

**Teknik Soft-Computing  
Materi Training PENS-ITS  
7-8 November 2006**

Oleh

Budi Santosa PhD  
Teknik Industri, ITS, Surabaya

©Copyright by Budi Santosa 2006  
All Rights Reserved

---

## Daftar Isi

---

<b>1</b>	<b>Pendahuluan</b>	<b>1</b>
<b>2</b>	<b>Beberapa Konsep Dasar</b>	<b>4</b>
2.1	Pengenalan Pola, Data Mining dan Machine Learning . . . . .	4
2.2	Variabel, Fitur dan Atribut . . . . .	6
2.3	Supervised dan Unsupervised Learning . . . . .	6
2.4	Jenis Nilai Variabel . . . . .	7
2.5	Klasifikasi dan Pendekatan Fungsi (Regresi) . . . . .	8
2.6	Klasifikasi Dua Kelas dan Multi Kelas . . . . .	9
2.7	Transformasi Data . . . . .	10
2.8	Konsep Jarak . . . . .	12
2.9	Programa Kuadratik (Quadratic Programming) . . . . .	14
2.10	Lagrange Multiplier . . . . .	14
<b>3</b>	<b>Teknik Klastering</b>	<b>17</b>
3.1	Pendahuluan . . . . .	17
3.2	Klastering Hirarki (Hierarchical Clustering) . . . . .	18
3.2.1	Kemiripan dan Ketidakmiripan . . . . .	18

3.3	K-means . . . . .	25
3.4	Fuzzy K-means . . . . .	26
3.5	Implementasi Klastering dengan Matlab . . . . .	27
<b>4</b>	<b>Analisis Diskriminan</b>	<b>37</b>
4.1	Pendahuluan . . . . .	37
4.2	Ide Dasar LDA Dua kelas . . . . .	37
4.3	Implementasi . . . . .	40
4.4	Implementasi Analisis Diskriminan Dengan Matlab	44
<b>5</b>	<b>Artificial Neural Networks</b>	<b>49</b>
5.1	Ide Dasar . . . . .	49
5.2	Model Komputasi untuk Neuron . . . . .	49
5.3	Model Neuron . . . . .	50
5.4	Macam-macam Fungsi Aktivasi . . . . .	52
5.5	Single-Layer Perceptrons . . . . .	54
5.6	Prosedur Learning . . . . .	54
5.6.1	Perceptron . . . . .	55
5.6.2	Gradient Descent . . . . .	58
5.6.3	Metoda Newton . . . . .	61
5.7	Algoritma Back-Propagasi . . . . .	62
5.7.1	Kasus 1, prosedur training untuk unit output	63
5.7.2	Kasus 2, prosedur training untuk unit hidden	64
5.8	Implementasi ANN dengan Matlab . . . . .	69
<b>6</b>	<b>Support Vector Machines</b>	<b>78</b>
6.1	Ide Dasar Support Vector Machine . . . . .	78
6.2	Formulasi Matematis . . . . .	80
6.3	Metoda Kernel . . . . .	83
6.4	Implementasi . . . . .	85
6.5	Implementasi SVM dengan Matlab . . . . .	90

<b>7</b>	<b>Support Vector Machines untuk Multi-Kelas</b>	<b>91</b>
7.1	Ide Dasar . . . . .	91
7.1.1	Metoda Satu-lawan-Semua (SLA) . . . . .	91
7.1.2	Metoda Satu-lawan-Satu (SLU) . . . . .	93
<b>8</b>	<b>Support Vector Regresi</b>	<b>97</b>
8.1	Pendahuluan . . . . .	97
8.2	Ide Dasar . . . . .	97
8.3	Formulasi SVR dalam QP Standar . . . . .	100
8.4	Loss Function . . . . .	103
8.5	Implementasi Support Vector Regression Dengan Matlab . . . . .	105
8.6	Pemilihan Metoda Prediksi . . . . .	110

# BAB 1

---

## Pendahuluan

---

1. Apa yang bisa kita lakukan terhadap data?
2. Kita perlu mengolahnya untuk mendapatkan manfaat dari data itu.
3. Kita perlu mengenali polanya sehingga akan kita temukan kecenderungan tertentu dari data tersebut.
4. Kita juga bisa melakukan pekerjaan prediksi
5. Pekerjaan-pekerjaan seperti ini dalam dunia ilmiah sering disebut dengan *pattern recognition* atau pengenalan pola.

Contoh-contoh berikut ini memperlihatkan masalah-masalah dalam *data mining*:

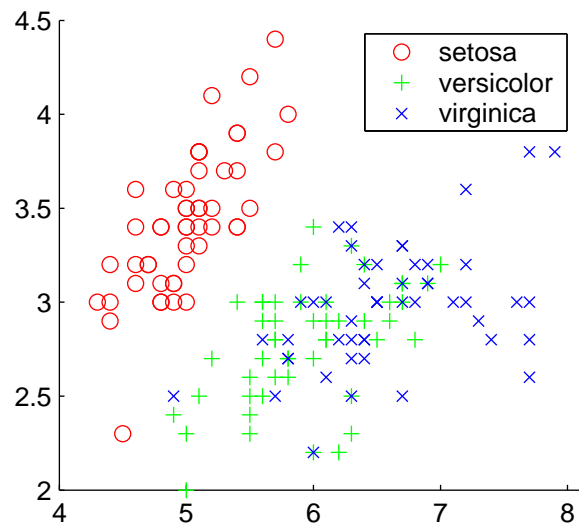
1. Memprediksi harga suatu saham dalam beberapa bulan ke depan berdasarkan performansi perusahaan dan data-data ekonomi
2. Memprediksi apakah seorang pasien yang diopname akan mendapat serangan jantung berikutnya berdasarkan catatan kesehatan sebelumnya dan pola makanannya

3. Memprediksi permintaan semen dalam beberapa tahun mendatang berdasarkan data permintaan semen di tahun-tahun sebelumnya
4. Memprediksi apakah akan terjadi tornado berdasarkan informasi dari sebuah radar tentang kondisi angin dan kondisi atmosfer yang lain
5. Identifikasi apakah sudah terjadi penipuan terhadap pengguna kartu kredit dengan melihat catatan transaksi yang tersimpan dalam database perusahaan kartu kredit
6. Barang apa yang biasanya dibeli oleh kustomer supermarket ketika dia membeli diaper bayi? Bagaimana manajemen supermarket memberi respon setelah mengetahui pola pembelian kustomer?

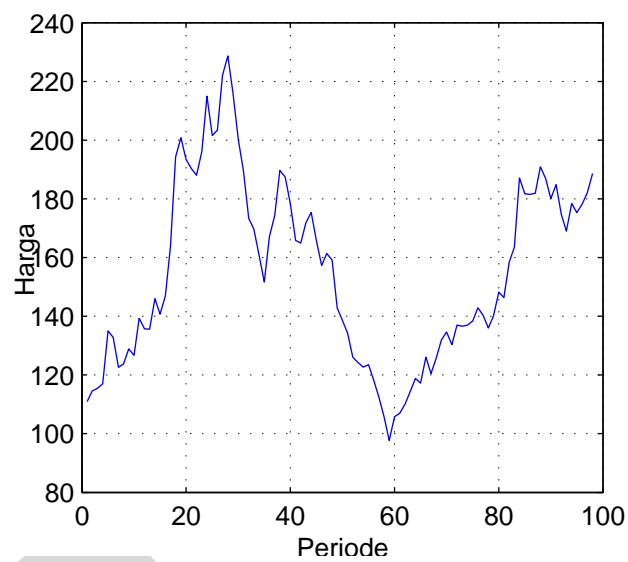
Data *Iris* Fisher (1936) dalam Tabel 1.1 berikut yang menandakan jenis bunga berdasar panjang sepal, lebar sepal, panjang petal dan lebar petal. Sedangkan jenis bunga bisa dikelompokkan dalam *Virginica*, *Setosa* dan *Versicolor*. Jenis-jenis bunga iris ini bisa diubah ke dalam nilai numerik, misalkan 1 untuk *Virginica*, 2 untuk *Setosa* dan 3 untuk *Versicolor*. Dalam hal ini, panjang sepal, lebar sepal, panjang petal dan lebar petal kita sebut sebagai atribut atau variabel. Nilai dari variabel ini kita sebut input. Sedangkan jenis bunga kita namakan sebagai output.

**Tabel 1.1:** Beberapa attribut dan nilainya yang digunakan untuk pengambilan keputusan

Panjang sepal	Lebar sepal	Panjang petal	Lebar petal	Jenis
6.50	2.80	4.60	1.50	Setosa
6.80	2.80	4.80	1.40	Setosa
7.40	2.80	6.10	1.90	Versicolor
7.70	2.80	6.70	2.00	Versicolor
4.40	2.90	1.40	0.20	Virginica
6.30	2.90	5.60	1.80	Versicolor



**Gambar 1.1:** Data Iris dengan output berupa tiga kelas/kategori



**Gambar 1.2:** Data harga tepung di Kansas dengan output berupa nilai kontinu

## 2.1 Pengenalan Pola, Data Mining dan Machine Learning

Pengenalan Pola adalah suatu disiplin ilmu yang mempelajari cara-cara mengklasifikasikan obyek ke beberapa kelas atau kategori dan mengenali kecenderungan data. Tergantung pada aplikasinya, obyek-obyek ini bisa berupa pasien, mahasiswa, pemohon kredit, image atau signal atau pengukuran lain yang perlu diklasifikasikan atau dicari fungsi regresinya. Biasanya subyek ini disebut dengan pengenalan pola atau pattern recognition.

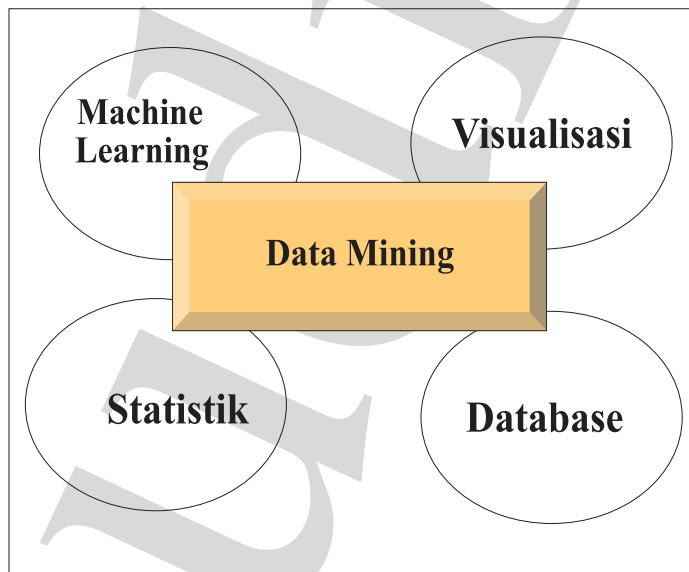
*Data mining*, sering juga disebut *knowledge discovery in database* (KDD), adalah kegiatan yang meliputi pengumpulan, pemakaian data historis untuk menemukan keteraturan, pola atau hubungan dalam set data berukuran besar. Keluaran dari data mining ini bisa dipakai untuk memperbaiki pengambilan keputusan di masa depan. Sehingga istilah pattern recognition sekarang jarang digunakan karena ia termasuk bagian dari data mining.

*Machine learning* adalah suatu area dalam *artificial intelligence* atau kecer-



dasar buatan yang berhubungan dengan pengembangan teknik-teknik yang bisa diprogramkan dan belajar dari data masa lalu. Pengenalan pola, data mining dan machine learning sering dipakai untuk menyebut sesuatu yang sama. Bidang ini bersinggungan dengan ilmu probabilitas dan statistik kadang juga optimasi. Machine learning menjadi alat analisis dalam data mining.

