

تکلیف نخست / مرجان مودت

```
import pandas as pd
import numpy as np
from matplotlib import Path
import matplotlib.pyplot as plt
```

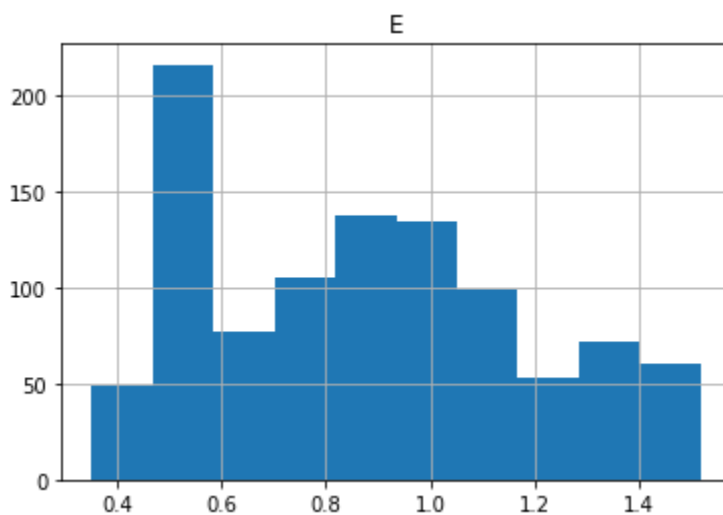
```
datafile=Path('SimData','/content/derive/MyDrive/data.xlsx')
df = pd.read_excel(datafile)
df.head()
```

ابتدا به کمک دستورهایی بالا و کتابخانه‌های موجود فایل داده‌ها را خوانده و آن‌ها را نمایش دادم:

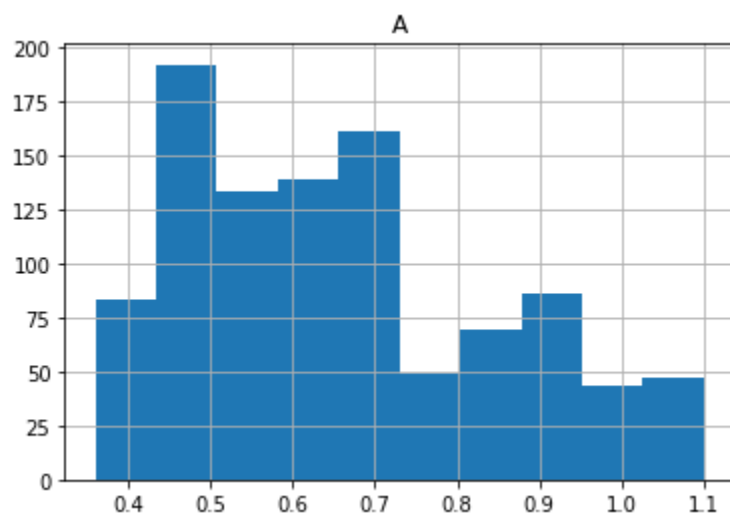
	age	Ischemia	Diastolic function	E	A	Grade	E/A	e	a	e/a
0	32	POS	NL	1.31	0.58	0	2.258621	0.20	0.08	2.500000
1	35	POS	NL	0.92	0.51	0	1.803922	0.15	0.07	2.142857
2	41	POS	NL	0.84	0.61	0	1.377049	0.13	0.08	1.625000
3	33	POS	NL	1.04	0.45	0	2.311111	0.17	0.06	2.833333
4	30	POS	NL	1.11	0.49	0	2.265306	0.18	0.07	2.571429

(آ) بافت نگار داده‌های موجود در ستون‌های E و A و e و a به ترتیب زیر است:

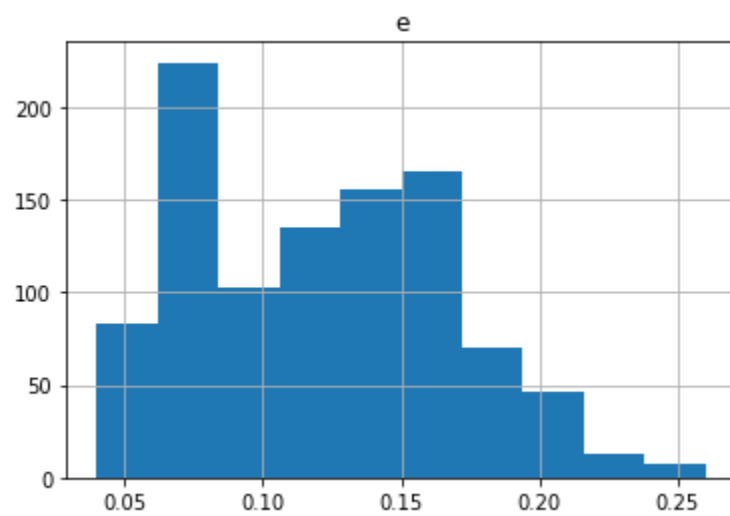
```
df.hist(column='E')
```



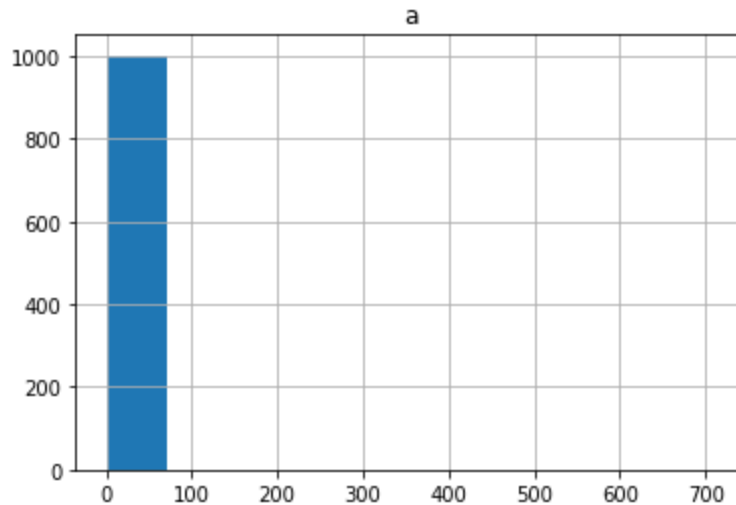
```
df.hist(column='A')
```



```
df.hist(column='e')
```



```
df.hist(column='a')
```

