دانشگاه تهران

دانشکده علوم مهندسی

الگوریتم‌ها و محاسبات

تکتم سمیعی

۸۱۰۸۹۶۰۵۴

یادگیری ماشین – دکتر سایه میرزایی

تمرین چهارم

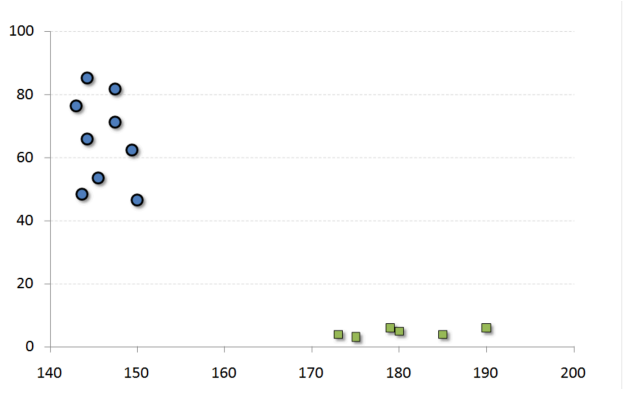
بهار ۰۰

سوال اول

Support vector machine

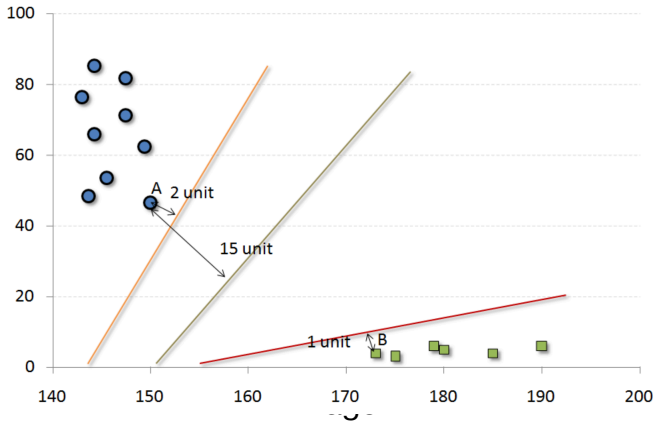
Support vector machine روشی است در عین محاسبات کم ، دقت بالایی دارد . بردارهای پشتیبان ، مجموعه ای از نقاط در فضای n بعدی داده ها هستند که مرز دسته ها را مشخص می کنند و مرزبندی و دسته بندی داده ها براساس آنها انجام می شود و با جابجایی یکی از آنها، خروجی دسته بندی ممکن است تغییر کند . در فضای دوبعدی ،بردارهای پشتیبان، یک خط، در فضای سه بعدی یک صفحه و در فضای n بعدی یک ابر صفحه را شکل خواهند داد. SVM یا ماشین بردار پشتیبان ، یک دسته بند یا مرزی است که با معیار قرار دادن بردارهای پشتیبان ، بهترین دسته بندی و تفکیک بین داده ها را برای ما مشخص می کند. به این مرز جدا کننده hyperplane گفته میشود .

به شکل زیر دقت کنید :



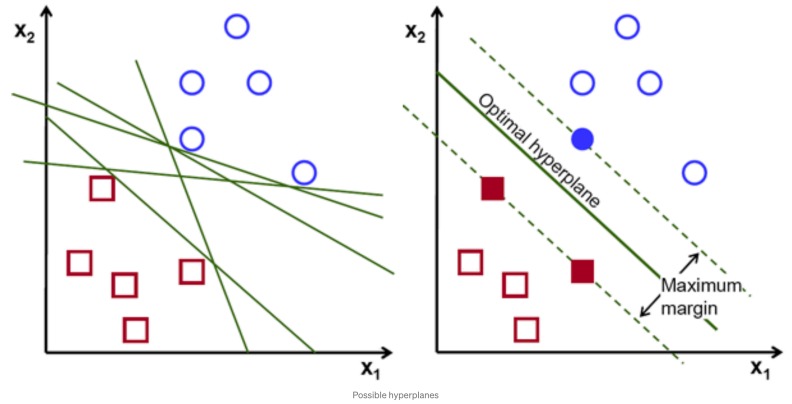
برای سادگی حالت دو بعدی را بررسی میکنیم . محور عمودی ویژگی X1 و ومحور افقی ویژگی X2 را نمایش میدهد .