- Statistik: lebih berdasarkan teori, lebih fokus pada pengujian hipotesis
- Machine learning : lebih bersifat heuristik, fokus pada perbaikan performansi dari suatu teknik learning, juga meliputi real-time learning dan robotic area yang tidak termasuk dalam data mining
- Data Mining: gabungan teori dan heuristik, fokus pada seluruh proses penemuan knowledge/pola termasuk data cleaning, learning, dan visualisasi dari hasilnya.



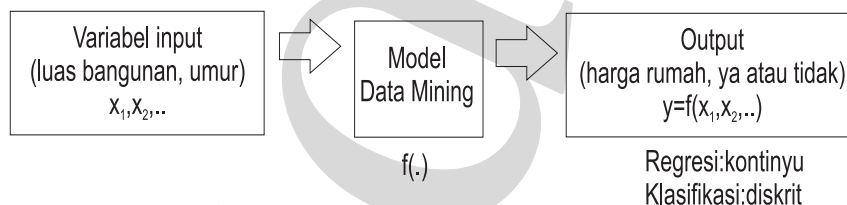
**Gambar 2.1:** Data Mining merupakan irisan dari berbagai disiplin

## 2.2 Variabel, Fitur dan Atribut

Suatu observasi, example, pattern (pola) atau obyek biasanya ditandai oleh beberapa atribut. Misalnya obyek orang, bisa ditandai dengan atribut tinggi badan, berat badan, bentuk muka, dan lain-lain. Atribut ini sering juga disebut dengan variabel. Juga ada yang menyebut dengan fitur.

## 2.3 Supervised dan Unsupervised Learning

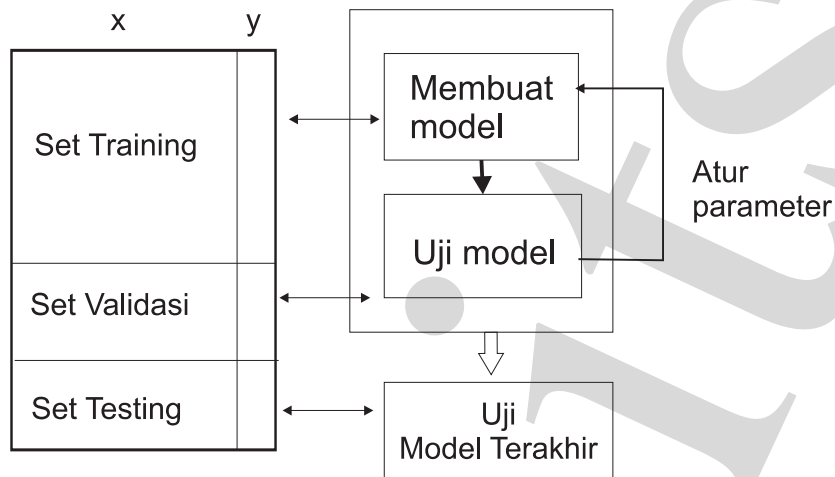
Dalam pendekatan pertama, unsupervised learning, metoda kita terapkan tanpa adanya latihan (training) dan tanpa ada guru (teacher). Misalkan kita punya sekelompok pengamatan atau data tanpa ada keluaran (output) tertentu, maka dalam unsupervised learning kita harus mengelompokkan data tersebut ke dalam beberapa kelas yang kita kehendaki. Ini terutama dilakukan karena memang keluaran tidak tersedia. Untuk melakukan *tugas* (task) ini kita bisa menerapkan metoda unsupervised learning. Masuk dalam kelompok ini adalah **clustering** dan **self organizing map** (SOM). *Supervised learning*



**Gambar 2.2:** Proses mencari hubungan antara variabel input,  $x$ , dan output,  $y$ , dalam supervised learning

yaitu metoda belajar dengan adanya latihan dan pelatih. Banyak teknik dalam pattern recognition masuk dalam kategori ini. Sebagai contoh adalah regresi, analisis diskriminan (LDA), artificial neural networks (ANN) dan support vector machine (SVM).

Mari kita lihat contoh lain. Misalkan seseorang akan menentukan kondisi suatu hari bagus untuk bermain sepakbola atau tidak. Ada beberapa ciri yang



**Gambar 2.3:** Pembagian data menjadi set training, validasi dan testing

bisa kita gunakan sebagai acuan untuk mengambil keputusan. Lihat di Tabel 2.1 berikut ini.

**Tabel 2.1:** Beberapa attribut dan nilainya yang digunakan untuk pengambilan keputusan

Suhu	Angin	Cuaca	Keputusan
Panas	Kuat	Hujan	Tidak Main
Dingin	Kecil	Tidak Hujan	Main
Sedang	Kuat	Hujan	Main

## 2.4 Jenis Nilai Variabel

Variabel berdasarkan nilainya bisa dikelompokkan sebagai berikut:

1. Nominal Variabel yang nilainya berupa simbol, nilainya sendiri hanya berfungsi sebagai label atau memberi nama, tidak ada hubungan antar nilai nominal, tidak bisa diurutkan atau diukur jaraknya dan hanya uji

persamaan yang bisa dilakukan. Sebagai contoh dalam variabel cuaca, ada *hujan* dan *tidak hujan*. Ini adalah nilai nominal, tidak bisa diurutkan atau dihitung jarak antara hujan dan tidak hujan.

## 2. Ordinal

Variabel yang nilainya berupa simbol tetapi bisa diurutkan, tidak bisa diukur jaraknya, tidak bisa dijumlahkan. Kadang perbedaannya dengan variabel nominal kurang tegas. Sebagai contoh variabel temperatur mempunyai nilai panas, sedang, dingin. Nilai ini bisa diurutkan dari panas-sedang-dingin.

## 3. Interval

Variabel yang nilainya bisa diurutkan, dan diukur dengan tetap dan unit yang sama. Misalkan variabel temperatur diukur dalam derajat Fahrenheit. Perbedaan nilai dalam interval bisa dihitung. Nilai nol tidak didefinisikan secara mutlak.

## 4. Rasio

Variabel yang mempunyai nilai nol yang mutlak. Sebagai contoh variabel jarak yang diukur dalam centimeter. Jarak antara suatu obyek dengan dirinya sendiri adalah nol. Nilai variabel ratio diperlakukan sebagai bilangan riil. Semua operasi matematika, seperti penjumlahan, pengurangan, pembagian, dsb. bisa dilakukan terhadap nilai rasio.

Dalam praktek, kebanyakan pengukuran dinyatakan sebagai kontinyu (ordinal) atau diskrit (kategoris, nominal). Dalam buku ini kita akan menggunakan term kontinyu dan diskrit untuk menyebut jenis variabel.

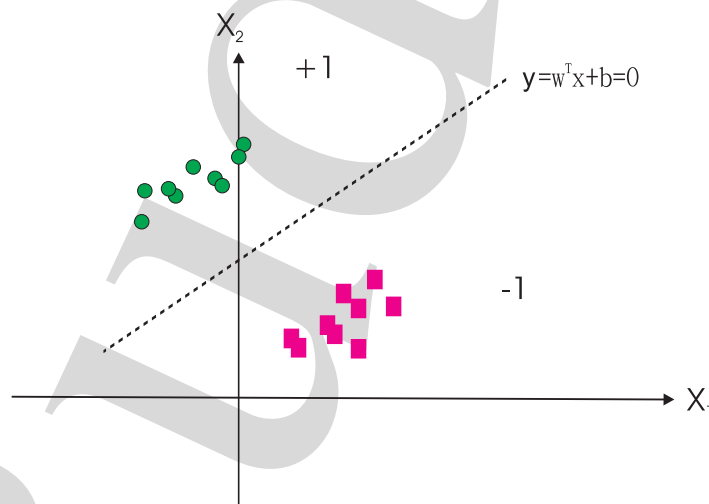
## 2.5 Klasifikasi dan Pendekatan Fungsi (Regresi)

Dalam klasifikasi, keluaran dari setiap data adalah bilangan bulat atau diskrit. Sedangkan dalam regresi keluaran dari setiap data adalah bilangan kontinu (continuous number). Dalam regresi, misalkan orang ingin meramalkan harga

rumah dengan mendasarkan harganya terhadap atribut seperti lokasi, umur rumah dan luas rumah maka keluarannya akan berupa bilangan kontinu. Keluarannya bisa berupa bilangan Rp 120 juta, Rp 100 juta atau Rp 51 juta.

## 2.6 Klasifikasi Dua Kelas dan Multi Kelas

Sebelum kita berkenalan dengan teknik-teknik yang akan kita pelajari ada baiknya kita mengetahui terlebih dahulu apa itu klasifikasi dua kelas dan multi kelas. Yang pertama bisa dijelaskan sebagai berikut. Misalkan kita punya set data untuk training  $(x_i, y_i)$ ,  $i = 1, \dots, \ell$  dengan data input  $X = \{x_1, x_2, \dots, x_\ell\} \subseteq \mathbb{R}^N$  dan output yang bersangkutan  $Y = \{y_1, \dots, y_\ell\} \subseteq \{\pm 1\}^\ell$ . Tujuan dari klasifikasi dua kelas adalah menemukan suatu *fungsi keputusan* (decision function)  $f(x)$  yang secara akurat memprediksi kelas dari data *test*  $(x, y)$  yang berasal dari fungsi distribusi yang sama dengan data untuk *training*, lihat Gambar 2.4. Set data  $\ell (x_i, y_i)$ ,  $i = 1, \dots, \ell$  biasa dinamakan set training, dimana  $x_i$  berkaitan dengan parameter input dan  $y_i$  menunjukkan parameter output. Untuk yang kedua, klasifikasi multi kelas, permasalahan sedikit



**Gambar 2.4:** Klasifikasi dua set obyek dari dua kelas

berbeda. Misalkan kita punya set data untuk training  $(x_i, y_i)$ ,  $i = 1, \dots, \ell$

dengan data input  $X = \{x_1, x_2, \dots, x_\ell\} \subseteq \mathbb{R}^N$  dan output yang bersangkutan  $Y = \{y_1, \dots, y_\ell\} \subseteq \{1, 2, \dots, k\}^\ell$ . Perhatikan sekarang bahwa output  $Y$  tidak lagi terbatas  $\pm 1$  seperti dalam kasus dua kelas. Output dari data kita bisa 1, 2, 3, 4 atau bahkan 10.

## 2.7 Transformasi Data

Ada beberapa cara transformasi data yang dilakukan sebelum kita menerapkan suatu metoda. Coba perhatikan bila kita punya data dengan susunan sebagai berikut:

$$X = \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1n} \\ x_{21} & x_{22} & \dots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix}.$$

dimana  $n$  adalah jumlah variabel/atribut dan  $m$  adalah banyaknya observasi. Beberapa cara untuk mentransformasi data adalah:

### 1. Centering

Dengan centering kita mengurangi setiap data dengan rata-rata dari setiap atribut yang ada. Misalkan kita ingin memtransformasikan data dalam suatu kolom dengan cara centering, maka bisa kita gunakan rumus berikut

$$\widehat{X} = X - \bar{X} \quad (2.1)$$

$\widehat{X}$  adalah vektor hasil setelah centering,  $X$  adalah vektor kolom dan  $\bar{X}$  adalah rata-rata dari kolom yang bersangkutan. Kita harus melakukan proses ini untuk semua kolom dari  $i = 1$  sampai  $i = n$ . Dari data yang sudah dicentering bisa kita lakukan beberapa operasi untuk mendapatkan beberapa besaran baru.

