به نام خدا

دانشکده مهندسی برق

پروژه نهایی

درس بهینه سازی محدب

محمدامین حاجی خداوردیان

۹۷۱۰۱۵۱۸

استاد:دکتر بابازاده

نیمسال دوم ۱۴۰۰-۱۴۰۱

بخش اول

برای قسمت اول پروژه با استفاده از CVX و کرنل گفته شده به حل مسئله دوگان پرداختیم. قطعه کد زیر برای این بخش زده شده است:

```
%Choose Gamma and C
i_G=1;
i_C=1;
        Gamma = GammaList(i_G);
        C = CList(i_C);
        N = size(XTrain,1);
        K = zeros(N,N);
            for i=1:N % RBF kernel
                for j = 1:N
                    K(i,j) = exp(-1*Gamma*sum((XTrain(i,:)-XTrain(j,:)).^2));
                end
            end
%CVX optimization
cvx_begin
    variable alpham(N);
    maximize (-0.5.*quad_form(YTrain.*alpham,K) + ones(N,1)'*(alpham));
    subject to
        alpham >= 0;
        YTrain'*alpham == 0;
        alpham <= C;
cvx_end
%Displaying the results.
disp({'C =' , num2str(C) ; 'Gamma=' , num2str(Gamma) ; 'OptimalValue =' , num2str(cvx_optval)});
```

نتایج به صورت زیر است:

```
Results of Optimization with CVX:
    'C ='                '0.01'
    'Gamma='             '10'
    'OptimalValue ='     '7.1303'
```

```
Results of Optimization with CVX:
    'C ='                '0.1'
    'Gamma='             '10'
    'OptimalValue ='     '49.4808'
```

```
Results of Optimization with CVX:
    'C ='                '0.5'
    'Gamma='             '10'
    'OptimalValue ='     '187.5612'
```

```
Results of Optimization with CVX:
    'C ='                '1'
    'Gamma='             '10'
    'OptimalValue ='     '326.3421'
```

```
Results of Optimization with CVX:
    'C ='                '0.01'
    'Gamma='             '50'
    'OptimalValue ='     '6.7163'
```

```
Results of Optimization with CVX:
    'C ='                '0.1'
    'Gamma='             '50'
    'OptimalValue ='     '31.2606'
```

```
Results of Optimization with CVX:
    'C ='                '0.5'
    'Gamma='             '50'
    'OptimalValue ='     '79.7071'
```

```
Results of Optimization with CVX:
    'C ='                '1'
    'Gamma='             '50'
    'OptimalValue ='     '116.6115'
```

```
Results of Optimization with CVX:
    'C ='                '0.01'
    'Gamma='             '100'
    'OptimalValue ='     '6.7948'
```

```
Results of Optimization with CVX:
    'C ='                '0.1'
    'Gamma='             '100'
    'OptimalValue ='     '28.3742'
```

```
Results of Optimization with CVX:
    'C ='                '0.5'
    'Gamma='             '100'
    'OptimalValue ='     '60.7804'
```

```
Results of Optimization with CVX:
    'C ='                '1'
    'Gamma='             '100'
    'OptimalValue ='     '83.4114'
```

```
Results of Optimization with CVX:
    'C ='                '0.01'
    'Gamma='             '500'
    'OptimalValue ='     '7.3065'
```

```
Results of Optimization with CVX:
    'C ='                '0.1'
    'Gamma='             '500'
    'OptimalValue ='     '42.4175'
```

```
Results of Optimization with CVX:
    'C ='                '0.5'
    'Gamma='             '500'
    'OptimalValue ='     '64.3648'
```

```
Results of Optimization with CVX:
    'C ='                '1'
    'Gamma='             '500'
    'OptimalValue ='     '73.9633'
```

بخش دوم

برای رسم خط تصمیم گیرنده نیاز است که با توجه به مطالب گفته شده و صورت مسئله اصلی، $w^Tx + b$ را محاسبه کنیم. برای اینکار نیاز است با استفاده از آلفاهای بدست آمده از حل مسئله CVX برای بدست آوردن W و b استفاده کنیم. برای محاسبه b با توجه به مطالب درس از کد زیر استفاده شده است:

```
%Calculating w'x + b
w = 0;
for i=1:N
    if alpham(i) > 1e-5 && alpham(i) < 1/2*C
        S = i;
        break;
    end
end
for i=1:N
    if alpham(i) > 1e-5
     w = w + alpham(i)*YTrain(i)*exp(-1*Gamma*sum((XTrain(S,:)-XTrain(i,:)).^2));
    end
end

b = YTrain(S) - w;
```

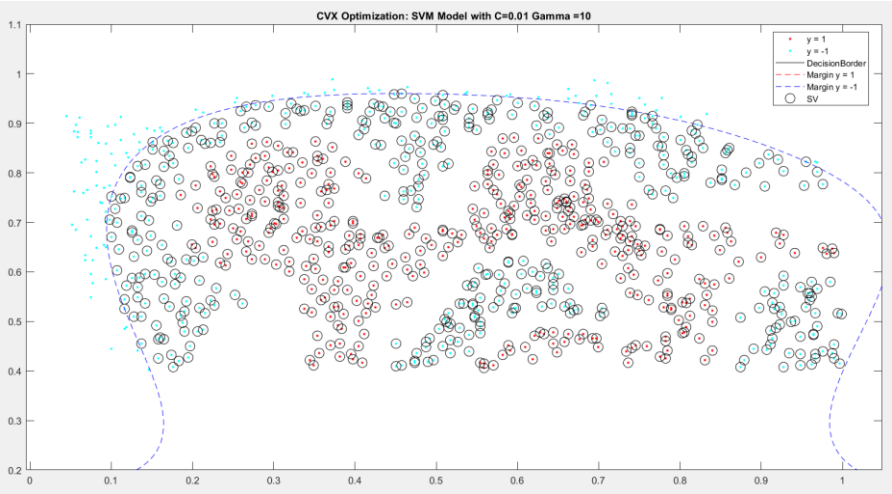
که در آن مقدار w با توجه شرایط KKT بدست آمده است و مقدار b به صورت یک مجهول و حل معادله خط تصمیم گیرنده بدست می‌آید.

