))
e_i2 = np.zeros((N, 1))

num = 0
den = 0
k = 0

def trimf(x, abc):
    return np.fmax(np.fmin((x-abc[0])/(abc[1]-abc[0]), (abc[2]-x)/(abc[2]-abc[1])), 0)
```

```
for i1 in range(1,N):
    for i2 in range(1,N):
        e_i1[i1-1,0] = -1 + h*(i1-1)
        e_i2[i2-1,0] = -1 + h*(i2-1)
        if i1==1:
            mu_A_x1 = trimf(x1, [-1,-1,-1+h])
        elif i1==N:
            mu_A_x1 = trimf(x1,[1-h, 1, 1])
        else:
            mu_A_x1 = trimf(x1,[-1+h*(i1-2), -1+h*(i1-1), -1+h*(i1)])

        if i2==1:
            mu_A_x2 = trimf(x2, [-1,-1,-1+h])
        elif i2==N:
            mu_A_x2 = trimf(x2,[1-h, 1, 1])
        else:
            mu_A_x2 = trimf(x2,[-1+h*(i2-2), -1+h*(i2-1), -1+h*(i2)])

        g_bar[k,0]= 1/(3+e_i1[i1, 0]+e_i2[i2, 0])
        num = num + g_bar[k,0]*mu_A_x1*mu_A_x2
        den=den+mu_A_x1*mu_A_x2
        k=k+1

f_x = num/den
g_x = 1/(1+x1**2+x2**2)

fig = plt.figure()
#ax = fig.gca(projection='3d')
ax = fig.add_subplot(projection = '3d')
E = g_x - f_x
surf = ax.plot_surface(x1, x2, E, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$Error$')
ax.set_title('$Error$')
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.savefig('fuzzy1.svg')
plt.show()
```

خروجی:



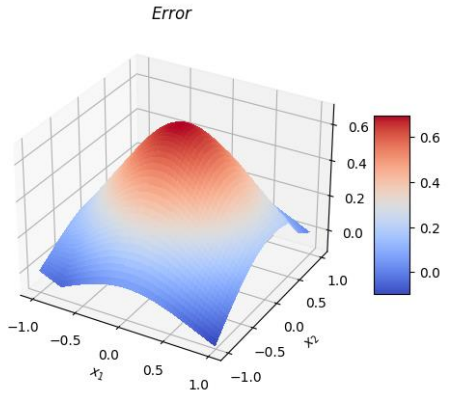
برنامه نمایش سیستم فازی با پایتون برای کران مرتبه دوم

* تفاوت با برنامه قبلی فقط در مقادیر نشان داده‌شده است.

```
import time
start_time = time.time()
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import warnings
warnings.filterwarnings('ignore')

alpha = -1
beta = 1
h = 0.25
N = 9

x1 = np.arange(alpha, beta, 0.01)
x2 = np.arange(alpha, beta, 0.01)
x1, x2 = np.meshgrid(x1, x2)

g_bar = np.zeros((N*N, 1))
e_i1 = np.zeros((N, 1))
e_i2 = np.zeros((N, 1))

num = 0
den = 0
k = 0

def trimf(x, abc):
    return np.fmax(np.fmin((x-abc[0])/(abc[1]-abc[0]), (abc[2]-x)/(abc[2]-abc[1])), 0)
```

```
for i1 in range(1,N):
    for i2 in range(1,N):
        e_i1[i1-1,0] = -1 + h*(i1-1)
        e_i2[i2-1,0] = -1 + h*(i2-1)
        if i1==1:
            mu_A_x1 = trimf(x1, [-1,-1,-1+h])
        elif i1==N:
            mu_A_x1 = trimf(x1,[1-h, 1, 1])
        else:
            mu_A_x1 = trimf(x1,[-1+h*(i1-2), -1+h*(i1-1), -1+h*(i1)])

        if i2==1:
            mu_A_x2 = trimf(x2, [-1,-1,-1+h])
        elif i2==N:
            mu_A_x2 = trimf(x2,[1-h, 1, 1])
        else:
            mu_A_x2 = trimf(x2,[-1+h*(i2-2), -1+h*(i2-1), -1+h*(i2)])

        g_bar[k,0]= 1/(3+e_i1[i1, 0]+e_i2[i2, 0])
        num = num + g_bar[k,0]*mu_A_x1*mu_A_x2
        den=den+mu_A_x1*mu_A_x2
        k=k+1

f_x = num/den
g_x = 1/(1+x1**2+x2**2)

fig = plt.figure()
#ax = fig.gca(projection='3d')
ax = fig.add_subplot(projection = '3d')
E = g_x - f_x
surf = ax.plot_surface(x1, x2, E, cmap=cm.coolwarm,
                       linewidth=0, antialiased=False)
ax.set_xlabel('$x_1$')
ax.set_ylabel('$x_2$')
ax.set_zlabel('$Error$')
ax.set_title('$Error$')
fig.colorbar(surf, shrink=0.5, aspect=5)
plt.savefig('fuzzy3.svg')
plt.show()
```

خروجی:



۲.

تولید داده‌ها

ابتدا با تعریف طول سری زمانی به میزان ۹۰۰ نقطه شروع می‌کنیم. سپس ۳۱ نقطه اول با مقادیر ثابت به علاوه یک مقدار تصادفی اضافه شده (در حدود ۱.۳ تا ۱.۵) مقداردهی اولیه می‌شوند. در ادامه، استفاده از یک دستور حلقه برای محاسبه باقی نقاط دنباله با استفاده از رابطه بازگشتی خاص انجام می‌شود.

پیش‌تخصیص

برای افزایش عملکرد، متغیرهای x و dataset_1 پیش‌تخصیص داده می‌شوند. این به کد کمک می‌کند تا سریعتر اجرا شود، زیرا اندازه آنها از پیش مشخص شده است.

رسم سری‌ها

این قسمت از کد یک نمودار را تولید می‌کند که نشان‌دهنده بخشی از دنباله زمانی تولید شده است. رسم خطوط با ضخامت ۲ نیز نشان‌دهنده وضوح بیشتر در نمودار است.

```
clc
clear
close all
```

```
%% data generation
n=900;

% preallocations
x=zeros(1,n);
dataset_1=zeros(n,7);

x(1,1:31)=1.3+0.2*rand;
for k=31:n-1
    x(1,k+1)=0.2*((x(1,k-30))/(1+x(1,k-30)^10))+0.9*x(1,k);
    dataset_1(k,2:6)=[x(1,k-3) x(1,k-2) x(1,k-1) x(1,k) x(1,k+1)];
end

dataset(1:600,2:6)=dataset_1(201:800,2:6);

t=1:600;

figure1 = figure('Color',[1 1 1]);
plot(t,x(201:800),'Linewidth',2)

[number_training,~]=size(dataset);
Rul=zeros(number_training/2,6);
Rules_total=zeros(number_training/2,6);
```

طراحی سیستم فازی

کد دو حالت مختلف برای تعداد توابع عضویت (MFs) را امتحان می‌کند: یک حالت با ۷ تابع عضویت و حالت دیگر با ۱۵ تابع عضویت. توابع مثلثی و ذوزنقه‌ای برای نمایش متغیرها استفاده می‌شوند.

```
%% design fuzzy system

% s=1 --> 7 MF
% s=2 --> 14 MF
for s=1:2
    switch s
        case 1
        num_membership_functions=7;
        c=linspace(0.5,1.3,5);
        h=0.2;
        membership_functions=cell(num_membership_functions,2);
        for k=1:num_membership_functions
            if k==1
                membership_functions{k,1}=[0,0,0.3,0.5];
                membership_functions{k,2}='trapmf';
            elseif k==num_membership_functions
                membership_functions{k,1}=[1.3,1.5,1.8,1.8];
                membership_functions{k,2}='trapmf';
            else
                membership_functions{k,1}=[c(k-1)-h,c(k-1),c(k-1)+h];
                membership_functions{k,2}='trimf';
            end
        end

        case 2
          num_membership_functions=15;
        c=linspace(0.3,1.5,13);
        h=0.1;
        membership_functions=cell(num_membership_functions,2);
        for k=1:num_membership_functions
            if k==1
                membership_functions{k,1}=[0,0,0.2,0.3];
                membership_functions{k,2}='trapmf';
            elseif k==num_membership_functions
                membership_functions{k,1}=[1.5,1.6,1.8,1.8];
                membership_functions{k,2}='trapmf';
            else
                membership_functions{k,1}=[c(k-1)-h,c(k-1),c(k-1)+h];
                membership_functions{k,2}='trimf';
            end
        end
    end
```

اختصاص درجه به قوانین

در این بخش، به ازای هر نقطه از داده‌ها، حداکثر درجه عضویت به هر یک از توابع عضویت محاسبه و در ماتریس قوانین ثبت می‌شود.

```
%% assign degree to each rule

vec_x=zeros(1,num_membership_functions);
vec=zeros(1,5);
for t=1:number_training
    dataset(t,1)=t;
      for i=2:6
          x=dataset(t,i);
              for j=1:num_membership_functions
                  if j==1
                  vec_x(1,j)=trapmf(x,membership_functions{1,1});
                  elseif j==num_membership_functions
                  vec_x(1,j)=trapmf(x,membership_functions{num_membership_functions,1});
                  else
                  vec_x(1,j)=trapmf(x,membership_functions{j,1});
                  end
              end
          [valu_x,column_x]=max(vec_x);
          vec(1,j-1)=max(vec_x);
          Rules(t,i-1)=column_x;
          Rules(t,6)=prod(vec);
          dataset(t,7)=prod(vec);
      end
end
```

حذف قوانین اضافی

این قسمت از کد قوانین تکراری یا اضافی را حذف می‌کند تا تنها قوانین ضروری باقی بمانند.

```
%% delete extra rules


rules_total(1,1:6)=Rules(1,1:6);
i=1;
for t=2:number_training
    m=zeros(1,i);
    for j=1:i
        m(1,j)=isequal(Rules(t,1:4),Rules_total(j,1:4));
        if m(1,j)==1 && Rules(t,6)>= Rules_total(j,6)
            Rules_total(j,1:6)=Rules(t,1:6);
        end
    end
    if sum(m) ==0
    Rules_total(i+1,1:6)=Rules(t,1:6);
    i=i+1;
    end
end
disp('*******************************************')
disp(['Final rules for',num2str(num_membership_functions),'membership functions for each input variables'])
final_Rules=Rules_total(1:i,:)
```

ایجاد یک سیستم استنتاج فازی

در این قسمت، یک سیستم استنتاج فازی معمولی ممدانی با استفاده از توابع MATLAB مربوطه ایجاد می‌شود. به این سیستم ورودی‌ها و خروجی‌ها اضافه می‌شود و توابع عضویت و قوانین بر اساس تجزیه‌وتحلیل قبلی درج می‌گردد.

```
%% create fuzzy inference system
Fisname='Prediction controller';
Fistype='mamdani';
Andmethod='prod';
Ormethod='max';
Impmethod='prod';
Agmethod='max';
Defuzzmethod='centroid';
fis=mamfis(Fisname,Fistype,Andmethod,Ormethod,Impmethod,Aggmethod,Defuzzmethod);
%% add variables
for num_input=1:4
    fis=addInput(fis,'input',['x',num2str(num_input)],[0.1 1.6]);
end
fis=addOutput(fis,'output','x5',[0.1 1.6]);
%% add mf function
for num_input=1:4
    for input_Rul=1:num_membership_functions
        fis=addMF(fis,'input',num_input,['A',num2str(input_Rul)],membership_functions{input_Rul,2},membership_functions{input_Rul,1});
    end
end
%% add rules
fis_Rules=ones(i,7);
fis_Rules(1:i,1:5)=Rules_total(1:i,1:5);
fis=addrule(fis,fis_Rules);
```

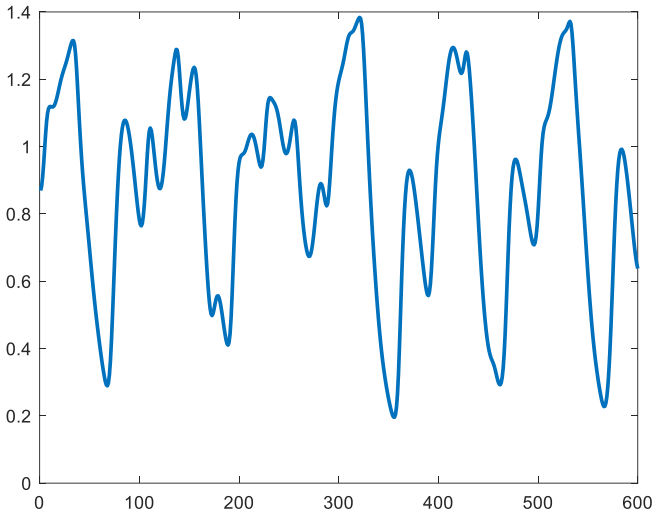
پیش‌بینی نقاط

کد به کمک سیستم فازی ایجاد شده، ۳۰۰ نقطه بعدی سری زمانی را پیش‌بینی می‌کند و در یک جدول ثبت می‌کند.

رسم پیش‌بینی‌ها

در انتها، نتایج پیش‌بینی شده و مقادیر واقعی در نموداری در کنار هم نمایش داده می‌شوند تا میزان دقت مدل قابل مشاهده و ارزیابی باشد.