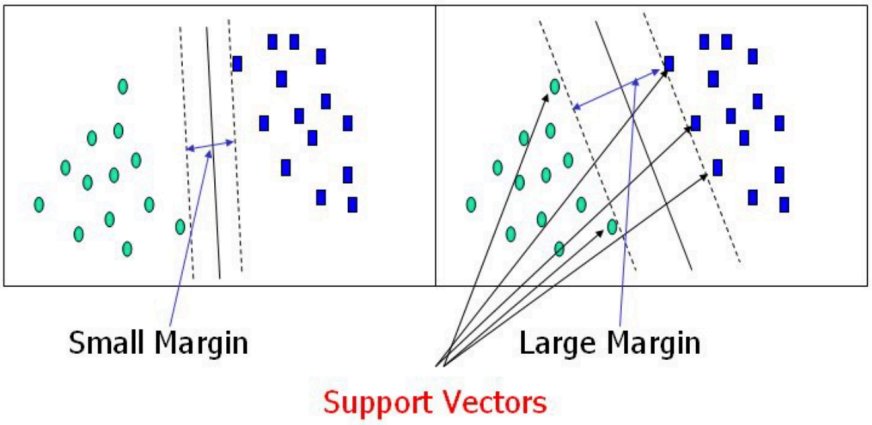
به ازای داده های موجود ، تعداد زیادی مرزبندی می توانیم داشته باشیم که سه تا از این مرزبندی ها در زیر نمایش داده شده است :



یک راه ساده برای انجام اینکار و ساخت یک دسته بند بهینه ، محاسبه فاصله ی مرزهای به دست آمده با بردارهای پشتیبان هر دسته (مرزی ترین نقاط هر دسته یا کلاس) و در نهایت انتخاب مرزیست که از دسته های موجود، مجموعاً بیشترین فاصله را داشته باشد که در شکل فوق خط میانی ، تقریب خوبی از این مرز است که از هر دو دسته فاصله ی زیادی دارد.

در الگوریتم SVM ما به دنبال ماکزیمم کردن مرز بین نقاط داده و hyperplane هستیم .





ریاضیات :

میدانیم فاصله ی هرنقطه ی (x_0, y_0) ، از خطی به معادله ی $ay + bx + c = 0$ برابر است با:

$$d = \frac{|a x_0 + b y_0 + c|}{\sqrt{a^2 + b^2}}$$

اگر x_0 و y_0 ره به صورت X_1 و X_0 و همچنین a و b را به صورت θ_0 و θ_1 باز نویسی کنیم ، رابطه ی بالارا به صورت برداری میتوان به صورت زیر نوشت :

$$d = \frac{|\theta_T X + \theta_0|}{||\theta||}$$

معادله ی

برای ما فاصله ی نقاط نزدیک به خط hyperplane مهم است و در انتخاب و جابه جایی خط اهمیت دارند .

این فاصله باید از مقدار پارامتر M که margin را تعیین میکند ، بیشتر باشد :

$$\frac{|\theta_T X + \theta_0|}{||\theta||} > M$$

در اینجا برای سادگی فرض میکنیم : $y \in \{-1,1\}$

هدف یافتن بردار θ و مقدار θ_0 به گونه ای که :

$$\theta_T X + \theta_0 \geq +1 \qquad for\, y^{(i)} = +1$$

$$\theta_T X + \theta_0 \leq -1 \qquad for\, y^{(i)} = -1$$

در واقع میخواهیم اندازه ی فاصله ی داده ها از hyperplane بیشتر از ۱ باشد. با این کار حاشیه ای مشخص میکنیم برای اطمینان از این که دادگان در یک حاشیه ی امنی قرار دارند .

حال عبارت زیر را تعریف میکنیم . میدانیم این عبارت همواره بزرگ تر از صفر است :

$$y^{(i)}(\theta_T X + \theta_0) \geq +1$$

هدف بیشینه کردن فاصله ی بردار های پشتیبان از از مرز تصمیم گیری است :

$$\frac{|\theta_T X + \theta_0|}{||\theta||} \geq M \quad \rightarrow \quad |\theta_T X + \theta_0| \geq M||\theta||$$

این معادله بینهایت جواب دارد اما با قرار دادن $M||\theta|| = 1$ خواهیم داشت : $M = \frac{1}{||\theta||}$

پس برای ماکزیمم کردن M باید اندازه ی θ را مینیمایز کنیم . پس مسئله تبدیل شد به یک مسئله ی بهینه سازی با شرط زیر :

$$min \frac{1}{2}||\theta||^2$$

$$s.t \qquad y^{(i)}(\theta^T X^{(i)} + \theta_0)) \geq +1$$

معادله ی فوق را باید تمام داده ها حل کرد . همانطور که میبینید تابع هدف $min\frac{1}{2}||\theta||^2$ یک تابع درجه ۲ است که تابعی محدب است و دارای میبینمم سراسری است و مینیمم های محلی ندارد و مطلوب ماست و قطعا در هر شرایط اولیه ای به جواب همگرا میشویم .

