Mesin Vektor Pendukung (SVM)
Kasus Non Linier

Beny Firman, 10 / 309779 / PTK / 07207 – TE
Jurusan Teknik Elektro & Teknologi Informasi FT UGM
Yogyakarta

2.5 SVM: Kasus Non Linier

Dalam menggunakan Teknik SVM untuk memecahkan masalah nonlinier, kita mengadopsi filosofi pemetaan vektor fitur dalam ruang dimensi yang lebih tinggi, seperti yang diharapkan, dengan probabilitas tinggi, kelas dipisahkan menjadi linier. Hal ini dijamin oleh Teorema Cover yang terkenal [Theo 09, Bagian 4.13]. Pemetaannya adalah sebagai berikut:

$$x \mapsto \phi(x) \in H$$

dimana H adalah sebuah dimensi yang lebih tinggi dari $\mathcal{R}^l$ dan tergantung pada pilihan dari φ(.) (nonlinier) bahkan bisa tidak terbatas . Selain itu, jika fungsi pemetaan dipilih secara cermat dari keluarga yang dikenal yang memiliki sifat tertentu, inner product antara gambar (φ(x1), φ(x2)) dari dua titik x1 , x2 dapat ditulis dalam:

$$< \phi(x_1), \phi(x_2) >= k(x_1, x_2)$$

di mana < ˚,˚ > menunjukkan operasi inner product sebuah nilai H dan k < ˚,˚ > adalah fungsi yang dikenal sebagai fungsi kernel. Ini berarti, inner product dalam ruang dimensi tinggi yang dapat dilakukan dalam hal fungsi kernel terkait yang bertindak dalam ruang dimensi rendah asli. Ruang H yang terkait dengan k < ˚,˚ > dikenal sebagai sebuah kernel yang mereproduksi ruang Hilbert atau (RKHS) (untuk definisi formal lebih lanjut dapat dilihat di [Theo 09, Bagian 4.18] dan referensi di dalamnya).

Karakteristik penting dari optimasi SVM adalah bahwa semua operasi dapat dicetak dalam hal inner product. Jadi, untuk memecahkan masalah linear dalam ruang dimensi tinggi (setelah pemetaan), yang harus kita lakukan adalah mengganti inner product dengan evaluasi kernel yang sesuai. Contoh umum dari fungsi kernel adalah pada fungsi dasar radial (RBF), didefinisikan sebagai:

$$k(x,y) = \exp\left(-\frac{||x-y||^2}{\sigma^2}\right)$$

di mana σ adalah parameter yang ditetapkan pengguna yang menentukan tingkat peluruhan k (x, y) menuju nol, sebagai y bergerak menjauh dari x dan (b) fungsi polinom, didefinisikan sebagai

$$k(x,y) = (x^T y + \beta)^n$$

B dan n adalah parameter yang ditentukan sendiri

Perhatikan bahwa dalam memecahkan masalah linear dalam ruang dimensi tinggi setara dalam memecahkan masalah nonlinier di ruang yang asli. Hal ini mudah untuk diverifikasi. Seperti dalam Pers. (2.12), hyperplane dihitung dengan metode SVM di ruang H dimensi atas yaitu:

$$w = \sum_{i=1}^{N} \lambda_i y_i \phi(x_i) \qquad (2.13)$$

Dengan nilai x, pertama kita petakan ke φ (x) lalu kemudian menguji apakah nilai berikut ini adalah kurang atau lebih besar dari nol:

$$g(x) \equiv < w, \phi(x) > + w_0 = \sum_{i=1}^{N} \lambda_i y_i < \phi(x), \phi(x_i) > + w_0$$

$$= \sum_{i=1}^{N} \lambda_i y_i k(x, x_i) + w_0 \qquad (2.14)$$

Dari hubungan sebelumnya, terlihat jelas bahwa bentuk eksplisit dari fungsi pemetaan φ (·) tidak diperlukan, yang harus kita ketahui adalah fungsi kernel karena data muncul hanya dalam inner product. Amati bahwa fungsi diskriminan yang dihasilkan, g (x), adalah nonlinier karena ketidaklinieritasnya dari fungsi kernel.

Untuk menghasilkan sebuah pengklasifikasi SVM nonlinier, fungsi SMO2 MATLAB, dibahas di sesi 2.4, dan dapat digunakan. Alasan Kernel masukan mengambil nilai-nilai 'poli' untuk polinomial Kernel atau RBF untuk kernel RBF. Dalam kasus yang pertama, kpar1 dan kpar2 sesuai dengan β dan parameter n, masing-masing; dalam kasus terakhir, kpar1 sesuai dengan parameter σ.

Contoh 2.5.1

1. Untuk menghasilkan 2-dimensi sekumpulan data X1 (aturan pelatihan) sebagai berikut. Pilih N = 150 titik data dalam 2-dimensi [-5, 5] × [-5, 5] area sesuai dengan distribusi seragam (mengatur bobot untuk fungsi rand sama dengan 0). Menetapkan titik $x=[x(1), x(2)]^T$ untuk kelas 1 (-1) sesuai dengan aturan 0,05 ($x^3(1) + x^2(1) + x(1) + 1$) > (<) x(2). (Jelas, dua kelas nonlinear dipisahkan; pada kenyataannya dipisahkan oleh kurva yang terkait dengan persamaan 0,05 ($x^3(1) + x^2(1) + x(1) + 1$) > (<) x(2) Plot titik-titik dalam X1 . Untuk mendapatkan suatu data tambahan yang ditetapkan X2 (test set) menggunakan cara yang sama seperti untuk X1 (dengan mengatur bobot untuk fungsi rand sama dengan 100).
2. Rancanglah pengklasifikasi SVM linear dengan menggunakan modifikasi pertama Algoritma Platt dengan parameter C = 2 dan tol = 0,001. Hitung kesalahan dan uji pelatihan kemudian hitung jumlah vektor pendukungnya.
3. Untuk mendapatkan sebuah pengklasifikasi SVM nonlinier menggunakan fungsi basis radial kernel untuk nilai σ = 0,1 dan 2. Gunakan modifikasi pertama dari Algoritma Platt, dengan C = 2 dan tol = ,001. Hitunglah tingkat pelatihan dan pengujian kesalahan kemudian menghitung jumlah vektor pendukungnya. Plot area keputusan yang didefinisikan oleh pengklasifikasi tersebut.
4. Ulangi langkah ke 3 menggunakan fungsi polinomial kernel $(x^T y + \beta)^n$ untuk (n, β) = (5,0) dan (3,1). Kemudian tarik kesimpulannya.

5. Pengklasifikasi desain SVM menggunakan fungsi dasar kernel radial dengan σ=1.5 dan menggunakan fungsi kernel polinomial dengan n = 3 dan β = 1. Gunakan modifikasi pertama dari algoritma Platt dengan tol = 0,001 untuk C = 0.2 20 200.

Solusi

Ikutilah beberapa langkah berikut:

Langkah 1. Untuk menghasilkan kumpulan data X1 dan vektor y1 mengandung label kelas untuk vektor dalam nilai X1, ketikkan program berikut:

```
l=2; % Dimensionality
N=150; % Number of vectors
% Generating the training set
rand('see d', 0)
X1=10*ran d(l ,N )- 5;
for i=1:N
      t=0.05*( X1 (1 ,i) ^3 +X 1(1 ,i )^ 2+X 1( 1, i)+ 1) ;
      if(t>X1( 2, i) )
            y1(i)=1;
      else
            y1(i)=-1 ;
      end
end
```

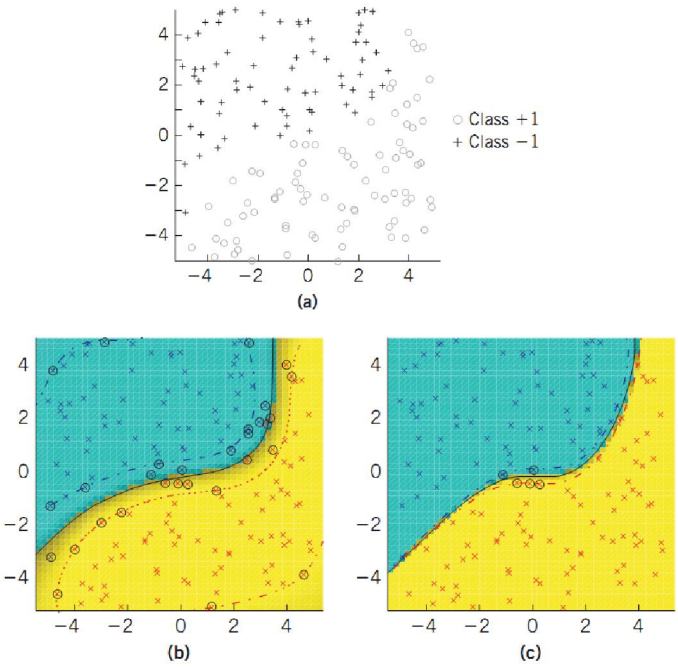
untuk plot kumpulan data X1 dapat dilihat pada gambar 2.5, ketikkan:

```
figure(1) , plot(X1( 1, y1 ==1 ), X1 (2, y1 == 1), 'r +' ,.. .
X1(1,y1== -1) ,X 1( 2,y 1= =- 1), 'b o' )
figure(1) , axis equal
```

Untuk menghasilkan nilai X2 sama seperti dalam kasus X1.

