تکلیف دوم/ مرجان مودت

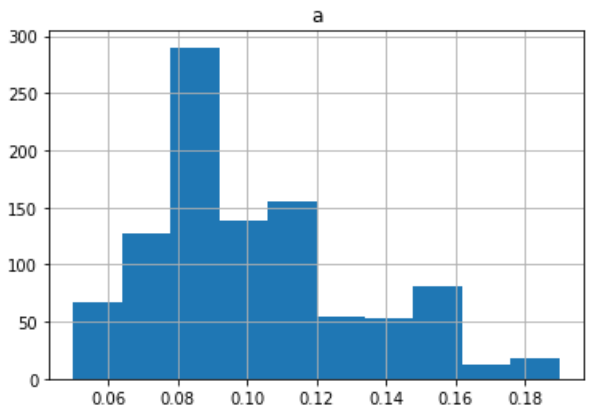
```
import pandas as pd
import numpy as np
from matplotlib import Path
import matplotlib.pyplot as plt

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

ابتدا به کمک دستورهای بالا و کتابخانه های موجود فایل داده ها را خوانده و آن ها 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

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

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



الف)

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

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

$$\min_{w,b} \underbrace{\frac{1}{2}||w||^2}_{\text{large margin}} + \underbrace{C\sum_n \ell^{(\text{hin})}(y_n, w\cdot x_n + b)}_{\text{small slack}} \qquad (7.48)$$

اگر رابطه بالا را در λ/C ضرب کنیم دقیقا میشه حل مسیله گرادیان نزولی با تابع loss hing .

و به راحتی مسیله svm را حل می کنیم و w و b را با روش گرادیان نزولی بدست می آوریم

در حالت کلی مسئله بهینه سازی ما شکل زیر را دارد:

$$\min_{w,b} \sum_n \ell(y_n, w\cdot x_n + b) + \lambda R(w,b)$$

اگر تابع loss hing را در فرمول فوق جایگذاری کنیم و از رگولایزر $\frac{\lambda}{2}||w||^2$ استفاده کنیم به این صورت در میاد:

$$\min_{w,b} \sum_n \ell^{(\text{hin})}(y_n, w\cdot x_n + b) + \frac{\lambda}{2}||w||^2$$

که الگوریتم آن را داریم، کافی است سیگما را در c ضرب کرده و λ را یک بگذاریم.

همچنین چون رابطه ۷-۴۸ ما مینیمم سازی می باشد، باید مشتق تابع مینیمم سازی به این صورت حساب شود:

برای زیتا بزرگتر مساوی صفر $dw = w - c \sum_{n} \max yi\ xi$, $db = - c \sum_{n} yi$

پس باید این ترم ها را در الگوریتم هینج گرادیان نزولی در c مناسبی ضرب شود تا svm حساب شود:

Algorithm 22 HINGEREGULARIZEDGD(**D**, λ, *MaxIter*)

```
w ← ⟨0, 0, ... 0⟩   ,   b ← 0                          // initialize weights and bias
for iter = 1 ... MaxIter do
    g ← ⟨0, 0, ... 0⟩   ,   g ← 0                      // initialize gradient of weights and bias
    for all (x,y) ∈ D do
        if y(w · x + b) < 1 then
            g ← g + y x  c                             // update weight gradient
            g ← g + y   c                              // update bias derivative
        end if
    end for
    g ← g − λw                                         // add in regularization term
    w ← w + ηg                                         // update weights
    b ← b + ηg                                         // update bias
end for
return w, b
```

الف)

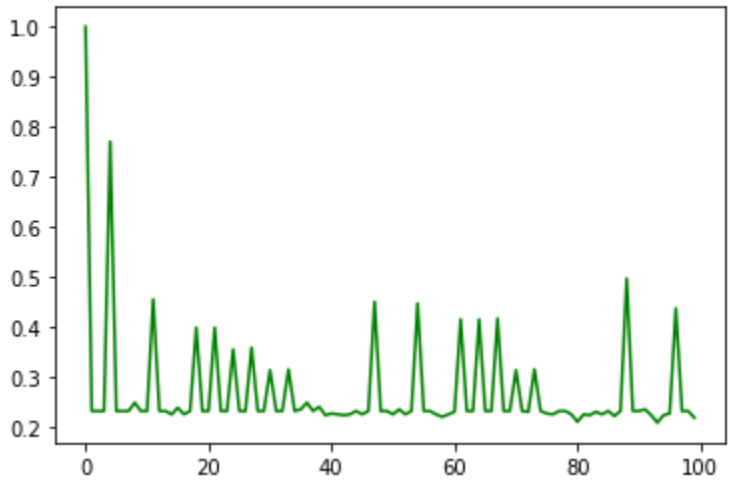
```
import numpy
from sklearn import preprocessing