در نهایت از با استفاده از ضرایب لاگرانژ محدودیت هارا از تابع هدف کم میکنیم :

$$L_p = min\frac{1}{2}||\theta||^2 - \Sigma\alpha^i[y^{(i)}(\theta^T X^{(i)} + \theta_0)) - 1]$$

سپس با محاسبه ی گرادیان این تابع هزینه ، مقدار هزینه در هر مرحله را به دست می آوریم :

$$cost = \frac{\Sigma theta}{N} \qquad if max(0,1 - y_i * (theta \,.\, x_i) = 0$$

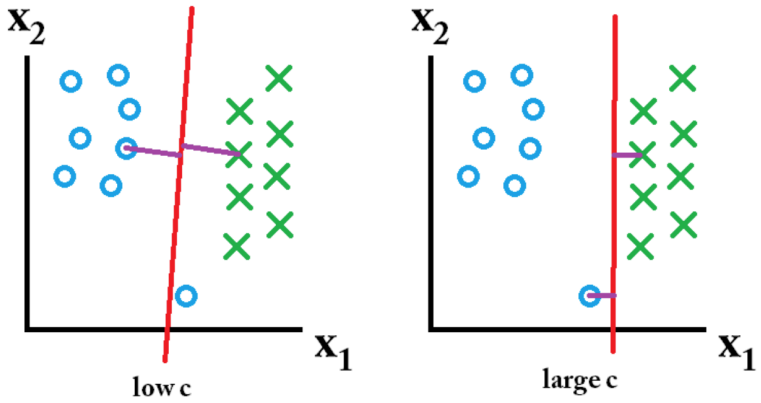
$$cpst = \frac{\Sigma theta - C y_i x_i}{N} \quad if max(0,1 - y_i * (theta \,.\, x_i) \neq 0$$

عبارت $1 - y_i * (theta \,.\, x_i)$ به تین معناست که داده هایی که در margin قرار دارند را ، فاصله شان را تا خط محاسبه میکند و به عنوان cost برمیگرداند . در این صورت فقط داده های داخل margin در تعیین خط نقش دارند .

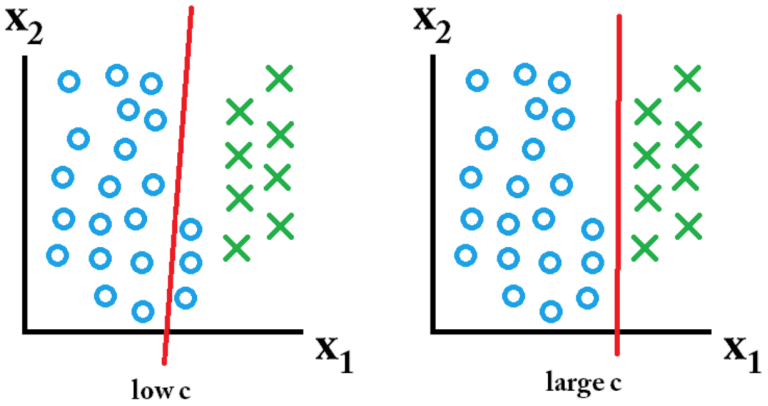
سوال دوم

در svm به دنبال دو چیز هستیم ، اول hyperplan ی با مقدار ماکزیمم margin و دوم ، hyperplane ی که تا جایی که میتواند به صورت صحیح داده های کلاس های مختلف را از هم جدا کند . مسئله این است که همیشه نمیتوان هردو شرط را ارضا کرد . C-parameter در Svm به ما میگوید که چقدر مورد دوم برای ما اهمیت دارد . در تصاویر زیر ، trade off بین دو شرط بالا را نشان میدهد که با تعیین پارامتر c ، به حالت های مختلف میپردازد .

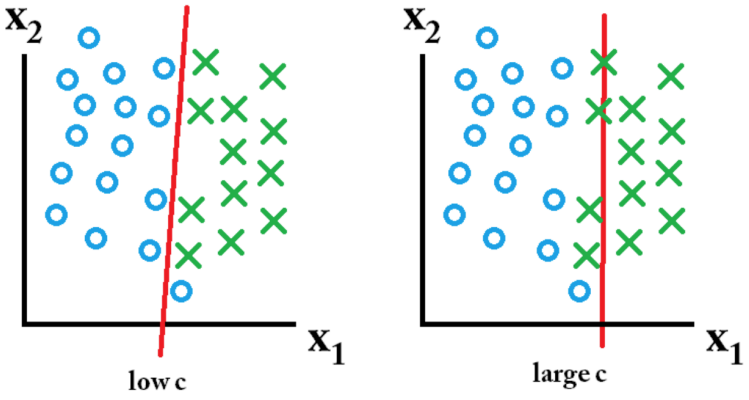
در تصویر اول ، برای پارامتر c بزرگ ، میبینم که برای این که داده ی پرت آبی به درستی جدا شود ، hyperplane با margin کوچک تری داریم و برای پارامتر c کوچک ، داده ی پرت آبی به اشتباه پیش بینی میشود ولی margin بزرگ تری داریم :



این که پارامتر c بزرگ یا کوچک باشد ، بستگی به توزیع داده ها دارد و باید با توجه به توزیع آن ها ، c را تعیین کنیم . در تصویر اول c بزرگ مطلوب ماست و در تصویر دوم c کوچک .