#### (a) Matriks Scatter

catter adalah jarak antar variabel  $X_1$  dengan  $X_2$  atau  $X_1$  dengan  $X_3$  dan seterusnya. Scatter antara setiap variabel dengan variabel

yang lain bisa dihitung dengan rumus:

$$Scatter = \widehat{X}'\widehat{X}$$

(b) Matriks Kovarian

Seperti scatter tapi kita bagi setiap entri dalam matriks dengan jumlah data  $m$ .

$$Kovarian = \frac{\widehat{X}'\widehat{X}}{m - 1}$$

2. Normalisasi

Setelah centering bisa juga dilanjutkan dengan proses berikutnya yaitu membagi setiap data yang sudah dicentering dengan standar deviasi dari atribut yang bersangkutan.

$$\widehat{X} = \frac{X - \bar{X}}{\sigma_x} \quad (2.2)$$

3. Scaling

Scaling adalah prosedur merubah data sehingga berada dalam skala tertentu. Skala ini bisa antara  $(0, 1)$ ,  $(-1, 1)$  atau skala lain yang dikehendaki. Misalkan kita punya data seperti dalam tabel 3.1, maka kita bisa mengkonversi data tersebut ke dalam skala  $(0, 1)$ . Dalam hal ini batas bawah (BB) adalah 0 dan batas atas (BA) adalah 1. Jika nilai maksimum tiap kolom adalah  $X_{max}$  dan nilai minimumnya adalah  $X_{min}$ , untuk mengubah data ke skala baru, untuk setiap data bisa dilakukan operasi

$$\widehat{X} = \frac{X - X_{min}}{X_{max} - X_{min}} * (BA - BB) + BB \quad (2.3)$$

Jika kita ingin data kita berada dalam skala  $(-1, 1)$ , maka bisa kita memakai rumus berikut:

$$\widehat{X} = \frac{X - X_{min}}{X_{max} - X_{min}} * (1 - (-1)) + (-1) \quad (2.4)$$

Berikut adalah contoh pengubahan skala dari suatu data ke dalam interval antara -1 dan 1. Pengalihan ini dilakukan dengan fungsi yang dinamakan *premnmx* yang tersedia dalam software Matlab (Santosa, 2006).

```
x =
    5.0000    2.0000    3.5000    1.0000
    6.0000    2.2000    4.0000    1.0000
    6.2000    2.2000    4.5000    1.5000
    6.0000    2.2000    5.0000    1.5000
    4.5000    2.3000    1.3000    0.3000
    5.0000    2.3000    3.3000    1.0000
    5.5000    2.3000    4.0000    1.3000
    6.3000    2.3000    4.4000    1.3000
    4.9000    2.4000    3.3000    1.0000
    5.5000    2.4000    3.7000    1.0000
```

```
>> [pn] = premnmx(x');
```

```
>> pn'
```

```
ans =
   -0.4444   -1.0000    0.1892    0.1667
    0.6667    0.0000    0.4595    0.1667
    0.8889    0.0000    0.7297    1.0000
    0.6667    0.0000    1.0000    1.0000
   -1.0000    0.5000   -1.0000   -1.0000
   -0.4444    0.5000    0.0811    0.1667
    0.1111    0.5000    0.4595    0.6667
    1.0000    0.5000    0.6757    0.6667
   -0.5556    1.0000    0.0811    0.1667
    0.1111    1.0000    0.2973    0.1667
```

## 2.8 Konsep Jarak

Jarak menjadi aspek penting dalam pengembangan metoda pengklasifikasian maupun regresi. Banyak metoda dikembangkan berangkat dari konsep jarak. Untuk lebih mudah memahami metoda yang akan kita pelajari dalam bab-bab berikut di sini kita bahas konsep jarak secara singkat. Untuk mengukur jarak dua titik  $x$  dan  $y$ , ( $d(x, y)$ ), kita bisa menggunakan beberapa konsep jarak.



Ukuran jarak harus memenuhi syarat-syarat sebagai berikut:

1.  $d(x, y) \geq 0$  (non-negatif)  
Tidak ada jarak yang mempunyai nilai negatif.
2.  $d(x, y) = 0$  jika dan hanya jika  $x = y$  (identity of indiscernibles)  
Jarak antara suatu obyek atau titik dengan obyek atau titik itu sendiri adalah nol.
3.  $d(x, y) = d(y, x)$  (simetri)  
Jarak dari  $x$  ke  $y$  adalah sama dengan jarak dari  $y$  ke  $x$
4.  $d(x, z) \leq d(x, y) + d(y, z)$  (ketidaksamaan segitiga).

Beberapa macam jarak yang sering digunakan dalam literatur machine learning/data mining adalah:

1. Jarak *Euclidean* ( $L_2$ -norm)

Jarak dua titik  $x$  dan  $y$  menurut Euclidean dirumuskan sebagai:

$$d(x, y) = \|x - y\|^2 = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \quad (2.5)$$

2. Jarak Manhattan atau Cityblock ( $L_1$ -norm) Menurut konsep ini jarak dua titik  $x$  dan  $y$  dirumuskan :

$$d(x, y) = \sum_{i=1}^n (|x_i - y_i|) \quad (2.6)$$

3. Jarak Minkowski

$$d(x, y) = \|x - y\|_q = (\sum |x - y|^q)^{\frac{1}{q}} \quad (2.7)$$

dimana  $q \geq 1$  adalah parameter yang bisa diseleksi. Dalam hal  $q = 1$ , maka jarak tersebut menjadi jarak Manhattan. Sedangkan untuk  $q = 2$ , jarak tersebut menjadi jarak Euclidean.

4. Jarak Chebyshev ( $L_\infty$ -norm)

$$d(x, y) = \|x - y\|_\infty = \max_{1 \leq i \leq n} \{|x_i - y_i|\} \quad (2.8)$$

5. Jarak Mahalanobis (energy-norm)

$$d(x, y) = [(x - y)^T \Sigma^{-1} (x - y)] \quad (2.9)$$

dimana  $\Sigma$  adalah covariance matrix. Dalam hal  $\Sigma = I$ , maka jarak ini menjadi Euclidean.

## 2.9 Programa Kuadratik (Quadratic Programming)

Salah satu bentuk persoalan optimisasi yang banyak digunakan selain program linear atau *linear programming* adalah *Programa Kuadratik* selanjutnya disingkat QP. Hal ini disebabkan karena bentuknya yang unik yang akan memberikan algoritma yang memungkinkan diperoleh solusi yang global optimal. Bentuk standar QP adalah

$$\begin{aligned} \min \quad & \frac{1}{2} x' Q x + f' x \\ \text{subject to} \quad & A_{eq} x = b_{eq} \\ & Ax \leq b \\ & LB \leq x \leq UB \end{aligned} \quad (2.10)$$

dimana  $LB$  adalah *lower bound* atau batas bawah nilai  $x$  dan  $UB$  adalah *upper bound* atau batas atas nilai  $x$ . Sedangkan  $Q, A, A_{eq}$  adalah matrik dan  $f, b_{eq}, b$  adalah vektor.

## 2.10 Lagrange Multiplier

*Lagrange multiplier* digunakan untuk menyelesaikan masalah optimisasi dengan pembatas (*constrained optimization*) di mana pembatasnya berupa persamaan

yang ditandai dengan  $=$ , bukan  $\leq$  atau  $\geq$ . Dengan Lagrange multiplier ini, masalah optimasi dengan pembatas akan dikonversikan menjadi masalah *optimasi tanpa pembatas* (*unconstrained optimization*). Dari situ akan dicari kondisi yang perlu untuk menentukan kandidat yang akan menjadi titik-titik optimal. Misalkan kita punya masalah optimasi dengan satu pembatas berupa persamaan sebagai berikut

$$\text{Minimize } f(x) \quad (2.11)$$

subject to

$$h(x) = 0 \quad (2.12)$$

Metoda Lagrange multiplier akan mengubah masalah ini menjadi masalah tanpa pembatas sebagai

$$\text{Minimize } L(x, \nu) = f(x) - \nu h(x) \quad (2.13)$$

Fungsi  $L(x, \nu)$  disebut dengan *fungsi Lagrange* dan  $\nu$  dinamakan *Lagrange multiplier*. Selanjutnya untuk mencari nilai  $x$  dan  $\nu$  yang optimal, kita turunkan fungsi Lagrange tersebut. Untuk ilustrasi perhatikan permasalahan berikut

$$\text{Minimize } f(x) = 2x_1 + x_2 \quad (2.14)$$

subject to

$$x_1^2 + x_2^2 - 1 = 0$$

Kita bisa mengubah masalah di atas menjadi  $L = 2x_1 + x_2 - \nu(x_1^2 + x_2^2 - 1) = 0$ . Kondisi optimalitas adalah

$$\frac{\partial L}{\partial x_1} = 2 - 2\nu x_1 = 0, \Rightarrow x_1 = \frac{1}{\nu}$$

$$\frac{\partial L}{\partial x_2} = 1 - 2\nu x_2 = 0, \Rightarrow x_2 = \frac{1}{2\nu}$$

Dengan memasukkan nilai  $x_1$  dan  $x_2$  ke dalam pembatas  $h = x_1^2 + x_2^2 - 1 = 0$  diperoleh

$$\nu = \frac{\sqrt{5}}{2}; \quad x = \left(-\frac{2}{\sqrt{5}}, -\frac{1}{\sqrt{5}}\right)$$

$$\nu = -\frac{\sqrt{5}}{2}; \quad x = \left(\frac{2}{\sqrt{5}}, \frac{1}{\sqrt{5}}\right)$$

Titik  $x = \left(-\frac{2}{\sqrt{5}}, -\frac{1}{\sqrt{5}}\right)$  adalah solusi optimal untuk masalah aslinya, persamaan (2.14).

### 3.1 Pendahuluan

Teknik klaster termasuk teknik yang sudah cukup dikenal dan banyak dipakai dalam *pattern recognition*. Sampai sekarang para ilmuwan dalam bidang *pattern recognition* masih melakukan berbagai usaha untuk melakukan perbaikan model klaster karena metoda yang dikembangkan sekarang masih bersifat *heuristik*.

Tujuan utama dari metoda klaster adalah pengelompokan sejumlah data/obyek ke dalam klaster (group) sehingga dalam setiap klaster akan berisi data yang semirip mungkin. Dalam klastering kita berusaha untuk menempatkan obyek yang mirip (jaraknya dekat) dalam satu klaster dan membuat jarak antar klaster sejauh mungkin. Ini berarti obyek dalam satu klaster sangat mirip satu sama lain dan berbeda dengan obyek dalam klaster-klaster yang lain.

Ada dua pendekatan dalam klastering: *partitioning* dan *hirarki*.

## 3.2 Klastering Hirarki (Hierarchical Clustering)

Dalam *klastering hirarki* kita hitung jarak masing-masing obyek dengan setiap obyek yang lain. Selanjutnya kita temukan pasangan obyek yang jaraknya terdekat. Sehingga tiap obyek akan berpasangan dengan satu obyek atau kelompok obyek yang lain yang paling dekat jaraknya. Langkah-langkah yang perlu dilakukan untuk melakukan klastering dengan cara klastering hirarki adalah :

1. Kelompokkan setiap obyek ke dalam kelompok/klasternya sendiri
2. Temukan pasangan paling mirip untuk dimasukkan ke dalam klaster yang sama dengan melihat data dalam matriks kemiripan (resemblance).
3. Gabungkan kedua obyek dalam satu klaster
4. Ulangi sampai tersisa hanya satu klaster

### 3.2.1 Kemiripan dan Ketidakmiripan

Untuk menggabungkan dua atau lebih obyek menjadi satu klaster, bisanya digunakan ukuran kemiripan atau ketidakmiripan. Semakin mirip dua obyek semakin tinggi peluang untuk dikelompokkan dalam satu klaster. Sebaliknya semakin tidak mirip semakin rendah peluang untuk dikelompokkan dalam satu klaster. Untuk mengukur *kemiripan* (similarity) dan *ketidakmiripan* (dissimilarity) diantara data/obyek bisa dipakai beberapa ukuran.