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

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

landa=1
#eta=0.01
eta=1e-3
c=15
MaxIter=100 #epochs
D= len(for_train) #Number of training samples
ce = np.zeros((MaxIter,))
eter = np.zeros((MaxIter,))
for m in range(MaxIter):
  G =[]
  G.append(0)
  G.append(0)
  G = np.array(G)
  g=0
  counter=0
  for d in range(D): # D number of samples
    t1=np.array(train.iloc[d:d+1,:]) #sample train row d
    w1=np.transpose(w)
    q=np.dot(t1,w1)  #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
    if h < 1:  #y*(w.x+b)<1

      G=G.reshape(1, 2)
      G=G+(Y*t1)*c
      g=g+Y*c
    #end if
  #end for 2

  #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
  G=G-landa*w
```

```
  w=w+eta*G
  b=b+eta*g
  df_shuffle=for_train.sample(frac =1)  #shuffle train
  train = df_shuffle.iloc[:, 3:5]  # A,E columns train #train
  labels_train = df_shuffle.iloc[:, 1:2]    #labels for train
#end for 1
plt.plot(eter, ce, 'g')
w=w.reshape((1, 2))

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

نتیجه:

```
w end =[[-2.91769031  4.82774142]]
b end =-1.8450000000000077
```



نمودار همگرایی همانطور که در شکل میبینیم همگرا نشده و همچنان دارای نوساناتی است چون به هر حال داده ها جداپذیر خطی نیستند اما نواسانات آن در مقایسه با نمودار همگرایی پرسپترون با همین داده ها به مراتب کمتر است و این نشان دهنده کارایی بالاتر الگوریتم svm نسبت به پرسپترون است جایی که همونه ها جداپذیرخطی نیستند.

شرح کد:

ابتدا کتابخانه های لازم را ایمپورت می کنیم، از from sklearn import preprocessing برای گرفتن نرخ یادگیری استفاده کردم.

```
import numpy
from sklearn import preprocessing
```

۶۰ درصد از داده ها را به عنوان داده های آموزشی انتخاب کردم

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

ستون های موردنظر را جدا نمودم

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

۴۰ درصد باقی مانده را برای تست باقی گذاشتم.

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

ستون های بردار ویژگی برای نمونه های تست را جدا می کنم

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

برچسب های نمونه های آموزشی را در متغیر موردنظر قرار دادم

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

برچسب های نمونه های آموزشی را از کل نمونه ها حذف کرده

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

و به این ترتیب برچسب نمونه های تست بدست می آید

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

بردار وزن در ابتدا با مقدار ۰ مقداردهی اولیه شده و به اندازه ستون های بردار ویژگی است

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

بایاس را نیز ۰ می گذارم

```
b=0
```

مقدار لاندا پارامتر رگولاریزیشن را ۱ قرار دادم تا هینج برای من svm را حساب کند.

```
landa=1
```

مقدار نرخ یادگیری را با کمک کتابخانه ای که پیش تر راجع به آن گفتم و به صورت تجربی مقدار آن را قرار دادم.

```
eta=1e-3
```

مقدار c را ۱۵ گذاشتم که این پارامتر میزان سخت گیری را برای slack در نظر می گیرد هرچه کمتر باشد به soft margin نزدیک تر است و شل می گیرد.

```
c=15
```

حداکثر تعداد تکرار

```
MaxIter=100 #epochs
```

تعداد نمون های آموزشی

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

آرایه ای که در آن مقدار خطا دسته بندی بر حسب خطا در عمل در هر مرحله از تکرار الگوریتم را در خود ذخیره می کند.

```
ce = np.zeros((MaxIter,))
```

آرایه زیر نیز برای ذخیره عدد هر مرحله از تکرار است این آرایه و آرایه قبلی با مقدار ۰ مقداردهی اولیه شدند و به تعداد مراحل اجرای الگوریتم سطر دارند.

```
eter = np.zeros((MaxIter,))
```

حلقه اجرا الگوریتم به اندازه حداکثر تعداد تکرار

```
for m in range(MaxIter):
```

آرایه ذخیره گرادیان در ابتدای هر ایتریشن با مقدار ۰ مقداردهی اولیه شده و به تعداد ستون های بردار ویژگی سطر دارد.

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

مقدار مشتق بر حسب بایاس در g ذخیره می شود.

```
  g=0
```

تعداد خطا در هر تکرار را ذخیره می کند.

```
  counter=0
```

برای تمام نمونه ها(در اینجا نمونه های آموزشی) این حلقه را اجرا کن

```
  for d in range(D): # D number of samples
```

در هر مرحله سطر شماره آن مرحله انتخاب می شود(به این ترتیب از مرحله اول تا آخر سطر اول تا آخر انتخاب می شود) برای همه سطر ها، سطر در بردار وزن ضرب می شود با بایاس جمع می شود، برچسب متناظر با سطر متناظر با بردار ویژگی جدا شده و در هم ضرب می شوند اگر حاصل کوچکتر مساوی ۰ بود یعنی برچسب و تخمین با هم یکی نیستند و باید به شمارنده خطا یک واحد اضافه شود.

```
    t1=np.array(train.iloc[d:d+1,:]) #sample train row d
    w1=np.transpose(w)
    q=np.dot(t1,w1)  #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
```