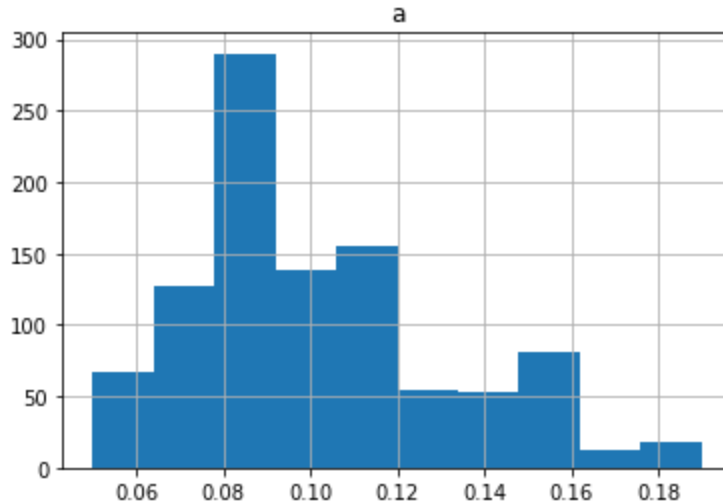


بله، با هیستوگرام می شود به وجود داده های پرت پی برد، همانطور که در شکل های بالا می بینید E و A و e داده پرتی ندارند چون رنج محور x که نشان دهنده بازه داده هاست بازه اعداد به هم نزدیک است و داده ها روی ستون های مختلف که نشان دهنده تعداد هر داده در هر بازه است پخش تقریباً نزدیکی دارند ولی در مورد a می بینیم که محور x را تا ۷۰۰ کشیده شده و در بازه ۰ تا ۱۰۰ که رنج خیلی بزرگی است نزدیک به ۱۰۰۰ عدد هست و ما متوجه می شویم که این بازه داده پرت دارد.

مقادیر ستون a باید کوچک تر از ۰.۲ باشد، پس سطرهایی که بیش تر از ۰.۲ است را حذف می کنیم:

```
indexNames = df[ (df['a'] > 0.2)].index
df.drop(indexNames , inplace=True)
df.hist(column='a')
```

در کد بالا ابتدا اندیس خانه هایی که این ستون را بیش تر از ۰.۲ دارند مشخص شد و سپس سطرهایی با این ایندکس از دیتا حذف شده و هیستوگرام ستون a را نمایش می دهیم.




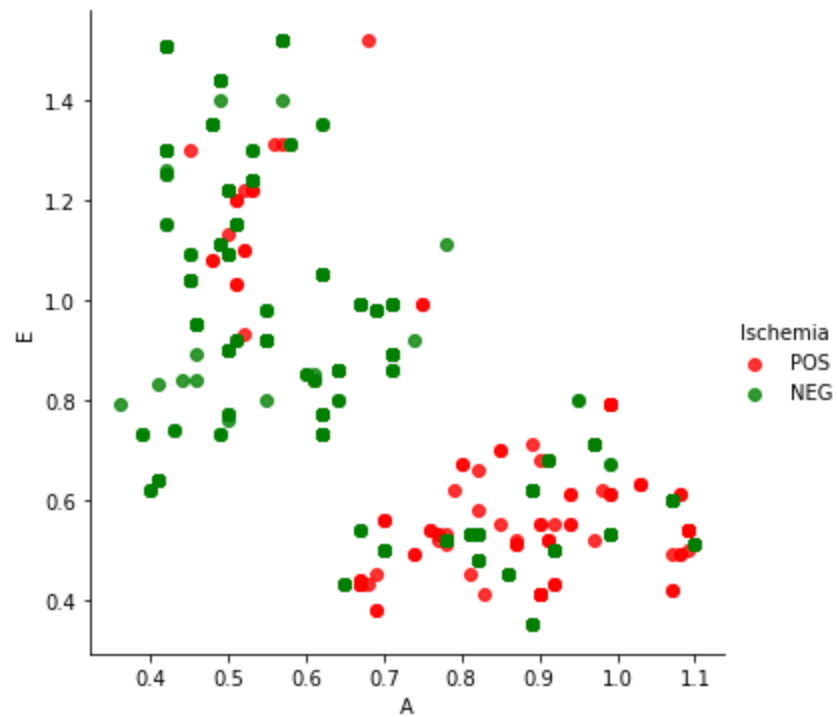
ب) ابتدا کتابخانه های لازم را import می کنیم:

```
# Pandas for managing datasets
import pandas as pd
# Matplotlib for additional customization
from matplotlib import pyplot as plt
# Seaborn for plotting and styling
import seaborn as sns

# Scatterplot arguments
sns.lmplot(x='A', y='E', data=df,
           fit_reg=False, # No regression line
           hue='Ischemia',
           palette=dict(POS="r", NEG="g")) # Color by evolution Ischemia
```

برای رسم نمونه ها از کتابخانه seaborn استفاده می کنیم و بر حسب مقدار **Ischemia** برای برچسب **pos** رنگ قرمز و برای برچسب **neg** رنگ سبز را در نظر می گیریم. چون **A** قرار است محور **x** را نمایش دهد آن را به عنوان متغیر اول به **x** می دهیم سپس **E** را به عنوان متغیر دوم به **y** می دهیم و این ها را از داده هایمان که در متغیر **df** قرار دادیم می خوانیم قرار نیست رگرسیون آن را نمایش دهیم پس **false** میگذاریم و بر حسب ستون **'Ischemia'** قرار است رنگ هر نمونه مشخص شود پس این ستون را به **hue** می دهیم و از پلت رنگی سفارشی خودمان نه پیش فرض برای برچسب مثبت قرمز و منفی سبز را در نظر گرفتیم

 <seaborn.axisgrid.FacetGrid at 0x7fc9cc3d56d0>



(پ)

```
df['Ischemia'] = np.where(df['Ischemia']=='POS', 1, -1) #pos => 1 , neg => -1
```