Langkah 2. Untuk menghasilkan pengklasifikasian SVM linier berdasar pada nilai X1 dengan C = 2 dan tol = 0.001, ketikkan program berikut:

```
kernel='linier';
kpar1=0;
```



Gambar 2.5 (a) Kumpulan Pelatihan untuk contoh 2.5.1 (b) Kurva keputusan yang terkait dengan pengklasifikasian menggunakan fungsi Kernel secara Radial (σ = 2) dan C = 2. (c) kurva Keputusan direalisasikan dengan pengklasifikasi menggunakan fungsi kernel polinomial (β = 1, n = 3) dan C = 2. Perhatikan bahwa hasil kernel yang berbeda di permukaan keputusan yang berbeda. Vektor pendukung adalah yang dilingkari. Garis – garis putus menunjukkan margin.

```
kpar2=0;
C=2;
tol=0.00 1;
steps=10 000 0;
eps=10^( -10 );
method=1 ;
[alpha, w0, w, evals, stp, glob] = SMO2(X1' , y1', ...
kernel, kpar1, kpar2, C, tol, steps, eps, method)
```

untuk menghitung kesalahan pelatihan, ketikkan:

```
Pe1=sum(( 2*( w* X1 -w0 >0 )- 1). *y 1< 0)/ le ng th( y1 )
```

Demikian pula menghitung kesalahan pengujian. Untuk menghitung jumlah vektor pendukung, ketikkan:

```
sup_vec=s um( al ph a>0 )
```

Langkah 3. Untuk menghasilkan sebuah pengklasifikasi SVM nonlinier menggunakan fungsi dasar kernel radial dengan σ = 0.1, bekerja seperti pada langkah 2, namun sekarang yang diatur adalah:

```
kernel='rbf' ;
kpar1=0.1 ;
kpar2=0;
```

Langkah yang sama untuk nilai σ yang lain. Untuk menghitung kesalahan pelatihan, proses adalah sebagai berikut:

- Vektor – vektor pendukung yang ditumpuk dalam matriks, sementara pengali Lagrange dan label kelasnya ditumpuk ke vektor. Kemudian ketikkan pseudocode berikut:

```
X_sup=X1( :,alpha '~= 0);
alpha_sup =alpha (alpha ~=0 )';
y_sup=y1( alpha ~=0 );
```

- Setiap vektor diklasifikasikan terpisah, ketikkan

```
for i=1:N
      t=sum((a lp ha _su p. *y _su p) .* ...
      CalcKern el (X _su p' ,X 1(: ,i )' ,ke rn el ,kp ar 1, kpa r2 )' )-w 0;
      if(t>0)
            out_train (i) =1 ;
      else
            out_train (i) =- 1;
      end
end
```

- Hitung kesalahan pelatihan seperti

```
Pe1=sum(o ut_ tr ai n.* y1 <0 )/l en gt h(y 1)
```

Kesalahan uji dihitung dengan cara yang sama. Untuk menghitung jumlah vektor pendukung, ketikkan:

```
sup_vec=s um( al ph a>0 )
```

daerah untuk merencanakan keputusan dibentuk oleh pengklasifikasi (lihat gambar 2.5(b)), ketikkan:

```
global figt4=3;
svcplot_b ook (X 1' ,y1 ', ke rne l, kp ar1 ,k pa r2, al ph a,- w0 )
```

Table 2.5 Results for the SVM Classifiers Designed in Steps 2, 3, and 4 of Example 2.5.1

	Training Error	Testing Error	No. Support Vectors
Linear	7.33%	7.33%	26
RBF (0.1)	0.00%	32.67%	150
RBF (2)	1.33%	3.33%	30
poly (5,0)	—	—	—
poly (3,1)	0.00%	2.67%	8

Note: *RBF(a) denotes the SVM classifier corresponding to the radial basis kernel function with $\sigma = a$; poly (n, β) denotes the SVM classifer with the polynomial kernel function of the form $(x^T y + \beta)^n$. The algorithm does not converge for the case poly(5,0).*

Langkah 4. Untuk menghasilkan pengklasifikasian SVM Nonlinier dengan fungsi Polinomial Kernel menggunakan n = 3 dan β = 1, sama halnya dengan cara sebelumnya namun yg diatur adalah:

```
kernel='poly';
kpar1=1;
kpar2=3;
```

Kesalahan pelatihan dan pengujian serta jumlah vektor pendukung dihitung seperti pada langkah sebelumnya (lihat juga Gambar 2.5 (c)). Demikian pula untuk kombinasi lain dari β dan n. Hasil yang diperoleh oleh SVM pengklasifikasi yang berbeda dirangkum pada Tabel 2 .5. Dari tabel ini, kesimpulan berikut dapat ditarik.

Pertama, kinerja pengklasifikasi linear lebih buruk daripada pengklasifikasi SVM Nonlinier. Ini diharapkan karena kelas yang berhubungan dengan masalah yang di tangani nonlinear dipisahkan. Kedua, pilihan untuk parameter fungsi kernel yang digunakan dalam pengklasifikasi SVM nonlinier secara signifikan mempengaruhi kinerja pengklasifikasian ini, parameter harus dipilih dengan hati-hati, setelah percobaan yang ekstensif (lihat, misalnya, RBF (0,1) dan mencoba RBF (5)). Akhirnya, kesalahan pelatihan yang kecil tidak selalu menjamin kesalahan pengujian yang kecil pula, perhatikan bahwa yang terakhir harus dipertimbangkan dalam mengevaluasi kinerja.

Langkah 5. Untuk menghasilkan pengklasifikasian SVM yang sesuai, sama seperti pekerjaan sebelumnya,n namun ubah nilai C menjadi 0.2 , 20 , 200.

Contoh 2.5.2

1. Untuk membangkitkan kumpulan data X1(pelatihan) seperti berikut. Perhatikan sembilan kotak [i, i + 1] × [j, j + 1], i = 0, 1, 2, j = 0, 1, 2 dan mengambil secara acak dari masing-masing 30 poin terdistribusi secara merata. Poin yang berasal dari kotak i + j bahkan (tidak datar) yang ditugaskan untuk kelas 1 (-1) (mengingatkan pada kotak putih dan hitam pada papan catur). Plot kumpulan data dan menghasilkan sebuah data tambahan yang ditetapkan X2 (pelatihan) berikut langkah yang digunakan untuk X1 (seperti pada Contoh 2.5.1, mengatur bobot untuk rand pada 0 untuk X1 dan 100 untuk X2).
2. (a) Desain pengklasifikasi SVM linear, dengan menggunakan modifikasi algoritma pertama Platt, dengan C = 200 dan tol = 0,001. Hitung kesalahan pelatihan dan uji dan menghitung jumlah vektor pendukungnya.

   (b) Menerapkan algoritma sebelumnya untuk rancangan pengklasifikasi SVM nonlinier, dengan fungsi kernel secara radial, untuk C, = 0.2 2, 20, 200, 2000, 20.000. Gunakan σ = 1, 1.5, 2, 5. Hitung kesalahan pelatihan dan uji dan menghitung jumlah vektor pendukung

   (c) Ulangi untuk fungsi polinomial kernel, menggunakan n = 3, 5 dan β = 1.
3. Tarik kesimpulan.

Solusi

Langkah 1, untuk membangkitkan kumpulan data X1, ketikkan:

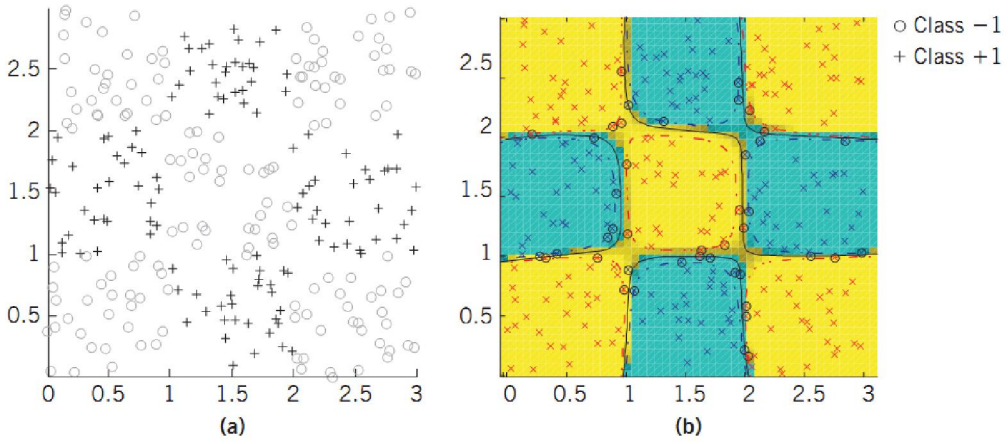
```
l=2; %Dimensionality
poi_per_square=30; %Points per square
N=9*poi_p ersquare; %Total no.of points
%Generating the training set
rand('seed', 0)
X1=[];
y1=[];
for i=0:2
      for j=0:2
            X1=[X1 rand(l,poi_per_square)+...
```

```
                [i j]'*ones (1,poi_per_square)];
                if(mod(i+j,2)==0)
                      y1=[y1 ones(1,poi_per_square)];
                else
                      y1=[y1 -ones(1,poi_per_square)];
                end
          end
    end
```

Untuk menggambarkan hasil X1 seperti pada Contoh 2.5.1 (lihat Gambar 2.6 (a)). Untuk menghasilkan X2, sama seperti pekerjaan dalam kasus X1.