طبق الگوریتم svm در حالتی که y*(w.x+b) کوچکتر از ۱ بود آپدیت داریم.
همانطور که در فرمول زیتا برای svm داشتیم و چون برابر با تابعloss hing بود سراغ این الگوریتم آمدیم:

$$\xi_n = \begin{cases} 0 & \text{if } y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) \geq 1 \\ 1 - y_n(\boldsymbol{w} \cdot \boldsymbol{x}_n + b) & \text{otherwise} \end{cases} \qquad (7.47)$$

```
if h < 1:  #y*(w.x+b)<1
```

حالا گرادیان و مشتق را برای svm همانطور که پیش تر توضیح دادم به این صورت بازنویسی می کنیم در واقع پارامتر c را در ترم قبلی کافی ست ضرب نموده تا w و b در مرحله بعد مطابق svm آپدیت و محاسبه شود:

```
    G=G.reshape(1, 2)
    G=G+(Y*t1)*c
    g=g+Y*c
  #end if
#end for 2
```

حالا برای آپدیت w وb ابتدا گرادیان را با توجه به مشتق تابع رگولایزر آپدیت کرده و سپس w وb را مانند تمام آپدیت هایی که با گرادیان و مشتق همراه است به این صورت آپدیت می کنیم که وزن جدید می شود وزن قبلی به اضافه نرخ یادگیری ضرب در گرادیان می شود برای بایاس و مشتق آن نیز آپدیت به همین صورت انجام می شود.

```
#update w, b
G=G-landa*w
w=w+eta*G
b=b+eta*g
```

در پایان هر ایتریشن تعداد خطا ها را بر تعداد کل نمونه ها(اینجا نمونه های آموزشی)تقسیم کرده و در آرایه مربوطه ذخیره می کنیم، همچنین عدد مرحله را نیز ذخیره می کنیم.

```
#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()
```

بُزدن تصادفی نمونه ها در این مرحله انجام می شود

```
#shuffle
df_shuffle=for_train.sample(frac =1)  #shuffle train
train = df_shuffle.iloc[:, 3:5]  # A,E columns train #train
labels_train = df_shuffle.iloc[:, 1:2]    #labels for train
#end for 1
```

رسم نمودار همگرایی با رنگ سبز

```
plt.plot(eter, ce, 'g')
```

وزن را به بردار 1x2 قرار بده

```
w=w.reshape((1, 2))
```

وزن و بایاس را چاپ کن.

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

ب)

```
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:

```
empirical error for train data =0.22203672787979967
empirical error for test data =0.24060150375939848
```

همانطور که مشاهده می شود خطا روی نمونه های آموزشی تقریباً ۰.۲۲ و روی نمونه های تست تقریباً ۰.۲۴ می باشد، یعنی روی نمونه های آموزشی خطای کمتری دارد نسبت به تست، اما تفاوت اندکی دارد

شرح کد:

تعداد داده های آموزشی را مشخص کرده و به ازای هر نمونه آمو زشی 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 $y*(wx+b) <= 0$ یعنی برچسب را اشتباه حساب کرده بدست آورده و بر تعداد کل نمونه های آموزشی تقسیم می کنیم.

```
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) )
```

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

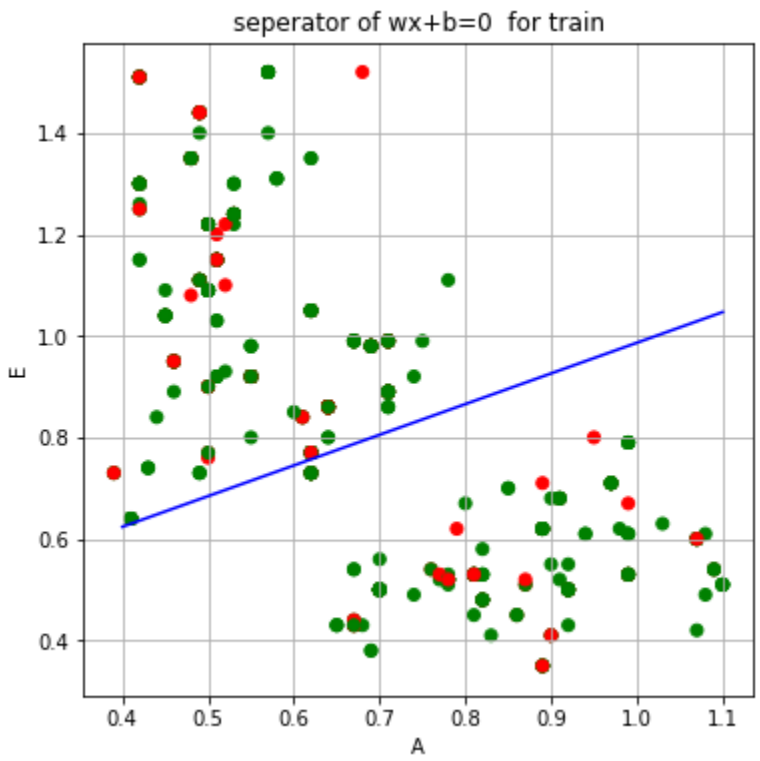
```
D_two= len(test) #Number of training samples
#empirical error
count_two=0
for d in range(D_two):
    t2=np.array(test.iloc[d:d+1,:]) #sample test row d
    w=w.reshape((2,1))
    q=np.dot(t2,w)  #w.x
    z=q+b  #(w.x+b)
    y=labels_test.iloc[d:d+1,:]
    Y=np.array(y).item() # label y
    j=Y*z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

