Matlab gyakorlat II.

Bázisfüggvényes hálók és szupport vektor gépek

1. RBF

A MATLAB Neural Network toolboxban kétfajta RBF megoldás található. Az első minden tanítópontra illeszt egy Gauss-görbét, így minden tanítópontban 0 hibát képes elérni. Ezt a fajta hálót létrehozó parancs a **newrbe** (Radial basis – exact fit). A másik megoldás iteratívan növeli a neuronszámot, amíg el nem ér egy kívánt átlagos négyzetes hibát (MSE). Ennek a parancsa a **newrb**.

Először is hozzunk létre egy egyszerű tanítóminta-készletet, és ábrázoljuk grafikonon.

```
P = -1:.1:1; %intervallum -1-tõl 1-ig, 0.1-es lépésközzel
T = [-.9602 -.5770 -.0729  .3771  .6405  .6600  .4609 ...
      .1336 -.2013 -.4344 -.5000 -.3930 -.1647  .0988 ...
      .3072  .3960  .3449  .1816 -.0312 -.2189 -.3201];
plot(P,T,'+'); %ábrázoljuk P függvényében T-t
title('Training Vectors'); %ábra címe
xlabel('Input Vector P'); %x tengely felirat
ylabel('Target Vector T'); %y tengely felirat
```

1.1. newrbe

A tanítómintáinkat használjuk is fel egy RBF tanításához. Vegyük észre, hogy itt nincs szükség a train parancsra.

```
sc = 1;     % szórás konstans
net = newrbe(P,T,sc);
```

Jelenítsük meg grafikonon a háló válaszát

```
plot(P,T,'+');
xlabel('Input');

X = -1:.01:1; %tizedakkora lépésköz
Y = sim(net,X); %RBF háló szimulációja

hold on; %elõzõ ábrára szeretnénk rajzolni
plot(X,Y); %elõzõ ábrára rajzoljuk rá a háló válaszát
hold off;
legend({'Target','Output'});
```

Vizsgáljuk meg, hogy a kapott RBF-ben hány neuron van. Ehhez nézzük meg a rejtett réteg kimeneti súlyait.

```
net.LW{2,1}
```

Látható, hogy 21 neuron található az RBF-ben, vagyis mind a 21 tanítópontra esik egy külön bázisfüggvény. Ez nagyobb problémák esetén jelentős teljesítményromláshoz vezethet, a túltanulásról nem is beszélve.

1.2. newrb

Vizsgáljuk meg a másik RBF módszert is. Tanítsunk most a mintapontjainkra egy másik RBF-et a **newrb** paranccsal. Itt most 0-tól különböző MSE-vel is megelégszünk, mivel nem akarjuk, hogy minden tanítópontra essen egy bázisfüggvény, így elkerülve az esetleges túltanulást.

```
clear net; %töröljük az elõzõ RBF-et
eg = 0.02; % cél MSE - nem biztos, hogy el akarjuk érni a 0-t
sc = 1;    % szórás konstans
net = newrb(P,T,eg,sc);
```

Jelenítsük meg az eredményt, és vizsgáljuk meg a neuronok számát.

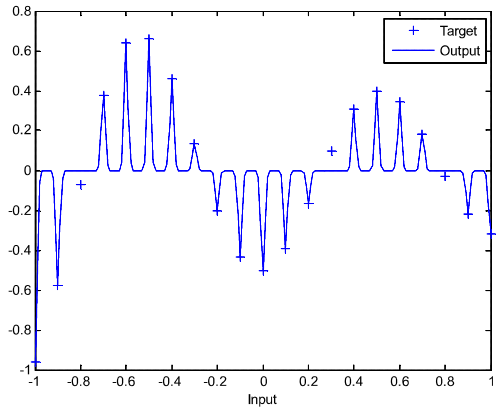
```
plot(P,T,'+');
xlabel('Input');

X = -1:.01:1; %tizedakkora lépésköz
Y = sim(net,X); %RBF háló szimulációja

hold on; %elõzõ ábrára szeretnénk rajzolni
plot(X,Y); %elõzõ ábrára rajzoljuk rá a háló válaszát
hold off;
legend({'Target','Output'});

net.LW{2,1}
```

Látható, hogy az illeszkedés nem teljesen pontos. Azonban mindezt 6 neuronnal sikerült elérni.