ابتدا در یک بلوک جدا گانه خط کد بالا را نوشتم که چند بار اجرا نشود.

```
import numpy
```

```
for_train=df.sample(frac = 0.6) #60% train
```

```
X = for_train.iloc[:, 3:5] # A,E columns
```

```
train=X.iloc[:,::-1] #train
```

```
for_test = df.drop(train.index) #40% reminder for test
```

```
test_a = for_test.iloc[:, 3:5] # A,E columns test
```

```
test=test_a.iloc[:,::-1] #test
```

```
labels_train = for_train.iloc[:, 1:2] #labels_train
```

```
Y = df.drop(labels_train.index) #drop labels for train
```

```
labels_test = Y.iloc[:, 1:2] #labels_test
```

```
w=[]
```

```
w.append(0)
```

```

w.append(0)
w = np.array(w)

b=0
u =[]
u.append(0)
u.append(0)
u = np.array(u)
B=0
c=1
MaxIter=100 #epochs
D= len(for_train) #Number of training samples
ce = np.zeros((MaxIter,))
eter = np.zeros((MaxIter,))
for m in range(MaxIter):
    counter=0
    for d in range(D): # D number of samples
        t1=np.array(train.iloc[d:d+1,:]) #sample train row d
        w=w.reshape((2,1))
        q=np.dot(t1,w) #x.w
        z=q+b # (w.x+b)
        y=labels_train.iloc[d:d+1,:]
        Y=np.array(y).item()
        h=Y*z #y*(w.x+b)

        if h <= 0:
            counter=counter+1
            w=w.reshape((1, 2))
            w=w+(Y*t1)
            b=b+Y
            u=u+(Y*c*t1)
            B=B+(Y*c)
            #print("iteration: {} train: {} ,w ={}".format(m,d,w) )
            #print("b ={}".format(b) )
            #print("u ={}".format(u) )
            #print("B ={}".format(B) )
        #end if
        c=c+1
    #end for
    #show iteration error
    #plot in iter
    CE=0
    CE=counter/D #classification error
    ce[m]=np.array(CE).item()
    eter[m]=np.array(m).item()

```

```

#shuffle
df_shuffle=df_train.sample(frac =1) #shuffle train
X = df_shuffle.iloc[:, 3:5] # A,E columns train
train=X.iloc[:, :-1] #train
labels_train = df_shuffle.iloc[:, 1:2] #labels for train
#end for
plt.plot(eter, ce, 'g')
w=w.reshape((1, 2))
w=w-(1/c)*u
b=b-(1/c)*B

print("w end ={}".format(w) )
print("b end ={}".format(b) )

```

توضیح کد:

```

df['Ischemia'] = np.where(df['Ischemia']=='POS', 1, -
1) #pos => 1 , neg => -1

```

در این خط کد برای ستون 'Ischemia' به جای POS و NEG از برچسب متناظر ۱ و -۱ استفاده می کنیم

```

import numpy

for_train=df.sample(frac = 0.6) #60% train
X = for_train.iloc[:, 3:5] # A,E columns
train=X.iloc[:, :-1] #train

```

کتابخانه لازم را ایمپورت کرده و سپس ۶۰٪ از داده ها را برای آموزش جدا کرده و ستون های مد A و E را جدا کرده و سپس با توجه به اینکه در صورت سوال قبلی A برای محور افقی و E برای محور عمودی بود من در train ویژگی اول را ستون A و ویژگی دوم را ستون E در نظر گرفتم

```

for_test = df.drop(train.index) #40% reminder for test
test_a = for_test.iloc[:, 3:5] # A,E columns test
test=test_a.iloc[:, :-1] #test

```

۴۰٪ باقی مانده را هم برای تست در نظر گرفتم

```

labels_train = for_train.iloc[:, 1:2] #labels_train
Y = df.drop(labels_train.index) #drop labels for train
labels_test = Y.iloc[:, 1:2] #labels_test

```

و سپس برچسب های آموزش و تست را مشخص کردم

```
w=[]  
w.append(0)  
w.append(0)  
w = np.array(w)
```

```
b=0  
u =[]  
u.append(0)  
u.append(0)  
u = np.array(u)  
B=0  
c=1
```

```
MaxIter=100 #epochs
```

، با توجه به اینکه طبق الگوریتم average perceptron در اجرای اول با ضرب مقدار اسکالر برچسب در نمونه اول که 1×2 است آپدیت می شود من بردار اولیه w را 1×2 و با مقدار \cdot پر کردم و برای بردار u که وزن های ذخیره شده را آپدیت می کند هم به همین صورت نوشتم و مقدار b و B هم اسکالر \cdot قرار دادم و مقدار اولیه c را هم یک در نظر گرفتم که برچسبی است که به تعداد بارهایی که w آپدیت نمی شود اختصاص دارد.

MaxIter=100 مقدار حد اکثر تعداد اجرای الگوریتم است

```
D= len(for_train) #Number of training samples  
ce = np.zeros((MaxIter,))  
eter = np.zeros((MaxIter,))
```

است و D تعداد نمونه های آموزش و ce و $eter$ که یک بردار 100 تایی است که با \cdot پر شده است

```
for m in range(MaxIter):  
    counter=0  
    for d in range(D): # D number of samples  
        t1=np.array(train.iloc[d:d+1,:]) #sample train row d  
        w=w.reshape((2,1))  
        q=np.dot(t1,w) #x.w  
        z=q+b # (w.x+b)  
        y=labels_train.iloc[d:d+1,:]  
        Y=np.array(y).item()  
        h=Y*z #y*(w.x+b)  
  
        if h <= 0:  
            counter=counter+1
```



```

w=w.reshape((1, 2))
w=w+(Y*t1)
b=b+Y
u=u+(Y*c*t1)
B=B+(Y*c)
#print("iteration: {} train: {} ,w ={}".format(m,d,w) )
#print("b ={}".format(b) )
#print("u ={}".format(u) )
#print("B ={}".format(B) )
#end if
c=c+1
#end for
#show iteration error
#plot in iter
CE=0
CE=counter/D #classification error
ce[m]=np.array(CE).item()
eter[m]=np.array(m).item()
#shuffle
df_shuffle=df_train.sample(frac =1) #shuffle train
X = df_shuffle.iloc[:, 3:5] # A,E columns train
train=X.iloc[:, :-1] #train
labels_train = df_shuffle.iloc[:, 1:2] #labels for train
#end for
plt.plot(eter, ce, 'g')
w=w.reshape((1, 2))
w=w-(1/c)*u
b=b-(1/c)*B

print("w end ={}".format(w) )
print("b end ={}".format(b) )

```

در ۱۰۰ تکرار و هر بار به تعداد نمونه های آموزش این مراحل را انجام می دهیم :