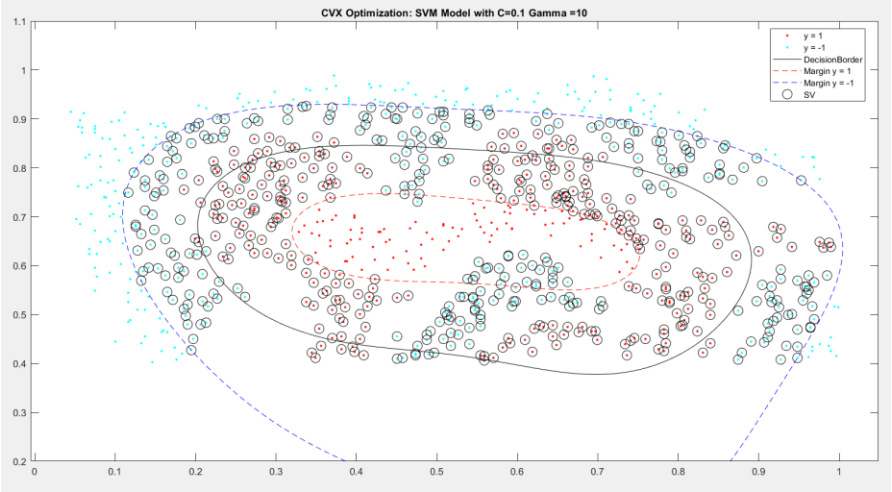
برای رسم خط از قطعه کد زیر استفاده شده است:

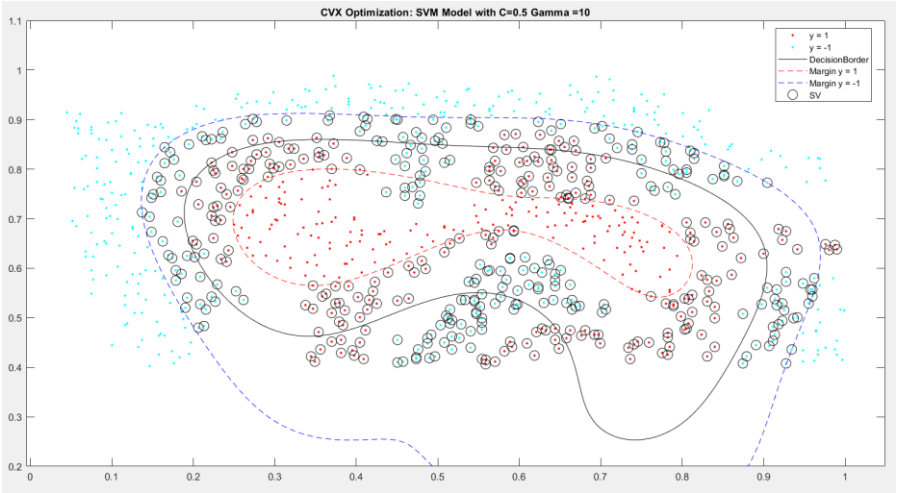
```
figure;

gscatter(XTrain(:,1),XTrain(:,2),YTrain);
hold on

X1Plot = linspace(0 , 1.1 , 200);
X2Plot = linspace(0.2 , 1.1 , 200);
[X,Y] = meshgrid(X1Plot,X2Plot);
Z = b;
f = zeros(size(X));
for i=1:size(X,1)
    for j=1:size(Y,1)
            xnew = [X(i,j) , Y(i,j)];
            wx = 0;
            for k=1:863
                if alpham(k) > 1e-5
                    wx = wx + alpham(k) *YTrain(k) * exp(-Gamma * ((xnew(1)-XTrain(k,1))^2 + (xnew(2)-XTrain(k,2))
                end
            end
            f(i,j) = wx + b;
     end
end
contour(X,Y,f,[0,0] , 'k');
contour(X,Y,f,[-1,-1] , 'r--');
contour(X,Y,f,[1,1]  , 'b--' );
```

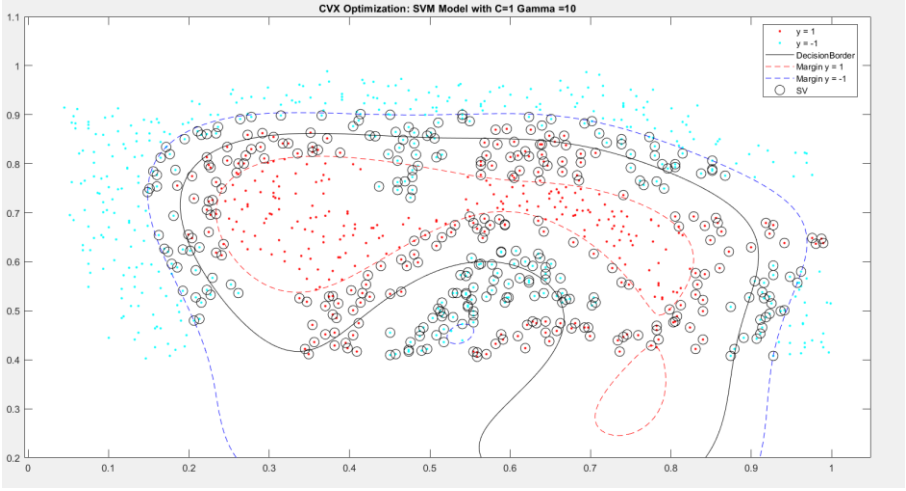
```
for i = 1:N
    if(SupVec(i))
        plot(XTrain(i,1), XTrain(i,2), 'ko', 'Markersize', 10);
        hold on;
    end
end
legend('y = 1' , 'y = -1' , 'DecisionBorder' , 'Margin y = 1' , 'Margin y = -1' , 'SV');
title(['SVM Model with C=' , num2str(C) , ' Gamma =' , num2str(Gamma)]);
disp(['Support Vectors =' , num2str(count)]);
```