Langkah 2. Untuk semua percobaan ini, hasilnya seperti pada contoh 2.5.1 (lihat juga gambar 2.6(b)).
Langkah 3. Dari hasil yang diperoleh, kesimpulan berikut dapat ditarik.

- Pertama, seperti yang diharapkan, pengklasifikasi linier kurang memadai untuk menangani masalah ini (pelatihan yang dihasilkan dan kesalahan uji lebih besar dari 40%). Hal yang sama berlaku untuk SVM dengan kernel polinomial (> 20% test kesalahan untuk semua kombinasi parameter). Hal ini berkaitan dengan sifat khusus dari contoh ini.
- Kedua, pengklasifikasi SVM dengan fungsi kernel secara radial memberikan hasil yang sangat baik untuk pilihan tertentu dari parameter σ. Pada Tabel 2 .6, hasil terbaik untuk pengklasifikasi dasar kernel SVM radial yang disajikan (semua dari mereka telah diperoleh untuk C = 2000). Dari tabel ini, sekali lagi dapat diverifikasi bahwa nilai-nilai yang sangat rendah mengakibatkan generalisasi σ sangat buruk (k pelatihan kesalahan, kesalahan tes tinggi). Penjelasan intuitif adalah bahwa sangat kecil σ menyebabkan k (x, xi) untuk drop dengan cepat menuju nol sekitar setiap xi X1, yang menyebabkan peningkatan jumlah vektor pendukung (karena banyak poin yang diperlukan untuk "menutupi" ruang mana yang terjadi ketidak validan data). Untuk setiap titik pelatihan, xi, penjumlahan dalam aturan klasifikasi dalam Pers. (2.14) sebagian besar dipengaruhi oleh istilah λi terkait yi k (xi, xi). Ini menjelaskan kesalahan pelatihan rendah. Sebaliknya, sejak k (x, xi) tidak cukup "menutupi" ruang jauh dari titik pelatihan, penjumlahan dalam Pers. (2.14) mungkin hampir nol untuk beberapa titik uji (dari kedua kelas) dan label masing-masing tidak dapat diprediksi secara akurat. Di sisi lain, nilai-nilai besar σ (dalam contoh kita σ = 5 adalah dianggap seperti itu) menyebabkan hasil yang buruk untuk kedua pelatihan dan set tes. Sebuah penjelasan intuitif untuk ini adalah bahwa, ketika σ adalah besar, semua k (x, xi) tetap hampir konstan di daerah di mana data poin berbohong. Hal ini juga menyebabkan peningkatan pendukung vektor (karena hampir semua titik sama pentingnya) dengan nilai yang hampir sama untuk λi. Jadi, penjumlahan dalam aturan klasifikasi dalam Pers. (2.14) pameran variasi rendah untuk berbagai nilai x (dari kedua pelatihan dan test set), yang mengarah ke kemampuan diskriminasi berkurang. Nilai antara dua ujung mengarah ke hasil yang lebih dapat diterima, dengan kinerja terbaik yang dicapai untuk σ = 1 dalam contoh ini. Pembahasan sebelumnya menjelaskan pentingnya memilih, untuk setiap masalah, nilai-nilai yang tepat untuk parameter yang terlibat.
- Ketiga, untuk parameter kernel tetap dan C bervariasi 0.2 hingga 20,000, jumlah SV (secara umum) menurun, seperti yang diharapkan.



Gambar 2.6 (a) Kumpulan pelatihan untuk contoh 2.5.2 (b) Kurva keputusan pengklasifikasi SVM, dengan fungsi Kernel secara radial (σ=1) dan C = 2000. Vektor pendukungnya adalah yang dilingkari, garis – garis putus adalah margin.

Table 2.6 Results for the SVM Classifiers Obtained for $\sigma = 0.1, 1, 1.5, 2, 5$ in Example 2.5.2

	Training Error	Test Error	No. Support Vectors
RBF(0.1)	0.00%	10.00%	216
RBF(1)	1.85%	3.70%	36
RBF(1.5)	5.56%	7.04%	74
RBF(2)	8.15%	8.52%	128
RBF(5)	35.56%	31.48%	216

Note: *RBF(a) denotes the SVM classifier with radial basis kernel functions with* $\sigma = a$.

2.6 Algoritma Perceptron Kernel

Dalam pemaparan sebelumnya, semua operasi dalam memperoleh pengklasifikasi SVM dapat dialihkan dalam bentuk inner product. Hal ini juga mungkin dilakukan dengan sejumlah algoritma lainnya. Algoritma Perceptron salah satu yang terkenal.

Untuk memanggil fungsi Algoritma Perceptron Kernel, ketikkan:

```
[a ,iter,count_misclas]=kernel_perce(X,y,kernel,kpar1,kpar2,max_iter)
```

Dimana,

`X` adalah matriks yang kolom-kolomnya adalah vektor data,
`y` adalah vektor yang mengandung label kelas dari titik data,
`kernel, kpar1, dan kpar2` didefinisikan sebagai dalam fungsi `SMO2`,
`max_iter` adalah jumlah maksimum iterasi algoritma yang dapat dilakukan,
`a` adalah sebuah vektor i^{th} adalah koordinat yang berisi jumlah titik i^{th} yaitu kesalahan klasifikasi
`iter` adalah jumlah iterasi yang dilakukan oleh algoritma,
`count_misclas` adalah jumlah titik kesalahan klasifikasi.

Sebuah vektor x tertentu diklasifikasikan ke kelas 1 atau kelas -1 tergantung apakah nilai berikut adalah positif atau negatif.

$$g(x) = \sum_{i=1}^{N} a_i y_i k(x, x_i) + \sum_{i=1}^{N} a_i y_i$$

Contoh 2.6.1

1. Perhatikan kumpulan data X1 (training set) dan X2 (test set) dari Contoh 2.5.1. Jalankan kernel perceptron algoritma menggunakan X1 sebagai pelatihan set dimana fungsi kernel adalah (a) linear, (b) secara radial fungsi dengan σ = 0.1, 1, 1.5, 2, 5, 10, 15, 20, dan (c) polinomial dari bentuk $(x^T y + \beta)^n$ untuk (n, β) = (3, 1), (5, 1), (15, 1), (3, 0), (5, 0). Untuk semua kasus ini tugasnya adalah menghitung kesalahan pelatihan dan tingkat pengujian kesalahan dan menghitung vektor pelatihan xi dengan i> 0 serta algoritma iterasi berjalan. Gunakan nilai 30.000 sebagai jumlah maksimum iterasi yang diijinkan.
2. Untuk tiap satu dari kasus-kasus sebelumnya, plot pada gambar yang sama training set X1 (gunakan warna yang berbeda dan simbol untuk setiap kelas) dan batas keputusan antara kelas.

Solusi

Langkah 1. Untuk menjalankan Algoritma Perceptron Kernel untuk Kernel Linier, ketikkan:

```
kernel='linear' ;
kpar1=0;
kpar2=0;
max_iter=30000;
[a,iter,count_misclas]=kernel_perce(X1,y1,kernel,...
kpar1,kpar2,max_iter);
```

dimana kpar1 dan kpar2 didefinisikan dalam fungsi MATLAB SMO2 tergantung pada jenis fungsi kernel yang dipertimbangkan.

Untuk menjalankan algoritma menggunakan fungsi kernel secara radial dengan σ = 0.1, ketikkan:

```
kernel='rbf';
kpar1=0.1;
kpar2=0;
max_iter=30000;
[a,iter,count_misclas]=kernel_perce(X1,y1,kernel,...
kpar1,kpar2,max_iter);
```

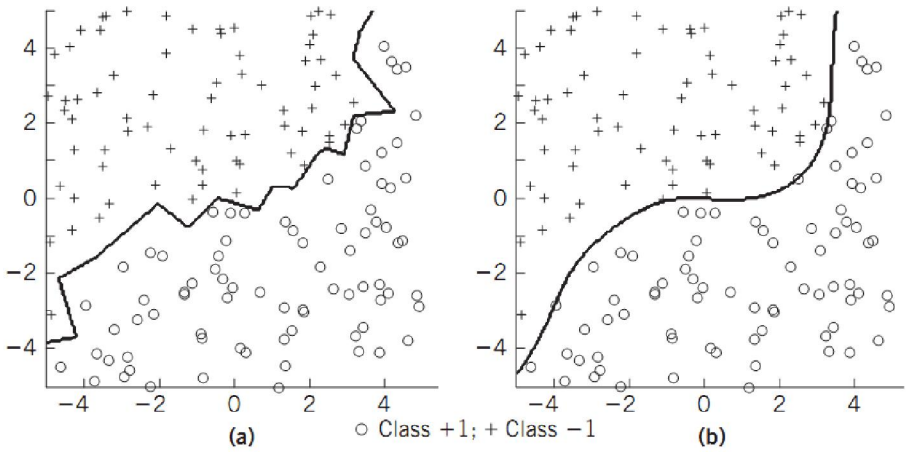
Untuk menjalankan algoritma menggunakan fungsi Polinomial Kernel dengan (n,β) = (3,1), ketikkan:

```
kernel='poly';kpar1=1;kpar2=3;
max_iter=30000;
[a,iter,count_misclas]=kernel_perce(X1,y1,kernel,...
kpar1,kpar2,max_iter);
```

Kasus-kasus lain dengan parameter yang berbeda diperlakukan sama.