```
%% prediction of 300 points of chosen dataset

table_prediction=zeros(300,2);
f=1;
for i=301:600
    input=dataset(i,2:6);
    output1=dataset(i,6);
x5=evalfis([input(1,1);input(1,2);input(1,3);input(1,4)],fis);
table_prediction(f,:)=[f,x5];
f=f+1;
end

figure;
plot(table_prediction(:,1),table_prediction(:,2),'r-.','Linewidth',2);
hold on;
plot(table_prediction(:,1),dataset(301:600,6),'b','Linewidth',2);
legend('estimate value','real value')
end
```



سری زمانی آشوب مکی گلاس

۳.

در این سوال، یک سیستم شناسایی به وسیله شبکه‌ای عصبی مبتنی بر توابع عضویت ارائه شده است. سیستم مورد نظر با استفاده از رشته‌ای از داده‌های ورودی تصادفی آموزش داده شده و سپس خروجی‌های سیستم را پیش‌بینی می‌کند.

تعداد توابع عضویت فازی (M) ، ۴ تعریف شده است.

num_training نشان‌دهنده تعداد نقاط آموزشی است.

total_num نشان‌دهنده کل تعداد نقاط است که شامل دوره‌های آموزش و آزمایش می‌شود.

landa نرخ یادگیری است.

پیش تخصیص

مقادیر اولیه صفر برای ماتریس‌ها و بردارهای مختلف تعیین می‌شوند.

x_bar، g_bar و sigm برای ذخیره مراکز، ارتفاعات و انحرافات استاندارد توابع عضویت در هر مرحله آموزش به‌کار می‌روند.

y و y_hat به ترتیب خروجی واقعی سیستم و خروجی مدل شناسایی را ذخیره می‌کنند.

ورودی اولیه تصادفی

ورودی اولیه u به صورت تصادفی انتخاب می‌شود.

g_u شامل محاسبه خروجی سیستم بر اساس فرمول داده شده است.

طراحی مرحله اول

این بخش مراکز اولیه توابع عضویت فازی را ایجاد می‌کند.

محدوده ورودی بین u_min و u_max مشخص شده است.

مراکز و ارتفاعات توابع عضویت براساس معادله داده‌شده محاسبه می‌شوند.

```
import numpy as np
import matplotlib.pyplot as plt

# initializing
M = 4  # number of mfs
num_training = 200
total_num = 700
landa = 0.1

# preallocate arrays
x_bar = np.zeros((num_training, M))
g_bar = np.zeros((num_training, M))
sigma = np.zeros((num_training, M))
y = np.zeros(total_num)
u = np.zeros(total_num)
x = np.zeros(total_num)
y_hat = np.zeros(total_num)
f_hat = np.zeros(total_num)
g_u = np.zeros(total_num)

# initial random input
u[0] = -1 + 2 * np.random.rand()
y[0] = 0
g_u[0] = 0.6 * np.sin(np.pi * u[0]) + 0.3 * np.sin(3 * np.pi * u[0]) + 0.1 * np.sin(5 * np.pi * u[0])
f_hat[0] = g_u[0]

# first step design
u_min = -1
u_max = 1
h = (u_max - u_min) / (M-1)
for k in range(M):
    x_bar[0, k] = u_min + h * k
    g_bar[0, k] = 0.6 * np.sin(np.pi * x_bar[0, k]) + 0.3 * np.sin(3 * np.pi * x_bar[0, k]) + 0.1 * np.sin(5 * np.pi * x_bar[0, k])
sigma[0, :] = (np.max(x_bar[0, :]) - np.min(x_bar[0, :])) / M
```

فاز آموزش

در این مرحله ورودی‌های تصادفی جدید تولید شده و به سیستم داده می‌شوند.

خروجی سیستم g_u و پیش‌بینی مدل f_hat محاسبه می‌شود.

پارامترهای مدل (g_bar، x_bar، sigma) بر اساس خطای پیش‌بینی و نرخ یادگیری به‌روزرسانی می‌شوند.

مقادیر نهایی پس از آموزش

x_bar_final، sigma_final و g_bar_final نشان‌دهنده پارامترهای نهایی توابع عضویت پس از فاز آموزش هستند.

خروجی شبکه پس از آموزش

پیش‌بینی مدل با استفاده از پارامترهای بدست‌آمده و داده‌های جدید آزمایش می‌شود.

در این فاز، ورودی‌های جدید با استفاده از یک تابع سینوسی با دوره مشخص تولید می‌شوند.

```
# training phase
for q in range(1, num_training):
    x[q] = -1 + 2 * np.random.rand()
    g_u[q] = 0.6 * np.sin(np.pi * x[q]) + 0.3 * np.sin(3 * np.pi * x[q]) + 0.1 * np.sin(5 * np.pi * x[q])
    z = np.exp(-np.square((x[q] - x_bar[q-1, :]) / sigma[q-1, :]))
    b = np.sum(z)
    a = np.dot(z, g_bar[q-1, :])
    f_hat[q] = a / b
    y[q] = 0.3 * y[q-1] + 0.6 * y[q-2] + g_u[q]
    y_hat[q] = 0.3 * y[q-1] + 0.6 * y[q-2] + f_hat[q]
    for l in range(M):
        g_bar[q, l] = g_bar[q-1, l] - landa * (f_hat[q] - g_u[q]) * z[l] / b
        x_bar[q, l] = x_bar[q-1, l] - landa * ((f_hat[q] - g_u[q]) * (g_bar[q-1, l] - a / b) * z[l] * 2 * (x[q] - x_bar[q-1, l])) / (sigma[q-1, l]**2)
        sigma[q, l] = sigma[q-1, l] - landa * ((f_hat[q] - g_u[q]) * (g_bar[q-1, l] - a / b) * z[l] * 2 * (x[q] - x_bar[q-1, l])**2) / (sigma[q-1, l]**3)

# final values after training
x_bar_final = x_bar[num_training - 1, :]
sigma_final = sigma[num_training - 1, :]
g_bar_final = g_bar[num_training - 1, :]

# network output after training
for q in range(num_training, total_num - 1):
    x[q] = np.sin(2 * np.pi * q / total_num)
    g_u[q] = 0.6 * np.sin(np.pi * x[q]) + 0.3 * np.sin(3 * np.pi * x[q]) + 0.1 * np.sin(5 * np.pi * x[q])
    z = np.exp(-np.square((x[q] - x_bar_final) / sigma_final))
    b = np.sum(z)
    a = np.dot(z, g_bar_final)
    f_hat[q] = a / b
    y[q+1] = 0.3 * y[q] + 0.6 * y[q-1] + g_u[q]
    y_hat[q+1] = 0.3 * y[q] + 0.6 * y[q-1] + f_hat[q]
```

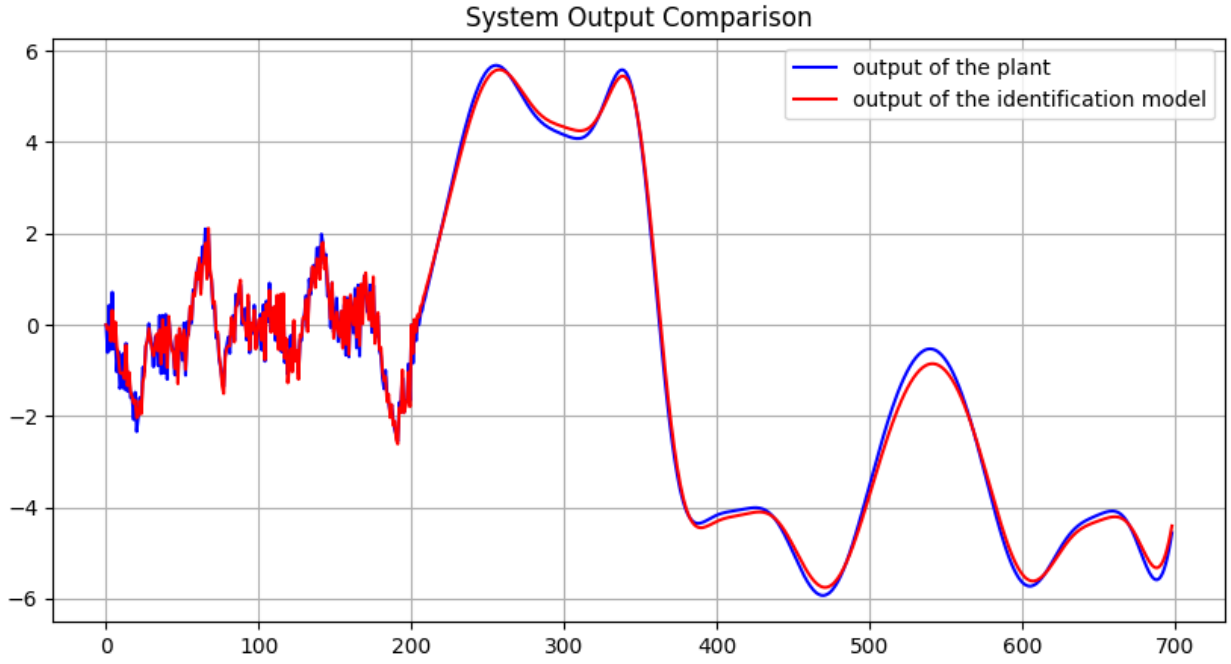
رسم داده‌ها

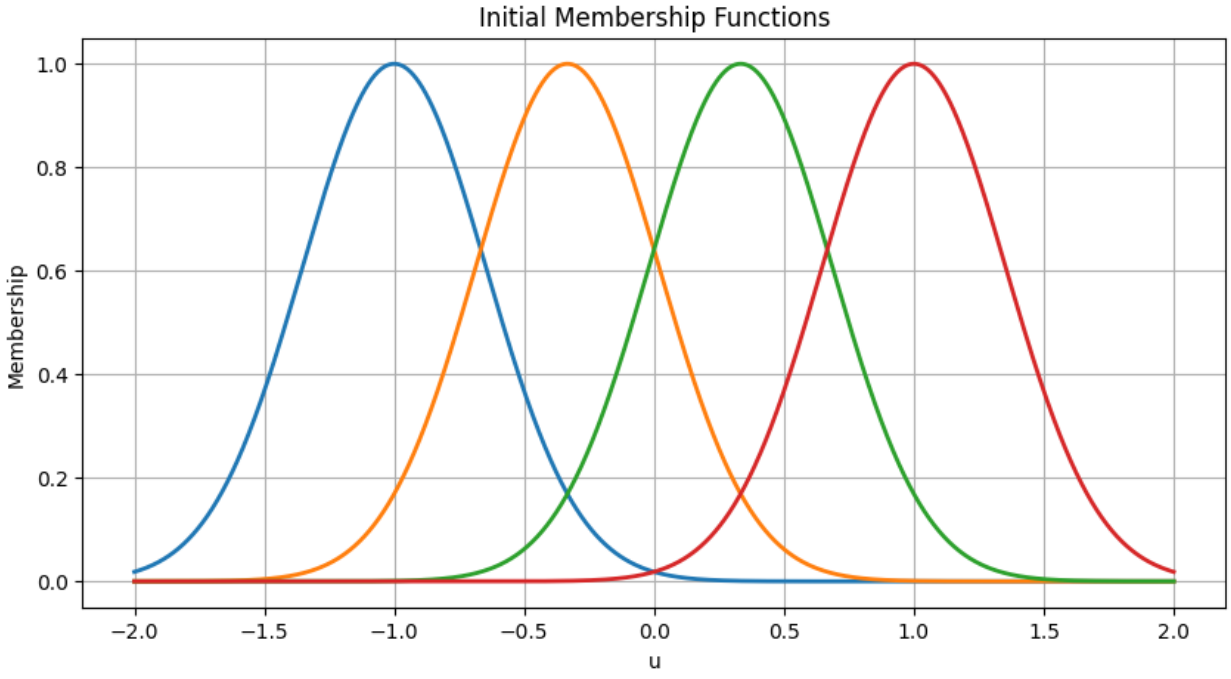
نمودار اول مقایسه خروجی واقعی سیستم (y) و خروجی حاصل از مدل شناسایی (y_hat) را نشان می‌دهد.

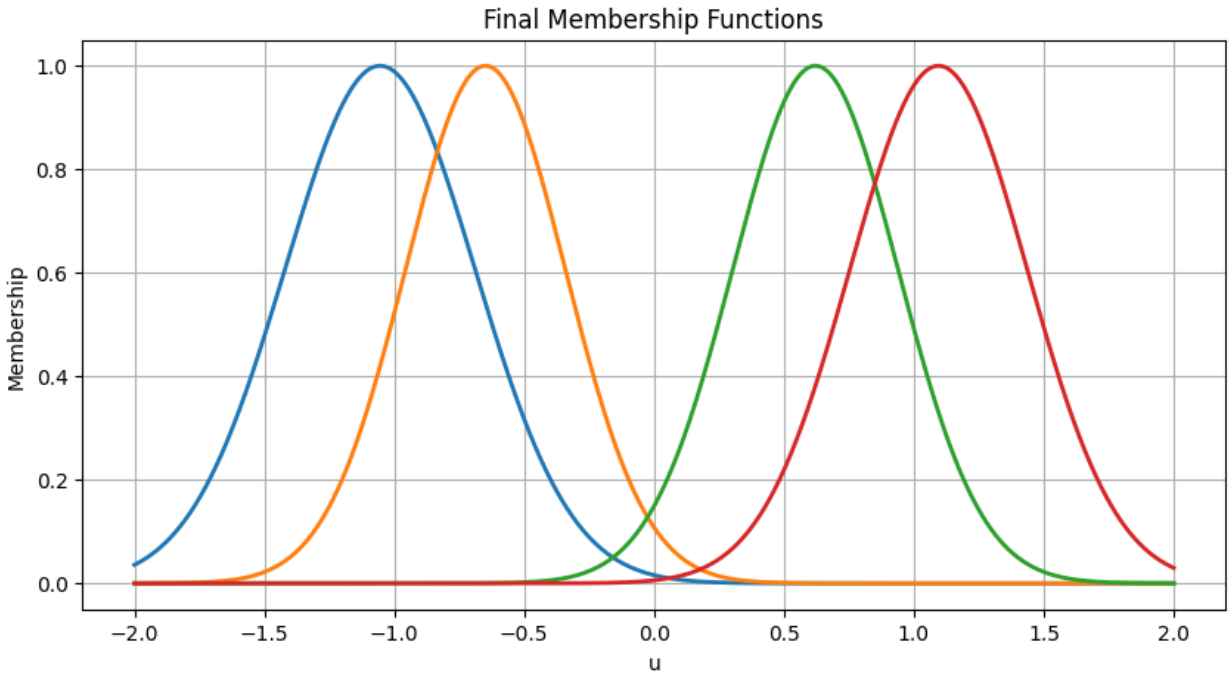
دو نمودار دیگر توابع عضویت اولیه و نهایی را برای تمامی M توابع عضویت نمایش می‌دهند.