که علاوه بر خط تصمیم گیرنده ، ساپورت وکتورها و مرز جدا کننده را نیز مشخص می‌کند. با افزایش مقدار C امکان رخ دادن خطا را افزایش می‌دهیم ولی این کار سبب می‌شود از بیش برازش جلوگیری کنیم. از طرفی مقدار گاما معیاری برای سنجش نزدیکی داده ها به یکدیگر است مقدار هردوی این پارامترها باید به گونه ای باشد که نه کم باشد و نه خیلی زیاد در غیر اینصورت نتایج خوبی بدست نمی‌آوریم.نتایج در زیر موجود است:

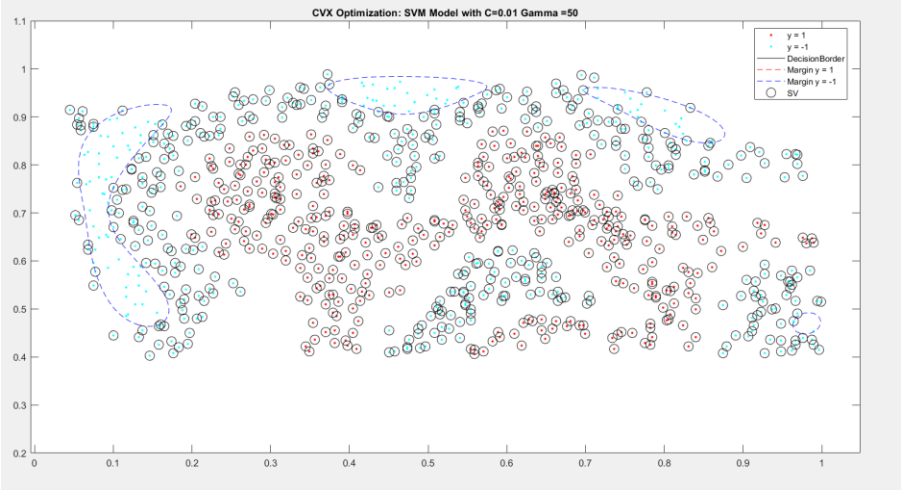


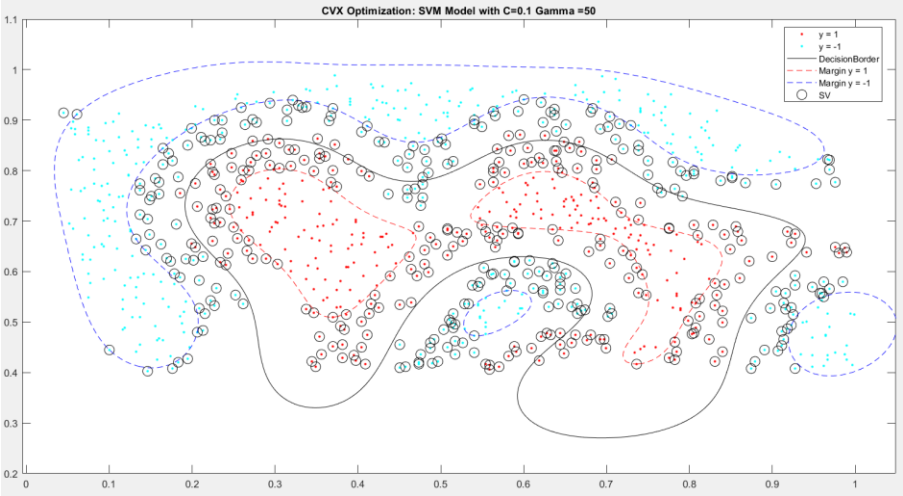


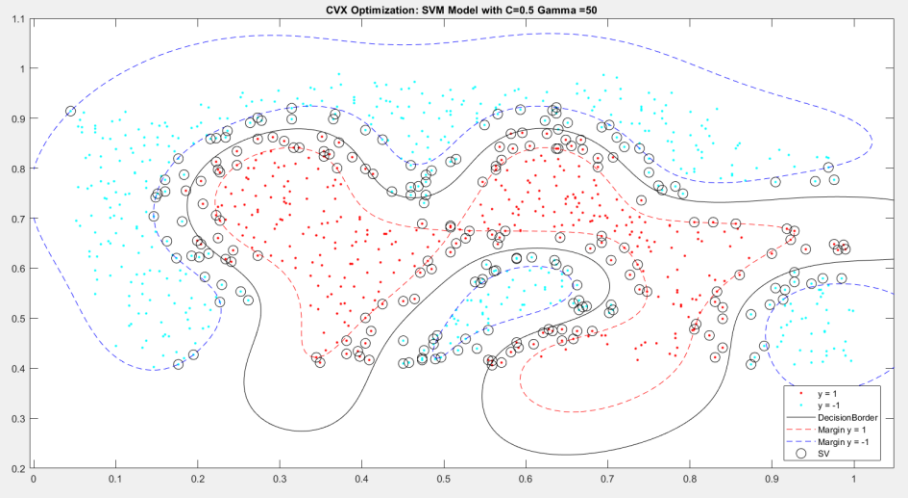


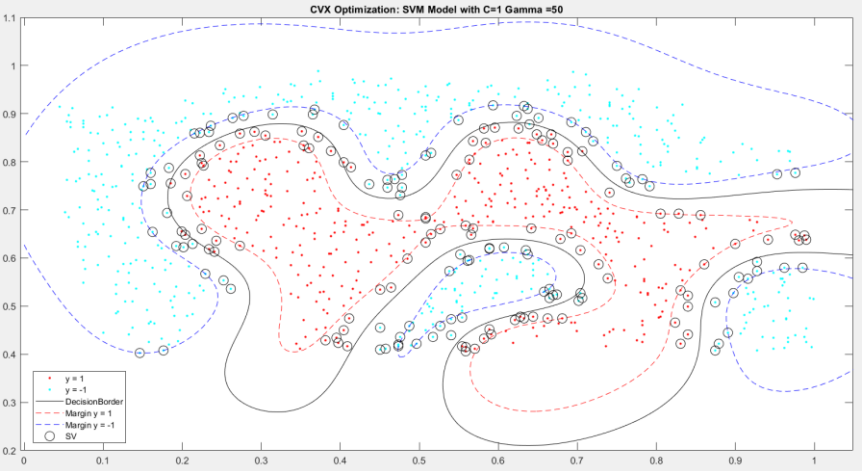