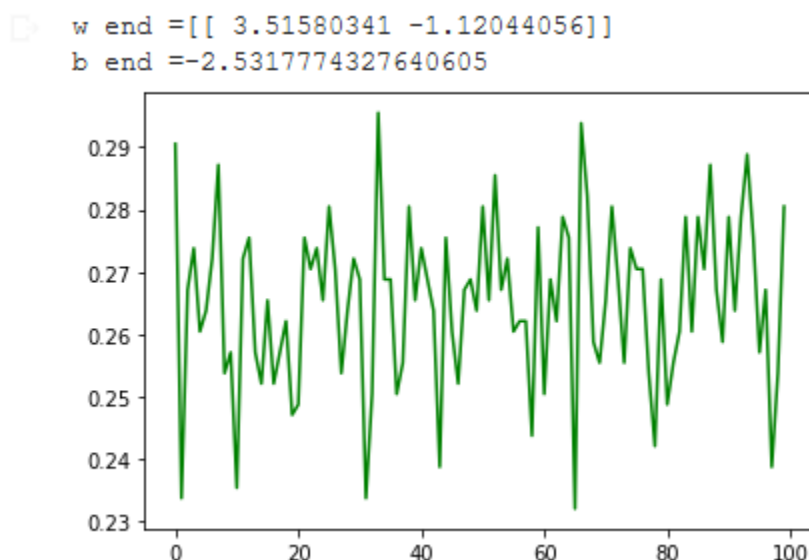
با استفاده از تابع `np.array()` ستون های بردار ورودی را به صورت یک آرایه در می آوریم و آن را به متغیر `t1` می دهیم و سپس `w` را به صورت `reshape(2x1)` می نویسیم چون `q=np.dot(t1,w)`

می خواهیم `t1` که `1x2` است را در `w` که `2x1` است ضرب کرده و حاصل که `1x1` می شود را با `b` جمع نموده و در متغیر `Z` قرار داده و `y=labels_train.iloc[d:d+1,:]` با این دستور مقدار سطر `d` ام را گرفته `np.array(y).item()` و مقدار این آیتم را در `Z` ضرب نموده که `h` را به ما می دهد `y*(w.x+b)`

، حالا اگر h کوچک تر مساوی صفر بود به این معنی که برچسب نمونه با برچسب w نخوانده(هم علامت نباشد) باید وزن و بایاس را آپدیت کنیم پس به `counter` یک واحد اضافه می کنیم و w را `reshape` می کنیم که w و b و u و B را آپدیت کنیم و بعد از آن در انتهای حلقه `for` یک واحد به c اضافه می شود.

بعد در انتهای هر `iter` تعداد برچسب های نادرست را به تعداد کل نمونه های آموزشی تقسیم می شود خطا در هر مرحله که در یک آرایه ذخیره کردیم و نهایتاً آن را نمایش دادیم اما بعد از هر `iter` نمونه های آموزش را بُر زدیم تا سریع تر به همگرایی برسیم.

نتایج حاصل از کد به این صورت است:



همانطور که میبینیم همگرا نشده است.

چون داده های ما توزیع یک نواختی (linear separable) نداشته اند.

(ت)

```
D= len(train) #Number of training samples

#empirical error
count=0
for d in range(D):
    t1=np.array(train.iloc[d:d+1,:]) #sample train row d
    w=w.reshape((2,1))
    q=np.dot(t1,w) #w.x
    z=q+b # (w.x+b)
    y=labels_train.iloc[d:d+1,:]
    Y=np.array(y).item() # label y
    j=Y*z
    if j <= 0:
        count=count+1

CE_one=0
CE_one=count/D #empirical error for train data
print("empirical error for train data ={}".format(CE_one) )

D_two= len(test) #Number of training samples
#empirical error
count_two=0
for d in range(D_two):
    t2=np.array(test.iloc[d:d+1,:]) #sample test row d
    w=w.reshape((2,1))
    q=np.dot(t2,w) #w.x
    z=q+b # (w.x+b)
    y=labels_test.iloc[d:d+1,:]
    Y=np.array(y).item() # label y
    j=Y*z
    if j <= 0:
        count_two=count_two+1

CE_two=0
CE_two=count_two/D_two #empirical error for test data
print("empirical error for test data ={}".format(CE_two) )
```

توضیح کد:

```
D= len(train) #Number of training samples

#empirical error
```

```

count=0
for d in range(D):
    t1=np.array(train.iloc[d:d+1,:]) #sample train row d
    w=w.reshape((2,1))
    q=np.dot(t1,w) #w.x
    z=q+b # (w.x+b)
    y=labels_train.iloc[d:d+1,:]
    Y=np.array(y).item() # label y
    j=Y*z
    if j <= 0:
        count=count+1

CE_one=0
CE_one=count/D #empirical error for train data
print("empirical error for train data ={}".format(CE_one) )

```

تعداد داده های آموزشی را مشخص کرده و به ازای هر نمونه آموز شی تعداد بارهایی که به ازای این نمونه ها الگوریتم خوب کار نکرده را مشخص می کنیم با استفاده از w و b که در مرحله قبل بدست آوردیم و $wx+b \leq 0$ یعنی برچسب را اشتباه حساب کرده بدست آورده و بر تعداد کل نمونه های آموزشی تقسیم می کنیم.

```

D_two= len(test) #Number of training samples
#empirical error
count_two=0
for d in range(D_two):
    t2=np.array(test.iloc[d:d+1,:]) #sample test row d
    w=w.reshape((2,1))
    q=np.dot(t2,w) #w.x
    z=q+b # (w.x+b)
    y=labels_test.iloc[d:d+1,:]
    Y=np.array(y).item() # label y
    j=Y*z
    if j <= 0:
        count_two=count_two+1

CE_two=0
CE_two=count_two/D_two #empirical error for test data
print("empirical error for test data ={}".format(CE_two) )

```

یک بار دیگر برای نمونه های تست این کار را انجام می دهیم یعنی به ازای نمونه های تست می بینیم که w چه برچسبی به آن ها می دهد و این برچسب منطبق بر جواب واقعی آن ها است یا خیر. به تعداد برچسب هایی که منطبق نیست تقسیم بر تعداد کل نمونه های تست می شود

نتایج حاصل:

```
empirical error for train data =0.19031719532554256
empirical error for test data =0.22305764411027568
```

خطا روی نمونه های آموزشی تقریباً 19. و روی نمونه های تست تقریباً 22. می باشد.