- Cosinus antara dua titik  $x$  dan  $y$  didefinisikan sebagai:

$$\cos\theta = \frac{x^T y}{\|x\| \|y\|} \quad (3.1)$$

dimana  $\|x\|$  didefinisikan sebagai

$$\sqrt{\sum_{i=1}^n x_i^2}$$

Dari data pertama di Tabel 3.1,  $[10 \ 5]$ , dan data kedua,  $[20 \ 20]$ , kita bisa menghitung besarnya cosinus diantara keduanya.

$$\cos\theta = \frac{[10 \ 5] * [20 \ 20]'}{\sqrt{[10 \ 5] * [10 \ 5]'}\sqrt{[20 \ 20] * [20 \ 20]'}} = \frac{300}{\sqrt{125}\sqrt{800}} = 0.9487 \quad (3.2)$$

- Kovarian

Kovarian antara dua data didefinisikan sebagai:

$$\text{cov}(x, y) = \frac{1}{n} \sum_{i=1}^n (X_i - \bar{X})(Y_i - \bar{Y}), \quad (3.3)$$

dimana  $x$  adalah data pertama dan  $y$  data kedua. Dari tabel 3.1 kita bisa menghitung kovarian antara data ke satu dan ke dua. Rata-rata dari data ke satu dan ke dua adalah

$$\bar{x} = \begin{bmatrix} 7.5 \\ 20 \end{bmatrix}.$$

$$\text{cov}(\text{data1}, \text{data2}) = \frac{1}{2}(10 - 7.5) * (20 - 20) + (5 - 7.5) * (20 - 20) = 0$$

- Koefisien korelasi

$$r(x, y) = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y} \quad (3.4)$$

Misalkan kita punya data sebagai berikut dan kita ingin melakukan klastering dengan klastering hirarki.

Dari tabel 3.1 di atas bisa dibuat matriks ketidakmiripan (angka di belakang koma dibulatkan), yaitu jarak. Selanjutnya kita susun dalam bentuk matriks. Dalam matriks ini jarak dari setiap pasang obyek dihitung. Nilai dalam satu entri menunjukkan jarak antar obyek dari indeks dari kolom dan baris. Misalnya entri dari kolom 1 baris 2, berarti jarak antar obyek 1 dan obyek 2. Karena matriks jarak ini simetris, maka kita hanya menampilkan

**Tabel 3.1:** Contoh set data dengan dua variabel/atribut

No.	$X_1$	$X_2$
1	10	5
2	20	20
3	30	10
4	30	15
5	5	10

separuh dari seluruh matriks.

	1	2	3	4	5
1	0	.	.	.	.
2	18.0	0	.	.	.
3	20.6	14.1	0	.	.
4	22.4	11.2	5	0	.
5	7.1	18.0	25	25.5	0

Biasanya untuk menentukan jarak antara dua kluster  $A$  dan  $B$  digunakan salah satu dari beberapa ukuran:

- Jarak maksimum antara elemen dalam kluster (complete linkage clustering).

Dengan cara ini jarak antara dua kluster didefinisikan sebagai

$$d(A, B) = \max_{x \in A, y \in B} \{s_{xy}\} \quad (3.5)$$

dimana  $s_{xy}$  adalah jarak dua data  $x$  dan  $y$  masing-masing dari kluster  $A$  dan  $B$ .

- Jarak minimum antara elemen dari setiap kluster (single linkage clustering).

Dengan cara ini jarak antara dua kluster didefinisikan sebagai

$$d(A, B) = \min_{x \in A, y \in B} \{s_{xy}\}, \quad (3.6)$$



- Rata-rata jarak antara elemen dari setiap kluster (average linkage clustering)

$$d(A, B) = \frac{1}{n_A n_B} \sum_{x \in A} \sum_{y \in B} s(x, y), \quad (3.7)$$

dimana  $n_A$  adalah banyaknya data dalam set A.

- Centroid Linkage Dengan cara ini jarak antara dua kluster didefinisikan sebagai

$$d(A, B) = s(\bar{x}, \bar{y}) \quad (3.8)$$

dimana

$$\bar{x} = \frac{1}{n_A} \sum_{x \in A} x$$

- Ward linkage

$$d(A, B) = \frac{n_A n_B s_{AB}^2}{n_A + n_B}, \quad (3.9)$$

dimana  $s_{AB}^2$  jarak antara kluster A dan B menggunakan *centroid linkage*.

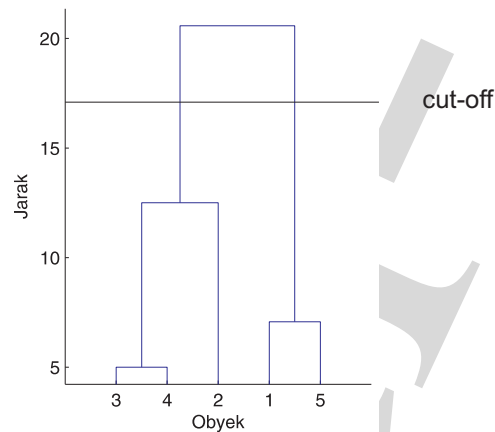
Untuk contoh dalam matriks kemiripan di atas kita tahu bahwa jarak minimum adalah 5 yaitu antara obyek 3 dan 4. Maka kita tempatkan obyek 3 dan 4 dalam satu kluster. Kemudian kita hitung lagi jarak antara tiap obyek dengan obyek yang lain. Untuk gabungan obyek 3 dan 4, sekarang koordinatnya kita ambil rata-rata dari keduanya yaitu [30 12.5]. matriks kemiripan didapat sebagai:

	1	2	3&4	5
1	0	.	.	.
2	18.0	0	.	.
3&4	21.4	12.5	0	.
5	<b>7.1</b>	18.0	25.1	0

Dengan prosedur yang sama seperti di atas, obyek 1 dan 5 kita tempatkan dalam satu klaster. Kita peroleh kemudian matriks sebagai berikut

	2	3&4	1&5
2	0	0	.
3&4	12.5	0	.
1&5	17.7	23.0	0

Kalau kita teruskan, kita akan mendapat pohon klaster atau *dendrogram* seperti diperlihatkan dalam Gambar 3.1. *Contoh*



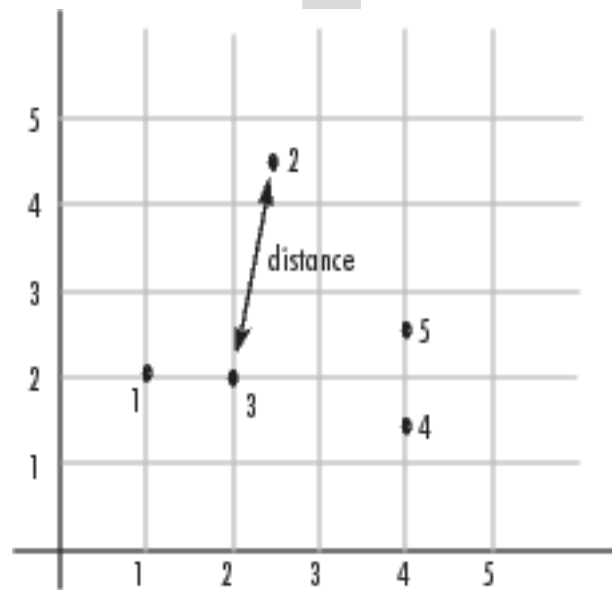
**Gambar 3.1:** Pohon klaster, dari bawah memperlihatkan bagaimana urutan lima obyek dikelompokkan menjadi satu klaster. Sumbu-y menunjukkan jarak antar klaster

Dalam contoh ini kita gunakan software Matlab. Misalkan kita mempunyai data dalam bentuk matriks  $X$ , dimana

$$X = \begin{bmatrix} 1 & 2 \\ 2.5 & 4.5 \\ 2 & 2 \\ 4 & 1.5 \\ 4 & 2.5 \end{bmatrix}$$

Jarak antar obyek dalam  $X$  dalam Matlab dihitung dengan fungsi *pdist*.  $\text{Pdist}(X)$  akan menghitung jarak antar obyek dalam matriks  $X$ .

```
>>Y = pdist(X)
Columns 1 through 7
    2.9155    1.0000    3.0414    3.0414    2.5495    3.3541    2.5000
Columns 8 through 10
    2.0616    2.0616    1.0000
```

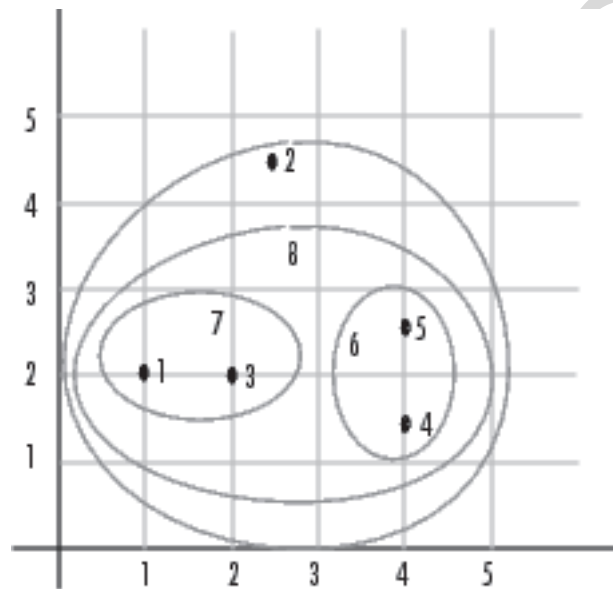


**Gambar 3.2:** Jarak antar dua obyek dalam dua dimensi

Kemudian dengan fungsi *squareform*(Y) akan diperoleh matriks jarak sebagai berikut:

0	2.9155	1.0000	3.0414	3.0414
2.9155	0	2.5495	3.3541	2.5000
1.0000	2.5495	0	2.0616	2.0616
3.0414	3.3541	2.0616	0	1.0000
3.0414	2.5000	2.0616	1.0000	0

Nilai dalam sel (1, 1) adalah jarak obyek 1 dengan dirinya sendiri. Nilai sel (1, 2) = sel (2, 1) menunjukkan jarak antara obyek 1 dan 2. Dari matriks jarak ini kemudian bisa kita dapatkan matriks *linkage*. Ini bisa dilakukan dengan menggunakan fungsi *linkage*. Kolom satu dan dua menunjukkan dengan obyek



**Gambar 3.3:** Hasil pengelompokan dengan menggunakan linkage

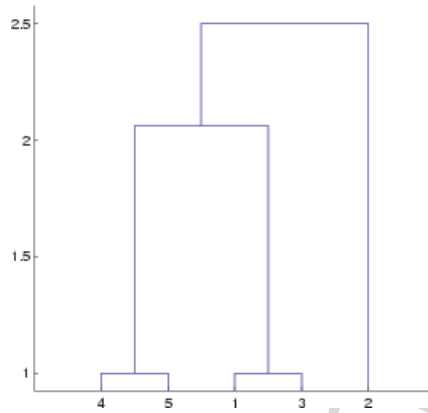
mana suatu obyek dikelompokkan. Kolom ke tiga adalah jarak antar obyek dalam kolom satu dan dua.

```
>>Z = linkage(Y)
Z =
    4.0000    5.0000    1.0000
    1.0000    3.0000    1.0000
    6.0000    7.0000    2.0616
    2.0000    8.0000    2.5000
```

Hasil dari klastering di atas ditunjukkan dalam gambar 3.4 dengan eksekusi perintah berikut:

```
>>dendrogram(Z)
```

Untuk mengukur tingkat akurasi hasil klastering kita dalam mewakili data, bisa digunakan suatu parameter yang dinamakan *koefisien korelasi cophenetic*. Parameter ini mengukur korelasi antara jarak yang dihitung selama



**Gambar 3.4:** Dendrogram

penyusunan pohon klaster (dendrogram) atau jarak cophenetic dan jarak sebenarnya. Dengan menggunakan notasi  $Y$  untuk jarak sesungguhnya yang dihitung dengan fungsi *pdist* dan jarak yang dihitung dengan *linkage*, formula penghitungan *koefisien korelasi cophenetic* adalah:

$$c = \frac{\sum_i^j (Y_{ij} - y)(Z_{ij} - z)}{\sqrt{\sum_i^j (Y_{ij} - y)^2 (Z_{ij} - z)^2}}, \quad (3.10)$$

dimana

$Z_{ij}$  adalah jarak dua obyek  $i$  dan  $j$  dari matriks  $Z$  kolom 3

$Y_{ij}$  adalah jarak antar obyek  $i$  dan  $j$  dari vektor  $Y$

$y$  dan  $z$  masing-masing adalah rata-rata dari  $Y$  dan kolom 3 dari  $Z$ .