Untuk menghitung kesalahan pelatihan, ketikkan:

```
for i=1:N
K=CalcKernel(X1',X1(:,i)',kernel,kpar1,kpar2)';
out_train(i)=sum((a.*y1).*K)+sum (a.*y1);
end
err_train=sum(out_train.*y1<0)/length(y1)
```



Gambar 2.7 Kumpulan Pelatihan untuk contoh soal 2.6.1 dan menghasilkan kurva keputusan yang berkaitan dengan Algoritma Perceptron Kernel menggunakan Fungsi Kernel basis radial dengan (a) σ = 0.1 dan (b) σ = 1.5

Dimana N adalah nomor vektor pelatihan. Untuk menghitung kesalahan pengujian, ketikkan:

```
for i=1:N
K=CalcKernel(X1',X2(:,i)',kernel,kpar1,kpar2)';
out_test(i)=sum((a.*y1).*K)+sum(a.*y1);
end
err_test=sum(out_test.*y2<0)/length(y2)
```

untuk menghitung jumlah pengklasifikasian yang hilang selama pelatihan, ketikkan:

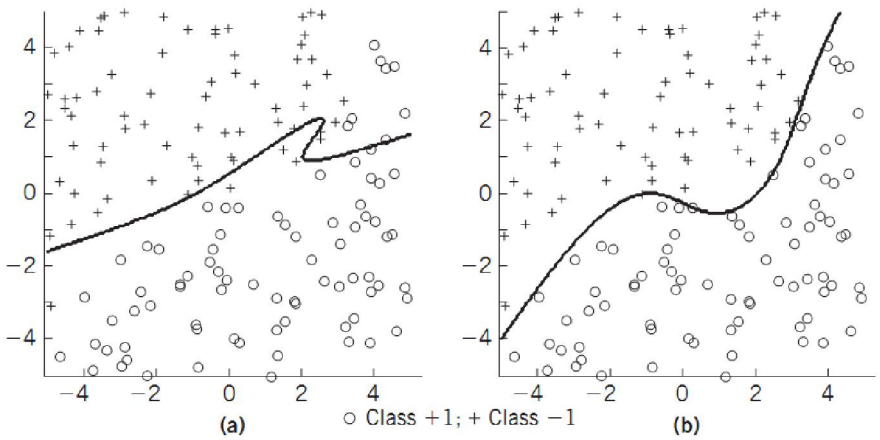
```
sum_pos_a =sum(a>0)
```

Langkah 2. Untuk menggambarkan Kumpulan Pelatihan (lihat gambar 2.7 dan 2.8), ketikkan:

```
figure(1),hold on
figure(1),plot(X1(1,y1==1),X1(2,y1==1),'ro',...
X1(1,y1==-1),X1(2,y1==-1),'b+')
figure(1),axis equal
```

Perhatikan bahwa vektor-vektor pelatihan ditetapkan dari kelas 1 (-1) yang ditandai dengan lingkaran (plus). Akhirnya, untuk merencanakan keputusan dalam batas angka yang sama, ketikkan:

```
bou_x=[-5 5];
bou_y=[-5 5];
resolu=.0 5;
fig_num=1 ;
plot_kernel_perce_reg(X1,y1,a,kernel,kpar1,kpar2,...
bou_x,bou_y,resolu,fig_num)
```



Gambar 2.8 (a) Pelatihan pada contoh 2.6.1 dan menghasilkan kurva keputusan yang berkaitan dengan Algoritma Kernel Perceptron menggunakan Fungsi Polinomial Kernel dengan (a) (n,β) = (3,0) dan (b) (n,β) = (3,1)

Dimana,

bou _x dan bou_y adalah vektor 2 – dimensi yang menentukan area (persegi) ruang di mana batas akan diambil, khususnya, (bou _x (1), bou _y (1)) mendefinisikan sudut kiri bawah area sementara (bou_x (2), bou_y (2)) mendefinisikan sudut kanan atas area tersebut,

resolu adalah resolusi dengan batas keputusan ditentukan (semakin rendah resolusi, gambar dari batas keputusan akan halus),

fig_num adalah jumlah gambar MATLAB dimana plot dari batas keputusan terjadi.

Hasil percobaan untuk fungsi dasar linier dan radial kernel disimpulkan dalam Tabel 2.7. Hasil percobaan untuk fungsi kernel polinomial disimpulkan dalam Tabel 2.8. Dari tabel dapat ditarik tiga kesimpulan.

- Pertama, untuk kernel linier, algoritma perceptron kernel tidak bertemu karena masalah tidak linear dipisahkan (berakhir setelah 30000 iterasi).
- Kedua, untuk fungsi kernel secara radial, algoritma perceptron kernel konvergen untuk berbagai nilai σ. Namun, sebagai nilai σ meningkat, algoritma iterasi kebutuhan lebih konvergen dan ketika σ menjadi sangat besar, algoritma gagal konvergen. Selain itu, garis keputusan yang diperoleh untuk kasus di mana σ = 0.1 agak "kasar" menunjukkan pengklasifikasi dengan kinerja generalisasi yang buruk (mengapa?). Nilai terbaik untuk σ tampaknya menjadi sekitar 1. Akhirnya, jumlah vektor dengan ai umumnya meningkat sebagai σ meningkat.
- Ketiga, untuk fungsi polinomial kernel, algoritma perceptron kernel tidak konvergen ketika parameter β = 0, tetapi tidak konvergen ketika β diset ke 1. Untuk β = 0 ruang dimensi tinggi (di mana pemetaan "yang tersirat" dilakukan) adalah dimensi yang lebih rendah dibandingkan dengan yang sesuai dengan β = 1, oleh karena itu berikut bahwa masalah tidak linear terpisah dalam ruang 4-dimensi didefinisikan oleh poly(3, 0), tetapi linear terpisah dalam ruang 9-dimensi didefinisikan oleh poly(3, 1)[3]

[3]The choice $n = 3, \beta = 0$ implies the mapping $[x(1), x(2)]^T \rightarrow [x^3(1), x^3(2), \sqrt{3}x^2(1)x(2), \sqrt{3}x(1)x^2(2)]^T$ (4-dimensional space); the choice $n = 3, \beta = 1$ implies the mapping $[x(1), x(2)]^T \rightarrow [x^3(1), x^3(2), \sqrt{3}x^2(1)x(2), \sqrt{3}x(1)x^2(2), \sqrt{3}x^2(1), \sqrt{3}x^2(2), \sqrt{6}x(1)x(2), \sqrt{3}x(1), \sqrt{3}x(2)]^T$ (9-dimensional space).

Remark

Seperti SVM, kita mengamati bahwa pilihan yang tepat dari fungsi kernel, serta pilihan parameter masing – masing, dilengkapi dengan eksperimen. Hal ini tergantung pada set data tertentu dan belum ada arahan ajaib (lihat Latihan 2.6.1).

Latihan 2.6.1

1. Perhatikan data X1 (training set) dan X2 (test set) dari Contoh 2.5.2. Jalankan algoritma perceptron menggunakan kernel X1 sebagai set pelatihan di mana fungsi kernel adalah (a) linear, (b) dasar radial dengan σ = 0.1, 0.5, 1, 1.5, 2, 5, dan (c) Polinomial dari bentuk $(x^T y + 1)^n$ untuk n = 3, 5, 15, 18, 20, 22. Untuk keseluruhan tiga kasus, hitunglah kesalahan pelatihan dan pengujian, jumlah misclassifications selama pelatihan dan jumlah iterasi algoritma dijalankan. Gunakan nilai 30000 sebagai jumlah maksimum iterasi yang diijinkan.
2. Untuk setiap plot kasus pada gambar pelatihan X1 yang sama, pengujian X2, dan batas keputusan antara dua kelas. Gunakan warna yang berbeda dan simbol untuk setiap kelas.

Petunjuk

Untuk melakukan eksperimen diperlukan, bekerja seperti pada contoh sebelumnya, sekarang mendefinisikan bou_x dan bou_y parameter sebagai:

```
bou_x=[0 3];
bou_y=[0 3];
```

Masalah ini lebih sulit dari yang dipertimbangkan dalam Contoh 2.6.1. Hasil menunjukkan bahwa baik linier dan polinomial kernel tidak dapat menyelesaikannya (dengan nilai-nilai tertentu untuk parameter). Fungsi Kernel Basis Radial (RBF) dapat memecahkan masalah pada kisaran nilai yang agak sempit untuk parameter σ.