پ)

```
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:



همانطور که در شکل کاملا مشهود است دسته بندی کلاسیفایر svm در مقایسه با دسته بندی کننده معادل خود در پرسپترون که نمودار آن را در پروژه پیشین دیدیم که اکثر نمونه ها را در یک دسته قرار می داد به خاطر جداپذیر خطی نبودن داده ها . اما اینجا الگوریتم svm این تفکیک را برای نمونه ها با وجود اینکه خطی جداپذیر نیستند با دقت بسیار بهتری انجام داده است و این نشان دهنده قدرت دسته بندی کننده svm در مقایسه با پرسپترون جایی که داده ها خطی جداپذیر نیستند، می باشد.

شرح کد:

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 `w · x + b` چون دو بعدی است `w1 * x1 + w2 * x2 + b` به این صورت مولفه ها نظیر به نظیر در هم ضرب می شوند برای ما اینجا `w1x + w2y + 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-intercept`:

```
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

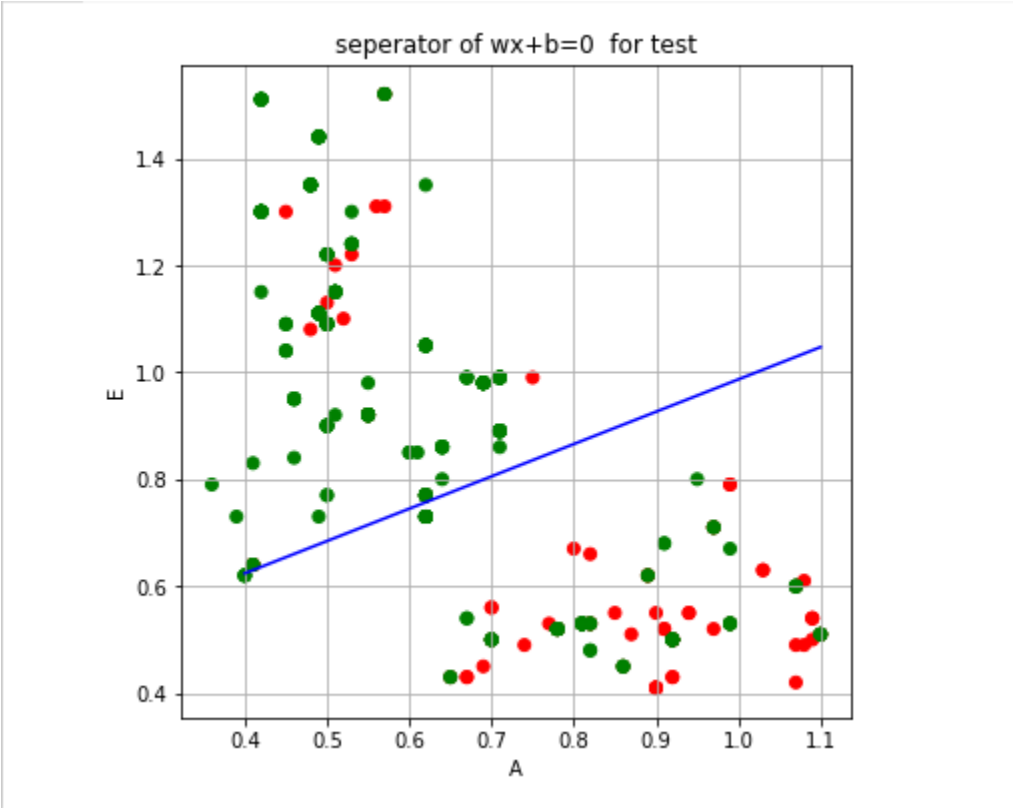
ت)

```
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()
```

شرح کد در قسمت پ آمده است.

نتیجه:



همانطور که در شکل دیده می شود برای نمونه های تست نیز دسته بندی خوبی داشته تقریبا با توجه به جداپذیر خطی نبودن نمونه ها.