Semakin dekat nilai  $c$  ke 1 semakin akurat hasil klaster kita dalam mewakili data.

### 3.3 K-means

Untuk melakukan klustering ini, nilai  $k$  harus ditentukan terlebih dahulu. Secara detail kita bisa menggunakan ukuran ketidakmiripan untuk mengelompokkan obyek kita. Ketidakmiripan bisa diterjemahkan dalam konsep jarak. Jika jarak dua obyek atau data atau titik cukup dekat, maka dua obyek itu

mirip. Semakin dekat berarti semakin tinggi kemiripannya. Semakin tinggi nilai jarak, semakin tinggi ketidakmiripannya. Algoritma *k-means* klastering bisa ringkas sebagai berikut:

1. Pilih jumlah klaster  $k$
2. Inisialisasi  $k$  pusat klaster Ini bisa dilakukan dengan berbagai cara. Yang paling sering dilakukan adalah dengan cara random. Pusat-pusat klaster diberi nilai awal dengan angka-angka random.
3. Tempatkan setiap data/obyek ke klaster terdekat Kedekatan dua obyek ditentukan berdasar jarak kedua obyek tersebut. Demikian juga kedekatan suatu data ke klaster tertentu ditentukan jarak antara data dengan pusat klaster. Dalam tahap ini perlu dihitung jarak tiap data ke tiap pusat klaster. Jarak paling dekat antara satu data dengan satu klaster tertentu akan menentukan suatu data masuk dalam klaster mana.
4. Hitung kembali pusat klaster dengan keanggotaan klaster yang sekarang Pusat klaster adalah rata-rata dari semua data/obyek dalam klaster tertentu. Jika dikehendaki bisa juga memakai median dari klaster tersebut. Jadi rata-rata (mean) bukan satu-satunya ukuran yang bisa dipakai.
5. Tugaskan lagi setiap obyek dengan memakai pusat klaster yang baru. Jika pusat klaster sudah tidak berubah lagi, maka proses pengklasteran selesai. Atau, kembali lagi ke langkah nomor 3 sampai pusat klaster tidak berubah lagi.

### 3.4 Fuzzy K-means

Berbeda dengan *k-means* klastering, dimana suatu obyek hanya akan menjadi anggota satu klaster, dalam *fuzzy k-means*, setiap data bisa menjadi anggota dari beberapa klaster. Sesuai dengan namanya *fuzzy* yang berarti samar. Batas-batas klaster dalam *k-means* adalah tegas (hard) sedangkan dalam *fuzzy k-means* adalah soft. Prosedur untuk *fuzzy k-means* sama dengan

*k-means* kecuali ada beberapa syarat sebagai berikut:

$$\sum_{i=1}^k u_k(x) = 1, \forall x \quad (3.11)$$

Dimana  $u_k(x)$  menunjukkan probabilitas  $x$  masuk dalam kluster tertentu. Dalam *fuzzy k-means*, pusat kluster dihitung dengan mencari rata-rata dari semua titik dalam suatu kluster dengan diberi bobot berupa tingkat keanggotaan (degree of belonging) dalam kluster tersebut yaitu:

$$\mu_k = \frac{\sum_x u_k(x)x}{\sum_x u_k(x)} \quad (3.12)$$

Tingkat keanggotaan  $x$  dalam kluster didefinisikan sebagai:

$$\begin{aligned} u_k(x) &= \frac{1}{d(\mu_k, x)^{-1}} \\ &= \frac{1}{\sum_j \frac{d(\mu_k, x)^{\frac{1}{m-1}}}{d(\mu_j, x)^{\frac{1}{m-1}}}} \end{aligned} \quad (3.13)$$

### 3.5 Implementasi Klustering dengan Matlab

Misalkan kita mempunyai set data yang kita dapat dengan mengenerate bilangan random dengan dua dimensi yang kita lakukan dalam Matlab sebagai berikut:

```
>>X = [randn(10,2)+2*ones(10,2); randn(10,2)-2*ones(10,2)]
```

```
X =
```

2.0000	2.5689
1.6821	1.7444
3.0950	1.6225
0.1260	1.7041
2.4282	0.5249
2.8956	1.7660
2.7310	2.1184
2.5779	2.3148

2.0403	3.4435
2.6771	1.6490
-1.3768	-1.6101
-1.2010	-1.9120
-1.0591	-2.6355
-2.9921	-2.5596
-1.7880	-1.5563
-1.7621	-2.9499
-3.0078	-1.2188
-2.7420	-1.4310
-0.9177	-2.8217
-2.1315	-2.2656

Dari data di atas kita lihat bahwa 10 data pertama mestinya berada dalam kelompok. Sedangkan 10 data berikutnya berada dalam kelompok yang lain. Untuk membuktikannya mari kita terapkan teknik K-means klastering. Kita eksekusi perintah berikut dalam Matlab.

```
>> [cidx, ctrs] = kmeans(X, 2, 'dist','SqEuclidean');
```

Output *cidx* adalah klaster untuk data pada baris yang sama. Sedangkan *ctrs* adalah pusat klaster dimana data tersebut terletak. Sedangkan *X* adalah input data dan 2 adalah jumlah klaster yang kita kehendaki. Di sini kita memilih untuk menggunakan jarak Euclidean. Untuk penyajian yang lebih baik kita bisa tulis perintah berikut:

```
>> [X cidx]
ans =
    2.0000    2.5689    1.0000
    1.6821    1.7444    1.0000
    3.0950    1.6225    1.0000
    0.1260    1.7041    1.0000
    2.4282    0.5249    1.0000
    2.8956    1.7660    1.0000
    2.7310    2.1184    1.0000
```



2.5779	2.3148	1.0000
2.0403	3.4435	1.0000
2.6771	1.6490	1.0000
-1.3768	-1.6101	2.0000
-1.2010	-1.9120	2.0000
-1.0591	-2.6355	2.0000
-2.9921	-2.5596	2.0000
-1.7880	-1.5563	2.0000
-1.7621	-2.9499	2.0000
-3.0078	-1.2188	2.0000
-2.7420	-1.4310	2.0000
-0.9177	-2.8217	2.0000
-2.1315	-2.2656	2.0000

Dua kolom pertama adalah data kita sedangkan kolom ketiga menunjukkan kluster untuk data pada baris yang bersangkutan. Bisa dilihat bahwa 10 data pertama memang berada pada kluster yang sama. Sedangkan 10 data yang lain berada dalam kluster yang berbeda. Untuk melihat lebih jelas, bisa kita plot dalam dua dimensi sebagai berikut:

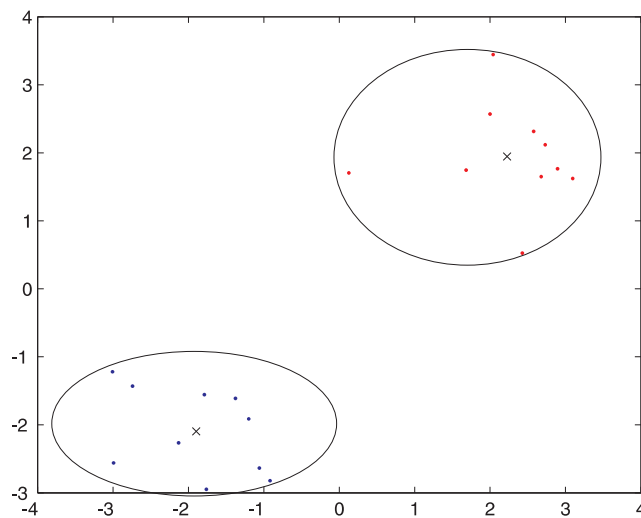
```
>> plot(X(cidx==1,1),X(cidx==1,2),'r.',X(cidx==2,1),...
X(cidx==2,2),'b.',ctr(:,1),ctr(:,2),'kx');
```

Hasil plotnya bisa dilihat pada Gambar 3.5. Bila kita ingin mengganti jenis jarak yang dipakai, misalnya dengan jarak Manhattan atau city block kita bisa gunakan perintah sebagai berikut:

```
>> [cidx, ctrs] = kmeans(X, 2, 'dist','city');
```

Selanjutnya kita terapkan klastering fuzzy K-means atau dalam Matlab disebut *fuzzy c-means*. Berikut ini adalah perintahnya

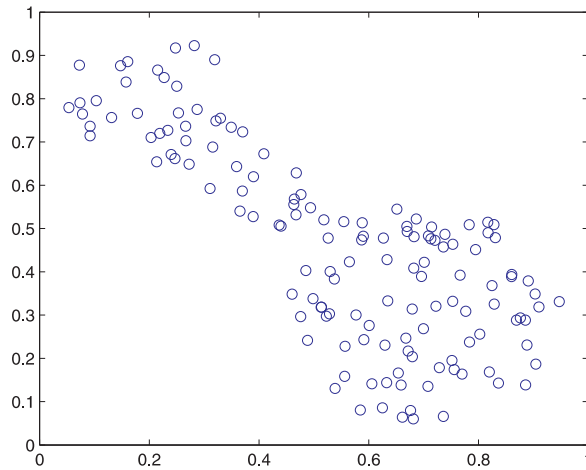
```
>>load fcndata.dat;% untuk meload data dalam Matlab
>>[center, U, obj_fcn] = fcm(fcndata, 2);
fuzzy c-means dengan fungsi fcm
```



**Gambar 3.5:** Data yang kita generate secara random, diklasterkan dalam 2 kelompok

Output *center* mengandung koordinat dari pusat klaster, dalam hal ini kita mempunyai 2 klaster. Sedangkan *U* mengandung tingkat keanggotaan untuk setiap titik data. Nilai yang lebih besar menunjukkan kecenderungan keanggotaan yang lebih besar. Untuk mengetahui data tertentu masuk ke klaster mana kita terapkan perintah berikut ini

```
>>maxU = max(U);
>>index1 = find(U(1, :) == maxU);
>>index2 = find(U(2, :) == maxU);
```