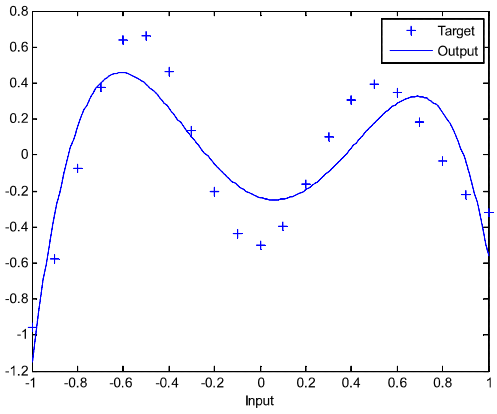
1.3. Szórás paraméter vizsgálata

Nézzük meg, hogy a szórás paraméter hogyan befolyásolja a tanulást. Ugyanezen a tanítókészleten futtassuk le az előző két kódot *sc = 0.01* és *sc = 100* paraméterekkel.

A kapott eredmények:



18 db neuron



21 db neuron

Látható, hogy sem a túl nagy, sem a túl kicsi konstans szórás nem előnyös.

1.4. Egy valós probléma megoldása

A feladat egy valós, ún. benchmark adathalmaz, a Boston Housing Data Set. A feladat célja az, hogy 14 paraméter alapján meghatározzuk az ingatlanárakat egy adott környéken. A paraméterek között olyanok találhatóak, mint az adó mértéke, régi épületek aránya, Boston belvárosától való távolság, vagy a bűnözési arány. További információ: http://archive.ics.uci.edu/ml/datasets/Housing

A probléma megoldásához először is bontsuk fel az adathalmazt bemeneti és célváltozóra, majd osszuk két részre a mintáinkat, hogy a tanítás után kiszámolhassuk az átlagos négyzetes hibát.

```
clear;
clc;
load 'housing.data'
P = housing(:,1:13); %az adatok elsõ 13 oszlopának átmásolása a P mátrixba
T = housing(:,14); %az adatok 14. oszlopa lesz a célváltozó

Pnormal = (P - ones(size(P,1),1)*mean(P)); %normalizálás, 0 várható érték
Pnormal = Pnormal ./ (ones(size(Pnormal,1),1)*sqrt(var(Pnormal)));
%egységnyi szórás

index = randperm(size(Pnormal,1)); %1..n -ig az egész számok egy véletlen
permutációja

trainP = Pnormal(index(1:450),:); %az index vektorban lévõ elsõ 450
indexnek megfelelõ érték
trainT = T(index(1:450),:);

testP = Pnormal(index(451:end),:); %a maradék pedig tesztminta lesz
testT = T(index(451:end),:);
```

Tanítsunk meg vele egy RBF-et, és nézzük meg az MSE-t több különböző szigmára, keressük meg a legjobb beállítást.

```
n = 50;
sigma = logspace(-2,0,n); %logaritmikus skála 0.01-től 1-ig
means = [];

for s=sigma
    net = newrbe(trainP',trainT', s);  %háló tanítása az adott szigma
paraméterrel
    Y = sim(net,testP'); %tesztmintákra a háló válasza
    err=(testT'-Y); %hiba kiszámítása
    means = [means, mean(err.^2)]; %MSE kiszámítása
end

plot(means);

optimal_sigma = sigma(find(means==min(means))); %legkisebb MSE-hez tartozó
szigma
net = newrbe(trainP',trainT', optimal_sigma); %ezzel megtanítjuk a
végleges hálót
```

2. MLP és RBF összehasonlítás – osztályozás

Az előző részben megismert kettős-spirál osztályozási problémát próbáljuk megoldani MLP-vel és RBF-el is.