Large C



Low C

سوال سوم

ابتدا ۱۵۰ داده در بازه ی گفته شده تولید میکنیم و داده های test و train را جدا میکنیم .

```
x1 = np.random.uniform(0,1.5,150).reshape(150,1)
x2 = np.random.uniform(0,1.5,150).reshape(150,1)
x3 = np.random.uniform(2,3.5,150).reshape(150,1)
x4 = np.random.uniform(2,3.5,150).reshape(150,1)
```

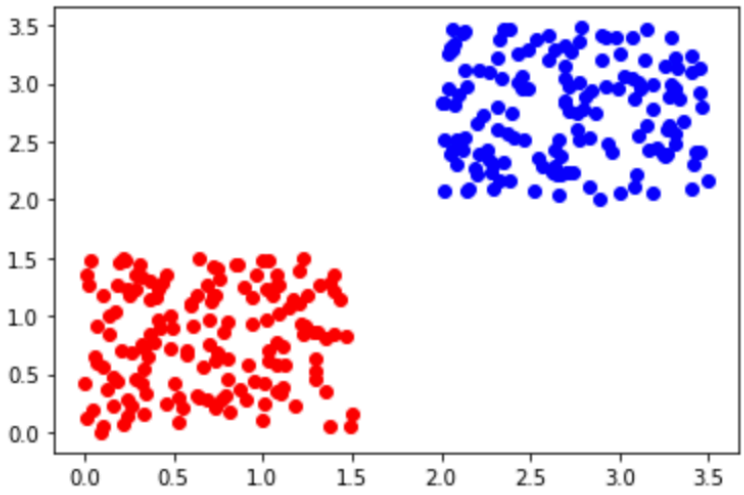
```
plt.plot(x1, x2, 'ro', x3, x4, 'bo')
plt.show()
```

```
class1 = np.concatenate((x1,x2),axis=1)
class2 = np.concatenate((x3,x4),axis=1)
class1 = np.c_[ class1 , np.ones(150) ]
class2 = np.c_[ class2 , (-1)*np.ones(150) ]
```

```
np.random.shuffle(class1)
np.random.shuffle(class2)
test1 = copy.deepcopy(class1[0:29,:])
test2 = copy.deepcopy(class2[0:29,:])
train1 = copy.deepcopy(class1[30:,:])
train2 = copy.deepcopy(class2[30:,:])
```

```
train = np.concatenate((train1, train2), axis=0)
np.random.shuffle(train)
```

```
X = copy.deepcopy(train[:,0:2])
Y = copy.deepcopy(train[:,2])
X = np.c_[ np.ones(240) , X ]
```



برای این مسئله ، پارامتر c را بزرگ در نظر گرفتم ، چون داده ها به طور شهودی فاصله ی زیادی از هم دارند و دیتاها در دو ناحیه ی تقریبا جدا قرار دارند ، پس امکان تداخل داده ها در ناحیه ی کلاس ها تقریبا وجود ندارد و میتوانیم از c بزرگ استفاده کنیم .

سپس با استفاده از روابط svm و گرادیان کاهشی که بیان شد ، ابتدا تابع Gradient_cost_Function را تعریف میکنیم که با استفاده از رابطه زیر ، به محاسبه ی cost میپردازد :

$$cost = \frac{\Sigma theta}{N} \qquad if max(0, 1 - y_i * (theta \,.\, x_i) = 0$$

$$cpst = \frac{\Sigma theta - C y_i x_i}{N} \qquad if max(0, 1 - y_i * (theta \,.\, x_i) \neq 0$$

همان طور که قبلا بیان شد ، داده های داخل margin. در تعین cost ، نقش دارند.

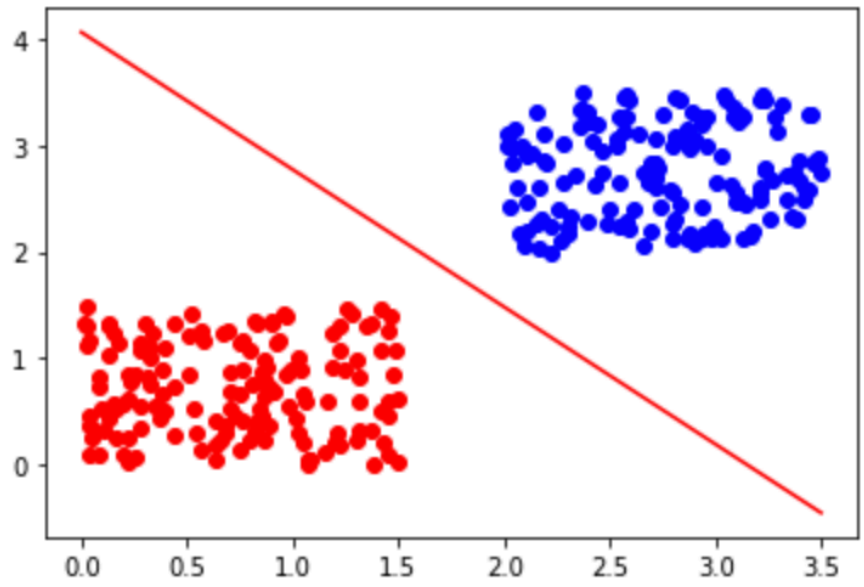
```
def Gradient_cost_Function(theta, X_batch, Y_batch):
    if type(Y_batch) == np.float64:
        Y_batch = np.array([Y_batch])
        X_batch = np.array([X_batch])
    margin = 1 - (Y_batch * np.dot(X_batch, theta))
    gradient_cost = np.zeros(len(theta))
    if max(0, margin) == 0:
        gradient_cost += theta
    else:
        gradient_cost += theta - (C * Y_batch[0] * X_batch[0])
    gradient_cost = gradient_cost/len(Y_batch)  # average
    return gradient_cost
```