(ث)

```
fig, ax = plt.subplots(figsize=(6,6))
arr1 = np.array(for_train.iloc[:, 4:5]) # A column
arr2 = np.array(for_train.iloc[:, 3:4]) # E column
labl = np.array(labels_train.iloc[:,0:1]) # label train
color= ['red' if l == 1 else 'green' for l in labl]

ax.scatter(arr1, arr2, color=color)

#y = -(b / w2) / (b / w1))x + (-b / w2)
x = np.linspace(0.7,1.1,25)
w=w.reshape(1,2)
s1=np.array(w[:,0:1]).item() #column 1 of w
s2=np.array(w[:,1:2]).item() #column 2 of w
y = -(b / s2) / (b / s1))*x + (-b / s2)
ax.plot(x, y, '-b', label='wx+b=0')

ax.set_xlabel('A')
ax.set_ylabel('E')
plt.title('seperator of wx+b=0 for train')

ax.grid()
plt.show()
```

توضیح کد:

```
fig, ax = plt.subplots(figsize=(6,6))
arr1 = np.array(for_train.iloc[:, 4:5]) # A column
arr2 = np.array(for_train.iloc[:, 3:4]) # E column
labl = np.array(labels_train.iloc[:,0:1]) # label train
color= ['red' if l == 1 else 'green' for l in labl]
ax.scatter(arr1, arr2, color=color)
```

ابتدا یک زیر پلات به اندازه 6×6 ایجاد کرده و سپس ستون های A و E را در مجموعه آموزش به همراه برجسب متناظر آن به متغیر ها نسبت داده سپس برای برجسب ۱ رنگ قرمز و در غیر این صورت رنگ سبز را در نظر گرفتیم و آن را نمایش دادیم.

```
#y = -(b / w2) / (b / w1))x + (-b / w2)
x = np.linspace(0.7,1.1,25)
w=w.reshape(1,2)
s1=np.array(w[:,0:1]).item() #column 1 of w
s2=np.array(w[:,1:2]).item() #column 2 of w
y = -(b / s2) / (b / s1))*x + (-b / s2)
ax.plot(x, y, '-b', label='wx+b=0')
```

برای رسم $w \cdot x + b$ چون دو بعدی است $w_1 * x_1 + w_2 * x_2 + b$ به این صورت مولفه ها نظیر به نظیر در هم ضرب می شوند برای ما اینجا $w_1x + w_2y + b$ به این صورت می شود معادله خط استاندارد $Ax + By - C = 0$ است که C اینجا برای ما صفر است با دو نقطه روی گراف می توانیم حل کنیم یکی x -intercept و دیگری y -intercept به این صورت:

```
x = -(b - w2y) / w1
if y == 0
x = -(b - w2 * 0) / w1
x = -b / w1
```

y -intercept که به این صورت می شود:

```
y = -(b - w1x) / w2
if x == 0
y = -(b - w1 * 0) / w2
y = -b / w2
```

بعد از اینکه دو نقطه از خط را بدست آوردیم برای محاسبه شیب با توجه به دو نقطه:

```
point_1 = (0, -b / w2)
point_2 = (-b / w1, 0)
```

```
m = (y2 - y1) / (x2 - x1)
m = (0 - -(b / w2)) / (-b / w1 - 0)
m = -(b / w2) / (b / w1)
```

و نهایتا با توجه به شیب و عرض از مبدا معادله خط به این صورت است:

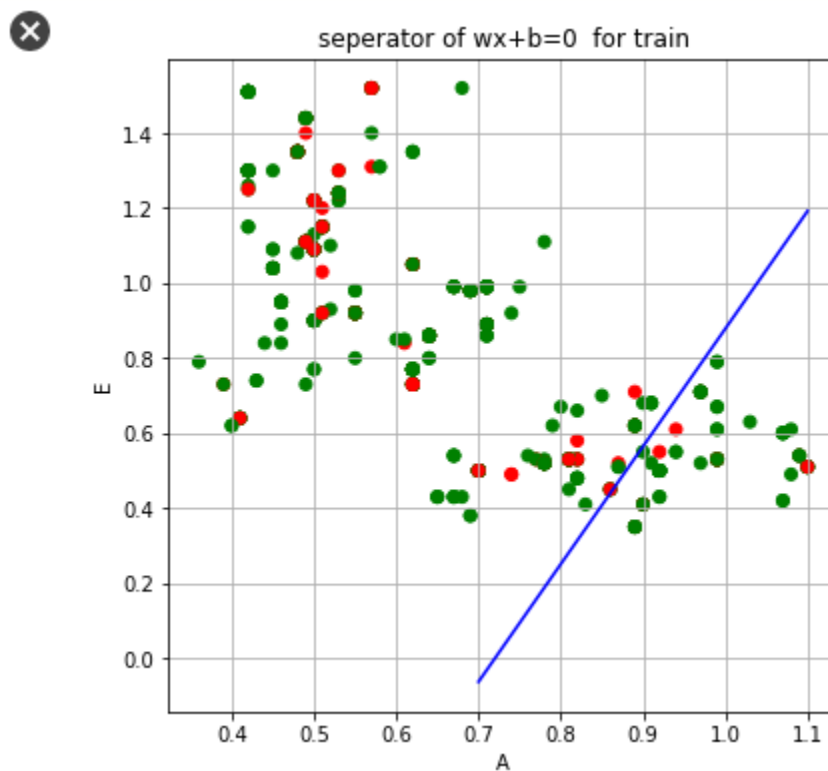
```
slope = -(b / w2) / (b / w1)
y-intercept = -b / w2
y = -(b / w2) / (b / w1))x + (-b / w2)
```

مقادیر را جایگذاری کرده و سپس معادله خط را رسم می کنیم.

```
ax.set_xlabel('A')
ax.set_ylabel('E')
plt.title('seperator of wx+b=0 for train')

ax.grid()
plt.show()
```

نتیجه:



چون داده ها linear separable نیست، نمی توانیم با خط کلاس ها را جدا کنیم و دسته بندی کننده(کلاسیفایر) بیشتر نمونه ها را در یک دسته قرار می دهد.

ج) و برای نمونه های تست هم به همین صورت عمل می کنیم:

```
fig, ax = plt.subplots(figsize=(6,6))
arr1 = np.array(for_test.iloc[:, 4:5]) # A column
arr2 = np.array(for_test.iloc[:, 3:4]) # E column
labl = np.array(labels_test.iloc[:,0:1]) # label train
color= ['red' if l == 1 else 'green' for l in labl]
```

```

ax.scatter(arr1, arr2, color=color)

#y = -(b / w2) / (b / w1))x + (-b / w2)
x = np.linspace(0.7,1.1,25)
w=w.reshape(1,2)
s1=np.array(w[:,0:1]).item() #column 1 of w
s2=np.array(w[:,1:2]).item() #column 2 of w
y = -(b / s2) / (b / s1))*x + (-b / s2)
ax.plot(x, y, '-b', label='wx+b=0')

ax.set_xlabel('A')
ax.set_ylabel('E')
plt.title('seperator of wx+b=0 for test')

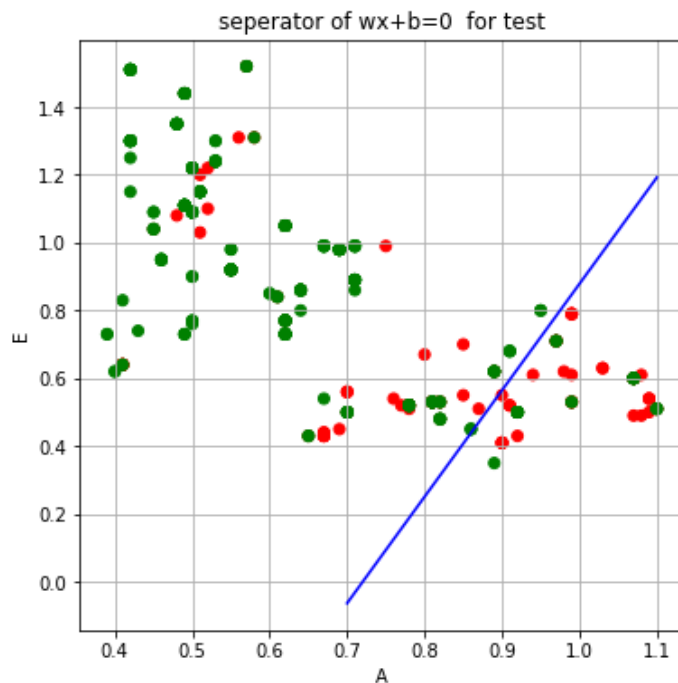
ax.grid()
plt.show()