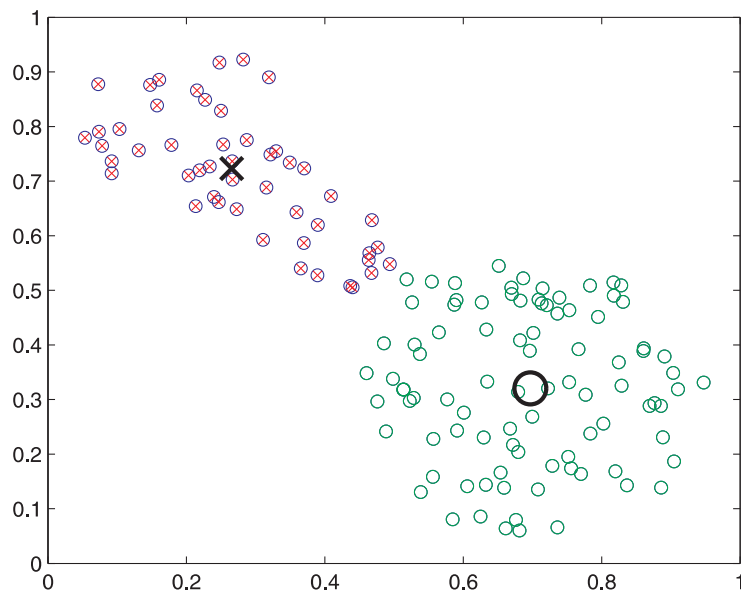
**Gambar 3.6:** Data untuk diklasterkan dengan fuzzy-c-means

Untuk membuat gambar dalam dua dimensi mengenai hasil klaster dari data di atas bisa digunakan rangkaian perintah berikut.

```
>> line(fcmdata(index1, 1), fcmdata(index1, 2), 'linestyle',...
'none','marker', 'o','color','g');
>> line(fcmdata(index2,1),fcmdata(index2,2),'linestyle',...
'none','marker', 'x','color','r');
>> hold on
>> plot(center(1,1),center(1,2),'ko','markersize',15,'LineWidth',2)
>> plot(center(2,1),center(2,2),'kx','markersize',15,'LineWidth',2)
```

Berikut adalah contoh implementasi klaster hirarki dengan Matlab. Yang pertama perlu kita generate set data dengan menggunakan bilangan random dalam 3 dimensi. Berikut ini adalah perintah dan hasilnya.

```
>> rand('state',0);
>> X = [rand(10,3); rand(10,3)+1.2; rand(10,3)+2.5];
X =
```



**Gambar 3.7:** hasil pengklasteran dengan fuzzy-c-means

0.9501	0.6154	0.0579
0.2311	0.7919	0.3529
0.6068	0.9218	0.8132
0.4860	0.7382	0.0099
0.8913	0.1763	0.1389
0.7621	0.4057	0.2028
0.4565	0.9355	0.1987
0.0185	0.9169	0.6038
0.8214	0.4103	0.2722
0.4447	0.8936	0.1988
1.2153	2.0381	1.3934

1.9468	1.2196	1.8822
1.6451	1.8813	1.5028
2.1318	1.5795	1.7417
1.6660	2.0318	1.3509
1.6186	1.7028	1.8979
2.0462	1.9095	1.5784
1.7252	1.6289	2.0600
1.4026	1.5046	2.0537
1.8721	1.3897	1.7936
2.9966	3.2271	3.2948
3.3998	2.8093	3.4568
3.3216	3.3385	3.0226
3.1449	3.0681	3.3801
3.3180	2.8704	2.6730
3.1602	3.2027	3.4797
2.8420	3.0466	2.7714
2.7897	2.9449	2.7523
2.8412	3.1946	3.3757
3.0341	3.1213	3.2373

Kemudian kita hitung jarak masing-masing data dengan data yang lain dengan perintah berikut

```
>> Y = pdist(X,'euclid');%menghitung jarak dengan jarak Euclidean
```

Y adalah vektor dengan ukuran  $(30 - 1 \times 30)/2 \times 1$ .

```
>> Z = linkage(Y,'single');
```

Sesudah diperoleh vektor jarak dari setiap obyek ke obyek yang lain, bisa kita tentukan pohon klaster dengan mencari pasangan terdekat dari setiap obyek atau kelompok obyek. Perintah *linkage* di atas melakukan tugas pengelompokan ini. Kita lihat hasilnya adalah sebagai berikut:

```
>> Z
```

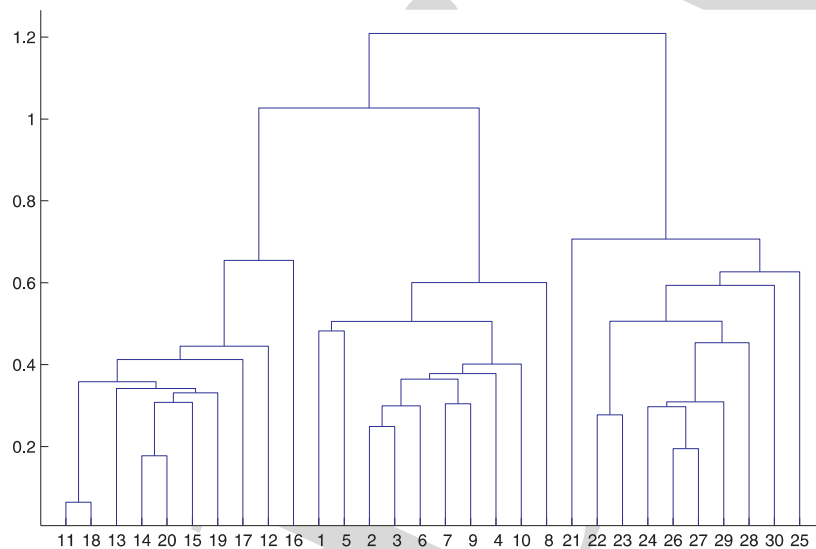
```
Z =
```

11.0000	18.0000	0.0639
14.0000	20.0000	0.1772
26.0000	27.0000	0.1948
2.0000	3.0000	0.2489
22.0000	23.0000	0.2774
24.0000	33.0000	0.2970
6.0000	34.0000	0.2995
7.0000	9.0000	0.3044
15.0000	32.0000	0.3078
29.0000	36.0000	0.3090
19.0000	39.0000	0.3310
13.0000	41.0000	0.3415
31.0000	42.0000	0.3580
37.0000	38.0000	0.3646
4.0000	44.0000	0.3782
10.0000	45.0000	0.4013
17.0000	43.0000	0.4120
12.0000	47.0000	0.4446
28.0000	40.0000	0.4533
1.0000	5.0000	0.4820
46.0000	50.0000	0.5055
35.0000	49.0000	0.5059
30.0000	52.0000	0.5936
8.0000	51.0000	0.6005
25.0000	53.0000	0.6266
16.0000	48.0000	0.6545
21.0000	55.0000	0.7063
54.0000	56.0000	1.0268
57.0000	58.0000	1.2085

Kolom pertama dan kedua adalah nomor kelompok dan kolom ketiga adalah jaraknya, yang dihitung dengan metoda single linkage. Penomoran kelompok dilakukan dengan nomor obyek aslinya, kecuali ada dua atau lebih obyek yang

tergabung dalam satu kelompok akan diberi nomor baru. Perintah berikut akan menghasilkan gambar dendrogram dari hasil perintah linkage di atas. Hasilnya diperlihatkan dalam Gambar 3.8.

```
>>H = dendrogram(Z)
```



**Gambar 3.8:** Hasil pengklasteran data di atas dengan klaster hirarki dalam bentuk dendrogram

Dengan perintah lain bisa kita tentukan di klaster mana masing-masing obyek ditempatkan. Berikut adalah perintah dan hasilnya

```
>> T = cluster(Z,'maxclust',3)
```

T =

```
1
1
1
1
1
1
1
```

```
1
1
1
2
2
2
2
2
2
2
2
2
2
2
2
2
2
3
3
3
3
3
3
3
3
3
3
3
```

Serangkaian perintah di atas, bisa dirangkum dalam satu perintah untuk mengetahui hasil pengklasteran sebagai berikut

```
>>T = clusterdata(X,'maxclust',3);
```

Seperti kita lihat dari hasil di atas, data 1 – 10 masuk dalam klaster 1, data 11 – 20 masuk dalam klaster 2 dan sisanya masuk dalam klaster 3. Untuk penggunaan Matlab dalam data mining secara lebih detail, lihat (Santosa, 2006).



## 4.1 Pendahuluan

Salah satu teknik yang sangat populer dan cukup sederhana untuk menyelesaikan masalah klasifikasi adalah *analisis diskriminan linear* (Linear Discriminant Analysis). Selanjutnya kita singkat LDA.

## 4.2 Ide Dasar LDA Dua kelas

Dalam teknik ini kita perlu mencari *fungsi diskriminan* atau *fungsi pemisah* yang merupakan kombinasi linear dari variabel-variabel, yang memisahkan obyek ke dalam dua kelompok atau kelas. Misalkan kita mempunyai sederet data sebagai berikut  $x_{11}, x_{12}, \dots, x_{1n_1}$  dari kelompok populasi satu dan  $x_{21}, x_{22}, \dots, x_{2n_2}$  dari kelompok populasi dua. Seperti biasa, setiap vektor  $x_{ij}$  terdiri dari  $p$  variabel. Fungsi diskriminan adalah kombinasi linier dari  $p$  variabel ini, yang memaksimalkan jarak antara dua kelompok setelah ditransformasikan. Kombinasi linier  $z = w'x$  mentransformasikan setiap vektor observasi menjadi be-

saran skalar:

$$z_{1i} = w'x_i = w_1x_{i1} + w_2x_{i2} + \dots + w_px_{ip}, i = 1, 2, \dots, n_1 \text{ (dari kelas 1)}$$

$$z_{2i} = w'x_i = w_1x_{i1} + w_2x_{i2} + \dots + w_px_{ip}, i = 1, 2, \dots, n_2 \text{ (dari kelas 2)}$$

Karena itu  $(n_1 + n_2)$  vektor observasi dalam dua sampel,

$$x_{11} \ x_{12} \dots x_{1n_1}$$

$$x_{21} \ x_{22} \dots x_{2n_2}$$

ditransformasikan menjadi besaran skalar,

$$z_{11} \ z_{12} \dots z_{1n_1}$$

$$z_{21} \ z_{22} \dots z_{2n_2}$$

Kita dapatkan rata-rata (mean) sebagai

$$\begin{aligned} \bar{z}_1 &= \frac{\sum_{i=1}^{n_1} z_{1i}}{n_1} = w' \bar{x}_1 \\ \bar{z}_2 &= \frac{\sum_{i=1}^{n_2} z_{2i}}{n_2} = w' \bar{x}_2 \end{aligned} \quad (4.1)$$

dimana

$$\bar{x}_1 = \frac{\sum_{i=1}^{n_1} x_{1i}}{n_1}$$

dan

$$\bar{x}_2 = \frac{\sum_{i=1}^{n_2} x_{2i}}{n_2}$$

Kita perlu menentukan nilai vector  $w$  yang memaksimalkan kuantitas  $(\bar{z}_1 - \bar{z}_2)/s_z$ . Karena  $(\bar{z}_1 - \bar{z}_2)/s_z$  bisa bernilai negatif, kita menggunakan jarak kuadrat yang distandarkan  $(\bar{z}_1 - \bar{z}_2)^2/s_z^2$  yang bisa diekspresikan sebagai

$$\frac{(\bar{z}_1 - \bar{z}_2)^2}{s_z^2} = \frac{[w'(\bar{x}_1 - \bar{x}_2)]^2}{w'S_{pl}w}, \quad (4.2)$$