ث) برای ۱۰۰۰ تکرار با بعد بردار ویژگی ۲

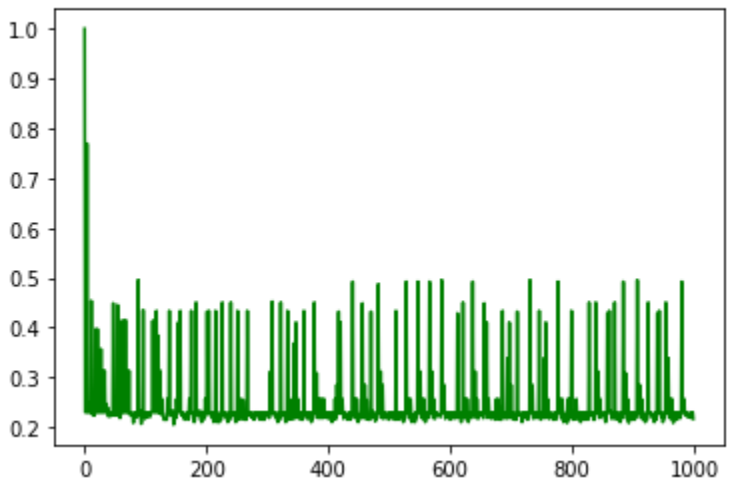
```
import numpy
from sklearn import preprocessing

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

landa=1
#eta=0.01
eta=1e-3
c=15
MaxIter=1000 #epochs
D= len(for_train) #Number of training samples
ce = np.zeros((MaxIter,))
eter = np.zeros((MaxIter,))
for m in range(MaxIter):
  G =[]
  G.append(0)
  G.append(0)
  G = np.array(G)
  g=0
  counter=0
  for d in range(D): # D number of samples
    t1=np.array(train.iloc[d:d+1,:]) #sample train row d
    w1=np.transpose(w)
    q=np.dot(t1,w1)  #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
    if h <= 1:
      #counter=counter+1
      G=G.reshape((1, 2))
      G=G+(Y*t1)*c
      g=g+Y*c
    #end if
  #end for 2
```

```
  #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
  G=G-landa*w
  w=w+eta*G
  b=b+eta*g
  df_shuffle=for_train.sample(frac =1)  #shuffle train
  train = df_shuffle.iloc[:, 3:5]  # A,E columns train #train
  labels_train = df_shuffle.iloc[:, 1:2]    #labels for train
#end for 1
plt.plot(eter, ce, 'g')
w=w.reshape((1, 2))

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

شرح کد در قسمت الف آمده است.

نتیجه:

```
w end =[[-1.17219357  5.58896485]]
b end =-3.000000000000035
```



خطا در عمل روی نمونه های آموزشی و تست برای ۱۰۰۰ تکرار با بعد بردار ویژگی ۲

```
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) )
```

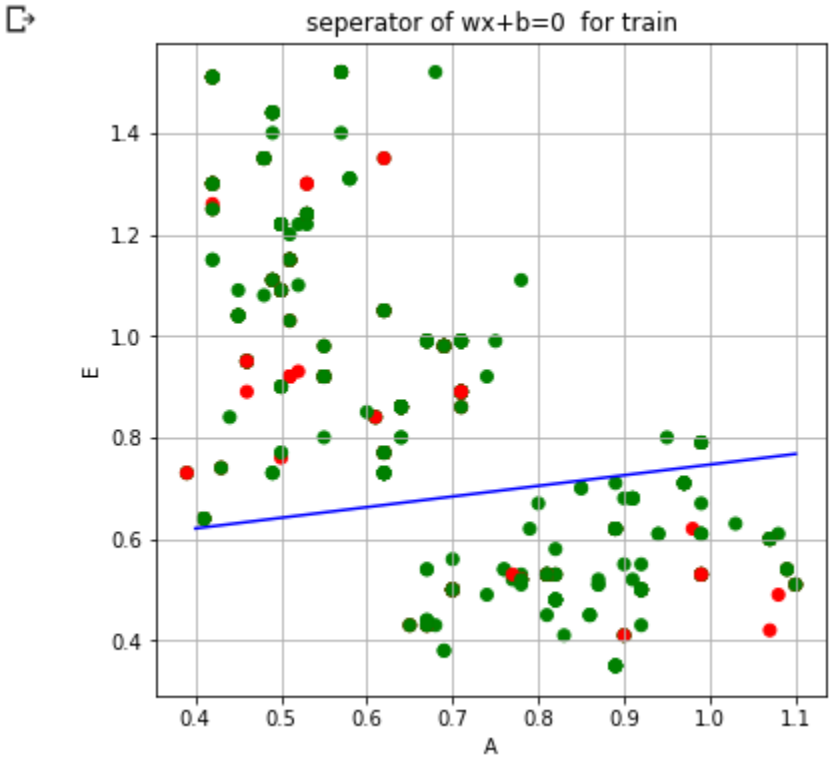
نتیجه:

```
empirical error for train data =0.22537562604340566
empirical error for test data =0.23809523809523808
```

منحنی اسکتر پلات به همراه خط جداکننده روی نمونه های آموزشی برای ۱۰۰۰ تکرار با بعد بردار ویژگی ۲

```
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:

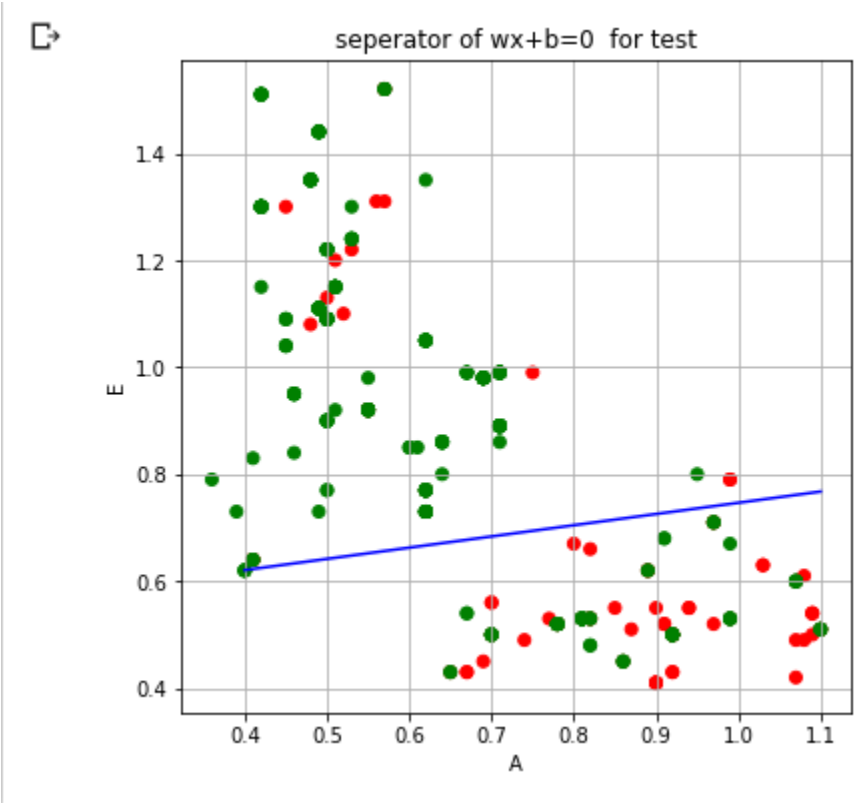


منحنی اسکترپلات به همراه خط جداکننده روی نمونه ها تست برای ۱۰۰۰ تکرار با بعد بردار ویژگی ۲

```
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()
```

نتیجه:



مصالحه بایاس و و اریانس، بیش برازش و خطا بیز

برای ۱۰۰ تکرار: اپسیلون-اپسیلون حد: ناشی از کمبود نمونه ها: ۰.۲۴۰-۰.۲۲۲=۰.۰۱۸ واریانس

و اپسیلون حد:ناشی از عدم غنای مجموعه f(که همان تابع خطی ما باشد در اینجا): ۰.۲۲۲ بایاس

برای ۱۰۰۰ تکرار: اپسیلون-اپسیلون حد: ناشی از کمبود نمونه ها: ۰.۲۲۵-۰.۲۳۸=۰.۰۱۳ واریانس

و اپسیلون حد:ناشی از عدم غنای مجموعه f(که همان تابع خطی ما باشد در اینجا): ۰.۲۲۵ بایاس

برای ۱۰۰۰ تکرار خطا ناشی از عدم غنا مجموعه توابع بیشتر شده یعنی مجموعه f توابع متنوعی در خودش ندارد خطی هستند و انعطاف پذیری کمی دارند چون داده ها جداپذیر خطی نیستند این نیاز وجود داردکه پیچیدگی توابع بیشتر باشد. باید یک تریدافی بین بایاس و واریانس برقرار شود.

اما در کل اینجا نه بیش برازش اتفاق نیافتاده چون اون برای حالتی است که بایاس کم و واریانس زیاد باشد ولی اینجا بایاس کمی بیش تر شده و واریانس کمتر شده است. وقتی بایاس زیاد باشد و واریانس کم باشد می گوییم آندرفیت اتفاق افتاده است.

خطا بهینه بیز برای حالتی است که خطا روی نمونه های آموزشی کمتر از آن نمی شود و این بهترین حالت ممکن خطا روی نمونه های آموزشی است ولی در اینجا برای ۱۰۰۰ تکرار این خطا روی نمونه های آموزشی بیش تر شده است و ما این حالت را نداریم.

ج) f score pricision recall برای ۱۰۰ تکرار

```
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) )
```

نتیجه:

```
 pricision =0.46153846153846156
recall =0.6976744186046512
f score pricision recall =0.5555555555555556
```

Fscore هرچه به یک نزدیک تر باشد دقت بهتر است و دقتی که برای پرسپترون محاسبه کردیم ۰.۳۸ بود اینجا دقت حدود ۰.۵۵ درصد است که نشان می دهد دقت svm بیش تر از پرسپترون است اما دقت ۰.۵ در حد یک دسته بندی کننده تصادفی است و این مطلوب ما نیست و دلیلش این است داده ما جداپذیر خطی نیست و در بهترین حالت ممکن یک خط نمی تواند داده هایی که با خط جداپذیر نیستند را دسته بندی کند ولی اگر قرار باشد یک خط این کار را برای ما انجام دهد svm بهترین نتیجه را به ما می دهد.