2.1. Mintapontok generálása

Először is generáljunk adatot az osztályozáshoz. Nyissunk egy új .m filet, és nevezzük el `twospirals.m` –nek.

```
function [x,u] = twospirals(variance)

r=0.1:0.002:1;
x1=sin(r*20).*r;
y1=cos(r*20).*r;
x2=-sin(r*20).*r;
y2=-cos(r*20).*r;

%normális eloszlású zaj hozzáadása
x1 = x1 + randn(1,length(x1))*variance;
y1 = y1 + randn(1,length(y1))*variance;
x2 = x2 + randn(1,length(x2))*variance;
y2 = y2 + randn(1,length(y2))*variance;

%osztályok megadása
u1=ones(length(r),1);
u2=-ones(length(r),1);

x=[x1', y1'; x2', y2'];
u=[u1;u2];

%generált mintapontok megjelenítése
```

```
plot(x1,y1,'xr')
hold on
plot(x2,y2,'xb')
hold off
```

Generáljunk mintákat, válasszuk tanító– és tesztminta-készletre.

```
[x, u] = twospirals(0);

index = randperm(size(x,1)); %1..n -ig az egész számok egy véletlen
permutációja
traincount = round(length(index)*0.8);
trainP = x(index(1:traincount),:); %az index vektorban lévõ elsõ 450
indexnek megfelelõ érték
trainT = u(index(1:traincount),:);

testP = x(index(traincount+1:end),:); %a maradék pedig tesztminta lesz
testT = u(index(traincount+1:end),:);
```

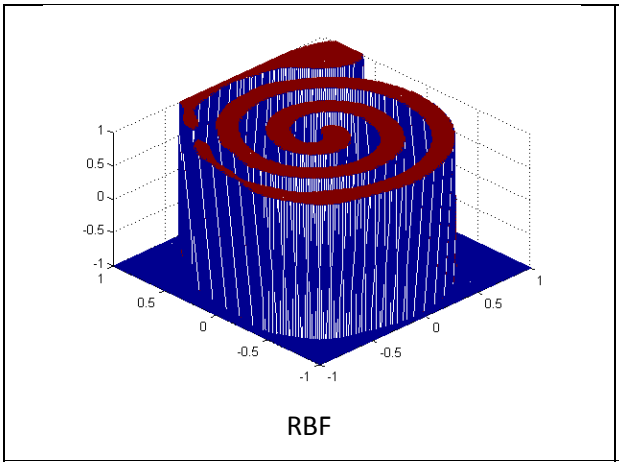
2.2. RBF osztályozás

Tanítsunk meg egy RBF-et. Számoljuk ki, hogy a tesztminták mekkora arányát osztályozza helyesen.

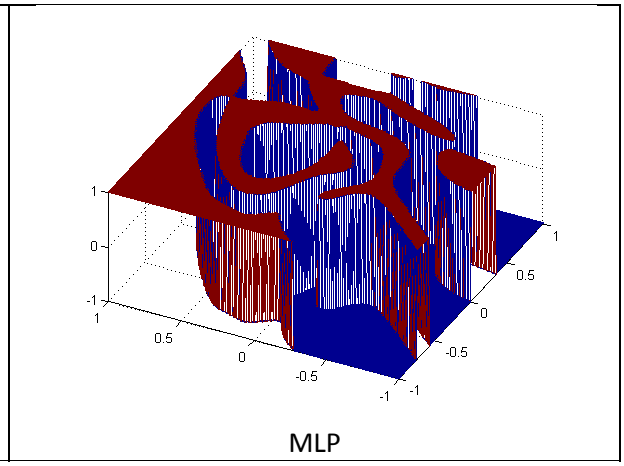
```
net = newrbe(trainP',trainT', 0.5);

Y = (sim(net,testP')>0)*2-1; %tesztmintákra a háló válasza
ok=(testT'==Y); %helyes értékek megszámolása
mean(ok) %mekkora arányban találjuk el
```