همانطور که در نتایج بالا دیده می‌شود با افزایش c نتایج بهتر می‌شود ولی هنگامی که مقدار ۱ را برای آن قرار می‌دهیم میزان قابل قبول بودن خطا را افزایش دادیم و نتایج به خوبی C=0.5 نیست.

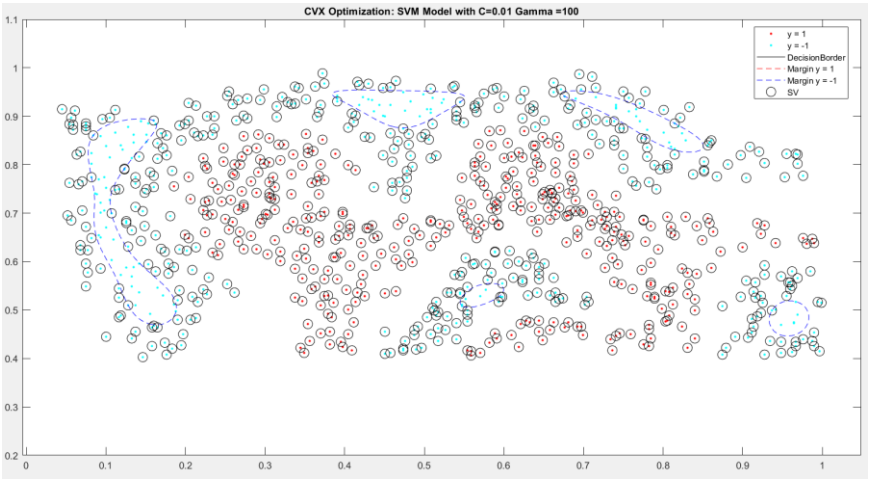

CVX Optimization: SVM Model with C=0.01 Gamma =50
y = 1
y = -1
DecisionBorder
Margin y = 1
Margin y = -1
SV

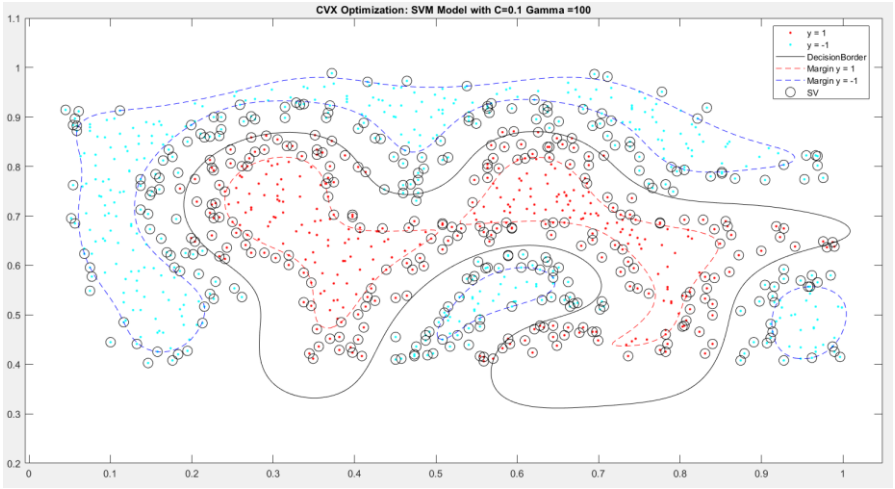

CVX Optimization: SVM Model with C=0.1 Gamma =50
y = 1
y = -1
DecisionBorder
Margin y = 1
Margin y = -1
SV

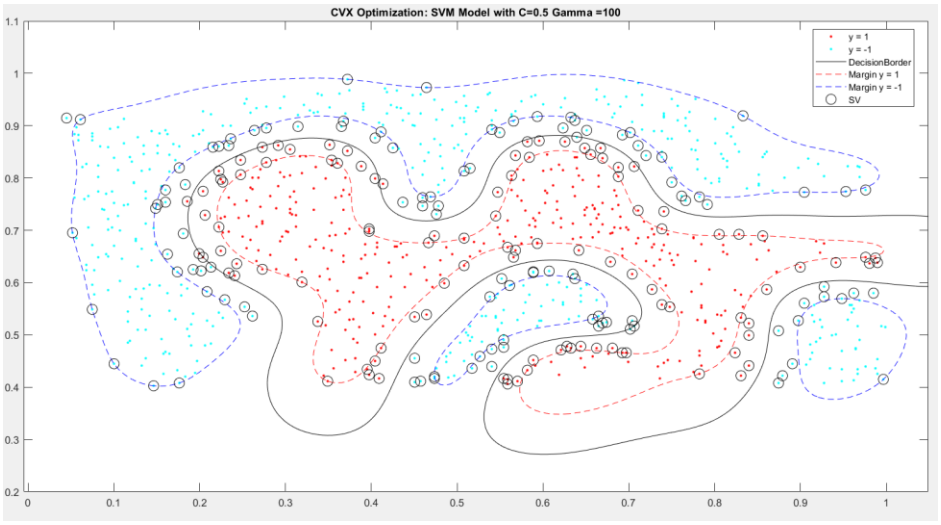

CVX Optimization: SVM Model with C=0.5 Gamma =50
y = 1
y = -1
DecisionBorder
Margin y = 1
Margin y = -1
SV