توضیح کد:

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

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

$$precision = tp/(tp+fp)$$

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

$$Recall = tp/(tp+fn)$$

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 تعدادنمونه هایی که کلاسیفایر درست pos را تشخیص داده شمردم یعنی حالاتی 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 بزگ تر از صفر ها tppf و نهایتا همه آن هایی که pos هستند. آن ها را شمرده و در فرمول بالا قرار داده و نتایج بدست می آید.

f score pricision recall برای ۱۰۰۰ تکرار

```
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) )
```

نتیجه:

```
 pricision =0.8181818181818182
recall =0.0967741935483871
f score pricision recall =0.17307692307692307
```

Fscore هرچه به یک نزدیک تر باشد دقت بهتر است اما برای ۱۰۰۰ تکرار میبینیم که دقت خیلی بدی روی نمونه های تست دارد چون هر دو precision وrecall باید مقدار بالایی داشته باشند.

ح) نمودار همگرایی برای ۱۰۰ تکرار با بعد بردار ویژگی ۴

```
import numpy
from sklearn import preprocessing

for_train_before = df.drop(df.columns[[5,6]], axis=1)
for_train=for_train_before.sample(frac = 0.6) #60% train
```

```
for_train['index'] = for_train.index
train = for_train.iloc[:, 3:7]  # E,A,e,a columns #train

for_test = df.drop(train.index)  #40% reminder for test
test = for_test.iloc[:, 3:7]   # E,A,e,a columns test #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.append(0)
w.append(0)
w = np.array(w)
b=0

landa=1
#eta=0.01
eta=1e-3
c=15
MaxIter=100 #epochs
D= len(for_train) #Number of training samples
ce = np.zeros((MaxIter,))
eter = np.zeros((MaxIter,))
for m in range(MaxIter):
  G =[]
  G.append(0)
  G.append(0)
  G.append(0)
  G.append(0)
  G = np.array(G)
  g=0
  counter=0
  for d in range(D): # D number of samples
    t1=np.array(train.iloc[d:d+1,:]) #sample train row d
    w1=np.transpose(w)
    q=np.dot(t1,w1)  #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
    if h < 1:
      counter=counter+1
      G=G.reshape((1, 4))
      G=G+(Y*t1)*c
      g=g+Y*c
    #end if
  #end for 2

  #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()

  G=G-landa*w
  w=w+eta*G
  b=b+eta*g

  #shuffle
  df_shuffle=for_train.sample(frac =1)  #shuffle train
  train = df_shuffle.iloc[:, 3:7]  # A,E columns train #train
  labels_train = df_shuffle.iloc[:, 1:2]    #labels for train
#end for 1
plt.plot(eter, ce, 'g')
w=w.reshape((1, 4))

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

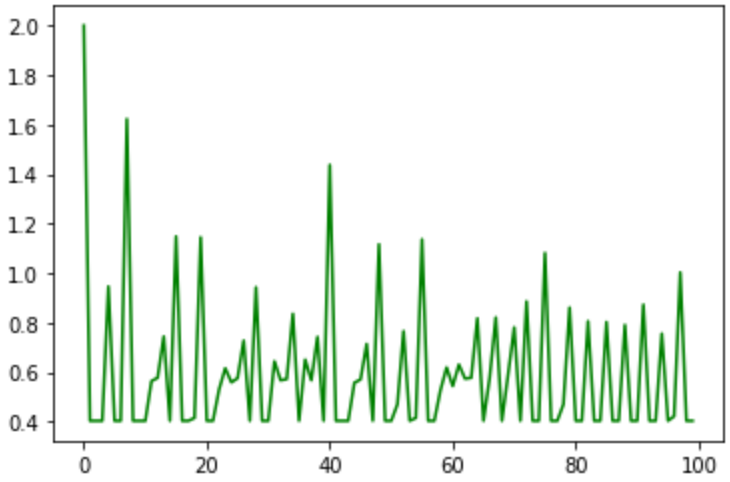
شرح کد در قسمت الف داده شده است فقط برای گرادیان و بردار وزن با توجه به اینکه بردار ویژگی را۴ تایی در نظر می گیریم این ها را هم باید بردار ۴ تایی در نظر بگیریم.

نتیجه:

نمودار همگرایی برای ۱۰۰ تکرار دارای نوسانات بیشتری می باشد در مقایسه با زمانی که بعد بردار ویژگی ۲ بود

این نمودار هم همگرا نشده است.