Ábrázoljuk a függvény értékét

```
figure(2);
[mx, my] = meshgrid(-1:0.01:1,-1:0.01:1);
test = (sim(net,[mx(:)';my(:)'])>0)*2-1;
test2 = reshape(test,201,201);
mesh(mx,my,test2);
```



RBF



MLP

2.3. MLP osztályozás

Most ugyanezt tegyük meg MLP-vel is.

```
net2 = newff(trainP',trainT',[10 10 10],{'tansig', ...
'tansig','tansig','purelin'},'traingda');
net2.trainParam.epochs = 2000;
net2.divideParam.testRatio = 0;
net2.divideParam.valRatio = 0;
net2.divideParam.trainRatio = 1;
net2 = train(net2,trainP',trainT');

Y = (sim(net2,testP')>0)*2-1; %tesztmintákra a háló válasza
ok=(testT'==Y); %helyes értékek megszámolása
mean(ok) %mekkora arányban találjuk el
```

Ábrázoljuk a függvény értékét

```
figure(2);
[mx, my] = meshgrid(-1:0.01:1,-1:0.01:1);
test = (sim(net2,[mx(:)';my(:)'])>0)*2-1;
test2 = reshape(test,201,201);
mesh(mx,my,test2);
```

3. Az SVM-KM toolbox telepítése

- Töltsük le a toolboxot a szerzők oldaláról:
  http://asi.insa-rouen.fr/enseignants/~arakoto/toolbox/index.html
  Tömörítsük ki a letöltött .zip filet a MATLAB\R2007b\toolbox\SVM-KM könyvtárba.
- Adjuk hozzá a Matlab path-hoz ezt a könyvtárat:
  File / Set Path… / Add folder with subfolders, majd válasszuk ki a könyvtárat. Ezután próbáljuk meg elmenteni a változtatásokat. Ha nem sikerül, az sem baj, csak következő futtatáskor ezt újra meg kell majd tennünk.

4. Osztályozás SVM-el

Ugyanezen kettősspirál feladatnak a zajos változatát oldjuk meg most SVM felhasználásával.

4.1. SVM osztályozó megtanítása

Egy másik, új .m fileba dolgozzunk mostantól. Először is bontsuk fel a mintapontjainkat tanító és tesztmintákra:

```
[x, u] = twospirals(0.01); %generáljunk egy zajos kettõsspirált

index = randperm(size(x,1)); %1..n -ig az egész számok egy véletlen
permutációja
traincount = round(length(index)*0.8); % 80%
trainP = x(index(1:traincount),:); %az index vektorban lév? els? 80%-nyi
indexnek megfelel? érték
trainT = u(index(1:traincount),:);
testP = x(index(traincount+1:end),:); %a maradék pedig tesztminta lesz
testT = u(index(traincount+1:end),:);
```

Majd állítsuk be az SVM paramétereit. A *c* paraméter értékét a két osztályhoz tartozó elemekre külön is megadhatjuk; itt most kompenzálunk az egyes osztályok számosságával, hogy ne tolódjon el a számítás az egyik osztály számbeli fölénye miatt.

Az első *c* érték végtelen lesz, vagyis nem engedünk meg pontot a biztonsági sávon belül.

Számos kernelfüggvény megadható, kommentezve látható pár. Ezekhez mind más és más *kerneloption* tartozik. Gauss kernel esetén ez a szórást jelenti.