dimana

$$s_z^2 = \frac{\sum_{i=1}^n (z_i - \bar{z})^2}{n - 1} = w'S_{pl}w$$

dan

$$S_{pl} = \sum_{i=1}^2 (n_i - 1) S_i / \sum_{i=1}^2 (n_i - 1)$$

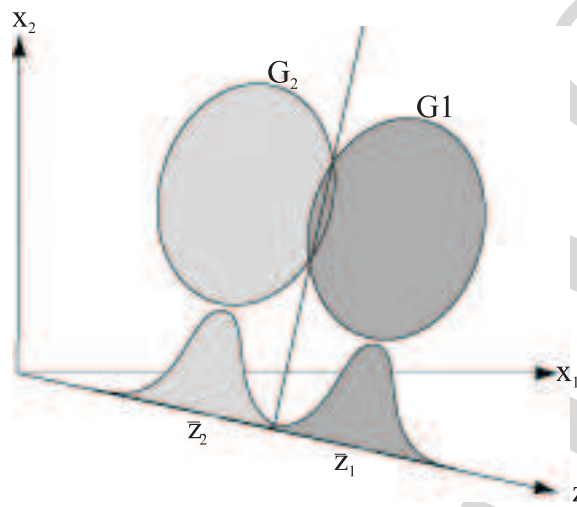
Dengan mengambil derivasi dari persamaan 4.2 terhadap  $w$  dan menyamakan dengan nol maka nilai maksimum dari 4.2 akan dicapai bila

$$w = S_{pl}^{-1}(\bar{x}_1 - \bar{x}_2), \quad (4.3)$$

atau kalau nilai  $w$  merupakan kelipatan dari  $S_{pl}^{-1}(\bar{x}_1 - \bar{x}_2)$ . Jadi nilai maksimum dari vektor  $w$  tidak unik. Tetapi, arah dari  $w$  unik, karena nilai relatif atau rasio  $w_1, w_2, \dots, w_p$  unik, dan  $z = w'x$  memproyeksikan titik  $x$  ke garis dimana  $(\bar{z}_1 - \bar{z}_2)^2 / s_z^2$  mencapai nilai maksimum. Perlu dicatat di sini bahwa  $S_{pl}^{-1}$  mempunyai nilai hanya jika  $n_1 + n_2 - 2 > p$ . Arah optimal  $w = S_{pl}^{-1}(\bar{x}_1 - \bar{x}_2)$  secara efektif paralel dengan garis yang menghubungkan  $\bar{x}_1$  dengan  $\bar{x}_2$ , karena jarak kuadrat  $(\bar{z}_1 - \bar{z}_2)^2 / s_z^2$  ekuivalen dengan jarak yang distandarkan antara  $x_1$  dan  $x_2$ . Ini bisa dilihat dengan mensubstitusi persamaan 4.3 ke dalam 4.2 dan kita peroleh:

$$\frac{(\bar{z}_1 - \bar{z}_2)^2}{s_z^2} = (\bar{x}_1 - \bar{x}_2)' S_{pl}^{-1} (\bar{x}_1 - \bar{x}_2) \quad (4.4)$$

untuk  $z = w'x$  dengan  $w = S_{pl}^{-1}(\bar{x}_1 - \bar{x}_2)$ . Karena  $w = S_{pl}^{-1}(\bar{x}_1 - \bar{x}_2)$ , persamaan 4.4 bisa kita tulis sebagai  $(\bar{z}_1 - \bar{z}_2)^2 / s_z^2 = w'(\bar{x}_1 - \bar{x}_2)$ , dan semua arah di luar arah ini akan mencapai jarak lebih kecil. Gambar 4.1 memberi ilustrasi bagaimana pemisahan dua kelompok obyek dengan analisis diskriminan setelah ditransformasikan ke dalam ruang satu dimensi. Dalam contoh ini covariance dari kedua populasi sama nilainya. Kombinasi linear  $[z_{1i} = w'x_{1i} = w_1x_{1i1} + w_2x_{1i2} + \dots + w_px_{1ip}, i = 1, 2, \dots, n_1$  dan  $z_{2i} = w'x_{2i} = w_1x_{2i1} + w_2x_{2i2} + \dots + w_px_{2ip}, i = 1, 2, \dots, n_2$  memproyeksikan titik-titik  $x_{1i}$  dan  $x_{2i}$  ke dalam garis pemisah yang optimal antara dua kelas tersebut. Jika variabel  $x_1$  dan  $x_2$  adalah *normal bivariate* maka kombinasi linear  $z = w_1x_1 + w_2x_2 = w'x$  adalah *normal univariate*. Dalam gambar 4.2 ditunjukkan alternatif proyeksi ke arah lain. gambar ini memperlihatkan bahwa analisis diskriminan mencapai jarak pemisahan maksimal.



**Gambar 4.1:** Analisis Diskriminan dua-kelas

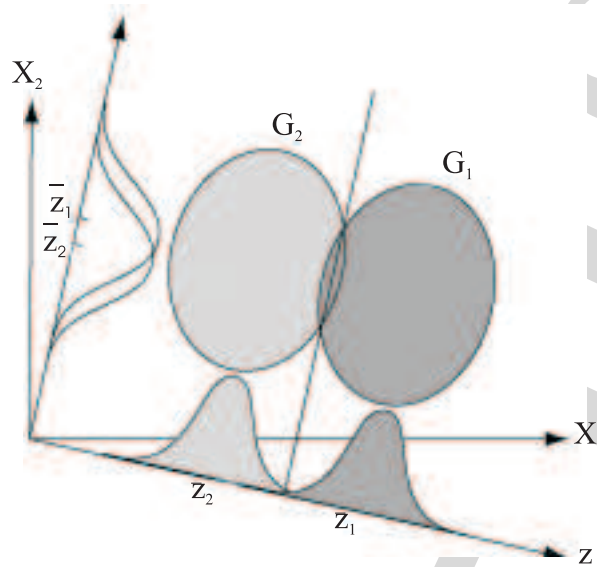
### 4.3 Implementasi

Contoh berikut menjelaskan bagaimana cara menemukan fungsi diskriminan seperti telah dijelaskan di atas. Perhatikan sampel dari produksi metal yang

**Tabel 4.1:** Produksi metal pada dua temperatur

No	Temperatur 1		No.	Temperatur 2	
	$x_1$	$x_2$		$x_1$	$x_2$
1	33	60	1	35	57
2	36	61	2	36	59
3	35	64	3	38	59
4	38	63	4	39	61
5	40	65	5	41	63
			6	43	65
			7	41	59

dilakukan pada dua temperatur yang berbeda dalam Tabel 4.1, variabel  $x_1 =$



**Gambar 4.2:** Tingkat separasi yang dicapai dengan Analisis Diskriminan

yield point dan  $x_2$ = ultimate strength. Dari data kita bisa menghitung

$$\bar{x}_1 = [36.4 \ 62.6]$$

$$\bar{x}_2 = [39.0 \ 60.4]$$

$$S_1 = \begin{bmatrix} 7.30 & 4.20 \\ 4.20 & 4.30 \end{bmatrix}$$

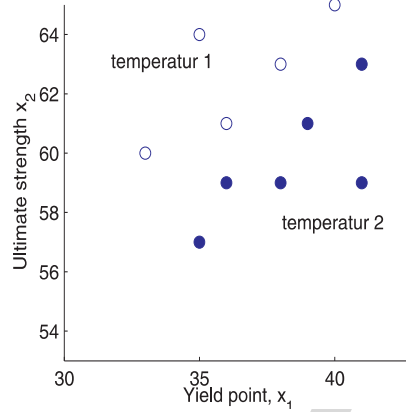
$$S_2 = \begin{bmatrix} 8.33 & 6.67 \\ 6.67 & 7.62 \end{bmatrix}$$

$$S_{pl} = (n_1 - 1)S_1 + (n_2 - 1)S_2 / (n_1 - 1)(n_2 - 1)$$

Dalam hal ini  $n_1 = 5$  dan  $n_2 = 7$

$$S_{pl} = \begin{bmatrix} 7.92 & 5.68 \\ 5.68 & 6.29 \end{bmatrix}$$

Plot dari data ini terlihat dalam gambar 4.3. Kita lihat bahwa kedua kelompok data ini bisa dipisahkan secara linier bila diproyeksikan ke arah yang



**Gambar 4.3:** Analisis Diskriminan dua-kelas

tepat. Tetapi bila diproyeksikan ke arah  $x_1$  atau  $x_2$  akan terjadi overlap. Satu dimensi dimana titik-titik itu akan diproyeksikan secara tepat adalah fungsi diskriminan

$$z = w'x = w_1x_1 + w_2x_2 = -1.633x_1 + 1.820x_2,$$

dimana  $w$  didapat dari

$$w = S_{pl}^{-1}(\bar{x}_1 - \bar{x}_2)'$$

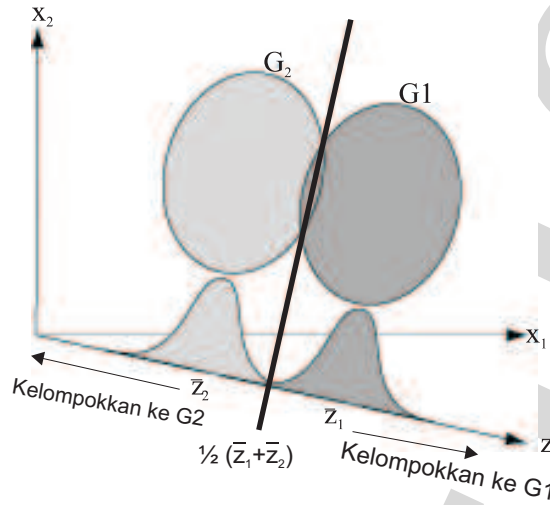
$$S_{pl}^{-1} = \begin{bmatrix} 0.3582 & -0.3234 \\ -0.3234 & 0.4509 \end{bmatrix}$$

$$\bar{x}_1 - \bar{x}_2 = \begin{bmatrix} -2.6000 & 2.1714 \end{bmatrix}$$

Setelah fungsi diskriminan ditemukan dan kita ingin mengelompokkan data baru ke kelas mana yang paling sesuai, bisa digunakan prosedur sebagai berikut:

Kelompokkan data baru  $x$  ke dalam group 1 atau  $G_1$  jika

$$w'x = S_{pl}^{-1'}(\bar{x}_1 - \bar{x}_2)x > \frac{1}{2}(\bar{x}_1 - \bar{x}_2)S_{pl}^{-1}(\bar{x}_1 + \bar{x}_2)$$



**Gambar 4.4:** Tingkat separasi yang dicapai dengan Analisis Diskriminan

atau

$$w'x = S_{pl}^{-1'}(\bar{x}_1 - \bar{x}_2)x - \frac{1}{2}(\bar{x}_1 - \bar{x}_2)S_{pl}^{-1}(\bar{x}_1 + \bar{x}_2) > 0$$

dan kelompokkan ke group 2 atau  $G_2$  bila

$$w'x = S_{pl}^{-1'}(\bar{x}_1 - \bar{x}_2)x - \frac{1}{2}(\bar{x}_1 - \bar{x}_2)S_{pl}^{-1}(\bar{x}_1 + \bar{x}_2) < 0$$

Dasar pikirnya adalah kita ingin mengelompokkan  $x$  ke  $G_1$  jika  $z = w'x$  lebih dekat ke  $\bar{z}_1$  daripada ke  $\bar{z}_2$ . Sebaliknya kita kelompokkan data baru  $x$  ke  $G_2$  bila  $z$  lebih dekat ke  $\bar{z}_2$ . Dari gambar 4.4 kita bisa melihat bahwa  $z$  lebih dekat ke  $\bar{z}_1$  bila

$$z > \frac{1}{2}(\bar{z}_1 + \bar{z}_2)$$

Secara umum ini tepat karena  $\bar{z}_1$  selalu lebih besar dari  $\bar{z}_2$  yang bisa dengan mudah dilihat dari:

$$(\bar{z}_1 - \bar{z}_2) = w'\bar{x}_1 - \bar{x}_2 = (\bar{x}_1 - \bar{x}_2)'S_{pl}^{-1}(\bar{x}_1 - \bar{x}_2) > 0$$

karena  $S_{pl}^{-1}$  positive definite. Sehingga  $\bar{z}_1 > \bar{z}_2$ . Karena  $\frac{1}{2}(\bar{z}_1 + \bar{z}_2)$  adalah titik tengah, maka  $z > \frac{1}{2}(\bar{z}_1 + \bar{z}_2)$  mengimplikasikan bahwa  $z$  lebih dekat ke  $\bar{z}_1$ .