سپس در تابع svm ، به تعداد ۵۰۰۰ حلقه ، theta را با استفاده از تابع Gradient_cost_Function آپدیت میکنیم . در نهایت ضرایب خط به عنوان لیست theta به ما داده خواهد شد :

```
def SVM(X, Y):
    theta = np.zeros(3)
    for epoch in range(5000):
        for i, x in enumerate(X):
            g_cost = Gradient_cost_Function(theta, x, Y[i])
            theta = theta - (alpha * g_cost)

    return theta
```

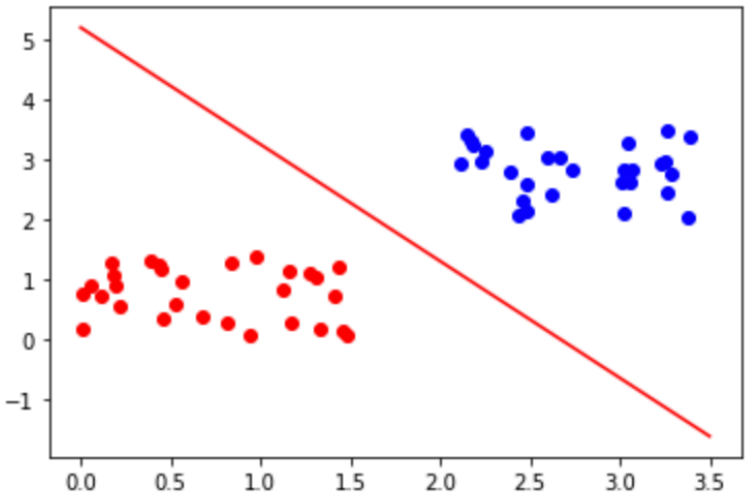
شکل نهایی به صورت زیر است :



Hyperplane ایجاد شده ، دقیقا میان داده های نزدیک به خط و با حدکثر margin ایجاد شده است .

سپس داده های تست را با hyperplane به دست آمده رسم میکنیم :

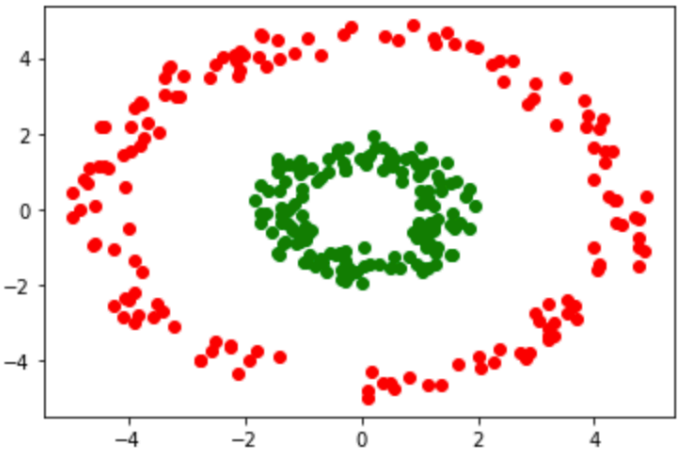
```
plt.plot(test1[:,0], test1[:,1], 'ro', test2[:,0], test2[:,1], 'bo')
x = np.linspace(0,3.5,100)
y = w[1]/(-w[2])*x + w[0]/(-w[2])
plt.plot(x, y, '-r', label='y=2x+1')
plt.show()
```