```
# plotting
plt.figure(figsize=(10, 5))
plt.plot(y[:-1], 'b', label='output of the plant')
plt.plot(y_hat[:-1], 'r', label='output of the identification model')
plt.title('System Output Comparison')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 5))
xp = np.linspace(-2, 2, 4000)
for l in range(M):
    miu_x = np.exp(-((xp - x_bar[0, l])**2) / (sigma[0, l]**2))
    plt.plot(xp, miu_x, linewidth=2)
plt.title('Initial Membership Functions')
plt.xlabel('u')
plt.ylabel("Membership")
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 5))
for l in range(M):
    miu_x = np.exp(-((xp - x_bar_final[l])**2) / (sigma_final[l]**2))
    plt.plot(xp, miu_x, linewidth=2)
plt.title('Final Membership Functions')
plt.xlabel('u')
plt.ylabel("Membership")
plt.grid(True)
plt.show()
```

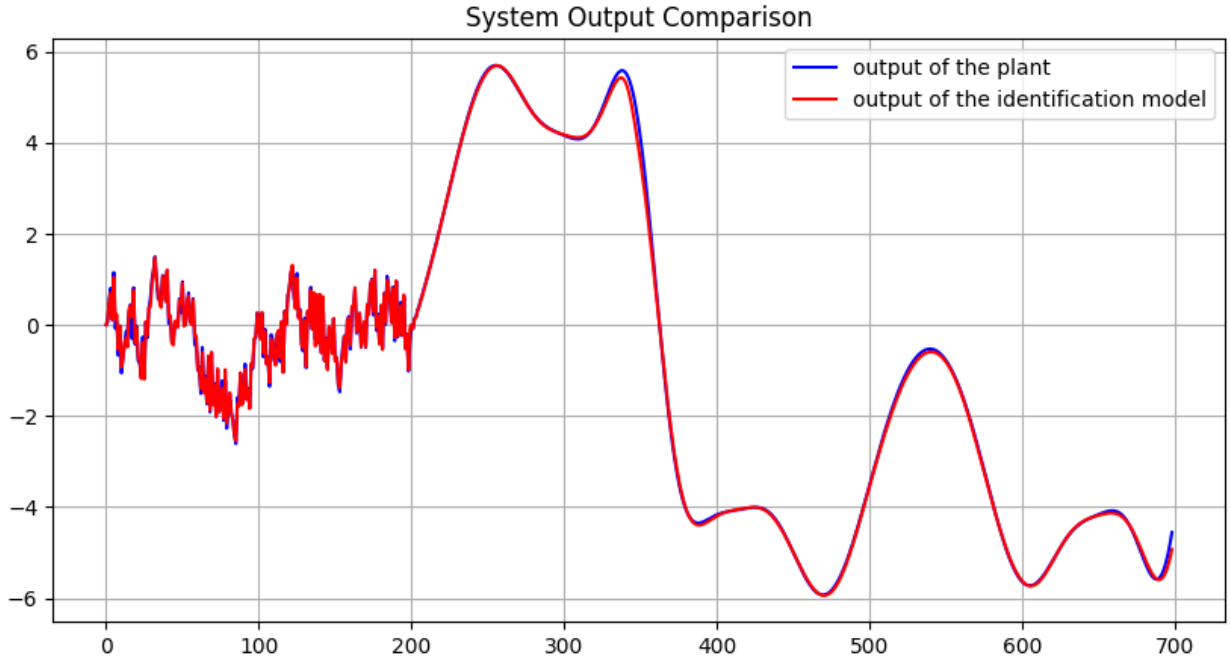
M = 4

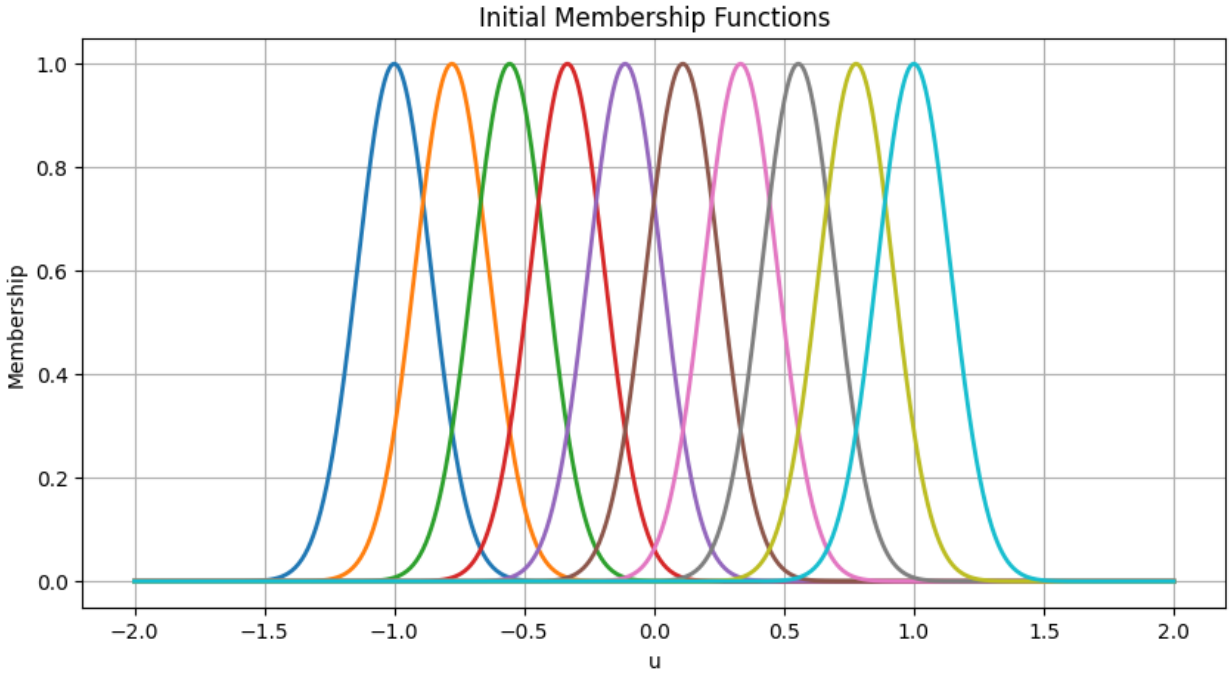

System Output Comparison
output of the plant
output of the identification model
6
4
2
0
−2
−4
−6
0
100
200
300
400
500
600
700

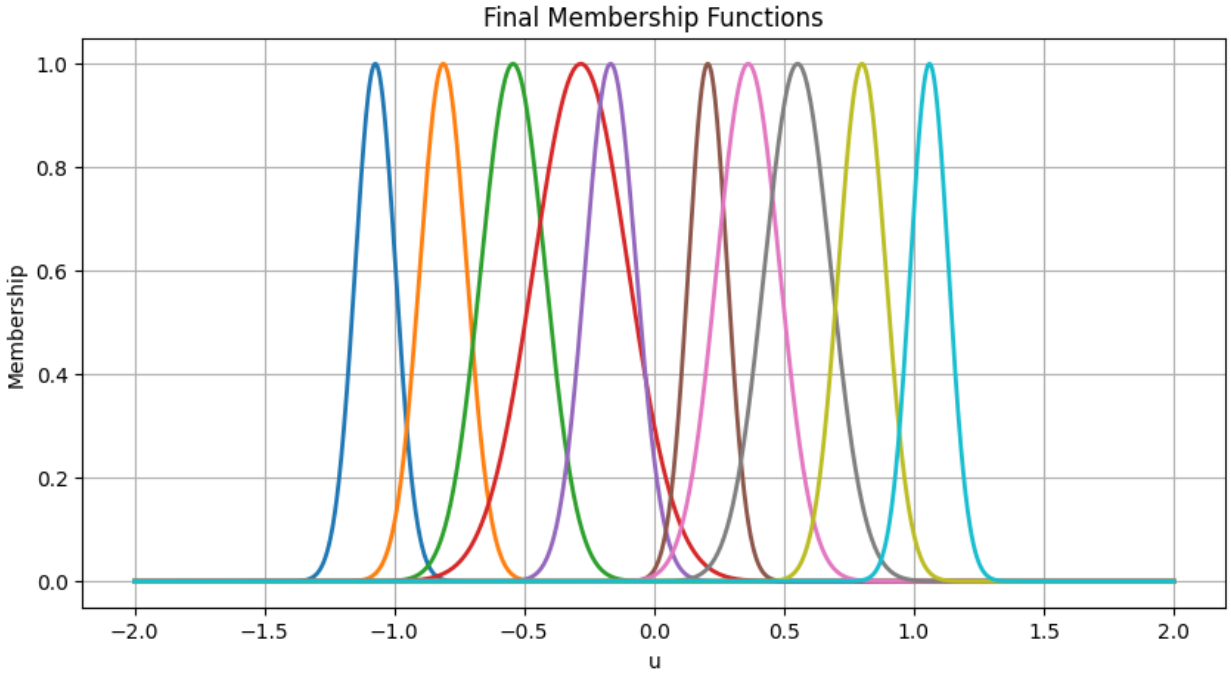

Initial Membership Functions
Membership
1.0
0.8
0.6
0.4
0.2
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
u


Final Membership Functions
Membership
u
1.0
0.8
0.6
0.4
0.2
0.0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

M=10


System Output Comparison
output of the plant
output of the identification model
6
4
2
0
−2
−4
−6
0
100
200
300
400
500
600
700





با افزایش تعداد توابع تعلق حجم محاسبات افزایش می‌یابد ولی تقریب با دقت بیشتری انجام می‌شود.

خروجی مدل شناسایی با ۱۰ تابع تعلق در مقایسه با ۴ تابع با دقت بیشتری خروجی سیستم را دنبال می‌کند.

.٤

-١.۴

کدنویسی درخت تصمیم بدون استفاده از کتابخانه سایکیت‌لرن

مجموعه داده مد نظر را با دستور gdown در محیط کولب بارگزاری می‌کنیم. داده از یک فایل CSV با نام covid.csv خوانده می‌شود.