2.7 Algoritma Adaboost

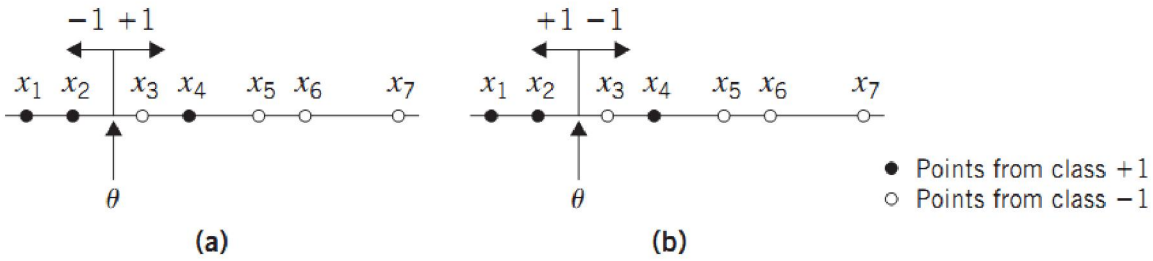
Algoritma lain adalah Algoritma Adaboost. Algoritma ini mengimplementasikan ide yang sangat menarik. Untuk memulai sebuah pengklasifikasi yang sangat sederhana dikenal sebagai base atau weak mulai diberlakukan. Secara sederhana, kita dapat mengartikan pengklasifikasian yang sedikit lebih baik daripada tebakan secara acak, yaitu menghasilkan tingkat kesalahan kurang dari 0.5 (untuk kasus 2 kelas). Algoritma Adaboost merupakan algoritma iteratif yang akan menghasilkan (akhir) dari pengklasifikasian yang didasarkan pada sejumlah pengklasifikasi dasar yang dirancang secara berurutan satu demi satu.

Rahasia dari sebuah Algoritma Adaboost adalah bahwa selama pelatihan dari proses pengklasifikasian dasar ke t (pada iterasi ke t), masing – masing pelatihan vektor xi berbobot wi, yang nilainya tergantung pada apakah nilai xi salah diklasifikasikan oleh pengklasifikasian dasar (t – 1), bobot wi dinaikkan atau diturunkan. Jadi vektor yang masih gagal menerima perhatian lebih dan (bobot) lebih. Pengklasifikasi akhir (didefinisikan setelah penghentian algoritma) diberikan sebagai bobot rata – rata dari semua pengklasifikasi dasar yang dirancang sebelumnya.

Ternyata skema tersebut konvergen menuju “kesalahan nol pada pelatihan”. Tingkat kesalahan pada set tes konvergen ke tingkat tertentu. Hal ini sangat menarik. Biasanya, pelatihan pengklasifikasi sampai kesalahan tercapai selama pelatihan diperoleh dalam hasil overfitting [Theo 09 Bagian 4,9].

Ini karena telah dinyatakan sebelumnya, pengklasifikasi bisa belajar banyak tentang "kespesifikan" dari set pelatihan tertentu dan cenderung menghasilkan tingkat kesalahan yang tinggi ketika menghadapi pengujian data, yang mana "yang tidak diketahui" untuk itu.

Asumsikan dimensi ruang fitur yang akan sama dengan l. Pertama, dimensinya katakanlah $j \leq l$, dipilih dan nilai – nilai minimum dan maksimum dari (dimensi l) dari himpunan vektor N pelatihan X sepanjang dimensi ke – j dihitung. Sejumlah niali θ, yang berfungsi sebagai ambang batas, secara acak dipilih antara dua nilai ekstrim dan vektor data dipisahkan menjadi dua kelas tergantung pada posisi koordinat ke – j nya, xi(j), i = 1, ..., N yang berhubungan dengan θ.



Gambar 2.9 Skenario pertama (a) dan skenario kedua (b) dari klasifikasi dasar diterapkan pada 7 titik data yang ditetapkan

Lebih khusus, berikut dua skenario yang dipertimbangkan: (a) vektor yang koordinat ke – j nya kurang dari θ ditugaskan ke kelas -1; (b) vektor yang koordinat ke – j nya kurang dari θ ditugaskan ke kelas +1 (jelas, ini kebalikan dari skenario pertama). Yang kedua, tingkat kesalahan klasifikasi e_a dan e_b dihitung (sebagai penjumlahan bobot dari semua vektor yang terkelompokan). Skenario yang menghasilkan tingkat klasifikasi yang lebih rendah diberlakukan untuk menentukan pengklasifikasi tersebut.

Untuk memahami lebih baik gagasan ini, perhatikan pengaturan dari Gambar 2.9, yang menunjukkan perkiraan dari tujuh titik dari data pelatihan diatur ke dimensi tertentu. Skenario (a) dan (b) diilustrasikan pada Gambar 2.9 (a) dan 2.9 (b), masing-masing. Pada Gambar 2.9 (a) kita mengamati bahwa enam vektor (semua kecuali x4) adalah kesalahan klasifikasi. Dengan asumsi bahwa bobot untuk semua titik data adalah sama, e_a = 6/7. Sebaliknya berlaku dalam Gambar 2.9 (b), di mana e_b = 1/7.

Klasifikasi dasar telah dijelaskan sebelumnya sepenuhnya ditetapkan oleh:

- Dimensi sepanjang berlangsungnya klasifikasi.
- Nilai ambang θ.
- Sebuah indeks yang mengambil nilai – nilai +1 atau -1 tergantung apakah skenario (a) atau (b) telah diseleksi. Ternyata klasifikasi yang dihasilkan umumnya memenuhi persyaratan untuk kesalahan klasifikasi kurang dari 0.5.

Contoh 2.7.1

Pertimbangkan tentang masalah klasifikasi 2 kelas 2 dimensi dimana kelas – kelas tersebut dijelaskan oleh pdf yang disajikan dalam contoh 1.6.2.

$m_{11} = [1.25, 1.25]^T$, $m_{12} = [2.75, 4.5]^T$, $m_{13} = [2, 11]^T$, $m_{21} = [2.75, 0]^T$, $m_{22} = [1.25, 2.75]^T$ $m_{23} = [4, 8]^T$.

Menghasilkan serangkaian data X yang terdiri dari titik N=100, seperti 50 bersumber dari kelas pertama dan 50 bersumber dari kelas kedua.

1. Jalankan algoritma AdaBoost pada kumpulan data X untuk menghasilkan sebuah "kuat" classifier sebagai deretan pengklasifikasi batang sederhana memiliki struktur digambarkan sebelumnya. Gunakan T_max = 3000, jumlah maksimum pengklasifikasi dasar (yaitu, iterasi).
2. Kelompokkan vektor – vektor dari sekumpulan pelatihan X menggunakan pengklasifikasi sebelumnya. Hitunglah kesalahan klasifikasi P bila hanya klasifikasi t dasar yang pertama yang diperhitungkan,, t = 1, ... , T_max. Gambarkan P dibandingkan dengan sejumlah pengklasifikasi dasarnya.
3. Hasilkan himpunan data pengujian Z menggunakan spesifikasi deretan nilai X. Kelompokkan vektor himpunan nilai Z menggunakan pengklasifikasi yang hasilnya didapat dari langkah sebelumnya. Hitunglah kesalahan klasifikasi P bila hanya klasifikasi t dasar yang pertama yang diperhitungkan, t = 1, ... , T_max. Gambarkan P dibandingkan dengan sejumlah klasifikasi dasar.
4. Amati gambaran yang dihasilkan dalam langkah 2 dan 3 lalu tarik kesimpulan.

Solusinya

Untuk menghasilkan himpunan data X dan Z, seperti pada contoh 1.6.2, ikuti langkah berikut:

Langkah 1. Gunakan fungsi boost_clas_coord pada MATLAB, dengan mengetikkan

```
[pos_tot,thres_tot,sleft_tot,a_tot,P_tot,K]=boost_clas_coord(X,y,T_max)
```

Dimana

`X adalah matriks l x N. Setiap kolom fitur vektor,`

`y adalah vektor dimensi N yang memiliki koordinat ke i merupakan label kelas (+1 atau -1) dari kelas di mana data vektor milik ke i,`

`T_max adalah jumlah klasifikasi dasar maksimal yang diijinkan,`

`Pos_tot adalah koordinat vektor ke – i yaitu nilai ambang dimensi yang dipilih untuk klasifikasi dasar ke – t,`

`Thres_tot adalah vektor koordinat ke t yang nilai ambang yang dipilih untuk klasifikasi dasar ke t,`

`Sleft_tot adalah vektor koordinat ke t ke variabel yang mengambil nilai +1 atau -1 berdasar pada apakah skenario (a) atau (b) yang dipilih pada klasifikasi dasar ke – t,`

`a_tot adalah vektor koordinat ke t yaitu bobot untuk klasifikasi dasar ke t,`

`P_tot adalah vektor koordinat ke t yaitu (bobot) probabilitas kesalahan klasifikasi untuk klasifikasi dasar ke t,`