```

ابتدا یک زیر پلات به اندازه 6x6 ایجاد کرده و سپس ستون های A و E را در مجموعه آموزش به همراه برچسب متناظر آن به متغیر ها نسبت داده سپس برای برچسب ۱ رنگ قرمز و در غیر این صورت رنگ سبز را در نظر گرفتیم و آن را نمایش دادیم.

برای رسم $y = -(b / w2) / (b / w1))x + (-b / w2)$ از $w \cdot x + b$ این معادله استفاده می کنیم و برای x هم از ۰.۷ تا ۱.۱ به میزان ۲۵ نمونه انتخاب کرده و خط y را با آن رسم می کنیم.

نتیجه:



چ) برای ۱۰۰۰ تکرار نمودار همگرایی را رسم می کنیم.

```
import numpy
```

```
for_train=df.sample(frac = 0.6) #60% train
```

```
X = for_train.iloc[:, 3:5] # A,E columns
```

```
train=X.iloc[:, :-1] #train
```

```
for_test = df.drop(train.index) #40% reminder for test
```

```
test_a = for_test.iloc[:, 3:5] # A,E columns test
```

```
test=test_a.iloc[:, :-1] #test
```

```
labels_train = for_train.iloc[:, 1:2] #labels_train
```

```
Y = df.drop(labels_train.index) #drop labels for train
```

```
labels_test = Y.iloc[:, 1:2] #labels_test
```

```
w=[]
```

```
w.append(0)
```

```
w.append(0)
```

```
w = np.array(w)
```

```
b=0
```

```
u = []
```

```

u.append(0)
u.append(0)
u = np.array(u)
B=0
c=1
MaxIter=1000 #epochs
D= len(for_train) #Number of training samples
ce = np.zeros((MaxIter,))
eter = np.zeros((MaxIter,))
for m in range(MaxIter):
    counter=0
    for d in range(D): # D number of samples
        t1=np.array(train.iloc[d:d+1,:]) #sample train row d
        w=w.reshape((2,1))
        q=np.dot(t1,w) #x.w
        z=q+b # (w.x+b)
        y=labels_train.iloc[d:d+1,:]
        Y=np.array(y).item()
        h=Y*z #y*(w.x+b)

        if h <= 0:
            counter=counter+1
            w=w.reshape((1, 2))
            w=w+(Y*t1)
            b=b+Y
            u=u+(Y*c*t1)
            B=B+(Y*c)
            #print("iteration: {} train: {} ,w ={}".format(m,d,w) )
            #print("b ={}".format(b) )
            #print("u ={}".format(u) )
            #print("B ={}".format(B) )
        #end if
        c=c+1
    #end for
    #show iteration error
    #plot in iter
    CE=0
    CE=counter/D #classification error
    ce[m]=np.array(CE).item()
    eter[m]=np.array(m).item()
    #shuffle
    df_shuffle=for_train.sample(frac =1) #shuffle train
    X = df_shuffle.iloc[:, 3:5] # A,E columns train
    train=X.iloc[:, :-1] #train
    labels_train = df_shuffle.iloc[:, 1:2] #labels for train

```

```

#end for
plt.plot(eter, ce, 'g')
w=w.reshape((1, 2))
w=w-(1/c)*u
b=b-(1/c)*B

print("w end ={}".format(w) )
print("b end ={}".format(b) )

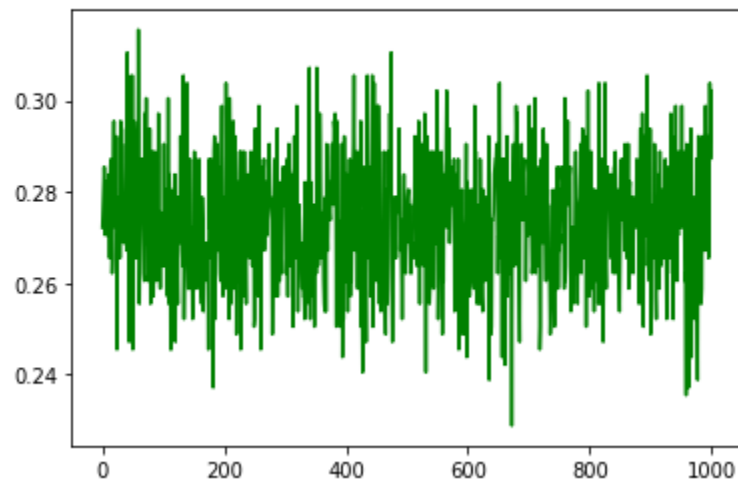
```

فقط کافی است `#epochs=1000` `MaxIter=1000` را قرار دهیم و کد را مثل حالت قبل اجرا کنیم:
نتیجه برای ۱۰۰۰ تکرار:

```

w end =[[ 2.36168147 -1.92020362]]
b end =-1.0258664008908165