```
w end =[[-2.94536465  4.49673464 -1.02393447 -1.85964574]]
b end =-0.8400000000000052
```



خطا در عمل برای نمونه های آموزش و تست برای ۱۰۰ تکرار با بعد بردار ویژگی ۴

```
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((4,1))
    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((4,1))
    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 #y*(w.x+b)
    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:

```
empirical error for train data =0.2337228714524207
empirical error for test data =0.22305764411027568
```

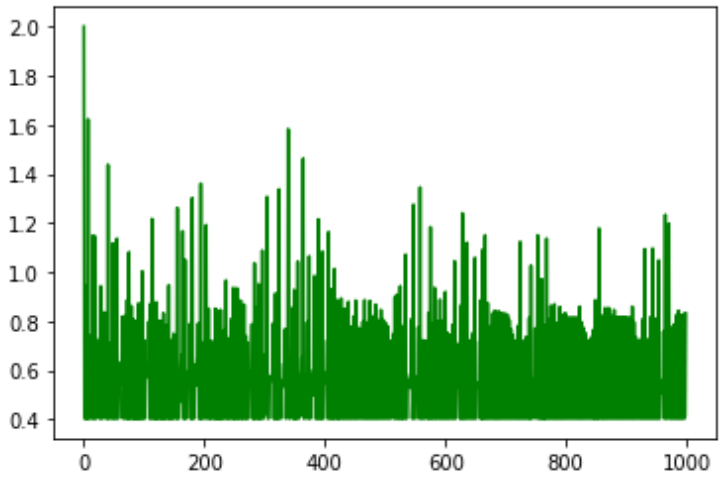
نمودار همگرایی برای ۱۰۰۰ تکرار با بعد بردار ویژگی ۴

```
import numpy
from sklearn import preprocessing

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

landa=1
#eta=0.01
eta=1e-3
c=15
MaxIter=1000 #epochs
D= len(for_train) #Number of training samples
ce = np.zeros((MaxIter,))
eter = np.zeros((MaxIter,))
for m in range(MaxIter):
  G =[]
  G.append(0)
  G.append(0)
  G.append(0)
  G.append(0)
  G = np.array(G)
```

```
    g=0
    counter=0
    for d in range(D): # D number of samples
      t1=np.array(train.iloc[d:d+1,:]) #sample train row d
      w1=np.transpose(w)
      q=np.dot(t1,w1)  #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
      if h < 1:
        counter=counter+1
        G=G.reshape((1, 4))
        G=G+(Y*t1)*c
        g=g+Y*c
      #end if
    #end for 2

    #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()

    G=G-landa*w
    w=w+eta*G
    b=b+eta*g

    #shuffle
    df_shuffle=for_train.sample(frac =1)  #shuffle train
    train = df_shuffle.iloc[:, 3:7]  # A,E columns train #train
    labels_train = df_shuffle.iloc[:, 1:2]    #labels for train
#end for 1
plt.plot(eter, ce, 'g')
w=w.reshape((1, 4))

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

نتیجه:

```
w end =[[ -4.2020349    5.24047834  -4.28367382 -14.04989263]]
b end =-5.520000000000039
```



برای ۱۰۰۰ تکرار نوسانات هم بیشتر شده

خطا در عمل برای نمونه های آموزش و تست برای ۱۰۰۰ تکرار با بعد بردار ویژگی ۴

```
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((4,1))
    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((4,1))
    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:

```
empirical error for train data =0.2020033388981636
empirical error for test data =0.2581453634085213
```

مصالحه بایاس و و اریانس، بیش برازش و خطا بیز

برای ۱۰۰ تکرار: اپسیلون-اپسیلون حد: ناشی از کمبود نمونه ها: ۰.۲۰۲-۰.۲۵۸=۰.۰۵۶ واریانس

و اپسیلون حد:ناشی از عدم غنای مجموعه f(که همان تابع خطی ما باشد در اینجا): ۰.۲۰۲ بایاس

برای ۱۰۰۰ تکرار: اپسیلون-اپسیلون حد: ناشی از کمبود نمونه ها: ۰.۲۳۴-۰.۲۲۳=-۰.۰۱۱ واریانس

و اپسیلون حد:ناشی از عدم غنای مجموعه f(که همان تابع خطی ما باشد در اینجا): ۰.۲۳۴ بایاس

برای ۱۰۰۰ تکرار خطا ناشی از عدم غنا مجموعه توابع بیشتر شده یعنی مجموعه f توابع متنوعی در خودش ندارد خطی هستند و انعطاف پذیری کمی دارند چون داده ها جداپذیر خطی نیستند این نیاز وجود داردکه پیچیدگی توابع بیشتر باشد. باید یک تریدافی بین بایاس و واریانس برقرار شود.

اما در کل اینجا نه بیش برازش اتفاق نیافتاده چون اون برای حالتی است که بایاس کم و واریانس زیاد باشد ولی اینجا بایاس زیاد شده و واریانس کم شده و آندرفیت اتفاق افتاده است.

خطا بهینه بیز برای حالتی است که خطا روی نمونه های آموزشی کمتر از آن نمی شود و این بهترین حالت ممکن خطا روی نمونه های آموزشی است ولی در اینجا برای ۱۰۰۰ تکرار این خطا روی نمونه های آموزشی بیش تر شده است و ما این حالت را نداریم.