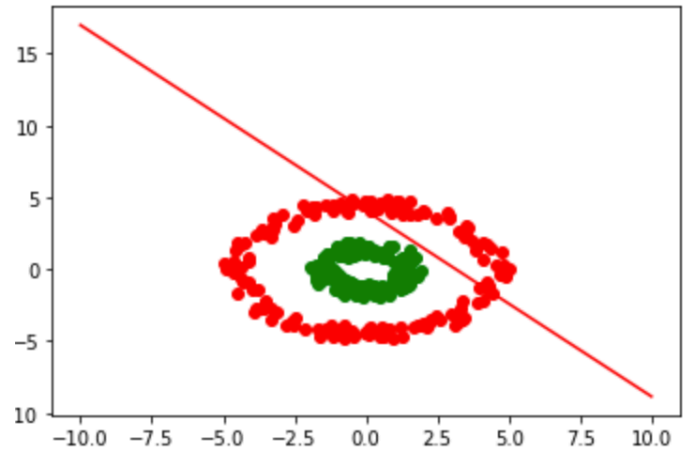


سوال چهارم

در این سوال داده ها با توزیع یکنواخت را در دو حلقه با محدوده شعاع های گفته شده در سوال ایجاد میکنیم :

```
import numpy
n = 150
phi1 = numpy.random.uniform(0, 2*numpy.pi, n).reshape(150,1)
r1= numpy.random.uniform(1, 2, n).reshape(150,1)
X1 = r1 * numpy.cos(phi1)
Y1 = r1 * numpy.sin(phi1)
phi2 = numpy.random.uniform(0, 2*numpy.pi, n).reshape(150,1)
r2 = numpy.random.uniform(4, 5, n).reshape(150,1)
X2 = r2 * numpy.cos(phi2)
Y2 = r2 * numpy.sin(phi2)
plt.plot(X1, Y1, 'go',X2, Y2, 'ro')
plt.show()
```



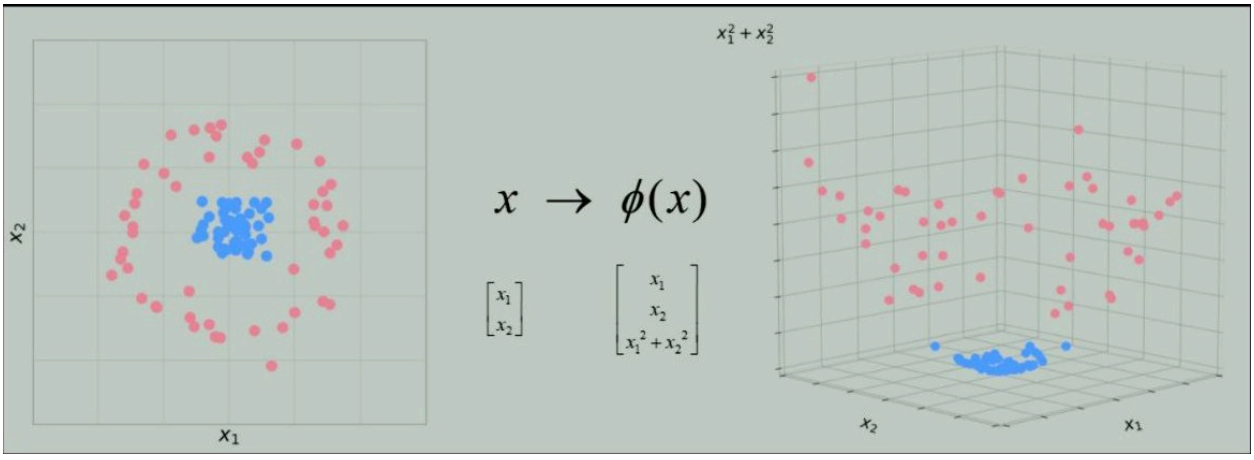


در این سوال ، با توجه به نتایج موجود ، میبینیم که مسائل classification غیر خطی را نمیتوان بدون از استفاده از کرنل طبقه بندی کرد .

سوال پنجم

زمانی که داده ها به صورت غیر خطی جدا نشدنی هستند ، طبقه بند های خطی در تعیین این مرز ممکن است موفق نباشند . اگرچه با بردن داده ها از یک فضا به یک فضای دیگر با بعد بالاتر میتواند میتواند به ما کمک کند تا با استفاده از طبقه بند های خطی این کار را انجام دهیم .

با کمک mapping function $\phi(x)$ میتوانیم داده هارا از حالت دو بعدی به حالت سه بعدی ببریم و سپس از طبقه بند های خطی استفاده کنیم .



اما اینکار هزینه ی محاسباتی بالایی دارد .

روش جایگزین استفاده از kernel function است . توابع کرنل به محاسبه ی ضرب داخلی بردار ویژگی ها میپردازد . تابع زیر کرنلی موسوم به کرنل چندجمله ای با درجه ی d است :

$$K(x, x') = \phi(x)\phi(x') = (x \,.\, x' + 1)^d$$

استفاده از توابع کرنل بسیار آسان تر از تعریف و استفاده از mapping function است .

در این مسئله از کرنل چندجمله ای با رابطه ی زیر استفاده کردم :

$$K(x, x') = \phi(x)\phi(x') = (x \,.\, x' + 1)^3$$

و تابع کرنل در نهایت ماتریسی برمیگرداند که هر درایه ی آن ، نمونه ی i و j داده های X را با استفاده از رابطه ی بالا map کرده است :

```python
def kernel(X):
    kernel_x = np.zeros((len(X),len(X)))
    for i in range(len(X)):
        for j in range(len(X)):
            kernel_x[i][j] = np.matmul(X[i].T,X[j]+1)**3
    return kernel_x
```