```
# imports

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

# DT from scratch

# load data
!pip install --upgrade --no-cache-dir gdown
!gdown 1hmLfXzalUn4FkQ_pV-V0VNSCdpBRA6Jg

data = pd.read_csv('/content/covid.csv')
print('- - - - - - - - - - - - -')
print("Original DataFrame:")
print(data)
```

```
- - - - - - - - - - - - -
Original DataFrame:
   Fever Cough Breathing issues Infected
0     No    No               No       No
1    Yes   Yes              Yes      Yes
2    Yes   Yes               No       No
3    Yes    No              Yes      Yes
4    Yes   Yes              Yes      Yes
5     No   Yes               No       No
6    Yes    No              Yes      Yes
7    Yes    No              Yes      Yes
8     No   Yes              Yes      Yes
9    Yes   Yes               No      Yes
10    No   Yes               No       No
11    No   Yes              Yes      Yes
12    No   Yes              Yes       No
13   Yes   Yes               No       No
- - - - - - - - - - - - -
```

مجموعه داده covid تعداد ۱۴ نمونه دارد. همچنین متشکل از ۳ ویژگی تب، سرفه و مشکلات تنفسی است. ۲ کلاس مربوط به ابتلا به کرونا یا خیر را دارد.

تابع entropy(labels) تعریف شده است که آنتروپی را بر اساس توزیع برچسب‌ها محاسبه می‌کند.

Labels: ستون برچسب‌ها که حاوی مقادیر برچسب‌ها است.

تعداد تکرار هر برچسب محاسبه شده و به تعداد کل نمونه‌ها تقسیم شده تا توزیع برچسب‌ها محاسبه شود. سپس از فرمول آنتروپی استفاده شده و مقدار آنتروپی با جمع‌کردن مقادیر به ازای هر برچسب محاسبه می‌شود. در نهایت، مقدار نتیجه قرینه شده و به عنوان خروجی تابع باز می‌گردد. برای ستون "Infected" از داده، مقدار آنتروپی محاسبه شده و نشان داده می‌شود.

```python
# model

## functional definition of entropy
def entropy(labels):
    p = labels.value_counts() / len(labels)
    return -sum(p * np.log2(p))
Infectad_entropy = entropy(data['Infected'])
print('- - - - - - - - - - - - -')
print("Infectad entropy:")
print(Infectad_entropy)
```

```
- - - - - - - - - - - - -
Infectad entropy:
0.9852281360342515
- - - - - - - - - - - - -
```

آنتروپی ستون Infected برابر با ۰.۹۸۵۲۲۸۱۳۶۰۳۴۲۵۱۵ است.

تابع information_gain(data, feature, target) بر اساس آنتروپی، بهره اطلاعات را محاسبه می‌کند.

Data: داده مورد استفاده که شامل ویژگی‌ها و برچسب‌ها است.

Feature: ویژگی مورد نظر که برای محاسبه بهره اطلاعات در نظر گرفته می‌شود.

Target: برچسب مورد نظر که بر اساس آن بهره اطلاعات محاسبه می‌شود.

آنتروپی والد (parent) محاسبه شده و در متغیر entropy_parent ذخیره می‌شود. سپس برای هر مقدار یکتا از ویژگی مورد نظر، زیرمجموعه‌ای از داده ایجاد شده و آنتروپی فرزند (child) محاسبه می‌شود. بهره اطلاعات با کم کردن آنتروپی فرزند از آنتروپی والد محاسبه می‌شود و به عنوان خروجی تابع باز می‌گردد. برای هر ویژگی، بهره اطلاعات محاسبه شده و نشان داده می‌شود. در نهایت ویژگی با بیشترین بهره اطلاعات به عنوان ویژگی انتخاب می‌شود و نشان داده می‌شود. این ویژگی تشکیل‌دهنده گره ریشه است.

```python
## information_gain
def information_gain(data, feature, target):
    # Entropy of parent
    entropy_parent = entropy(data[target])

    # Entropy of child
    entropy_child = 0
    for value in data[feature].unique():
        subset = data[data[feature] == value]
        wi = len(subset) / len(data)
        entropy_child += wi * entropy(subset[target])

    return entropy_parent - entropy_child

data.iloc[:, :-1].columns
[information_gain(data, feature, 'Infected') for feature in data.iloc[:, :-1].columns]
max_entropy_node = np.argmax([information_gain(data, feature, 'Infected') for feature in data.iloc[:, :-1].columns])
print('- - - - - - - - - - - - -')
print("max entropy is for coloumn ", max_entropy_node)
```

```
- - - - - - - - - - - - -
Max Information Gain is for coloumn  2
- - - - - - - - - - - - -
```

ستون ۲، مربوط به ویژگی Breathing Issues بیشترین بهره اطلاعات را دارد در نتیجه به عنوان گره ریشه انتخاب می‌شود.

یک کلاس Node تعریف شده است که نمایانگر یک گره درخت تصمیم است.

Feature: ویژگی مرتبط با گره، که در صورت نبودن آن نشان‌دهنده یک گره برگ است.

Label: برچسب مرتبط با گره، که در صورت نداشتن ویژگی نشان‌دهنده برچسب گره برگ است.

Children: یک دیکشنری که نمایانگر زیردرخت‌های هر گره می‌باشد.

متد repr جهت تولید نمایش متنی از یک نمونه از کلاس استفاده می‌شود. اگر ویژگی موجود باشد، نمایش متنی برای گره تصمیم و در غیر این صورت برای گره برگ ایجاد می‌شود.

```python
## DT and nodes
class Node:

    def __init__(self, feature=None, label=None):
        self.feature = feature
        self.label = label
        self.children = {}

    def __repr__(self):
        if self.feature is not None:
            return f'DecisionNode(feature="{self.feature}", children={self.children})'
        else:
            return f'LeafNode(label="{self.label}")'
```

یک تابع make_tree(data, target) بر اساس ورودی‌های داده و هدف، درخت تصمیم را ساخته و برمی‌گرداند. در ابتدا، تابع بررسی می‌کند که آیا تمام نمونه‌های داده دارای یک برچسب هستند یا خیر. در صورتی که تمامی برچسب‌ها یکسان باشند، یک گره برگ با برچسب یکتا ایجاد می‌شود. در صورتی که داده یا ویژگی‌ها به پایان برسند، یک گره برگ با برچسبی که بیشترین تکرار را دارد ایجاد می‌شود. برای هر ویژگی، بهره اطلاعات محاسبه شده و در لیست gains ذخیره می‌شود. ویژگی با بیشترین ابهره اطلاعات به عنوان بهترین ویژگی انتخاب می‌شود. با استفاده از بهترین ویژگی انتخاب شده، یک گره جدید با این ویژگی ایجاد می‌شود. برای هر مقدار ممکن از بهترین ویژگی، یک زیرمجموعه ایجاد شده و با فراخوانی بازگشتی تابع make_tree، زیردرخت مربوطه ایجاد می‌شود. درخت تصمیم با فراخوانی make_tree بر روی داده covid با هدف Infected ایجاد می‌شود.

```python
def make_tree(data, target):
    # leaf node?
    if len(data[target].unique()) == 1:
        return Node(label=data[target].iloc[0])

    features = data.drop(target, axis=1).columns
    if len(features) == 0 or len(data) == 0:
        return Node(label=data[target].mode()[0])

    # calculate information gain
    gains = [information_gain(data, feature, target) for feature in features]

    # greedy search to find best feature
    max_gain_idx = np.argmax(gains)
    best_feature = features[max_gain_idx]

    # make a node
    node = Node(feature=best_feature)

    # loop over the best feature
    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value].drop(best_feature, axis=1)
        # display(subset)

        node.children[value] = make_tree(subset, target)

    return node

## make tree
tree = make_tree(data, 'Infected')
print('- - - - - - - - - - - - -')
print('tree :', tree)

root_node = tree.feature
print('- - - - - - - - - - - - - - -')
print('root node is :', root_node)
```

```
- - - - - - - - - - - - -
tree : DecisionNode(feature="Breathing issues", children={'No': DecisionNode(feature="Fever", children={'No': LeafNode(label="No"), 'Yes': DecisionNode(feature="Cough", children={'Yes': LeafNode(label="No")})})
- - - - - - - - - - - - -

, 'Yes': DecisionNode(feature="Fever", children={'Yes': LeafNode(label="Yes"), 'No': DecisionNode(feature="Cough", children={'Yes': LeafNode(label="Yes")})})})

- - - - - - - - - - - - - - -
root node is : Breathing issues
```

تحلیل درخت تصمیم:

این درخت تصمیم، متشکل از گره‌های تصمیم (DecisionNode) و گره‌های برگ (LeafNode) است. درخت بر اساس مشکلات تنفسی شروع می‌شود و بر اساس وجود یا عدم وجود تب و سرفه به تصمیم‌های بعدی می‌رسد. هر گره تصمیم ویژگی‌های مشخصی را برای انتخاب انجام می‌دهد و هر گره برگ نهایتاً به یک تشخیص (برچسب) منتهی می‌شود. این مدل به صورت سلسله مراتبی علائم را بررسی می‌کند و به دسته‌بندی "Yes" یا "No" در مورد ابتلا به کووید-۱۹ می‌پردازد.

ویژگی "Breathing issues" گره تصمیم ابتدایی(ریشه) است که بر اساس مشکلات تنفسی تصمیم‌گیری را شروع می‌کند.

اگر مشکلات تنفسی وجود نداشته باشد (No):

ویژگی "Fever" به عنوان گره تصمیم بعدی در نظر گرفته و بررسی می‌شود:

اگر تب وجود نداشته باشد (No):

اولین گره برگ با برچسبNo.

اگر تب وجود داشته باشد (Yes):

ویژگی "Cough" به عنوان گره تصمیم بعدی در نظر گرفته و بررسی می‌شود:

اگر سرفه وجود داشته باشد (Yes):

گره برگ با برچسب "No" خواهیم داشت.

اگر مشکلات تنفسی وجود داشته باشد (Yes):

ویژگی "Fever" بررسی می‌شود:

اگر تب وجود داشته باشد (Yes):

گره برگ با برچسب Yes.

اگر تب وجود نداشته باشد (No):

ویژگی "Cough" بررسی می‌شود:

اگر سرفه وجود داشته باشد (Yes):

گره برگ با برچسب Yes.

از کتابخانه graphviz برای تصویرسازی درخت تصمیم استفاده شده است. درخت ابتدایی و یک نسخه بهبود یافته از درخت با اطلاعات اضافی مشخص شده‌اند.

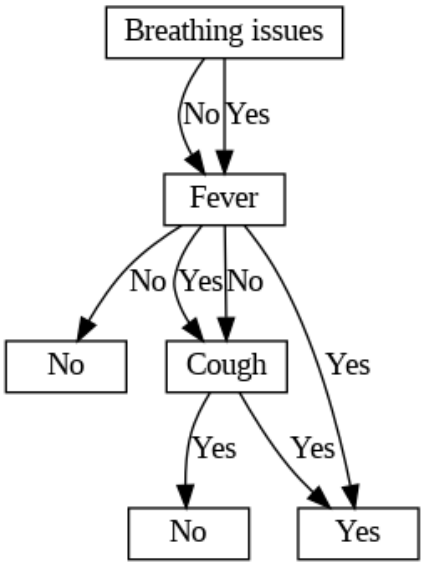
در این قسمت از کتابخانه graphviz برای تصویرسازی درخت تصمیم ابتدایی استفاده شده است. یک گراف جدید با نام decision-tree ایجاد شده است که تنظیمات خاصی برای نمایش گره‌ها دارد. تابع plot_tree به منظور ایجاد ویژگی‌ها و ارتباطات بین گره‌ها تعریف شده است. هر گره با نام و برچسب مناسب نمایش داده شده و اتصالات بین گره‌ها بر اساس روابط درخت تصمیم ایجاد می‌شوند. تابع plot_tree فراخوانی شده و گراف با استفاده از render به فرمت PNG ذخیره شده است. فایل تصویری با نام decision_tree ایجاد شده تا بتوان آن را مشاهده کرد.