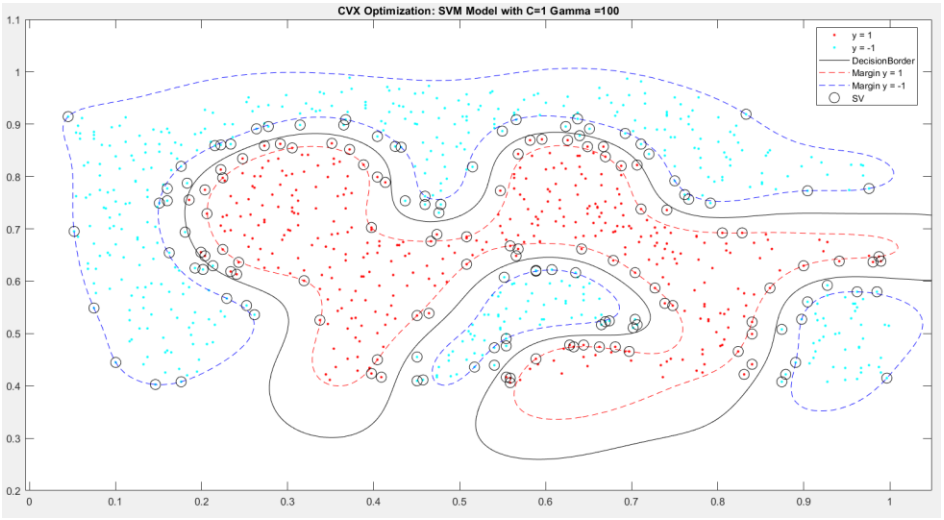


دقت خط تصمیم گیری نسبت به حالت قبل که گاما کمتر بود بهتر شده است.