`K adalah sejumlah hasil klasifikasi dasar oleh algoritma.`

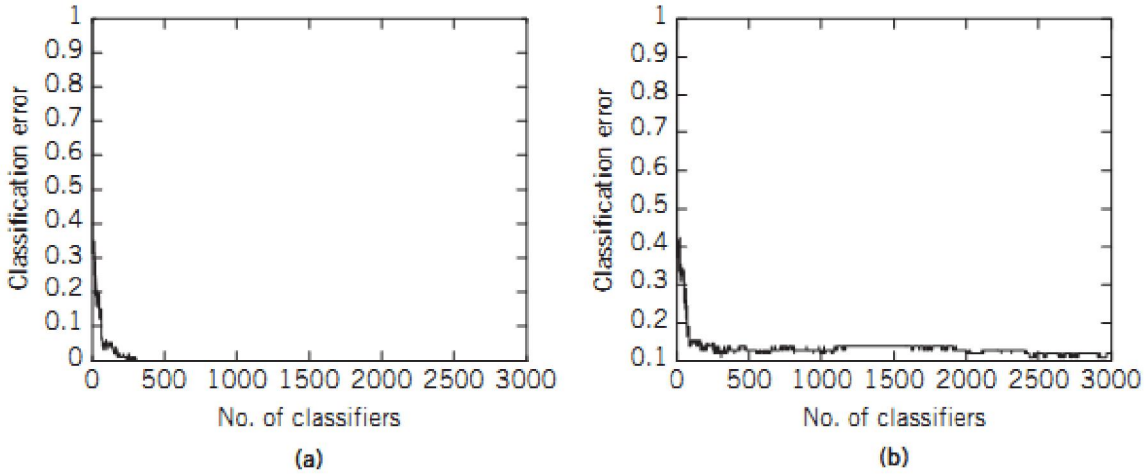
Langkah 2. Gunakan fungsi boost_clas_coord pada MATLAB, dengan mengetikkan

```
[y_out,P_error]=boost_clas_coord_out(pos_tot,thres_tot,sleft_tot,a_tot,P_tot,K,X,y)
```

Untuk menggambarkan kesalahan klasifikasi pada X dibandingkan dengan sejumlah pengklasifikasi dasar (lihat gambar 2.10(a)), ketikkan

```
figure(3), plot(P_error)
```

Langkah 3. Seperti langkah 2, ganti nilai X dengan Z (lihat gambar 2.10(b))



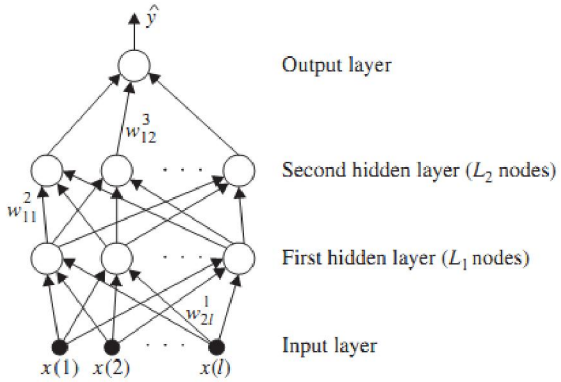
Gambar 2.10 Hasil dari contoh 2.7.1: (a) kesalahan klasifikasi untuk himpunan pelatihan dibanding dengan sejumlah pengklasifikasi dasar; (b) kesalahan klasifikasi untuk himpunan pengujian dibanding sejumlah pengklasifikasi dasar.

Langkah 4. Untuk pelatihan himpunan X, kesalahan klasifikasi cenderung menuju 0 sebagai meningkatnya pengklasifikasi dasar (lihat gambar 2.10(a)). Untuk pengujian himpunan Z, kesalahan klasifikasi cenderung menuju batas positif sebagai meningkatnya jumlah pengklasifikasi dasar.

2.8 Multilayer Perceptron

Algoritma Perceptron dan terkait dengan "arsitektur" elemen dasar perceptron dibahas pada sesi 2.2. Perceptron dapat dianggap sebagai upaya untuk memodelkan elemen bangunan dasar, neuron, otak manusia. Setiap neuron sangat bersemangat dengan menerima sinyal input x(1), x(2), ... , x(l). Kemudian setiap masukan ditimbang dengan bobot w1, w2, ..., wl, yang juga dikenal sebagai bobot synaptic, dianalogikan dengan terminologi yang digunakan dalam ilmu saraf. Jumlah bobot kemudian hilang melalui fungsi aktivasi, dan jika nilainya lebih tinggi dari pada nilai threshold (-w0) jaringannya "fires" Artinya, itu memberikan nilai output, jika tidak tetap tidak aktif.

Seperti pada gambar 2.11 merupakan Perceptron Multilayer dengan 2 layer yang tersembunyi dan node tunggal pada layer keluarannya. Bobot yang menghubungkan titik i dengan j layer k dinotasikan w_{ji}^{k}.



Gambar 2.11 Perceptron Multilayer dengan 2 layer yang tersembunyi dan node tunggal pada layer keluaran

Tujuan dari pelatihan Perceptron Multilayer adalah untuk memperkirakan bobot, serta nilai-nilai ambang batas, dari semua neuron yang terkait dalam jaringan. Untuk tujuan ini, fungsi harga dipilih. Pilihan yang paling populer adalah fungsi kehilangan keuadrat paling kecil. Tujuan dari perhitungan dalam Perceptron Multilayer adalah mencari bobot yang tidak diketahui sampai nilainya minimum,

$$J = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \qquad (2.15)$$

Skema algoritma untuk melakukan minimisasi sebelumnya adalah berulang-ulang dan secara luas dikenal sebagai Algoritma Backpropagation (BP). Algoritma dimulai dari beberapa nilai awal bebas untuk semua parameter yang tidak diketahui, dan konvergen ke minimum lokal dari fungsi harga dalam Persamaan (2.15). Secara umum, fungsi harga memiliki sejumlah minima lokal. Dengan demikian, pilihan nilai awal mempengaruhi solusi yang diperoleh dengan algoritma. Dalam prakteknya, algoritma berjalan beberapa kali, mulai dari nilai awal yang berbeda, bobot sesuai dengan solusi terbaik dipilih. (Lebih lanjut tentang sifat-sifat Perceptrons backpropagation multilayer dan dapat ditemukan di [Theo 09, Bab 4].)

Berikut cara pemahaman tentang arti dari parameter yang terkait dengan kode MATLAB. Ini milik anggota keluarga turunan algoritma gradien, dan langkah iterasi dasar adalah dalam bentuk:

$$w(new) = w(old) + \Delta w \qquad (2.16)$$

new & old merupakan perkiraan dari langkah sebelumnya dan saat ini. Jangka waktu koreksi terkait dengan harga gradien, dihitung pada w(old):

$$\Delta w = -\mu \frac{\partial J}{\partial w} \qquad (2.17)$$

Dan w mengacu pada parameter bobot (termasuk batas ambang) jaringan neuron.

ebuah perilaku yang lebih baik dan populer dari algoritma backpropagation adalah yang disebut dengan istilah momentum. Versi ini membutuhkan parameter tambahan, α, yang dikenal sebagai istilah momentum (biasanya antara 0.1 dan 0.8), yang mengontrol koreksi, sekarang menjadi

$$\Delta w(new) = \alpha \Delta w(old) - \mu \frac{\partial J}{\partial w}$$

Dalam penyelesaiannya menggunakan MATLAB, untuk membangkitkan Perceptron Multilayer dapat menggunakan fungsi NN_trainning dengan mengetikkan,

```
[net,tr] = NN_training(X,y,k,code,iter,par_vec)
```

`X adalah vektor pelatihan kolom,`

`Y adalah vektor yang berisi label kelas untuk vektor data,`

`Code menentukan algoritma pelatihan yang digunakan (1 untuk BP, 2 untuk BP dengan syarat momentum dan 3 untuk BP dengan pesat pembelajaran adaptif)`

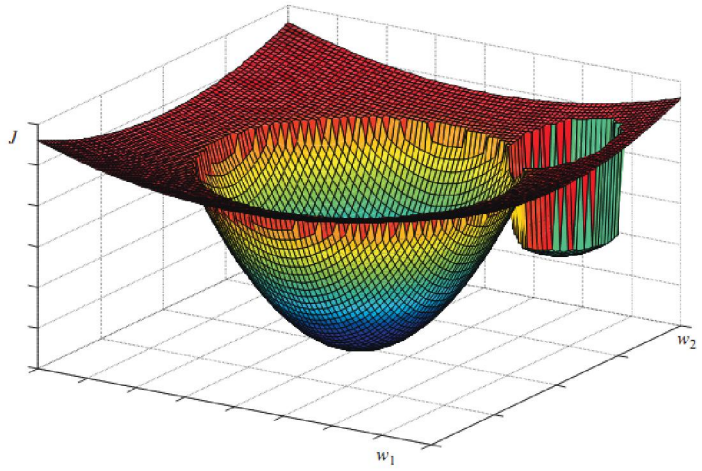
`iter adalah jumlah iterasi yang akan dilakukan oleh algoritma`

`par_vec adalah vektor 5 dimensi yang mengandung nilai – nilai (a) pesat pembelajaran yang digunakan dalam standar algoritma BP, (b) istilah yang digunakan dalam BP dengan syarat momentum dan (c) 3 nilai yang terkait dengan BP dalam pesat pembelajaran adaptif,`

net adalah struktur jaringan yang dikembalikan, yang mengikuti struktur pemrograman yang digunakan oleh MATLAB,

tr adalah struktur yang berisi antara lain jumlah, kinerja jaringan selama pelatihan sehubungan dengan jumlah epoch.

Pada gambar 2.12 berikut menunjukkan fungsi harga 2 dimensi yang terdiri dari 2 minima, satu yang luas dan satu yang sempit.