```
## visualize
from graphviz import Digraph, nohtml

g = Digraph('g', filename='decision-tree.gv', node_attr={'shape': 'record', 'height': '.1'})

def plot_tree(tree, g):
    root_node = tree.feature
    if root_node is None:
        return g
    g.node(root_node, nohtml(root_node))
    child_nodes = tree.children.keys()
    for i, child in enumerate(child_nodes):
        node = tree.children[child]
        name = node.feature if node.feature is not None else child+node.label
        label = node.feature if node.feature is not None else node.label
        g.node(name, nohtml(label))
        g.edge(root_node, name, label=child)
        plot_tree(node, g)
    return g

g = plot_tree(tree, g)
g.render('decision_tree', format='png', view=True)
```

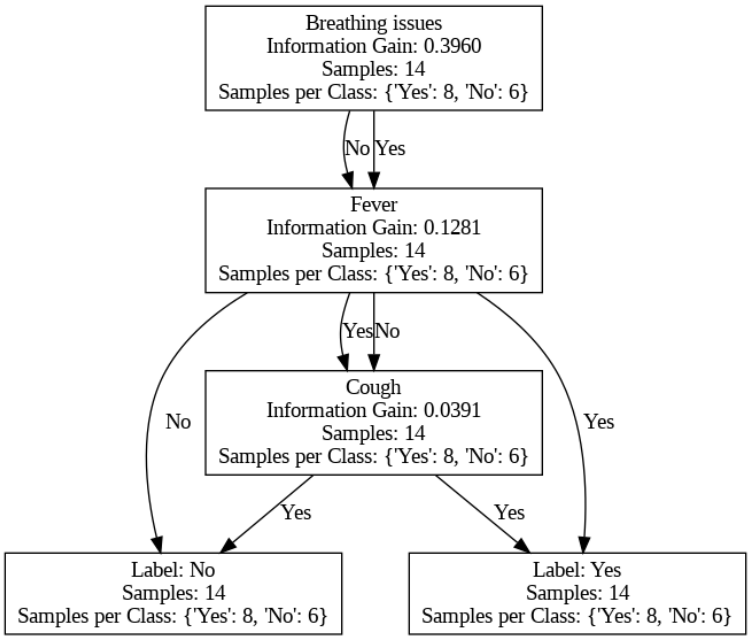
درخت بهبود یافته نیز حاصل اطلاعات مهمی مانند ویژگی با بیشترین بهره اطلاعات و توزیع برچسب‌ها هر گره را نمایش می‌دهد. توابع جدیدی برای تولید برچسب‌های گره‌ها ایجاد شده است که شامل اطلاعات بیشتری از درخت می‌شوند. این اطلاعات شامل نام ویژگی، بهره اطلاعات مربوط به ویژگی فعلی و تعداد نمونه‌ها گره می‌باشند. در صورتی که تعداد بیشتری از برچسب‌ها در یک گره وجود داشته باشد، اطلاعات توزیع برچسب‌ها به عنوان بخشی از برچسب گره نمایش داده می‌شود. این توزیع نشان‌دهنده تنوع برچسب‌ها در گره است. تابع plot_tree به گونه‌ای بهبود یافته است که اطلاعات جدید گره‌ها را نمایش می‌دهد. درخت با استفاده از کتابخانه graphviz به صورت فایل تصویری ذخیره و قابل مشاهده است. یک گراف جدید با نام decision-tree-enhanced ایجاد شده است که اطلاعات بهبود یافته را نمایش می‌دهد. فایل تصویری این گراف با فرمت PNG ذخیره شده و قابل مشاهده است.

```python
## tree with more inf
from graphviz import Digraph, nohtml

def plot_tree(tree, g, parent_name=None, edge_label=None):
    if tree.feature is not None:
        current_node_name = tree.feature
        node_label = (
            f"{tree.feature}\n"
            f"Information Gain: {information_gain(data, tree.feature, 'Infected'):.4f}\n"
            f"Samples: {len(data)}\n"
            f"Samples per Class: {data['Infected'].value_counts().to_dict()}"
        )

    else:
        current_node_name = f"Leaf_{tree.label}"
        node_label = (
            f"Label: {tree.label}\n"
            f"Samples: {len(data)}\n"
            f"Samples per Class: {data['Infected'].value_counts().to_dict()}"
        )

    g.node(current_node_name, label=nohtml(node_label))

    if parent_name is not None:
        g.edge(parent_name, current_node_name, label=edge_label)

    for value, child_node in tree.children.items():
        plot_tree(child_node, g, current_node_name, str(value))

    return g

# Create a new graph
g = Digraph('g', filename='decision-tree-enhanced.gv', node_attr={'shape': 'box'})

# Plot the tree and save the visualization
g = plot_tree(tree, g)
g.render('decision_tree_enhanced', format='png', view=True)
```

خروجی



درخت تصمیم ابتدایی



درخت تصمیم بهبود یافته

تحلیل نتایج درخت

تحلیل انتخاب گره‌های تصمیم و برگ و نحوه تصمیم‌گیری در بخش قبلی توضیح داده شد. در اینجا کمی توضیحات اضافه‌تر آورده شده است.

گره ریشه: در این گره تمام ۱۴ نمونه حضور دارند. ۸ نمونه مربوط به کلاس Yes و ۶ نمونه مربوط به کلاس No هستند. بهره اطلاعات در این گره ۰.۳۹۶۰ است. می‌دانیم که گره با بیشترین بهره اطلاعات به‌عنوان گره ریشه انتخاب می‌شود که با توجه به درخت این نیز این موضوع صادق است. در واقع حداسازی کلاس‌ها با این ویژگی بسیار دشوار است به دلیل اینکه ناخالصی زیادی دارد. حال به ازای داشتن یا نداشتن مشکلات تنفسی باید ویژگی تب را بررسی کنیم. در این گره نیز تمام ۱۴ نمونه حضور دارند. ۸ نمونه مربوط به کلاس Yes و ۶ نمونه مربوط به کلاس No هستند. بهره اطلاعات در این گره ۰.۱۲۸۱ است. در این گره نیز نمی‌توان جداسازی کلاس‌ها را انجام داد پس به سراغ آخرین ویژگی می‌رویم. ویژگی سرفه با بهره اطلاعاتی ۰.۰۳۹۱ باز هم نمی‌تواند کلاس‌ها را از هم جدا کند!

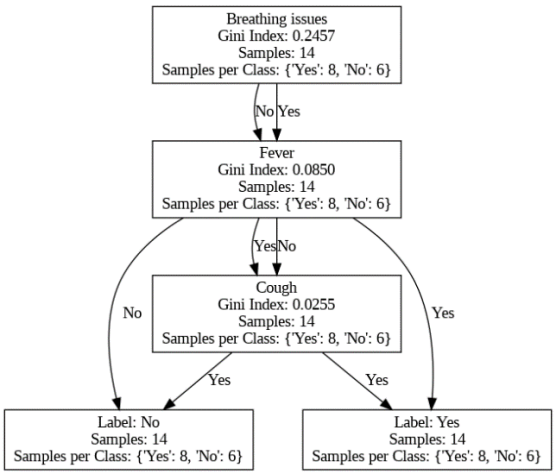
در کل درخت خوبی از آب درنیامد.

جهت دستیابی به نتیجه بهتر با معیار gini به جای آنتروپی هم کدنویسی را انجام دادم اما در نتیجه درخت بهتری حاصل نشد!

```python
# Model

## Functional definition of Gini index
def gini(labels):
    class_probabilities = labels.value_counts() / len(labels)
    gini_index = 1 - sum((p ** 2) for p in class_probabilities)
    return gini_index

Infected_gini = gini(data['Infected'])
print('- - - - - - - - - - - -')
print("Infected Gini Index:")
print(Infected_gini)

## Gini index-based information_gain
def information_gain_gini(data, feature, target):
    # Gini index of parent
    gini_parent = gini(data[target])

    # Gini index of child
    gini_child = 0
    for value in data[feature].unique():
        subset = data[data[feature] == value]
        wi = len(subset) / len(data)
        gini_child += wi * gini(subset[target])

    return gini_parent - gini_child

data.iloc[:, :-1].columns
[information_gain_gini(data, feature, 'Infected') for feature in data.iloc[:, :-1].columns]
max_gini_node = np.argmax([information_gain_gini(data, feature, 'Infected') for feature in data.iloc[:, :-1].columns])
print('- - - - - - - - - - - -')
print("Max Information Gain (Gini Index) is for column ", max_gini_node)
```

```python
## Decision Tree and Nodes
class Node:

    def __init__(self, feature=None, label=None):
        self.feature = feature
        self.label = label
        self.children = {}

    def __repr__(self):
        if self.feature is not None:
            return f'DecisionNode(feature="{self.feature}", children={self.children})'
        else:
            return f'LeafNode(label="{self.label}")'

def make_tree_gini(data, target):
    # Leaf node?
    if len(data[target].unique()) == 1:
        return Node(label=data[target].iloc[0])

    features = data.drop(target, axis=1).columns
    if len(features) == 0 or len(data) == 0:
        return Node(label=data[target].mode()[0])

    # Calculate information gain using Gini index
    gains = [information_gain_gini(data, feature, target) for feature in features]

    # Greedy search to find the best feature
    max_gain_idx = np.argmax(gains)
    best_feature = features[max_gain_idx]

    # Make a node
    node = Node(feature=best_feature)

    # Loop over the best feature
    for value in data[best_feature].unique():
        subset = data[data[best_feature] == value].drop(best_feature, axis=1)
        node.children[value] = make_tree_gini(subset, target)

    return node
```

```python
## Make tree with Gini index
tree_gini = make_tree_gini(data, 'Infected')
print('- - - - - - - - - - - -')
print('Tree with Gini Index:', tree_gini)

## Visualize with Gini index
from graphviz import Digraph, nohtml

def plot_tree_gini(tree, g, parent_name=None, edge_label=None):
    if tree.feature is not None:
        current_node_name = tree.feature
        node_label = (
            f"{tree.feature}\n"
            f"Gini Index: {information_gain_gini(data, tree.feature, 'Infected'):.4f}\n"
            f"Samples: {len(data)}\n"
            f"Samples per Class: {data['Infected'].value_counts().to_dict()}"
        )

    else:
        current_node_name = f"Leaf_{tree.label}"
        node_label = (
            f"Label: {tree.label}\n"
            f"Samples: {len(data)}\n"
            f"Samples per Class: {data['Infected'].value_counts().to_dict()}"
        )

    g.node(current_node_name, label=nohtml(node_label))

    if parent_name is not None:
        g.edge(parent_name, current_node_name, label=edge_label)

    for value, child_node in tree.children.items():
        plot_tree_gini(child_node, g, current_node_name, str(value))

# Create a new graph
g_gini = Digraph('g_gini', filename='decision-tree-gini.gv', node_attr={'shape': 'box'})

# Plot the tree with Gini index and save the visualization
plot_tree_gini(tree_gini, g_gini)
g_gini.render('decision_tree_gini', format='png', view=True)
```



درخت تصمیم با معیار gini

۲.۴-

دیتاست سرطان سینه

مجموعه داده مربوط به سرطان سینه ۵۶۹ داده دارد. متشکل از ۳۰ ویژگی و ۲ کلاس خوش‌خیم و بدخیم است. کلاس خوش‌خیم ۲۱۲ نمونه و بدخیم ۳۵۷ نمونه است. هدف تشخیص نوع بیماری از روی ویژگی‌هاست و با یک مسئله طبقه‌بندی سر و کار داریم.

دیتاست سرطان سینه را از کتابخانه Scikit-Learn فراخوانی می‌کنیم. دیتاست به دو بخش آموزش و آزمون تقسیم می‌شود.

```
# imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_breast_cancer
from sklearn import tree

# classification
breast_cancer = load_breast_cancer()
X, y = breast_cancer.data, breast_cancer.target

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42, test_size=0.15)
X_train_size = X_train.shape
X_test_size = X_test.shape
print('Xtrain size =', X_train_size ,'Xtest size =', X_test_size)
print('')
```

```
Xtrain size = (483, 30) Xtest size = (86, 30)
```

در این بخش از کد، یک Grid Search برای بهینه‌سازی هایپرپارامترهای مدل درخت تصمیم انجام می‌شود. زیرا انتخاب درست هایپرپارامترها می‌تواند تأثیر زیادی در کارایی مدل داشته باشد.

Grid Search برای بهینه‌سازی هایپرپارامترها:

param_grid: یک دیکشنری که حاوی مقادیر مختلف برای هایپرپارامترهای max_depth (حداکثر عمق درخت) ، ccp_alpha (ضریب هرس) و criterion (معیار ارزیابی) است.

ساخت Grid Search Object:

GridSearchCV: یک الگوریتم Grid Search که به عنوان تخمین‌گر مدل از DecisionTreeClassifier استفاده می‌کند. این تابع برای جستجوی مقادیر بهینه این هایپرپارامترها با استفاده از معیارهای مختلف و اعتبارسنجی(cv) ۵-fold استفاده می‌کند.

grid_search.fit(X_train, y_train): اجرای Grid Search بر روی داده‌های آموزش. در این مرحله، الگوریتم Grid Search تمامی ترکیب‌های مختلف هایپرپارامترها را امتحان می‌کند و بهترین ترکیب را بر اساس عملکرد مدل در طول اعتبارسنجی انتخاب می‌کند.

best_params: مقادیر بهینه‌سازی شده برای max_depth، ccp_alpha و criterion که توسط Grid Search انتخاب شده‌اند.

best_clf: یک مدل جدید از DecisionTreeClassifier با استفاده از بهترین پارامترهای یافته می‌شود.

best_clf.fit(X_train, y_train): مدل جدید با استفاده از داده‌های آموزش، آموزش داده می‌شود.

درخت تصمیم بهینه‌سازی شده با استفاده از tree.plot_tree رسم می‌شود. با استفاده از tree.export_text، اطلاعات مدل بهینه‌سازی شده به صورت متنی نمایش داده می‌شود.