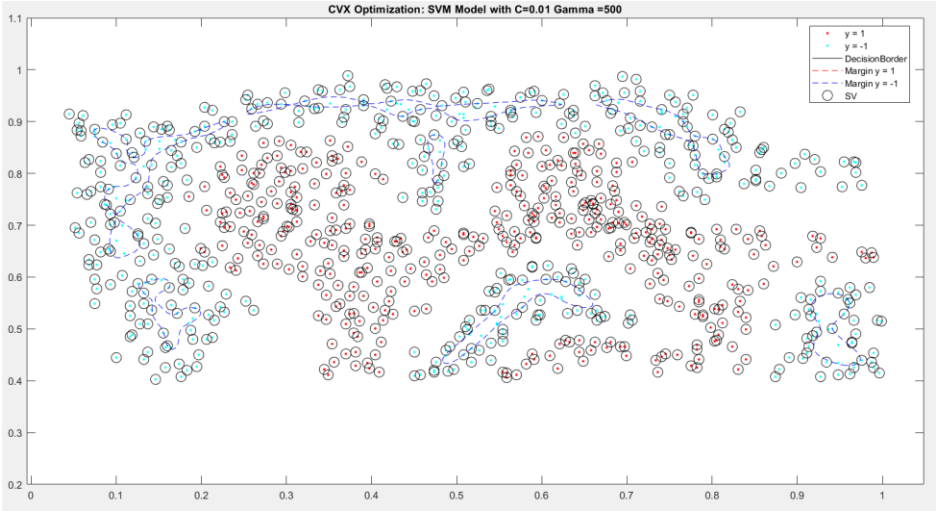


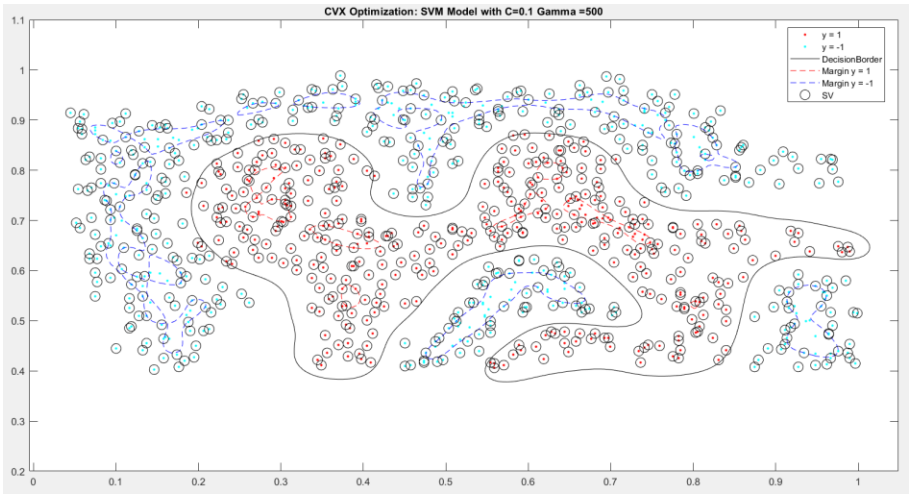


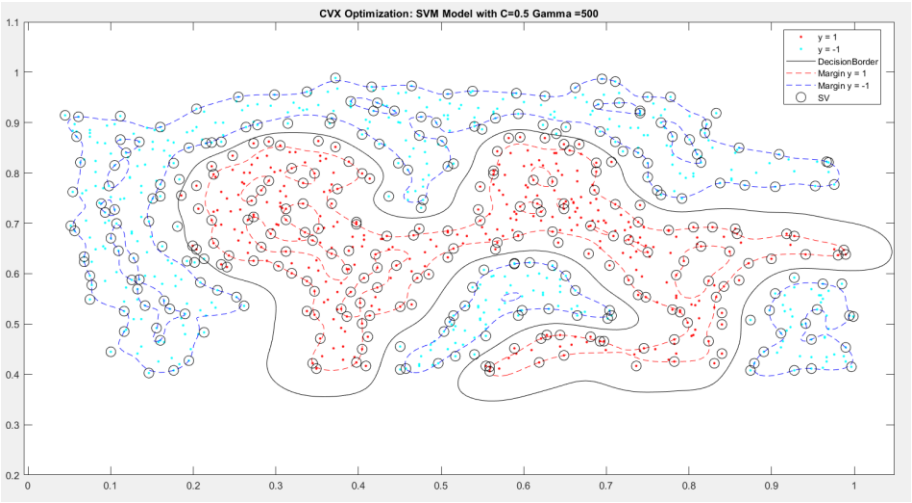


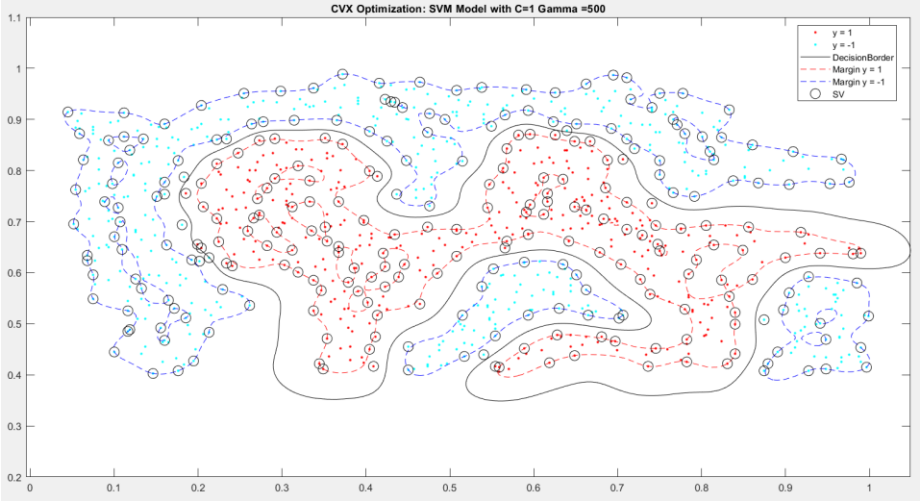


با افزایش گاما نتایج بهتری کسب کرده ایم و همچنین مشابها افزایش C تا یک جایی کاربردی است.




CVX Optimization: SVM Model with C=0.1 Gamma =500
y = 1
y = -1
DecisionBorder
Margin y = 1
Margin y = -1
SV


CVX Optimization: SVM Model with C=0.5 Gamma =500
y = 1
y = -1
DecisionBorder
Margin y = 1
Margin y = -1
SV


CVX Optimization: SVM Model with C=1 Gamma =500
y = 1
y = -1
DecisionBorder
Margin y = 1
Margin y = -1
SV

نتیجه گیری نهایی: با توجه به تصاویر بدست آمده حالت هایی که c=1 و یا c=0.01 است که در واقع جایی است که اجازه خطای بسیار زیاد و خطای بسیار کم داده شده است به خوبی جدا سازی انجام نشده است و برای مقادیر c=0.1 و c=0.5 جداسازی خوب رخ داده است که در واقع اجازه خطا دارد ولی مقدار آن بسیار زیاد نیست. از طرفی مقدار گاما که نزدیکی نقاط را نشان می‌دهد با افزایش آن که تغییری در کرنل ایجاد می‌کند ممقدار کرنل کاهش میابد و با استفاده از آن تفکیک بهتری صورت می‌گیرد.

بخش سوم

برای این بخش با توجه به مقادیر آلفا بدست آمده از بخش اول تعداد ساپورت وکتورها را حساب کرده ایم برای راحتی کار این مقادیر را در جدول زیر میاوریم:

γ \ C	0.01	0.1	0.5	1
10	768	584	447	401
50	770	448	249	185
100	770	449	210	157
500	796	745	347	282

کمترین تعداد Support Vector ها به ازای زمانی است که در بازه بین C = 0.5 تا C = 1 و از طرفی در بازه Gamma = 50 تا Gamma = 100 قرار داریم.

بخش چهارم

برای این بخش تابعی با نام Comp_H_G زده شده است. کد آن به صورت زیر است:

```
function [GradientF,HesianF] = Comp_H_G(XTrain, YTrain, Alpha, Mu, C, Gamma)
%   Detailed explanation goes here

    N = size(XTrain,1);
    K = zeros(N,N);
    for i=1:N
        for j = 1:N
                K(i,j) = exp(-1*Gamma*sum((XTrain(i,:)-XTrain(j,:)).^2));
        end
    end

    Barrier_Grad = zeros(863, 1);
    Barrier_Hesian = zeros(863, 1);
    for i = 1:N
        Barrier_Grad(i) = (-1)/Alpha(i) + (-1)/(C - Alpha(i));
        Barrier_Hesian(i,i) = 1/Alpha(i)^2 + 1/((C - Alpha(i))^2);
    end
    XTild = diag(YTrain) * K;
    GradientF = -diag(YTrain) * K * diag(YTrain) * Alpha + ones(863,1) + Mu * Barrier_Grad;
    HesianF = -diag(YTrain) * K * diag(YTrain) + Mu * Barrier_Hesian;

end
```

در کد بالا مشتق های توابع گرفته شده است و قرار داده شده است در این بخش توضیح خاصی وجود ندارد و مشتق گیری از توابع f و barrier مشخص است و در کد کاملا می‌توان آن را دنبال کرد.

بخش پنجم

برای این بخش چند تابع مختلف نوشته شده است که ابتدا به توضیح آن‌ها می پردازیم.