Az osztályozó tanítását az **svmclass** paranccsal tehetjük meg.

```
%svm parameters
lambda = 1e-6;
verbose = 0;

%svm kernel parameter.
%use poly with degree 1 for a linear kernel

%kernel='poly';
kernel = 'gaussian';
%kernel = 'wavelet';

kerneloption = 0.1;
c = inf;

%compensate for the unbalanced training set
costfactor = sum(trainT < 0) / sum(trainT > 0);
C = ones(size(trainT,1),1) * c;
C(trainT > 0) = c * costfactor;


%train
[xsup,w,w0,pos,tps,alpha] =
svmclass(trainP,trainT,C,lambda,kernel,kerneloption,verbose);
```

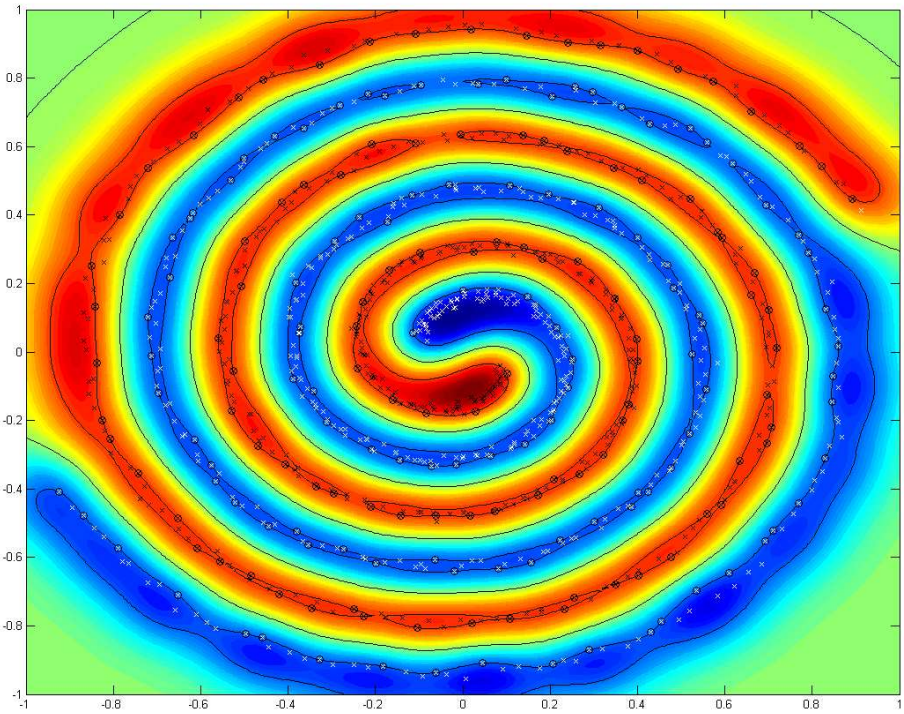
Míg a MATLAB Neural Network toolbox beli hálók (MLP, RBF, stb…) egy net struktúrában tárolódnak, addig az SVM-KM több változóban tárol egy SVM-et. Ez pedig az *xsup*, szupport vektorokat tartalmazó mátrix, a súlyokat tartalmazó *w* mátrix, illetve a biasokat tartalmazó *w0* vektor.

A létrehozott SVM tesztelése, helyes osztályozás arányának kiszámítása:

```
%evaluate
testY = (svmval(testP,xsup,w,w0,kernel,kerneloption)>0)*2-1;
testOK=(testT==testY);
mean(testOK)
```

SVM által megtanult felület, szupport vektorok, margó, határvonal:

```
figure(2)
[mx, my] = meshgrid(-1:0.01:1,-1:0.01:1);
test = svmval([mx(:),my(:)],xsup,w,w0,kernel,kerneloption);
testmatrix = reshape(test,201,201);
%eredmény ábrázolása szintvonalakkal, színekkel
contourf(mx,my,testmatrix,50);shading flat;
hold on
%margó ábrázolása, ahol +1 és -1 a kimenet. A 0 a határfelület.
contour(mx,my,testmatrix,[-1 0 1],'k');
%szupport vektorok bekarikázása zöld színnel
plot(xsup(:,1),xsup(:,2),'og');
%a két osztályba tartozó összes pont berajzolása
plot(x(1:size(x,1)/2,1),x(1:size(x,1)/2,2),'xk');
plot(x(size(x,1)/2:end,1),x(size(x,1)/2:end,2),'xw');
hold off
```



Látható, hogy a szupport vektorok a biztonsági sáv határán fekszenek. A biztonsági sávon belüli pontok pedig tesztpontok, ezekkel nem tanítottuk az SVM-et.