```



(چ) ت

```

D= len(train) #Number of training samples

#empirical error

```

```

count=0
for d in range(D):
    t1=np.array(train.iloc[d:d+1,:]) #sample train row d
    w=w.reshape((2,1))
    q=np.dot(t1,w) #w.x
    z=q+b # (w.x+b)
    y=labels_train.iloc[d:d+1,:]
    Y=np.array(y).item() # label y
    j=Y*z
    if j <= 0:
        count=count+1

CE_one=0
CE_one=count/D #empirical error for train data
print("empirical error for train data ={}".format(CE_one) )

D_two= len(test) #Number of training samples
#empirical error
count_two=0
for d in range(D_two):
    t2=np.array(test.iloc[d:d+1,:]) #sample test row d
    w=w.reshape((2,1))
    q=np.dot(t2,w) #w.x
    z=q+b # (w.x+b)
    y=labels_test.iloc[d:d+1,:]
    Y=np.array(y).item() # label y
    j=Y*z
    if j <= 0:
        count_two=count_two+1

CE_two=0
CE_two=count_two/D_two #empirical error for test data
print("empirical error for test data ={}".format(CE_two) )

```

نتیجه خطای ایمپریکال روی نمونه های آموزش و تست:

```

❏ empirical error for train data =0.18864774624373956
   empirical error for test data =0.23809523809523808

```

(چ)ث

```

fig, ax = plt.subplots(figsize=(6,6))
arr1 = np.array(for_train.iloc[:, 4:5]) # A column

```

```

arr2 = np.array(for_train.iloc[:, 3:4]) # E column
labl = np.array(labels_train.iloc[:,0:1]) # label train
color= ['red' if l == 1 else 'green' for l in labl]
ax.scatter(arr1, arr2, color=color)

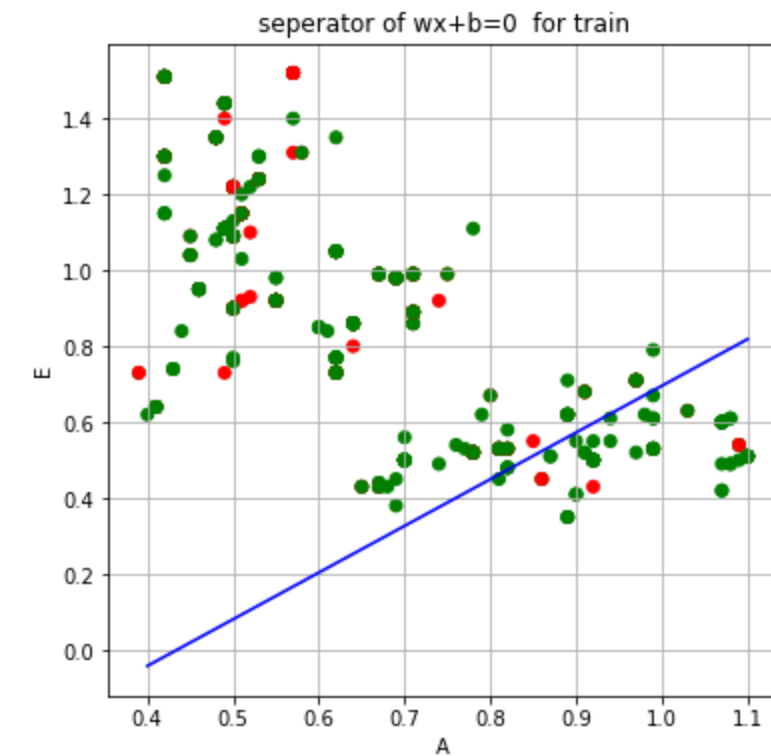
#y = -(b / w2) / (b / w1))x + (-b / w2)
x = np.linspace(0.4,1.1,25)
w=w.reshape(1,2)
s1=np.array(w[:,0:1]).item() #column 1 of w
s2=np.array(w[:,1:2]).item() #column 2 of w
y = -(b / s2) / (b / s1))*x + (-b / s2)
ax.plot(x, y, '-b', label='wx+b=0')

ax.set_xlabel('A')
ax.set_ylabel('E')
plt.title('seperator of wx+b=0 for train')

ax.grid()
plt.show()

```

نتیجه:



(ج)

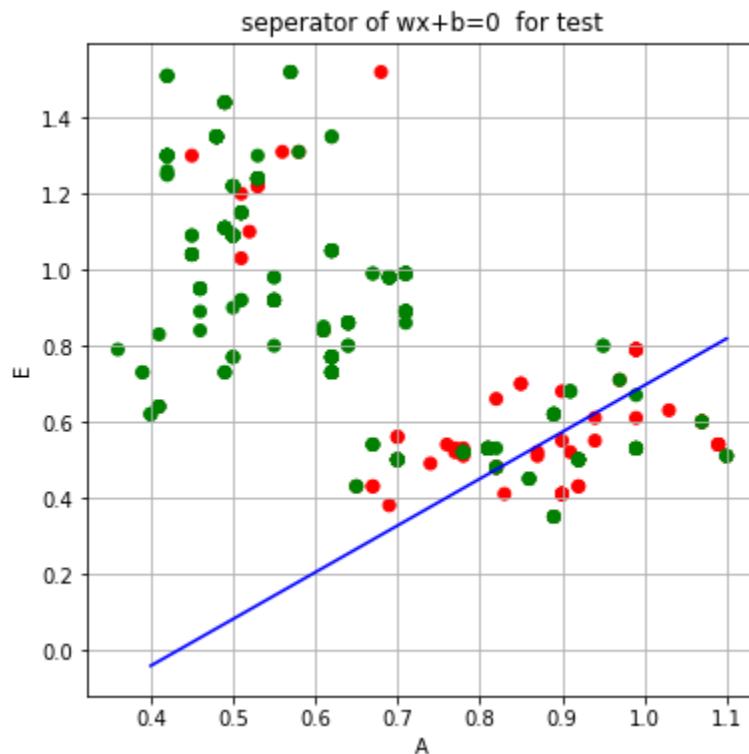
```
fig, ax = plt.subplots(figsize=(6,6))
arr1 = np.array(for_test.iloc[:, 4:5]) # A column
arr2 = np.array(for_test.iloc[:, 3:4]) # E column
labl = np.array(labels_test.iloc[:,0:1]) # label train
color= ['red' if l == 1 else 'green' for l in labl]
ax.scatter(arr1, arr2, color=color)

#y = (-(b / w2) / (b / w1))x + (-b / w2)
x = np.linspace(0.4,1.1,25)
w=w.reshape(1,2)
s1=np.array(w[:,0:1]).item() #column 1 of w
s2=np.array(w[:,1:2]).item() #column 2 of w
y = (-(b / s2) / (b / s1))*x + (-b / s2)
ax.plot(x, y, '-b', label='wx+b=0')

ax.set_xlabel('A')
ax.set_ylabel('E')
plt.title('seperator of wx+b=0 for test')

ax.grid()
plt.show()
```

نتیجہ:



نتیجه گیری بایاس واریانس و بیش برآزش و خطا بیز

$$\text{Error}(f) = \underbrace{[\text{error}(f) - \min_{f' \in \mathcal{H}} \text{error}(f')]}_{\text{Estimation error}} + \underbrace{\min_{f' \in \mathcal{H}} \text{error}(f')}_{\text{Approximation error}}$$

همانطور که می بینیم خطا روی نمونه های آموزشی در ۱۰۰ دور ۰.۱۹ است که با ۱۰۰۰ تکرار به ۰.۱۸ رسیده و ۰.۰۱ خطا روی نمونه های آموزشی کم شده و لی روی نمونه های تست خطا در ۱۰۰ دور ۰.۲۲ بوده که با ۱۰۰۰ تکرار به ۰.۲۳ رسیده ۰.۰۱ بیشتر شده و در واقع افزایش تعداد تکرار همیشه به معنی بهتر شدن نتایج نیست چون ممکن است روی نمونه های آموزشی خطا کمتر شود ولی روی نمونه های تست خطا بیشتر شود اما اینجا اینجا اختلاف کوچکی دارد و نمی توانیم بگوییم بیش برآزش اتفاق افتاده بلکه برعکس

اما آنچه که اینجا مشهود تر است نزدیک بودن خطا آموزشی و تست به هم دیگر در ۱۰۰ دور با توجه به فرمول بالا:

$$\text{Error}(f) = \underbrace{[0.22 - 0.19]}_{\text{Estimation error}} + \underbrace{0.19}_{\text{Approximation error}} = 0.03 + 0.19$$

همانطور که می بینید با توجه به مصالحه بایاس و واریانس و همین طور پیچیدگی داده های آموزشی که توزیع یک نواختی ندارند و linear separable نیستند با خط جدا پذیر نیستند و به توابع پیچیده تری نیاز داریم برای دسته بندی نمونه ها و با یک مدل خطی ساده نمی شود این کار را انجام داد و در واقع ما اینجا بایاس بالا (۰.۱۹) داریم و واریانس پراکندگی داده ها کم (۰.۰۳) است که نشان می دهد underfitting اتفاق افتاده است یعنی مدل توانایی انعکاس رفتار نمونه ها را ندارد مدل پیچیدگی اش کمتر از پیچیدگی نمونه هاست و درخت خالی یا عمق کمی دارد. در واقع inductive bias به درستی انتخاب نشده است و ناشی از عدم غنای مجموعه \mathcal{F} است.

Approximation error بالا و Estimation error پایین است و در مورد خطا بهینه بیز که کم ترین خطا ممکن است و هیچ وقت اپسیلون حد یا بایاس کمتر از این مقدار نمی تواند بشود و در واقع حد پایین Approximation error می باشد و زمانی می تواند اتفاق بیافتد که inductive bias به درستی انتخاب شده باشد که ما اینجا در مسیله با این قضیه رو برو نیستیم چون بایاس بالاست.

و همچنین راجع به بیش برآزش که زمانی رخ می دهد که پیچیدگی مجموعه \mathcal{F} بالاست به هر دیتایی می تواند fit شود و خطا اپسیلون حد یا empirical error روی نمونه های آموزشی خیلی کم می شود

حتی به خطا بهینه بیز نزدیک می شود ولی خوب نیست چون روی نمونه های آموزشی خیلی `fit` می شود اما خطا واقعی ممکن است زیاد باشد. اما اینجا بیش برآزش ندارم چون مجموعه `f` ما پیچیدگی ندارد.

$$\text{Error}(f) = [0.23 - 0.18] + 0.18 = 0.05 + 0.18$$

و همین داستان بالا `under fitting` در ۱۰۰۰ تکرار هم مشاهده می شود.

چون داده ها جداپذیر خطی نیست نمی توانیم با خط کلاس ها را از هم جدا کنیم و کلاسیفایر بیشتر نمونه ها را در یک کلاس قرار می دهد. (تفسیر نمودارهای با خط جدا شونده)

(ح)

```
tedad= len(test) #Number of test samples
#empirical error
tp=0
tppf=0
tpfn=0
f=0
pricision=0
recall=0
for tt in range(tedad):
    ta=np.array(test.iloc[tt:tt+1,:]) #sample test row tt
    w=w.reshape((2,1))
    q=np.dot(ta,w) #w.x
    z=q+b #(w.x+b)
    y=labels_test.iloc[tt:tt+1,:]
    Y=np.array(y).item() # label y
    if (Y == 1 and z > 0) :
        tp = tp+1
        if z > 0: #tp+fp
            tppf = tppf+1
        if Y == 1: #tp+fn
            tpfn = tpfn+1
#end for
pricision = tp/tppf #pricision
recall = tp/tpfn #recall

f = 2*pricision*recall/(pricision+recall) #f score
print(" pricision ={}".format(pricision) )
print("recall ={}".format(recall) )
print("f score pricision recall ={}".format(f) )
```


توضیح کد:

با توجه به مبحث precision recall هر چه f_{score} بیش تر باشد دقت بالاتر است

$$f_{score} = 2 \cdot \text{precision} \cdot \text{recall} / (\text{precision} + \text{recall})$$

$$\text{precision} = \text{tp} / (\text{tp} + \text{fp})$$

نمونه هایی که کلاسیفایر درست pos را تشخیص داده تقسیم بر همه اونایی که pos تشخیص داده است.

$$\text{Recall} = \text{tp} / (\text{tp} + \text{fn})$$

نمونه هایی که کلاسیفایر درست pos را تشخیص داده تقسیم بر همه اونایی که واقعا pos هستند.

ابتدا تعداد نمونه های تست را مشخص کردم سپس برای هر نمونه از تست تعداد نمونه هایی که کلاسیفایر درست pos را تشخیص داده شمردم یعنی حالتی که دسته بندی کننده به نمونه برچسب ۱ می دهد و برچسب واقعی هم یک است و آن را در tp قرار دادم، همه آن هایی که کلاسیفایر بر چسب ۱ داده یعنی بزرگ تر از صفر ها tppf و نهایتا همه آن هایی که pos هستند. آن ها را شمردم و در فرمول بالا قرار داده و نتایج بدست می آید.

نتیجه:

```
prcision =0.4838709677419355
recall =0.3225806451612903
f score prcision recall =0.3870967741935484
```

با توجه به مبحث precision recall هر چه f_{score} بیش تر باشد دقت بالاتر است

همانطور که از نتیجه ۰.۳۸ مشخص است این کلاسیفایر اصلا عملکرد خوبی ندارد چون precision recall مقدار کمی دارد و با توجه به داده هایمان که linear separable نیست نیاز به تابع کلاسیفایر پیچیده تری داریم.