تابع اول برای محاسبه مقدار تابع در نقطه دلخواه است. این تابع در فایل Comp_F.m وجود دارد و به شکل زیر است:

```
function F = Comp_F(XTrain,YTrain,Alpha , C , Mu , Gamma)
%   Detailed explanation goes here
Sum = 0;
for i=1:length(Alpha)

        Sum = Sum - Mu * log(Alpha(i)) - Mu * log(C - Alpha(i));

end
N = size(XTrain,1);
    K = zeros(N,N);
    for i=1:N % RBF kernel
        for j = 1:N
                K(i,j) = exp(-1*Gamma*sum((XTrain(i,:)-XTrain(j,:)).^2));
        end
    end

F = (-1/2)*Alpha'*(diag(YTrain) * K * diag(YTrain))*Alpha + ones(863, 1)' * Alpha + Sum;
end
```

متغیر sum در واقع مقدار تابع barrier را حساب می‌کند و K نمایانگر کرنل است. در نهایت مقدار تابع در متغیر F با استفاده از مجموع تابع اصلی و تابع barrier به عنوان خروجی برگردانده می‌شود. برای آلفاها در این بخش هیچ محدودیتی در نظر گرفته نشده است ولی میدانیم اگر خارج بازه 0 و C شود مقدار تابع لگاریتمی به صورت مختلط می‌شود. این را در تابع بعد برای آن فکری کرده‌ایم.

تابع دوم که برای backtracking و پیدا کردن t نوشته شده است اسم آن BackTracking.m است و به شکل زیر است:

```
function t = BackTracking(XTrain,YTrain,Alpha , C , Mu , AlphaBT , BetaBT ,Gamma , t_Initial , V)
%   Detailed explanation goes here
[GF HF] = Comp_H_G(XTrain, YTrain, Alpha, Mu, C, Gamma);
Alphaprime = Alpha + t_Initial *V;
while(1)
    F = Comp_F(XTrain,YTrain,Alphaprime , C , Mu , Gamma);
    while(1)
        if(imag(F) == 0)
            break;
        end
        t_Initial = BetaBT * t_Initial;
        Alphaprime = Alpha + t_Initial *V;
        F = Comp_F(XTrain,YTrain,Alphaprime , C , Mu , Gamma);
    end
    if (Comp_F(XTrain,YTrain,Alphaprime , C , Mu , Gamma) <= Comp_F(XTrain,YTrain,Alpha , C , Mu , Gamma) + AlphaBT*t_Initial*GF'*V)
        break;
    end
    t_Initial = BetaBT * t_Initial;
    Alphaprime = Alpha + t_Initial * V;

end
t = t_Initial;
end
```

همانطور که دیده می‌شود در این بخش یک شرطی وجود دارد که تا زمانی که مقدار خروجی تابع اصلی دارای بخش موهومی است آلفا را تغییر دهد تا در بازه 0 تا C قرار بگیرد و مشکلی برای شرط نابرابری بوجود نیاید. در ادامه نیز مشابه گفته های درس به حساب کردن t پرداختیم.

در این تابع با استفاده از تابع Comp_F شرط خروجی که در درس گفته شد را چک می‌کنیم در صورتی که نقض نشده باشد مقدار t با بتای گفته شده در صورت پروژه جایگزین می‌شود و مقدار جدید آلفا با عنوان alphaprime حساب می‌شود. این مراحل انقد تکرار می‌شود که به t مطلوب برسیم. حال به سراغ بدنه اصلی این بخش می‌رویم. کد آن به شکل زیر است:

```
%newton's method
while(2*863 * Mu > Epsilone)
    T_Initial = 1e-4;
    while(1)
        [GF, HF] = Comp_H_G(XTrain, YTrain, Alpha_Initial, Mu, C, Gamma);
        V = zeros(863, 1);
        W = 0;
        A = [HF YTrain; YTrain' 0];
        Temp = inv(A) * [-GF ; 0];
        V = Temp(1:863);
        T_Initial = 1e-4;
        t = BackTracking(XTrain,YTrain,Alpha_Initial , C , Mu , AlphaBT , BetaBT ,Gamma , T_Initial , V);
        Alpha_Initial = Alpha_Initial + t*V;
        if norm(t*V) < 1e-6
            Alpha_Opt = Alpha_Initial;
            break;
        end
    end
    OptVal_Barrier(Iteration) = Comp_F(XTrain,YTrain,Alpha_Opt , C , Mu , Gamma);
    Iteration = Iteration + 1;
    Mu = Mu * 0.5;
end
```

کد بالا الگوریتم نیوتون است. در ابتدا تا هنگامی که Mu در تعداد قیود که برابر است با ۲*۸۶۳ کمتر از مقدار دلخواهی نشود کار ادامه میابد. پس از آن با استفاده از یک t اولیه به شروع الگوریتم می‌پردازیم. در ابتدا مسئله نیوتون را با استفاده از یک آلفای اولیه دلخواه که توضیحات آن را در بخش پارامتر می‌دهیم حساب می‌کنیم. مشابه درس نیاز است که یک معادله حل شود که پارامتر گام برای الگوریتم نیوتون مشخص شود و آن هم معادله زیر است:

$$\begin{bmatrix} \nabla^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} -\nabla f(x) \\ 0 \end{bmatrix}$$

که در مسئله ما A برابر است با YTrain و هسیان نیز با استفاده از تابع گفته شده در قبل حساب شده است. از طرفی گرادیان نیز با استفاده از همان تابع بدست آمده است. بنابراین با حل این معادله به v می‌رسیم.

پس از آن با استفاده از backtracking بهترین t را برای طول گام انتخاب می‌کنیم با استفاده از این گام و جهت آن که با حل معادله قبل بدست آمد نقطه جدید آلفا را بدست می‌آوریم.برای شرط خروج از روش نیوتون این را در نظر گرفتیم که کار را تا جایی ادامه دهد که فاصله بین نقطه انتخابی فعلی و قبلی کمتر از 1e-6 شد در این صورت به اتمام برسد. پس از آن mu با استفاده از ضریبی به مقدار 0.5 کوچک می‌شود و روش barrier ادامه میابد.