Gambar 2.12 Bentangan fungsi 2 dimensi yang mencakup luasan dan minima yang sempit

Contoh 2.8.1

Berikut adalah contoh penyelesaian masalah jaringan syaraf,

1. Terdapat sejumlah data X1 pada himpunan pelatihan yang berisi 60 titik dari kelas +1 (kira – kira 20 dari setiap distribusinya) dan 80 titik dari kelas -1 (perkiraan 20 titik dari setiap distribusinya). Gunakan langkah yang sama untuk membangkitkan himpunan X2.
2. Berdasar pada X1, latihlah 2 layer jaringan syaraf feedforward (FNN) dengan dua atau 4 titik dalam layer yang tersembunyi. Semua titik layer yang tersembunyi menggunakan hyperbolic tangent (tanh) sebagai fungsi aktivasi, selama titik keluaran menggunakan fungsi aktivasi linier. Jalankan Algoritma BP standar untuk 9000 iterasi dengan pesat pembelajaran 0.01 . hitunglah kesalahan pelatihan dan pengujian (pada X1 dan X2). Lalu gambarkan titik – titik pelatihan serta daerah keputusan yang dibentuk oleh masing – masing jaringan. Juga gambarkan kesalahan pelatihan dibanding sejumlah iterasi.
3. Ulangi langkah ke2 dengan pesat pembelajaran 0.0001 dalam Algoritma standar Backpropagation.
4. Ulangi langkah ke 2, terapkan Algoritma BP Adaptif untuk 6000 iterasi, dengan pesat pembelajaran 0.00001 dan ri = 1.05, rd = 0.7 dan c = 1.04.
5. Berikan keterangan atas hasil yang diperoleh pada langkah 2, 3 dan 4.

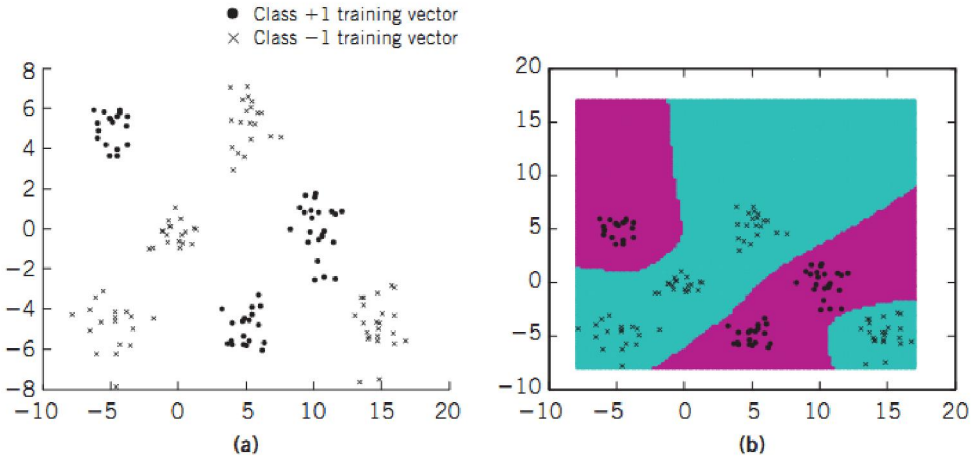
Solusinya,

```
randn('seed' ,0 )
%Parameter definition
```

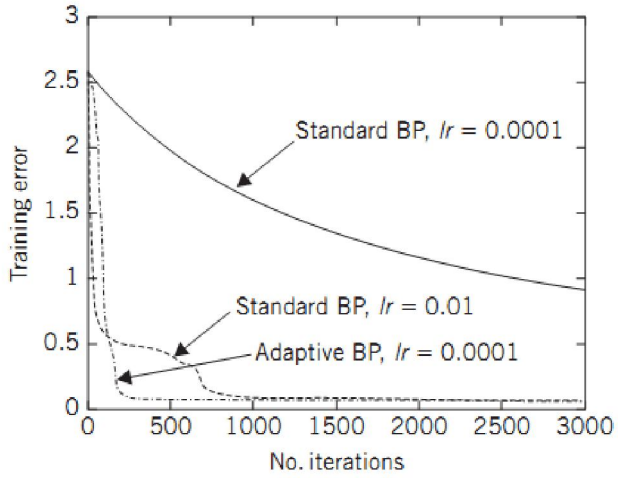
```
l=2; % Dimensionality
m1=[-5 5; 5 -5; 10 0]'; %Means of Gaussians
m2=[-5 -5; 0 0; 5 5; 15 -5]';
[l,c1]=si ze( m1 ); %no of gaussians per class
[l,c2]=si ze( m2 );
P1=ones(1 ,c1 )/ c1 ; %Probabilities of the gaussians per class
P2=ones(1 ,c2 )/ c2 ;
s=1; %variance
%%%%%%%%%%%%%%%%%%
% Generation of training data from the first class
N1=60; %Number of first class data points
for i=1:c1
S1(:,:,i) =s*eye( l) ;
end
sed=0; %Random generato r seed
[X1,y1]=mixt_model(m1,S1,P1,N1,sed) ;
%%%%%%%%%%%%%%%%%%
% Generation of training data from the second class
N2=80; %Number of second class data points
for i=1:c2
S2(:,:,i )= s*eye (l );
end
sed=0; %Random generator seed
[X2,y2]= mixt_model (m2, S2, P2 ,N2,sed );
%%%%%%%% %%% %% %% %%
%Production of the unified data set
X1=[X1 X2]; %Data vectors
y1=[ones (1,N1) -ones(1,N2)] ; %Class labels
figure(1 0), hold on
figure(1 0), plot(X1( 1, y1==1 ),X1(2,y1==1), 'r .', .. .
X1(1,y1= =-1), X1 (2, y1==-1),'bx')
rand('seed', 100) %Random generators initialization
randn('seed' ,100 )
iter=9000 ; %Number of iterations
code=1; %Code for the chosen training algorithm
k=2; %number of hidden layer nodes
lr=.01; %learning rate
par_vec=[ lr 0 0 0 0];
```

```
[net,tr]= NN_training( X1 ,y1 ,k ,code ,iter,par_vec) ;
pe_train=NN_evaluation( net ,X 1, y1)
pe_test=NN_evaluation (net, X2 ,y2)
maxi=max( max ([ X1 '; X2']));
mini=min( min ([ X1 '; X2']));
bou=[mini maxi];
fig_num=1 ; %Number of figure
resolu=(bou(2)- bou(1))/ 100 ; %Resoluti on of figure
plot_NN_reg( net, bou,resolu,fig_num) %Decision region plot
figure(fig_num ), hold on %Plotting training set
figure(fig_num ), plot(X1(1 ,y 1= =1) ,X 1( 2,y 1= =1 ),' r. ', ...
X1(1,y1==-1) ,X 1( 2,y1==- 1),'bx' )
figure(11 ), plot(tr.perf)

iter=6000 ; %Number of iterations
code=3; %Code for the chosen training algorithm
k=2; %number of hidden layer nodes
lr=.0001; %learning rate
par_vec=[ lr 0 1.05 0.7 1.04]; %Parameter vector
```



Gambar 2.13 (a) Himpunan Pelatihan pada contoh 2.8.1. (b) area keputusan yang dibentuk oleh FNN 2 layer dengan empat titik layer yang tersembunyi dilatih dengan Algoritma standar BP dengan pesat pembelajaran 0.01 dan 9000 iterasi.



Gambar 2.14 Plot kesalahan pelatihan dibanding dengan sejumlah iterasi untuk BP standar dengan lr = 0.01 (garis putus – putus), BP Standar dengan lr = 0.00001 (garis padat), dan BP adaptif dengan lr = 0.00001 (garis titik putus – putus) untuk contoh 2.8.1

Table 2.9 Results for the FNNs Trained by the Standard BP Algorithm, with Learning Rate 0.01 and 9000 Iterations, from Example 2.8.1

	Two Nodes	Four Nodes
Training error	29.29%	0
Test error	30.71%	0

Note: *When fewer than the minimum required hidden-layer nodes are employed, the resulting FNN is unable to solve the problem.*

Table 2.10 Results for the FNNs Trained by the Adaptive BP Algorithm, with Learning Rate 0.0001 and 6000 Iterations, from Example 2.8.1

	Two Nodes	Four Nodes
Training error	29.29%	0
Test error	32.14%	0

Seperti pada kasus – kasus sebelumnya, nilai k harus diubah menjadi 4 guna merancang sebuah FNN 2 lapis dengan empat titik layer yang tersembunyi. Hasilnya dirangkum dalam Tabel 2.1.0.

Dari percobaan sebelumnya berikut dapat ditarik kesimpulan. Pertama, jumlah node yang tersembunyi secara langsung mempengaruhi kemampuan belajar dari jaringan saraf. Percobaan menunjukkan bahwa FNN dengan dua titik lapisan tersembunyi tidak mampu untuk belajar masalah klasifikasi dipertimbangkan dalam contoh ini. Sebaliknya, sebuah FNN dengan empat lapisan tersembunyi node memiliki node yang cukup untuk mewujudkan kurva keputusan diperlukan (lihat Tabel 2.9 dan 2.10 serta daerah keputusan dibentuk oleh FNN masing-masing).

Kedua, tingkat belajar mempengaruhi kecepatan konvergensi dari algoritma backpropagation. Kecil nilai tukar pembelajaran dapat memperlambat kecepatan konvergensi dari algoritma BP (lihat Gambar 2.14).