```python
# Define the parameter grid
param_grid = {
    'max_depth': [3, 5, 7],
    'ccp_alpha': [0.001, 0.01, 0.1],
    'criterion': ['gini', 'entropy']
}

# Create the grid search object
grid_search = GridSearchCV(tree.DecisionTreeClassifier(random_state=42), param_grid, cv=5)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

# Create a new classifier with the best parameters
best_clf = tree.DecisionTreeClassifier(random_state=42, **best_params)
best_clf.fit(X_train, y_train)

DT = tree.plot_tree(best_clf)
print(DT)
print('')

r = tree.export_text(best_clf)
print(r)
print('')
```

```
|--- feature_7 <= 0.05
|   |--- feature_20 <= 16.83
|   |   |--- feature_13 <= 48.70
|   |   |   |--- class: 1
|   |   |--- feature_13 >  48.70
|   |   |   |--- class: 0
|   |--- feature_20 >  16.83
|   |   |--- feature_1 <= 16.19
|   |   |   |--- class: 1
|   |   |--- feature_1 >  16.19
|   |   |   |--- class: 0
|--- feature_7 >  0.05
|   |--- feature_22 <= 114.45
|   |   |--- feature_21 <= 25.65
|   |   |   |--- class: 1
|   |   |--- feature_21 >  25.65
|   |   |   |--- class: 0
|   |--- feature_22 >  114.45
|   |   |--- class: 0
```

۱. اگر مقدار feature_7 کمتر یا مساوی با ۰.۰۵ باشد:

اگر مقدار feature_20 کمتر یا مساوی با ۱۶.۸۳ باشد:

اگر مقدار feature_13 کمتر یا مساوی با ۴۸.۷۰ باشد:

- کلاس: ۱

اگر مقدار feature_13 بیشتر از ۴۸.۷۰ باشد:

- کلاس: ۰

اگر مقدار feature_20 بیشتر از ۱۶.۸۳ باشد:

اگر مقدار feature_1 کمتر یا مساوی با ۱۶.۱۹ باشد:

- کلاس: ۱

اگر مقدار feature_1 بیشتر از ۱۶.۱۹ باشد:

- کلاس: ۰

۲. اگر مقدار feature_7 بیشتر از ۰.۰۵ باشد:

اگر مقدار feature_22 کمتر یا مساوی با ۱۱۴.۴۵ باشد:

اگر مقدار feature_21 کمتر یا مساوی با ۲۵.۶۵ باشد:

- کلاس: ۱

اگر مقدار feature_21 بیشتر از ۲۵.۶۵ باشد:

- کلاس: ۰

اگر مقدار feature_22 بیشتر از ۱۱۴.۴۵ باشد:

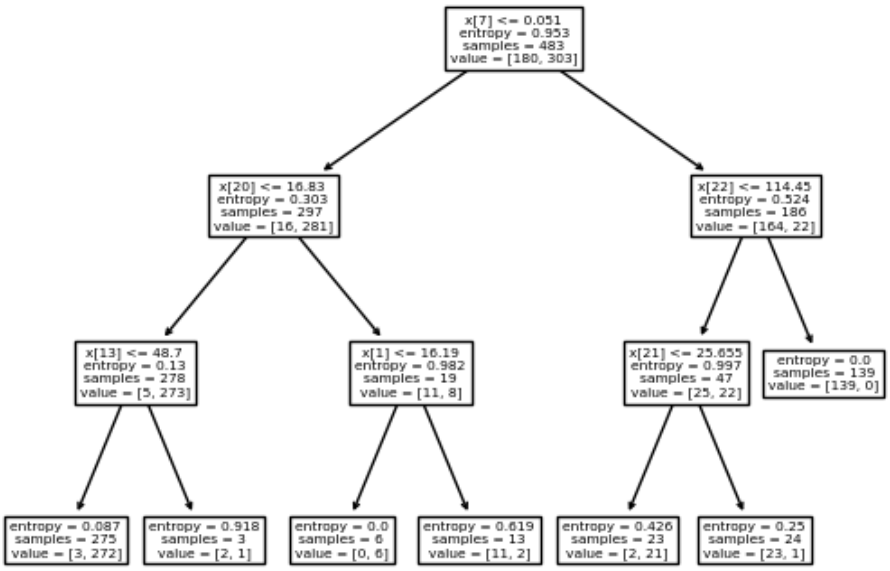
- کلاس: ۰

در اینجا هر یک از شاخه‌ها از گره اصلی با توجه به مقدار یک ویژگی مشخص شده‌اند. این درخت به صورت بازگشتی و بر اساس انتخاب ویژگی‌هایی که بهترین تقسیم را ایجاد می‌کنند، ساخته شده است.

برای مثال، اگر مقدار `feature_7` کمتر یا مساوی با ۰.۰۵ باشد و مقدار `feature_20` نیز کمتر یا مساوی با ۱۶.۸۳

باشد، مدل تصمیم گرفته که نمونه از کلاس ۱ است. این نحوه تفسیر برای هر گره و شاخه ادامه دارد.

- تعداد کل گره‌ها: ۱۱
- تعداد گره‌های برگ (گره‌هایی که دارای تصمیم نهایی یا کلاس هستند): ۶
- تعداد گره‌های تصمیم گیری (گره‌هایی که شرط تصمیم‌گیری دارند): ۵



بهترین درخت‌ها متعلق به max_depth = 3 , ccp_alpha = 0.01, entropy و

max_depth = 3 , ccp_alpha = 0.001, entropy و

max_depth = 3 , ccp_alpha = 0.001, gini است.

معیار جینی:

معیار جینی یک معیار اندازه‌گیری از تبدیل یک مجموعه از اشیاء به یک وضعیت یکنواخت است. این معیار برای اندازه‌گیری عدم اطمینان (impurity) در یک توزیع احتمال استفاده می‌شود. در متغیر تصادفی با توزیع P که به K کلاس تقسیم می‌شود، جینی بر اساس فرمول زیر محاسبه می‌شود:

$$Gini(P) = 1 - \sum_{k=1}^{K} P(k)^2$$

در اینجا:

P(k) : احتمال وقوع کلاس (k).

(K): تعداد کل کلاس‌ها.

برای یک توزیع کاملا یکنواخت (همه احتمالات یکسان)، مقدار Gini صفر خواهد بود و برای یک توزیع کاملا ناهمگن (یک کلاس با احتمال ۱ و سایر کلاس‌ها با احتمال ۰)، مقدار Gini برابر با ۱ است.

آنتروپی:

آنتروپی نیز یک معیار اندازه‌گیری عدم اطمینان (impurity) در یک توزیع احتمال است. برخلاف معیار جینی که در مفهوم درخت تصمیم معمولاً مورد استفاده قرار می‌گیرد، آنتروپی از زمینه تئوری اطلاعات مشتق شده است. آنتروپی برای توزیع احتمال P به صورت زیر محاسبه می‌شود:

$$H(P) = -\sum_{k=1}^{K} P(k) \log_2(P(k))$$

در اینجا:

P(k) : احتمال وقوع کلاس (k).

(K): تعداد کل کلاس‌ها.

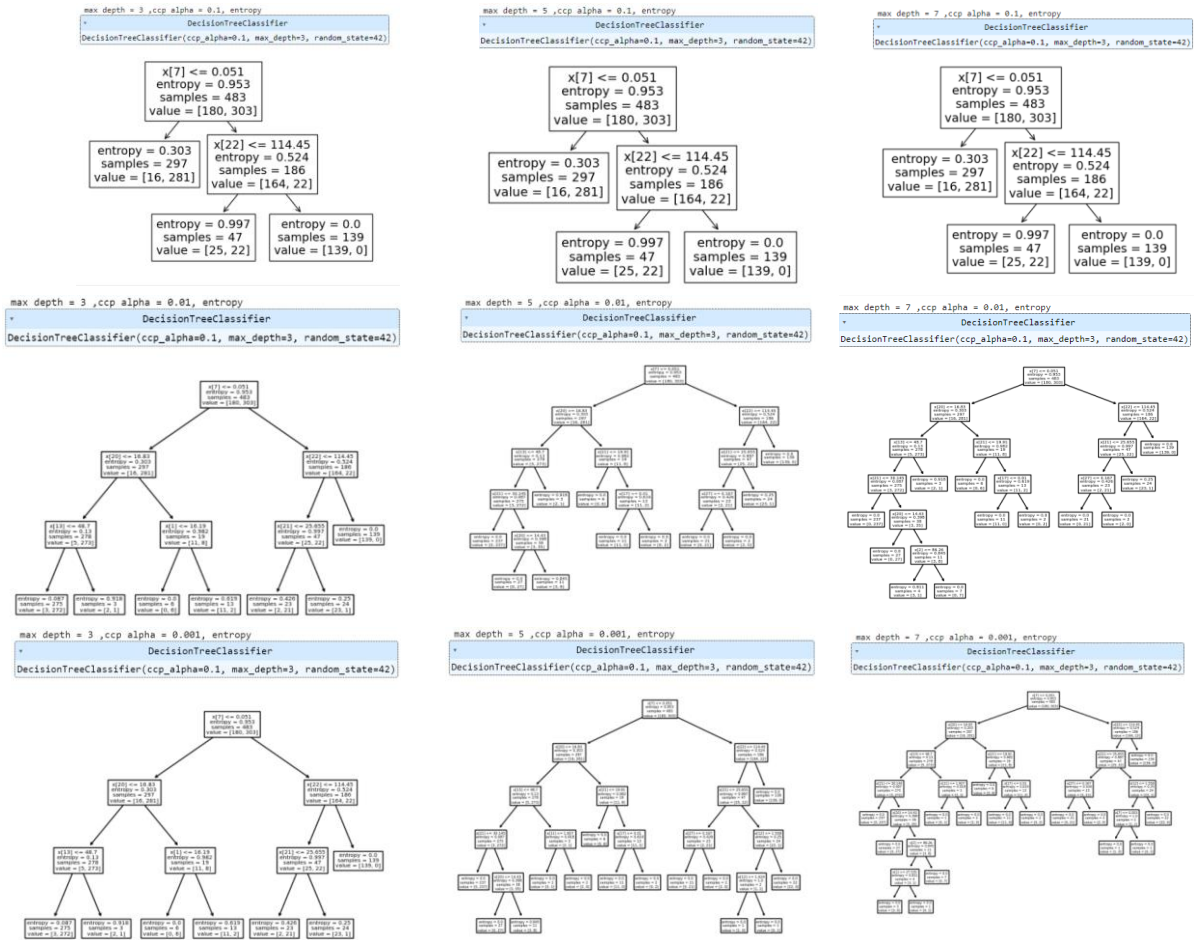
آنتروپی باعث می‌شود که توزیع‌های یکنواخت دارای مقدار آنتروپی بیشتری شوند و توزیع‌های ناهمگن (با احتمالات متفاوت) دارای مقدار آنتروپی کمتری باشند. آنتروپی نیز در بازه [۰, ۱] قرار دارد و مقدار کمتر به معنای یک توزیع ناهمگن (کم اطمینان) و مقدار بیشتر به معنای یک توزیع یکنواخت (زیاد اطمینان) است.

معیار جینی بر اساس مجموع مربعات احتمالات کلاس‌ها محاسبه می‌شود. آنتروپی بر اساس لگاریتم احتمالات کلاس‌ها محاسبه می‌شود.

معیار جینی دارای مقدار اولیه کمتری برای توزیع‌های یکنواخت است. آنتروپی دارای مقدار اولیه بیشتری برای توزیع‌های یکنواخت است.

معیار جینی در بازه [۰, ۱] قرار دارد. آنتروپی نیز در بازه [۰, ۱] قرار دارد.

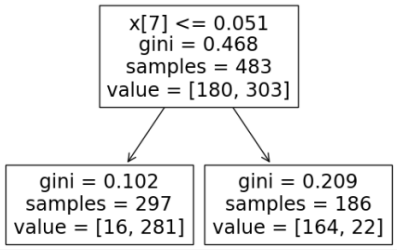
رسم درخت‌های متفاوت



max depth = 3 ,ccp alpha = 0.1, gini

DecisionTreeClassifier

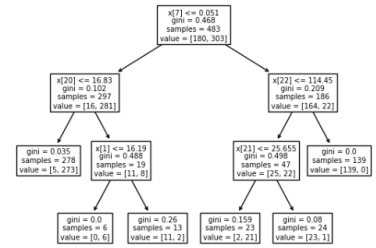
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



max depth = 3 ,ccp alpha = 0.01, gini

DecisionTreeClassifier

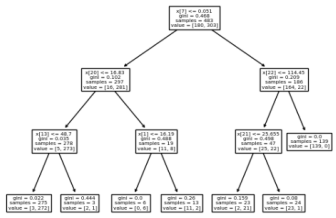
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



max depth = 3 ,ccp alpha = 0.001, gini

DecisionTreeClassifier

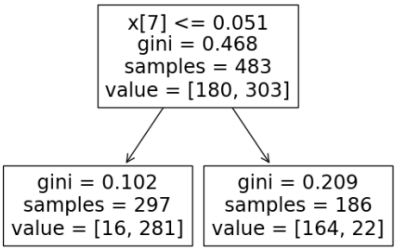
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



max depth = 5 ,ccp alpha = 0.1, gini

DecisionTreeClassifier

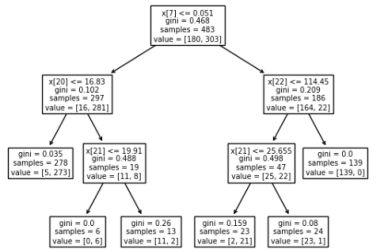
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



max depth = 5 ,ccp alpha = 0.01, gini

DecisionTreeClassifier

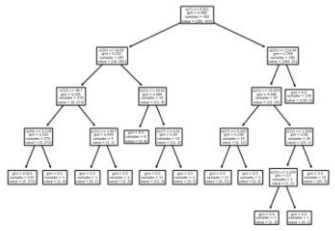
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



max depth = 5 ,ccp alpha = 0.001, gini

DecisionTreeClassifier

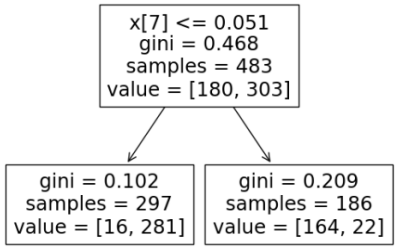
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



max depth = 7 ,ccp alpha = 0.1, gini

DecisionTreeClassifier

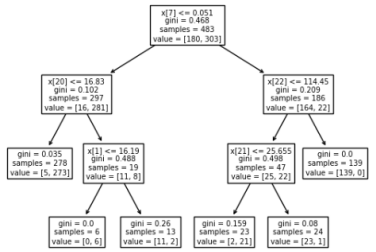
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



max depth = 7 ,ccp alpha = 0.01, gini

DecisionTreeClassifier

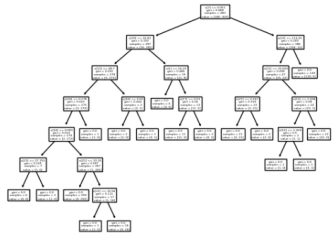
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



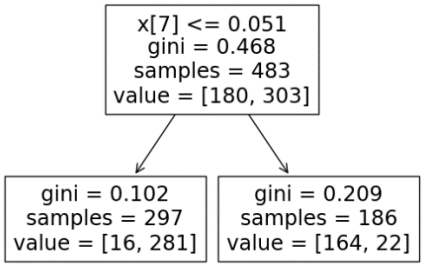
max depth = 7 ,ccp alpha = 0.001, gini

DecisionTreeClassifier

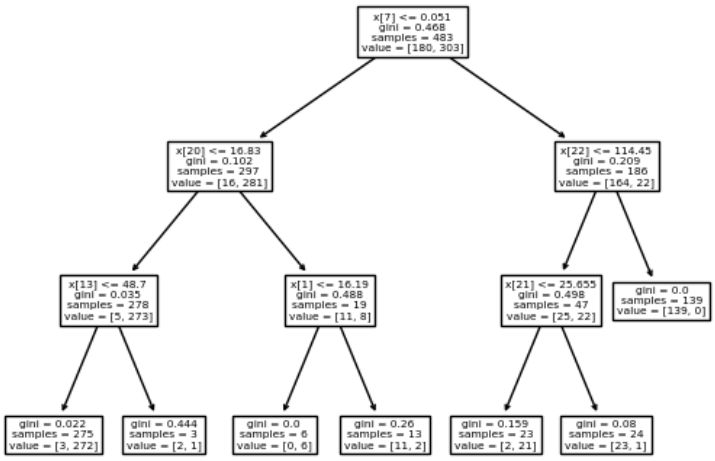
DecisionTreeClassifier(ccp_alpha=0.1, max_depth=3, random_state=42)



مقایسه فراپارامتر ccp alpha کوچک و بزرگ



Ccp alpha = 0.2



Ccp_alpha = 0.001

ضریب هرس کوچکتر نتیجه دقیق‌تری ارائه می‌دهد.

مدل بهینه‌سازی شده روی داده‌های آزمون پیش‌بینی می‌شود. از توابع predict و score برای پیش‌بینی و ارزیابی دقت مدل بر روی داده‌های آزمون استفاده می‌شود.