4.2. Paraméterek hangolása

Az előző példában a zaj nem sokat számított azon túl, hogy az egyes mintapontok nem egyenlő távolságra voltak egymástól. Nézzük most meg, hogy mi történik erősebb zaj hatására, mikor a két osztályba kerülő pontok már át is fednek egymással.

Állítsuk a zaj paramétert nagyobbra, 0.05-re.

```
[x, u] = twospirals(0.05);
```

A megtanult felület ekkor a legkevésbé sem hasonlít a kettős spirálra. Ennek oka, hogy minden egyes mintapontot helyesen szeretnénk osztályozni, noha lehet, hogy eleve a mintakészletünkben találhatóak fals értékek. A *c* paraméter állításával megengedhetjük, hogy a biztonsági sávon belül is essenek pontok, sőt, akár rosszul osztályozott pontokat is megengedünk.

Nézzünk meg több *c* értéket, 100, 10, 2, 1, 0.1

```
c = 1;
```

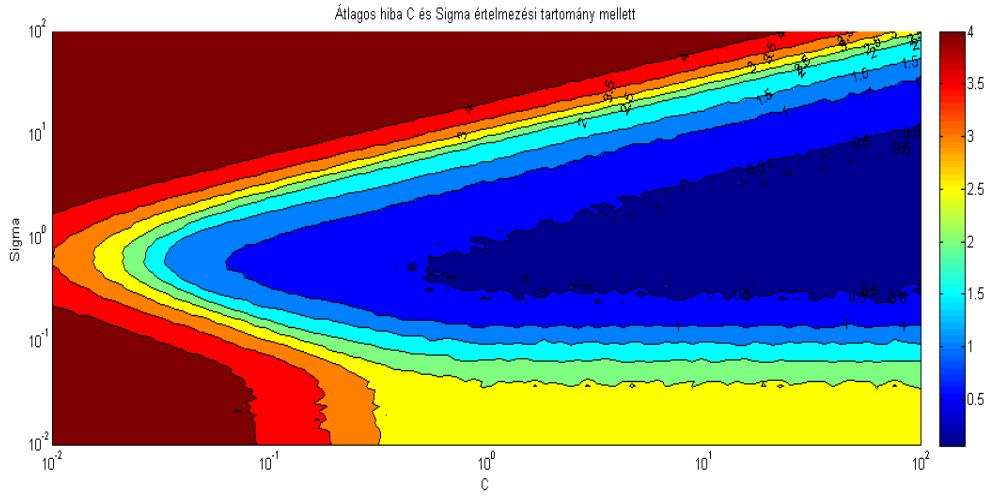
Látható, hogy a *c* csökkentésével egyre jobb eredményeket kapunk, a legjobb *c=1* körül tapasztalható. Azonban a *c* további csökkentése eredményromlással jár, túl sok mintapontra engedjük meg a hibás osztályozást.

Állítsuk a c értéket 1-re, és most vizsgáljuk meg a Gauss kernelünk szórás paraméterét. Próbáljuk ki a következő értékeket: 1, 0.5, 0.2, 0.1, 0.05, 0.03

```
kerneloption = 0.1;
```

Látható, hogy a nagy szórás értékekre az SVM egyszerűen nem tudja megtanulni a feladatot, hiszen az összes tanítópontra igaz, hogy ezen a szóráson belül nagyon sok ellenkező osztályba tartozó minta van. 0.1 körüli értéknél kapjuk a legjobb eredményt. Ennél kisebb értéknél a szórás annyira kicsi lesz, hogy a mintapontokra illesztett Gauss-görbék nem is „érnek össze”, elveszik az általánosító képesség. Megfigyelhető, hogy a legkisebb érték már nagyon sokáig fut, ennek oka, hogy gyakorlatilag minden mintapontot ki kell választani szupport vektornak.