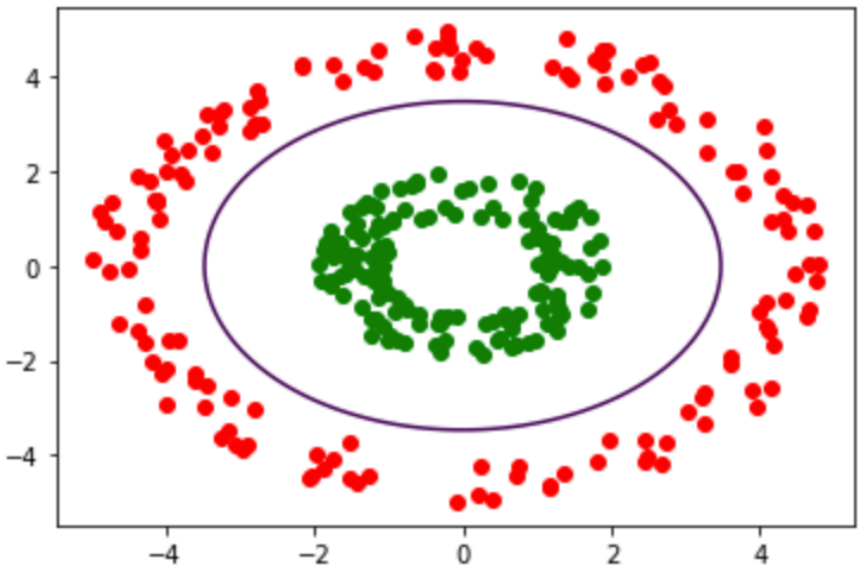
در تابع nonlinear kernel ، ابتدا کرنل X را محاسبه میکنیم ، سپس به تعداد ۵۰۰۰ بار حلقه ی for را برای محاسبه ضرایب beta اجرا میکنیم :

$$\beta := \beta + \alpha(Y - K\beta)$$

در این رابطه ، K کرنل X است و alpha ، learning rate است :

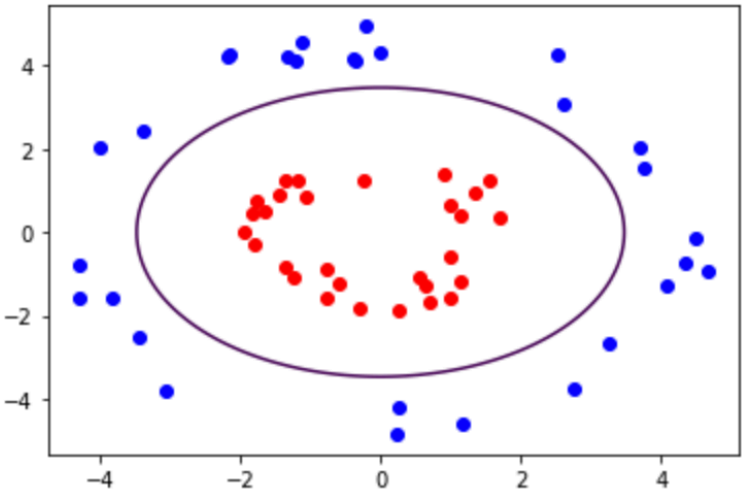
```python
def nonlinear_kernel(X , Y ):
    kernel_x = kernel(X)
    beta = np.zeros(240)
    for epoch in range(5000):
        ascent = Y - np.matmul(kernel_x,beta)
        beta = beta + (alpha * ascent)
    theta = np.matmul(beta.T,X)
    return theta
```

بعد از محاسبه ی beta ، ضرایب تتا را به دست می آوریم و به صورت کانتور ، مرز های مسئله را رسم میکنیم :



سپس داده های تست را با hyperplane به دست آمده رسم میکنیم :

```
plt.plot(test1[:,0], test1[:,1], 'ro', test2[:,0], test2[:,1], 'bo')
x = np.linspace(-4, 4, 100)
y = np.linspace(-4, 4, 100)
X, Y = np.meshgrid(x,y)
F = +theta[0] - theta[1]*X**2 - theta[1]*Y**2
plt.contour(X,Y,F,[2])
plt.show()
```



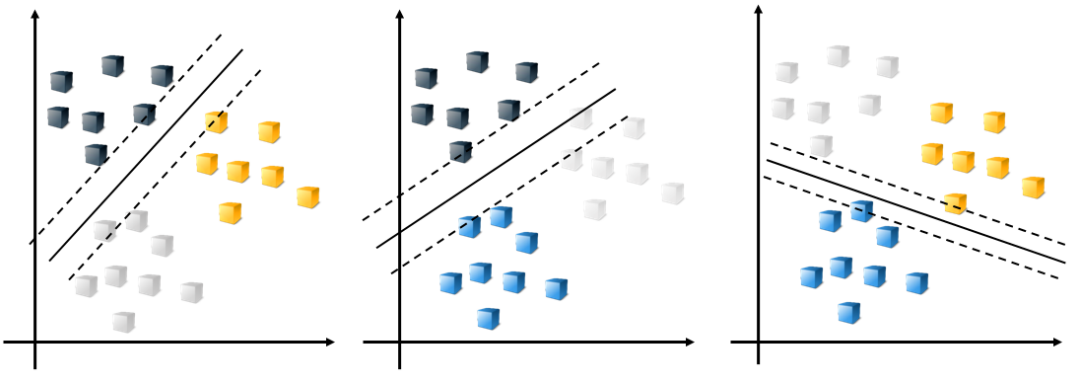
سوال ششم

Svm به صورت کلی طبقه بندی چند کلاسه را support نمیکند و طبقه بندی های باینری را انجام میدهد. برای طبقه بندی های چند کلاسه باید مسئله را به چند مسئله ی طبقه بندی باینری تبدیل کنیم .