```
best_clf.predict(X_test)
best_clf.score(X_test, y_test)
best_clf.predict_proba(X_test)
```

مسیر تصمیم برای داده‌های آزمون (X_test[[i]]) با استفاده از decision_path نمایش داده می‌شود.

این قسمت از کد یک حلقه for بر روی تمام نمونه‌های داده تست ایجاد می‌کند. در هر مرحله از حلقه، مسیر تصمیم گیری برای یک نمونه از داده تست محاسبه شده و به همراه کلاس پیش‌بینی شده نمایش داده می‌شود. برخی از اطلاعاتی که این قسمت از کد ارائه می‌دهد، عبارتند از: مسیر تصمیم گیری که داده تست از آن عبور کرده‌است و کلاس پیش‌بینی شده برای هر نمونه از داده تست

```python
for i in range(len(X_test)):
    decision_path = best_clf.decision_path(X_test[[i]])
    print(f"Decision Path for Sample {i+1}:", decision_path.toarray())
    prediction = best_clf.predict(X_test[[i]])
    print(f"Predicted Class for Sample {i+1}: {prediction[0]}\n")
```

```
Decision Path for Sample 1: [[1 1 1 1 0 0 0 0 0 0 0 0 0]]
Predicted Class for Sample 1: 1

Decision Path for Sample 2: [[1 0 0 0 0 0 0 0 1 0 0 0 1]]
Predicted Class for Sample 2: 0

Decision Path for Sample 3: [[1 0 0 0 0 0 0 0 1 0 0 0 1]]
Predicted Class for Sample 3: 0
```

بر اساس درخت تصمیم، نمونه با شماره ۱ در گره‌های ۱، ۲، ۳، ۴ حضور داشته و پیش‌بینی کلاس ۱ انجام شده است.

بر اساس درخت تصمیم، نمونه با شماره ۲ در گره‌های ۱، ۹ و ۱۳ حضور داشته و پیش‌بینی کلاس ۰ انجام شده است.

دیتاست Drugs

در این دیتاست هدف ما تشخیص این است که برای هر بیمار چه دارویی مناسب است. بدین صورت که چند نمونه داریم با ویژگی‌های مختلف شامل سن، جنس، فشار خون ، سدیم-پتاسیم و کلسترول، با این ویژگی‌ها انتخاب می‌کنیم که برای هر بیمار کدام یک از ۵ داروی A، B، C، x و یا y مناسب است. این دیتاست شامل ۵ ستون ویژگی بوده و یک ستون هدف که Drug است.

این یک نمونه از طبقه‌بندی‌کننده چندکلاسه است و می‌توان از بخش آموزش مجموعه داده برای ساختن درخت تصمیم استفاده کرد و سپس از آن برای پیش‌بینی کلاس یک بیمار ناشناخته یا تجویز دارو برای بیمار جدید استفاده کرد.

بارگزاری دیتاست

```
from google.colab import files

# Upload kaggle.json
uploaded = files.upload()
```

```
Choose Files drug200.csv
- drug200.csv(text/csv) - 5827 bytes, last modified: 1/31/2024 - 100% done
Saving drug200.csv to drug200.csv
```

```
!pip install kaggle
!mkdir -p ~/.kaggle
!mv kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json
!kaggle datasets download -d pablomgomez21/drugs-a-b-c-x-y-for-decision-trees
!unzip drugs-a-b-c-x-y-for-decision-trees.zip
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (1.5.16)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle) (2023.11.17)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from kaggle) (4.66.1)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle) (8.0.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle) (2.0.7)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle) (6.1.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle) (3.6)
mv: cannot stat 'kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
Traceback (most recent call last):
  File "/usr/local/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/usr/local/lib/python3.10/dist-packages/kaggle/__init__.py", line 23, in <module>
    api.authenticate()
  File "/usr/local/lib/python3.10/dist-packages/kaggle/api/kaggle_api_extended.py", line 403, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in /root/.kaggle. Or use the environment method.
unzip:  cannot find or open drugs-a-b-c-x-y-for-decision-trees.zip, drugs-a-b-c-x-y-for-decision-trees.zip.zip or drugs-a-b-c-x-y-for-decision-trees.zip.ZIP.
```

دیتاست را بصورت دیتافریم در می‌آوریم.

```
import pandas as pd

# Load the dataset
df = pd.read_csv('/content/drug200.csv')

# Explore the dataset and preprocess if needed
```

عملیات پیش‌پردازش روی دیتاست را انجام می‌دهیم تا داده‌های غیرعددی را به عدد تبدیل کنیم.

```
# encode categorical features with label_encoding
from sklearn.preprocessing import LabelEncoder

# initiating the class
label_enc = LabelEncoder()

# columns that are categorical
cols = drugs.select_dtypes(include='O').columns
# looping on each column in the dataset
for col in cols:
    # Label encoding each column
    drugs[col] = label_enc.fit_transform(drugs[col])

# displaying the data after encoding
drugs
```

	Age	Sex	BP	Cholesterol	Na_to_K	Drug
0	23	0	0	0	25.355	4
1	47	1	1	0	13.093	2
2	47	1	1	0	10.114	2
3	28	0	2	0	7.798	3
4	61	0	1	0	18.043	4
...	...	...	...	...	...	...
195	56	0	1	0	11.567	2
196	16	1	1	0	12.006	2
197	52	1	2	0	9.894	3
198	23	1	2	1	14.020	3
199	40	0	1	1	11.349	3

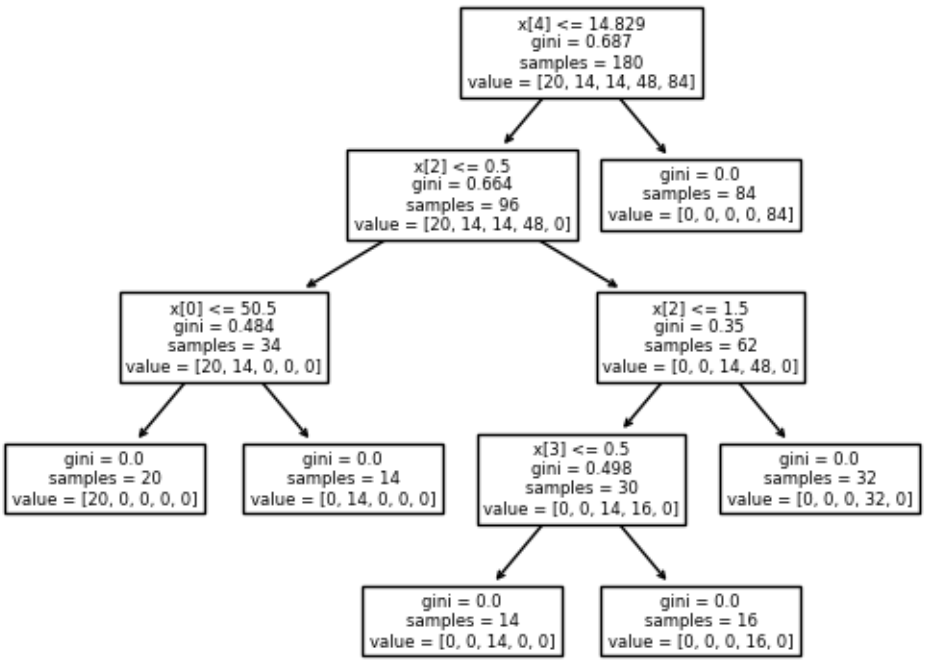
200 rows × 6 columns

۵ ستون اول را ویژگی و ستون آخر را هدف در نظر می‌گیریم. سپس عملیات آموزش مدل و رسم درخت را مشابه بخش قبل انجام می‌دهیم.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier as DTC, DecisionTreeRegressor as DTR
from sklearn.tree import plot_tree

# dividing the data into X, y
X = drugs.drop(columns='Drug')
y = drugs['Drug']
# split the data into train set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.1, random_state=42)
# Building the model
tree_clf0 = DTC()
# Fitting the model
tree_clf0.fit(X_train,y_train)
DT = tree.plot_tree(tree_clf0)
print(DT)
print('')

r = tree.export_text(tree_clf0)
print(r)
print('')
```

```
[Text(0.625, 0.9, 'x[4] <= 14.829\ngini = 0.687\nsamples = 180\nvalue = [20, 14, 14, 48, 84]'), Text(0.5, 0.7, 'x[2] <= 0.5\ngini = 0.664\nsamples = 96\nvalue = [20, 14, 14, 48, 0]'), Text(0.25, 0.5, 'x[0] <= 50
```

```
|--- feature_4 <= 14.83
|   |--- feature_2 <= 0.50
|   |   |--- feature_0 <= 50.50
|   |   |   |--- class: 0
|   |   |--- feature_0 >  50.50
|   |   |   |--- class: 1
|   |--- feature_2 >  0.50
|   |   |--- feature_2 <= 1.50
|   |   |   |--- feature_3 <= 0.50
|   |   |   |   |--- class: 2
|   |   |   |--- feature_3 >  0.50
|   |   |   |   |--- class: 3
|   |   |--- feature_2 >  1.50
|   |   |   |--- class: 3
|--- feature_4 >  14.83
|   |--- class: 4
```



درخت با عمق ۴ و معیار gini رسم شده‌است. در نهایت تمامی کلاس‌ها بصورت دقیق و بدون خطا و با دقت ۱۰۰٪ از یکدیگر جدا شدند.