Ha ábrázolnánk a *c* és szórás paraméterek terében az SVM teljesítményét, egy ehhez hasonló ábrát kapnánk:



A paramétertérben meghatározott hibafelületen a közel optimális osztályozáshoz tartozó paraméterek egy összefüggő területen találhatók. Megállapítható hogy a minimális hiba viszonylag nagy környezetében találhatóak közel optimális pontosságú osztályozásra vezető paraméterkombinációk. (Az ábrán a kék rész)

5. Regresszió SVM-el

5.1. Mintapontok generálása

Vizsgáljuk meg az SVM függvényapproximációs képességeit a sinc függvényen. Generáljunk zajos tanítópontokat belőle. Egy másik, új .m fileba dolgozzunk mostantól.

```
n=200;
x=linspace(-20,20,n)';
y = sin(x)./x; %az eredeti sinc függvény
y(x==0)=1; %x==0-nál ne legyen NaN
d = y + randn(length(x),1)*0.05; %zaj hozzáadása
h = plot(x,y,'g',x,d,'b+');
hold on
```

5.2. SVM tanítása

Az SVM regresszióra történő használata kicsit másképp történik, mint az osztályozásnál. Most az **svmreg** parancsot kell használni.

```
%paraméterek beállítása
C = 10;  lambda = 0.001;
epsilon = 0.05;
kerneloption = 2;
kernel='gaussian';
verbose=0;
%tanítás
[xsup,ysup,w,w0] = ...
    svmreg(x,d,C,epsilon,kernel,kerneloption,lambda,verbose);
```

Az SVM válaszának kiszámítása és megjelenítése

```
%az svm válasza
rx = svmval(x,xsup,w,w0,kernel,kerneloption);

h = plot(x,rx,'b');
h = plot(xsup,ysup,'or');
plot(x,rx+epsilon,'b--');
plot(x,rx-epsilon,'b--'); hold off;
hold off
```

5.3. Paraméterek hangolása

Mit is csinálnak az egyes paraméterek? Kezdjük megint a C paraméterrel. Próbáljuk ki a következő értékeket: inf, 1000, 10, 0.1, 0.01, 0.001

```
C = 10;
```

Látható, hogy nagy C értékeknél hasonlóan jó lesz a háló válasza, mint C=10 esetén, kis értékeknél azonban már nem követi rendesen a függvényt.

Először írjuk át a szórás paramétert 2-ről 0.5-re, majd változtassuk a lambda értéket: 10, 1, 0.1, 0.01, 0.000001, 0

```
lambda = 0.1;

kerneloption = 0.5;
```

Látható, hogy nagy lambda értékeknél a kimenet jóval simább, mint kis lambda értékeknél. Egészen kis értékeknél pedig gyakorlatilag a zajra tanulunk rá.

Változtassuk most az érzéketlenségi sáv szélességét az epsilon paraméterrel. 0.20, 0.15, 0.10, 0.5, 0.1

```
epsilon = 0.05;
```

Látható, hogy minél nagyobbra vesszük, annál kevesebb szupport vektorunk lesz, azonban annál kevésbé lesz pontos az SVM válasza. Nagyon kis érték esetén pedig szinte minden mintapont szupport vektor lesz.

Végezetül vizsgáljuk meg a Gauss kernel szórásparaméterének hatását. 0.1, 0.5, 1, 2, 4

```
kerneloption = 2;
```

Látható, hogy nagyon függ a megoldás minősége ettől. Túl kis szórás esetén a zajra tanul rá az SVM, túl nagy esetén pedig nem lehet megtanulni a prolémát.