حال به پارامترهای مسئله نگاه میندازیم.

```
%Parameters
AlphaBT = 0.01;
BetaBT = 0.5;
Epsilone = 1e-8;
Mu = 1e-4;
C = 0.1;
Gamma = 500;
Alpha_Initial = 9*C/10*ones(863, 1) + 0.01*rand(863, 1);
T_Initial = 1e-4;
GF = zeros(863, 1);
HF = zeros(863, 863);
```

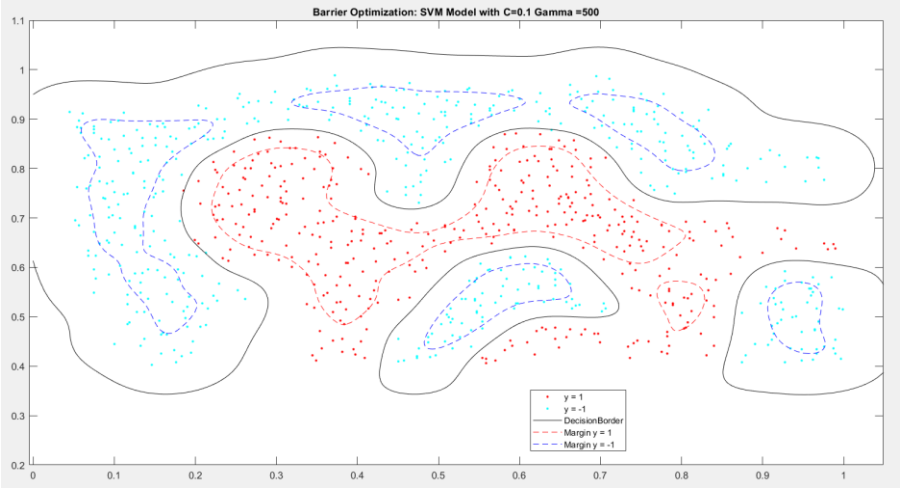
این پارامترها توسط صورت مسئله داده شده است و تنها دو پارامتر توسط ما انتخاب شده است. آلفای اولیه که به صورتی انتخاب شده است که شرط alpha’*YTrain =0 این شرط برقرار شود و برای اینکار مقدار آن را به ضریب ۰.۰۱ با یک عدد رندوم جمع کرده‌ایم.

برای پارامتر t نیز 1e-4 را در نظر گرفتیم چون در حالت عادی نیز بازه کوچکی در اختیار داریم در واقع بازه ما بین ۰ و ۰.۱ است.

نتایج این الگوریتم به صورت زیر است:

```
Results of Optimization with Barrier:
    'C ='                '0.1'
    'Gamma='             '500'
    'OptimalValue ='     '40.2015'
```

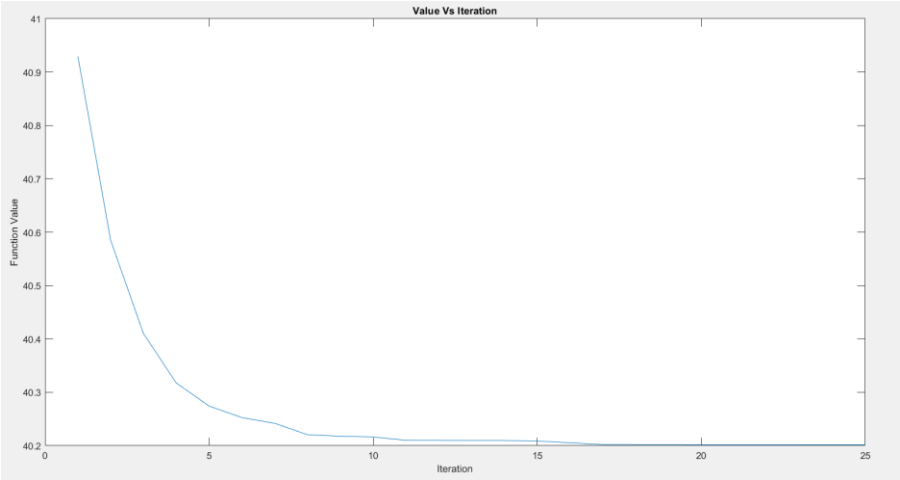
شکل مرز جداکننده نیز در صفحه بعد رسم شده است.



تنها مشکل این روش این است که تمامی نقاط به عنوان ساپورت وکتور در نظر گرفته شده اند.

بخش ششم

مقدار تابع با توجه به تعداد ایتریشن ها در زیر آورده شده است:



حال به مقایسه روش barrier و cvx می‌پردازیم.

```
Compare:
    'CVXTime ='       '45.1094'
    'BarrierTime='    '878.0156'

    'CVXOptVal ='       '42.4175'
    'BarrierOptVal='    '40.2015'

    'CVX_SV ='       '745'
    'Barrier_SV='    '864'
```

همانطور که دیده می‌شود روش barrier کندتر است و تمام نقاط ساپورت وکتور انتخاب شده است. مقدار تابع در هردو نزدیک به یکدیگر است. از لحاظ بهینگی هردو توانسته اند با توجه به خط تصمیم گیرنده جداسازی خوبی را در اختیار ما قرار دهند. ولی روش barrier با انتخاب آلفاهای متفاوت نتایج ممکن است عوض شود و شاید اصلا همگرا نشویم. همچنین در این روش اورفیت رخ داده و تمام نقاط ساپورت وکتور شده‌اند. در کل روش barrier برای دریافت تقریبی از جواب بهینه مناسب است و برای کارهای دقیق بهتر است از ابزارهای دیگر استفاده کرد.