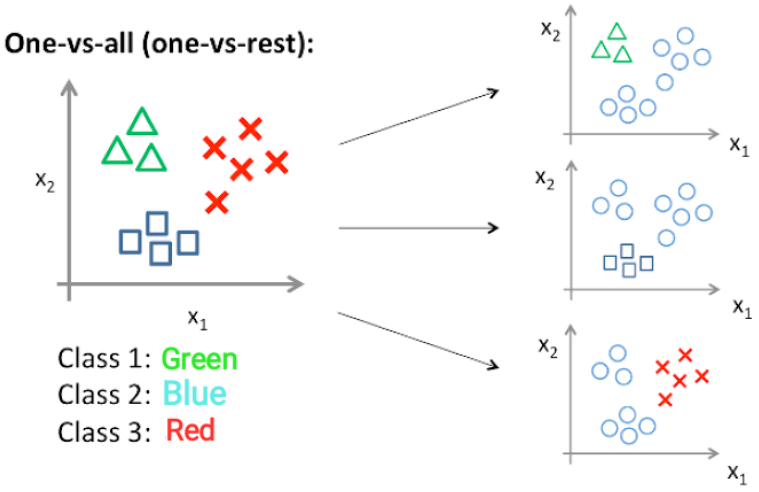
دو رویکرد برای حل این مسئله داریم :

one vs one

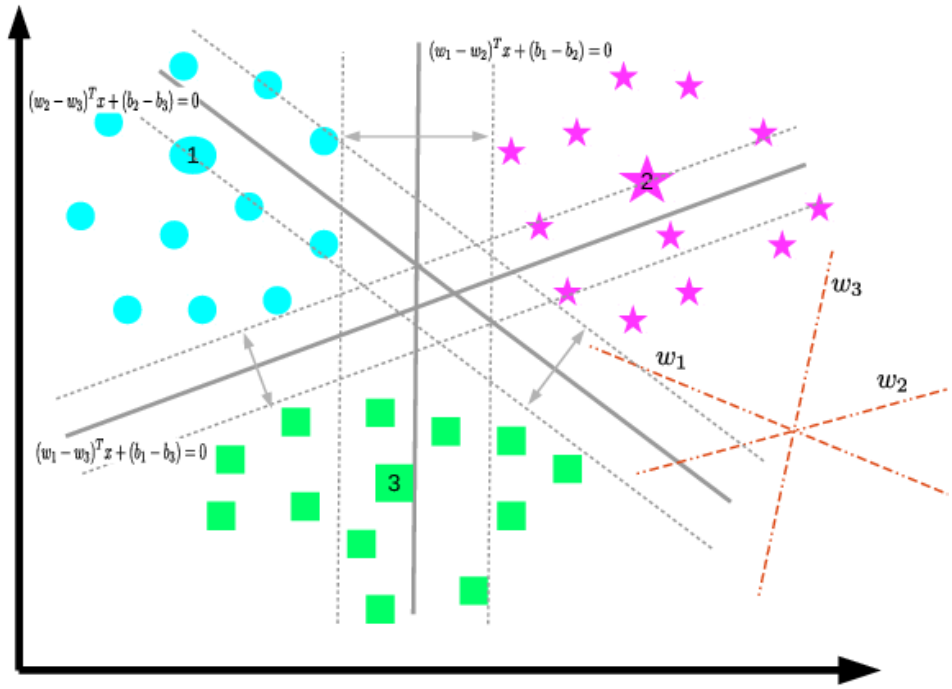
ایده ی کلی به صورت است که نقاط داده را به فضای ابعاد بزرگ تری map کنیم تا بین هردو کلاس ، خط جداکننده ی متقابل را بیابیم . یعنی hyperplane را برای هردو کلاس به صورت جدا محاسبه میکنیم .



one vs all



در این روش hyperplane هر کلاس را نسبت به داده های تمام کلاس های دیگر محاسبه میکنیم . به این صورت که داده هارا در هر مرحله به دو گروه تمام داده های کلاس مورد نظر برای طبقه بندی و بقیه داده های تمام کلاس ها تقسیم بندی میکنیم .



در طبقه بندی به روش svm ، اگر ویژگی ها بیشتر از ۲ تا باشند ، hyperplane از حالت خط به ابعاد بالاتری میشود . مثلا اگر ویژگی ها ۳ تا باشند ، hyperplane به صورت صفحه خواهد بود و اگر ویژگی ها ۴ تا باشند ، به صورت یک ابر یا حجمی از فضا خواهد بود . در ابعاد بالاتر هم قابل تصویر سازی نیست .