Ketiga, bahkan ketika itu dimulai dengan nilai tingkat belajar yang kecil, algoritma adaptif BP menunjukkan konvergensi dengan sangat cepat dibandingkan dengan algoritma BP standar karena menyesuaikan nilai pada setiap iterasi untuk "kecocokan" daerah bentangan (lihat Gambar 2.14). Harga untuk ini perilaku yang diinginkan adalah beberapa pemeriksaan tambahan selama eksekusi algoritma untuk menyesuaikan tingkat belajar.

Remark

Ketika dimensi data yang lebih tinggi dari 3 (yang sering terjadi), tidak ada visualisasi data adalah memungkinkan. Dalam kasus tersebut, jumlah node lapisan tersembunyi yang dipilih biasanya merupakan hasil dari eksperimen yang luas.

Contoh 2.8.2

Pertimbangan pada masalah klasifikasi 2 kelas 2 dimensi. Titik dari kelas pertama(kedua) dinotasikan +1 -1, batangan untuk empat (lima) distribusi Gausian dalam arti $[-10, 0]^T$, $[0, -10]^T$, $[10, 0]^T$, $[0, 10]^T$, $[-10, -10]^T$, $[0, 0]^T$, $[10, -10]^T$, $[-10, 10]^T$, $[10, 10]^T$ dengan probabilitas yang sama. Matriks kovarian dari masing – masing distribusi adalah $\sigma^2 I$, dimana $\sigma^2 = 4$ dan I adalah matriks identitas 2x2.

1. Untuk menghasilkan dan menggambarkan himpunan data X1 (himpunan pelatihan) berisi 80 titik dari kelas +1 (20 titik masing – masing distribusi) dan 100 titik dari kelas -1 (20 titik dari masing – masing distribusi). Gunakan langkah yang sama untuk menghasilkan himpunan data X2 (himpunan pelatihan).
2. Berdasarkan pada nilai X1, latihlah tiga layer 2 jaringan syaraf feedforward (FNN) dengan tiga, empat dan 10 titik pada layer yang tersembunyi. Fungsi aktivasi titik layer tersembunyi adalah hyperbolic tangent (tanh); titik keluarannya mempunyai fungsi aktivasi linier. Gunakan Algoritma BP Adaptif untuk 10000 iterasi dengan pesat pembelajaran 0.01 (untuk sisa dari parameter algoritma, gunakan nilai pada contoh 2.8.1). Hitunglah kesalahan pelatihan dan pengujian (X1 dan X2) dan gambarkan titik data serta area keputusan yang dibentuk pada masing – masing jaringan. Tarik keseimpulannya.
3. Rancanglah FNN 2 layer dengan 10 titik layer tersembunyi dan gunakan algoritma BP Adaptif untuk melatihnya pada X1. Hitunglah kesalahan pesat pelatihan dan kesalahan setelah iterasi 300, 400, 1000, 2000, 3000, 4000, 5000 dan 10000. Ubah parameter pesat pembelajaran menjadi 0.01 dan parameter lain hingga sama seperti nilai yang diberikan pada contoh 2.8.1 kemudian tarik kesimpulannya.

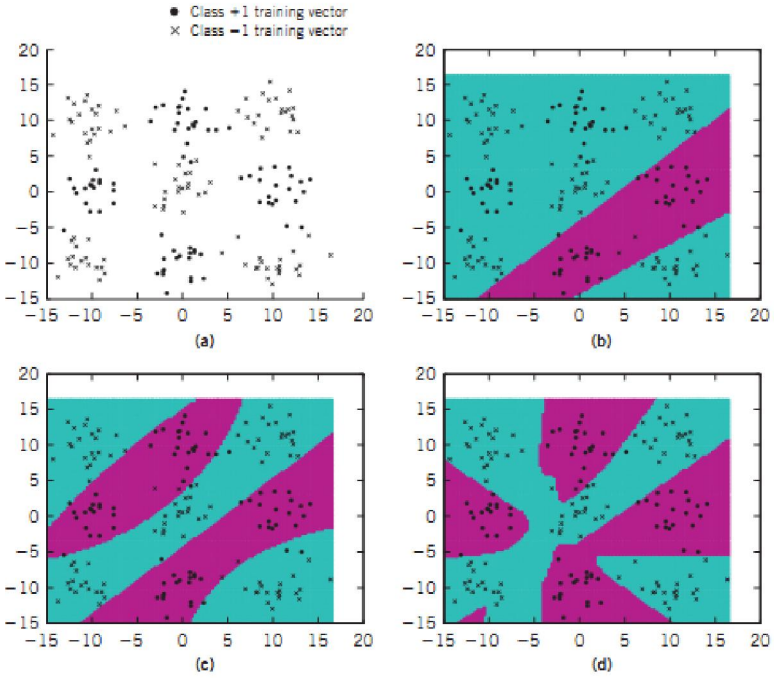
Solusinya,

Langkah 1. Tulislah nilai X1 dan X2 pada contoh 2.8.1

Langkah 2. Percobaan yang dibutuhkan dapat dilakukan langkah kerjanya seperti pada langkah 2 dari Contoh 2.8.1. Ubah data dan daerah keputusan yang terbentuk oleh tiga FNN sebelumnya yang ditampilkan pada gambar 2.16. Hasil klasifikasi ditunjukkan dalam tabel 2.11.

Langkah 3. Dari hasil yang disajikan dalam Tabel 2.12, mengamati bahwa perkiraan dari 500 – 3000 iterasi pengujian kesalahan (yang merupakan indikasi dari kemampuan generalisasi dari jaringan)tersisa 3,89%. Untuk iterasi selebihnya, kesalahan pelatihan selanjutnya menurun sedangkan kesalahan uji meningkat dengan nilai lebih tinggi dari 3,89%. Hal ini menunjukkan bahwa, dalam kasus ini, lebih dari

(sekitar) 3000 iterasi mengarah ke overtraining jaringan. Artinya, jaringan mulai fokus pada "kekhususan" dari X1, yang mengarah pada kinerja generalisasi terdegradasi. Ini merupakan isu penting dalam pelatihan FNN. Salah satu cara untuk mendeteksi overtraining adalah untuk menerapkan himpunan data (satu set validasi) yang berbeda dari X1, dan secara berkala mengukur pada itu kinerja FNN yang diperoleh selama pelatihan. Jika kesalahan yang dihasilkan dari serangkaian data yang mulai meningkat, sementara kesalahan pelatihan menurun, pelatihan berhenti.



Gambar 2.16 (a) Himpunan pelatihan pada contoh 2.8.2 (b) area keputusan yang dibentuk oleh FNN 2 layer dengan 3, 4 dan 10 titik layer tersembunyi, masing – masing dilatih dengan Algoritma BP Adaptif dengan pesat pembelajaran 0.01 dan 10000 iterasi

Table 2.11 Results for the FNNs Trained by the Adaptive BP Algorithm, with Learning Rate 0.01 and 10,000 Iterations from Example 2.8.2

	Three Nodes	Four Nodes	Ten Nodes
Training error	25.56%	6.67%	1.67%
Test error	30.00%	8.33%	7.22%

Note: *The other parameters of the algorithm are specified as in Example 2.8.1.*

Table 2.12 Results for a Two-Layer FNN with 10 Hidden-Layer Nodes Trained by the Adaptive BP Algorithm, with Learning Rate 0.01 and for Various Numbers of Iterations from Example 2.8.2

	No. Iterations							
	300	500	1000	2000	3000	4000	5000	10,000
Training error	3.89%	2.22%	2.22%	2.22%	2.22%	1.67%	1.67%	1.67%
Test error	4.44%	3.89%	3.89%	3.89%	3.89%	5.56%	6.11%	7.22%

Latihan 2.8.2

Pertimbangan pada masalah klasifikasi 2 kelas 2 dimensi. Titik dari kelas pertama(kedua) dinotasikan +1 -1, batangan untuk delapan distribusi Gausian dalam arti $[-10, 0]^T$, $[0, -10]^T$, $[10, 0]^T$, $[0, 10]^T$, $[-10, -20]^T$, $[10, 20]^T$, $[20, 10]^T$, $[20, -10]^T$, $[-10, -10]^T$, $[0, 0]^T$, $[10, -10]^T$, $[-10, 10]^T$, $[10, 10]^T$, $[20, 20]^T$, $[20, 0]^T$, $[0, 20]^T$ dengan probabilitas yang sama. Matriks kovarian dari masing – masing distribusi adalah $\sigma^2 I$, dimana $\sigma^2 = 4$ dan I adalah matriks identitas 2x2.

1. Untuk menghasilkan dan menggambarkan himpunan data X1 (himpunan pelatihan) berisi 160 titik dari kelas +1 (20 titik masing – masing distribusi) dan 160 titik dari kelas -1 (20 titik dari masing – masing distribusi). Gunakan langkah yang sama untuk menghasilkan himpunan data X2 (himpunan pelatihan).
2. Jalankan Algoritma BP Adaptif dengan pesat pembelajaran 0.01 dan 10000 iterasi untuk melatih FNN 2layer dengan 7, 8, 10, 14, 16, 20, 32 dan 40 titik layer yang tersembunyi. Nilai parameter untuk Algoritma ini diambil seperti pada contoh 2.8.1.
3. Ulangi langkah ke 2 untuk $\sigma^2 = 2, 3, 4$ dan tarik kesimpulannya.