محاسبه معیارهای ارزیابی

```
# calculating the accuracy
from sklearn.metrics import accuracy_score, classification_report
y_pred = tree_clf0.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('accuracy =',accuracy)

report = classification_report(y_test, y_pred)
print(report)
```

```
accuracy = 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         3
           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00         2
           3       1.00      1.00      1.00         6
           4       1.00      1.00      1.00         7

    accuracy                           1.00        20
   macro avg       1.00      1.00      1.00        20
weighted avg       1.00      1.00      1.00        20
```

۳.۴-

این مجموعه داده با ترکیب اطلاعات امید به زندگی، عوامل سلامت و اقتصادی از سازمان جهانی بهداشت و سازمان ملل متحد، از سال ۲۰۰۰ تا ۲۰۱۵ برای ۱۹۳ کشور جمع‌آوری شده است. مطالعه به بررسی تأثیر عوامل ایمن‌سازی، مرگ و میر، عوامل اقتصادی و اجتماعی بر امید به زندگی می‌پردازد. با حذف داده‌های از دست رفته و تقسیم متغیرهای پیش‌بینی‌کننده به دسته‌های مختلف، این مجموعه داده شامل ۲۲ ستون و ۲۹۳۸ ردیف است. هدف اصلی این مجموعه داده بررسی عوامل مختلفی است که ممکن است بر امید به زندگی تأثیر بگذارند. مطالعه به منظور پاسخ به سوالاتی از جمله تأثیر عوامل ایمن‌سازی، مرگ و میر، عوامل اقتصادی و اجتماعی بر امید به زندگی می‌پردازد. از طریق تحلیل این داده‌ها، تلاش برای شناخت علل کاهش یا افزایش امید به زندگی در کشورها و تأثیر عوامل مختلف بر این متغیر مهم انجام شده است. این تحقیق به سوالاتی پیرامون تأثیرات زندگی روزمره، شیوه زندگی، عوامل بهداشتی و اقتصادی بر امید به زندگی می‌پردازد و می‌کوشد الگوها و ارتباطات میان این متغیرها را تجزیه و تحلیل کند.

ابتدا کتابخانه‌های مورد نیاز را فراخوانی می‌کنیم. سپس دیتاست را بارگزاری کرده و آن را بصورت دیتافریم درمی‌آوریم.

```
# imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import plot_tree
```

```
# load data
!pip install --upgrade --no-cache-dir gdown
!gdown 1cX-RoHkW3ZLYxE0yIyUoz73q8ljh1vmf
df = pd.read_csv('/content/Life_Expectancy_Data.csv')
```

```
Requirement already satisfied: gdown in /usr/local/lib/python3.10/dist-packages (5.0.1)
Requirement already satisfied: beautifulsoup4 in /usr/local/lib/python3.10/dist-packages (from gdown) (4.11.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from gdown) (3.13.1)
Requirement already satisfied: requests[socks] in /usr/local/lib/python3.10/dist-packages (from gdown) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from gdown) (4.66.1)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4->gdown) (2.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (2023.11.17)
Requirement already satisfied: PySocks!=1.5.7,>=1.5.6 in /usr/local/lib/python3.10/dist-packages (from requests[socks]->gdown) (1.7.1)
Downloading...
From: https://drive.google.com/uc?id=1cX-RoHkW3ZLYxE0yIyUoz73q8ljh1vmf
To: /content/Life_Expectancy_Data.csv
100% 306k/306k [00:00<00:00, 122MB/s]
```

جهت اطلاع از انواع داده‌ها و اطلاعات دیتاست، پنج ردیف اول آن را نمایش می‌دهیم.

```
df.head()
```

	Country	Continent	Year	Status	Life_expectancy	Adult_Mortality	infant_deaths	Alcohol	percentage_expenditure	Hepatitis_B	...	Polio	Total_expenditure	Diphtheria	HIV/AIDS	GDP	Population	thinness 1-19 years	thinness 5-9 years	Income_composition
0	Afghanistan	Asia	2015	Developing	65.0	263	62	0.01	71.279624	65.0	...	6.0	8.16	65	0.1	584.259210	33736494	17.2	17.3	
1	Afghanistan	Asia	2014	Developing	59.9	271	64	0.01	73.523582	62.0	...	58.0	8.18	62	0.1	612.696514	327582	17.5	17.5	
2	Afghanistan	Asia	2013	Developing	59.9	268	66	0.01	73.219243	64.0	...	62.0	8.13	64	0.1	631.744976	31731688	17.7	17.7	
3	Afghanistan	Asia	2012	Developing	59.5	272	69	0.01	78.184215	67.0	...	67.0	8.52	67	0.1	669.959000	3696958	17.9	18.0	
4	Afghanistan	Asia	2011	Developing	59.2	275	71	0.01	7.097109	68.0	...	68.0	7.87	68	0.1	63.537231	2978599	18.2	18.2	

5 rows × 23 columns

برای پیداکردن نام دقیق ستون هدف نام تمامی ستون‌ها را استخراج می‌کنیم.

```
print(df.columns)
```

```
Index(['Year', 'Life_expectancy', 'Adult_Mortality', 'infant_deaths',
       'Alcohol', 'percentage_expenditure', 'Hepatitis_B', 'Measles', 'BMI',
       'under_five_deaths',
       ...
       'Population_9939678', 'Population_99429', 'Population_996698',
       'Population_9971727', 'Population_9977446', 'Population_99789',
       'Population_997961', 'Population_9987333', 'Population_9999617',
       'Population_Unknown'],
      dtype='object', length=2492)
```

این بخش شامل:

پر کردن مقادیر از دست رفته با میانگین هر ستون.

تبدیل ستون‌های دسته‌بندی به فرمت عددی با استفاده از رمزگذاری one hot.

تنظیم ستون هدف

تعیین ستون ویژگی‌ها (X) و هدف (y) برای مدل

مجموعه داده به مجموعه‌های آموزش و آزمون تقسیم می‌شود. ۱۵ درصد از داده‌ها برای آزمون استفاده می‌شود.

در اینجا درخت تصمیم رگرسیون مقداردهی اولیه شده و به داده‌های آموزش فیت می‌شود. پس از آن، پیش‌بینی‌هایی بر روی داده‌های آزمون انجام می‌شود.

عملکرد مدل با استفاده از میانگین مربعات خطا و r2score ارزیابی می‌شود.

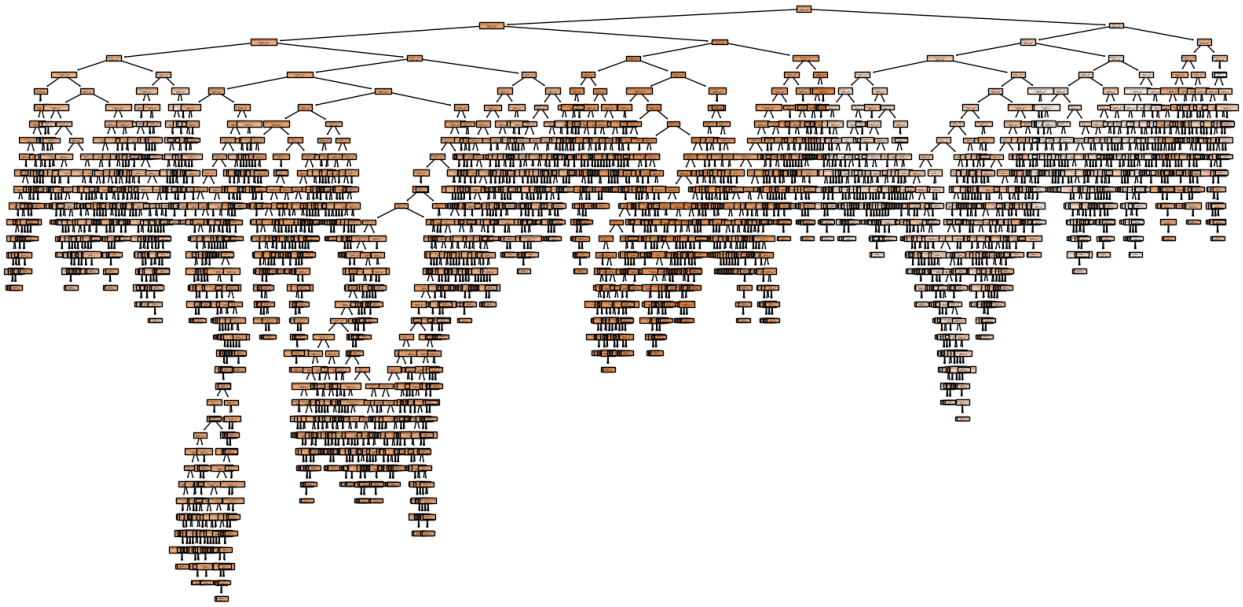
در نهایت، درخت تصمیم با استفاده از tree.plot_tree() رسم می‌شود.

در این کد داده ها را از قبل پیش‌پردازش می‌شوند، یک مدل رگرسیون درخت تصمیم را ایجاد و فیت می‌کند، عملکرد آن را ارزیابی می‌کند و درخت تصمیم حاصل را رسم می‌کند.

```python
# imports
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# Preprocess the data
# Check missing values
df.fillna(df.mean(), inplace=True)

# convert categorical columns to numerical
df = pd.get_dummies(df, drop_first=True)

# 'Life expectancy' is target column
target_column = 'Life_expectancy'

# If column names have any leading or trailing spaces, remove them
df.columns = df.columns.str.strip()

# define features (X) and target (y)
X = df.drop(columns=[target_column]).values
y = df[target_column].values

# Split train and test data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=13)

# Initialize the DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=13)

# Fit the model on the training data
regressor.fit(X_train, y_train)

# Predict on the test data
y_pred = regressor.predict(X_test)

# Evaluate the model
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred)}")
print(f"R^2 Score: {r2_score(y_test, y_pred)}")

# Visualize the decision tree
plt.figure(figsize=(20,10))
tree.plot_tree(regressor, filled=True, feature_names=df.drop(columns=[target_column]).columns)
plt.show()
```

```
Mean Squared Error: 5.500702702702704
R^2 Score: 0.9317072903266679
```

مقادیر معیارهای ارزیابی نشان می‌دهد دقت مدل خوب است.



تنظیم هایپرپارامترهای مختلف

```python
from sklearn.model_selection import GridSearchCV
# Hyperparameter grid
param_grid = {
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2']
}

# Initialize the DecisionTreeRegressor
regressor = DecisionTreeRegressor(random_state=13)

# Initialize the GridSearchCV
grid_search = GridSearchCV(estimator=regressor, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1)

# Fit the model on the training data
grid_search.fit(X_train, y_train)

# Best hyperparameters
print(f"Best hyperparameters: {grid_search.best_params_}")

# Predict on the test data using the best model
best_regressor = grid_search.best_estimator_
y_pred = best_regressor.predict(X_test)

# Evaluate the model
print(f"Mean Squared Error: {mean_squared_error(y_test, y_pred)}")
print(f"R^2 Score: {r2_score(y_test, y_pred)}")
```

```
Fitting 5 folds for each of 135 candidates, totalling 675 fits
Best hyperparameters: {'max_depth': 15, 'max_features': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
Mean Squared Error: 5.799563930913609
R^2 Score: 0.9279968474625598
```

۵.