## 4.4 Implementasi Analisis Diskriminan Dengan Matlab

Perhatikan data training berikut berikut

```
>> training
training =
    0.9285    2.3808
    1.2792    2.3198
    2.3733    0.0906
    1.1798   -1.3056
    0.4580    2.7887
    2.6342    1.3908
    1.8252    1.0203
    1.2308    0.5940
    1.6716   -0.5349
    0.4919    1.2214
    1.8564   -0.3745
    1.2685    0.1607
    1.6250    0.7914
   -0.0473    1.7559
    2.5357    1.3757
    1.4344   -0.3454
   -0.9171    2.4819
    1.4699    1.0327
    2.2744    2.8705
    1.6385   -0.2090
   -2.1068   -2.2594
   -2.0851   -0.8033
   -1.1516   -1.3340
   -2.3818   -1.1067
   -2.3633   -1.5071
   -4.3647   -3.9382
```



-2.1854	-1.2030
-0.6361	0.9705
-1.2600	-0.0782
-1.2370	-1.5333
-1.1654	-1.9354
-0.7617	-0.6479
-1.7088	-1.5424
-1.9971	-1.7474
-0.2813	-0.9217
-1.5953	0.7731
-0.6760	-4.5638
-2.3569	-0.1729
-1.2065	-2.4256
0.0529	0.3354

dengan label

```
>> group
```

```
group =
```

```
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1  
1
```

```
1
1
1
1
1
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
```

```
>> testing
testing =
  4.5013 -4.4211
 -2.6886 -1.4713
  1.0684  3.1317
 -0.1402 -4.9014
  3.9130 -3.6111
```

2.6210	-2.9723
-0.4353	-3.0128
-4.8150	1.0379
3.2141	-2.2781
-0.5530	-3.0119
1.1543	-4.8473
2.9194	2.4679
4.2181	-0.5490
2.3821	4.3181
-3.2373	-0.3401
-0.9429	-0.8135
4.3547	3.4622
4.1690	0.2515
-0.8973	-2.9735
3.9365	1.7214

Dengan menggunakan analisis diskriminan linier, kita akan mendapatkan kelas untuk data testing. Berikut ini adalah hasil implemetasi dalam Matlab dengan data, training *training*, dan data testing, *testing*, dan kelas dari data testing yang dihasilkan oleh analisis diskriminan ini disimpan dalam variabel *c*.

```
>> c = classify(testing, training, group)
```

```
c =
```

```

1
2
1
2
1
1
2
2
1
2
2
```

1  
1  
1  
2  
2  
1  
1  
2  
1

Buddhists

## 5.1 Ide Dasar

Artificial neural networks (ANN) atau *jaringan saraf buatan*, selanjutnya disebut ANN, awalnya mendapat inspirasi dari sistem jaringan saraf makhluk hidup. Observasi bahwa sistem belajar dari makhluk hidup terutama manusia terdiri dari jaringan yang sangat kompleks yang terdiri dari neuron yang saling terhubung telah memberi inspirasi ini. Dalam ANN, jaringan saraf makhluk hidup ingin ditiru susunannya. ANN muncul sebagai alternatif pendekatan konvensional yang biasanya kurang fleksibel terhadap perubahan struktur masalah. ANN menawarkan kelebihan dimana bisa mengatasi beberapa persoalan tanpa mengadakan perubahan drastis terhadap modelnya.

## 5.2 Model Komputasi untuk Neuron

McCulloch-Pitt mengajukan unit batas binari sebagai model komputasi untuk ANN. Model ini menghitung jumlah dari  $n$  signal input  $x_j, j = 1, 2, \dots, n$  yang diberi bobot dan menghasilkan nilai 1 bila jumlah tersebut di atas batas

tertentu dan 0 bila dibawah batas tersebut. Secara matematis bisa ditulis (Haykin, 1999).

$$y = \varphi \left( \sum_j^n w_j x_j - u \right), \quad (5.1)$$

dimana  $\varphi(.)$  adalah fungsi aktivasi dan  $w$  adalah bobot sesuai dengan input ke- $j$ .

### 5.3 Model Neuron

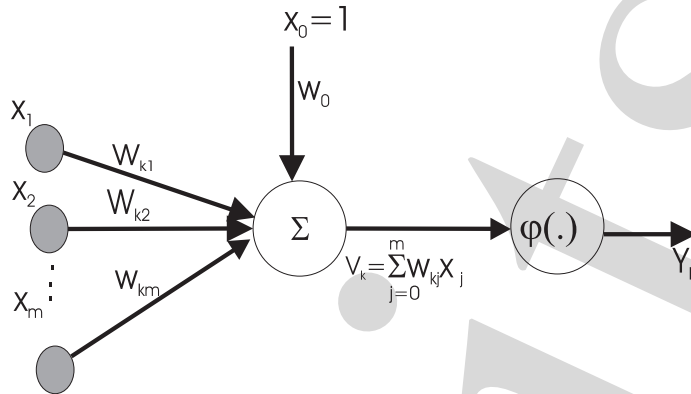
Sebuah neuron adalah unit pemroses informasi yang sangat vital dalam operasi suatu neural network. Diagram 5.1 memperlihatkan suatu model neuron yang menjadi dasar desain suatu neural network. Elemen-elemen dasar model neuron adalah:

1. Set synapsis atau link penghubung, yang ditandai dengan adanya bobot atau kekuatan dari link ini. Secara lebih detail, suatu signal  $x_j$  pada synapsis  $j$  dihubungkan ke neuron  $k$  dikalikan dengan bobot  $w_{kj}$ . Perlu dicatat bagaimana indeks pada bobot synapsis  $w_{kj}$  ini dituliskan. Indeks pertama yaitu  $k$  menunjukkan neuron dan indeks kedua  $j$  menunjukkan input seberapa.
2. Penambah, yaitu untuk menjumlahkan signal input yang diberi bobot. Operasi ini adalah kombinasi linier.
3. Fungsi aktivasi (activation function), untuk membatasi besarnya output dari suatu neuron

Dalam model neuron, gambar 5.1, termasuk di situ adalah bias dinyatakan sebagai  $b_k$  atau  $W_0$ . Bias  $b_k$  mempunyai fungsi untuk menaikkan atau menurunkan net input untuk fungsi aktivasi, tergantung nilainya positif atau negatif.

Neuron  $k$  bisa didiskripsikan secara matematis

$$u_k = \sum_{j=1}^m w_{kj} x_j \quad (5.2)$$



**Gambar 5.1:** Model neuron

dan

$$y_k = \varphi(u_k + b_k) \quad (5.3)$$

dimana  $x_1, x_2, \dots, x_m$  adalah signal input dan  $w_{k1}, w_{k2}, \dots, w_{km}$  adalah bobot dari tiap synapsis  $k$ ,  $u_k$  adalah kombinasi linier dari output yang dihasilkan signal-signal tersebut;  $b_k$  adalah bias;  $\varphi(.)$  adalah fungsi aktivasi dan  $y_k$  adalah signal output dari neuron yang bersangkutan. Pemakaian bias memberi pengaruh terhadap output dari neuron sebagai berikut

$$v_k = u_k + b_k$$

atau

$$v_k = \sum_{j=0}^m w_{kj} x_j \quad (5.4)$$

dan

$$y_k = \varphi(v_k) \quad (5.5)$$

Dalam persamaan 5.4 kita tambahkan satu synapsis untuk mengakomodasi term  $b$

$$x_0 = +1$$

dan bobot untuk synapsis ini adalah

$$w_{k0} = b_k$$

## 5.4 Macam-macam Fungsi Aktivasi

Jenis-jenis fungsi aktivasi  $\varphi(\cdot)$  yang bisa dipakai dalam neural networks adalah

### 1. Fungsi Threshold

Untuk fungsi ini kita punya dua output:

$$\varphi(v) = \begin{cases} 1 & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases} \quad (5.6)$$

### 2. Fungsi Linear Piecewise

$$\varphi(v) = \begin{cases} 1, & v \geq \frac{1}{2} \\ v + \frac{1}{2}, & -\frac{1}{2} < v < \frac{1}{2} \\ 0, & v \leq -\frac{1}{2} \end{cases} \quad (5.7)$$

### 3. Fungsi Sigmoid

$$\varphi(v) = \frac{1}{1 + \exp(-av)} \quad (5.8)$$

Ada beberapa syarat suatu fungsi bisa menjadi fungsi aktivasi  $\varphi(\cdot)$ .

#### 1. Nonlinear

Dengan memakai fungsi aktivasi yang tidak linier akan memperbaiki kemampuan network dalam melakukan tugasnya.

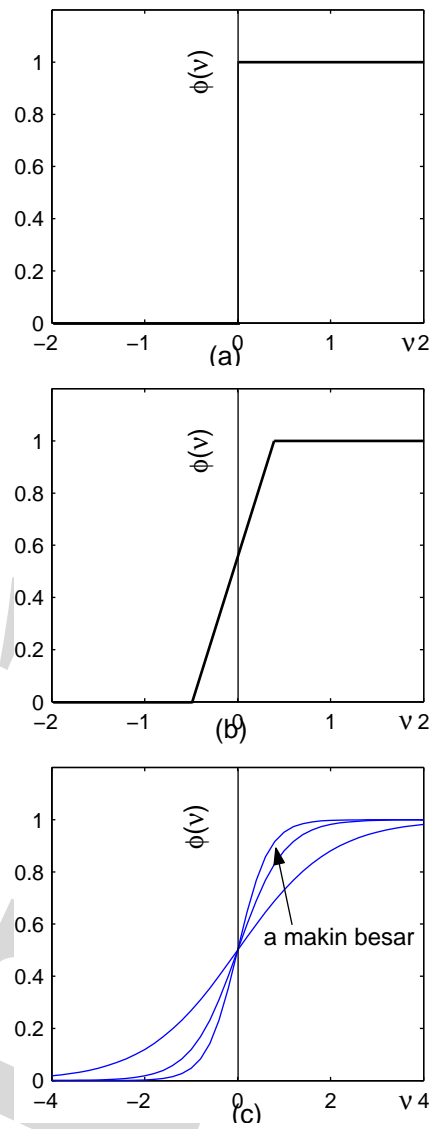
#### 2. Saturate

Suatu fungsi dinamakan *saturate* bila mempunyai output dengan nilai minimum dan maksimum. Dengan demikian akan menjaga nilai bobot  $w$  dan bias  $b$  *bounded* (terbatas). Sehingga, waktu yang dibutuhkan untuk training menjadi terbatas juga.

#### 3. Kontinuitas dan *smoothness*

Dengan syarat ini suatu fungsi aktivasi  $\varphi(\cdot)$  dan  $\varphi'(\cdot)$  terdefinisi dalam range dari argumennya. Ini sangat penting seperti dalam back-propagasi dimana kita memerlukan turunan  $\varphi'(\cdot)$ .





**Gambar 5.2:** Grafik (a) Fungsi Threshold (b) Fungsi Linear-piecewise (c) Fungsi Sigmoid

Macam-macam fungsi aktivasi yang sering digunakan dalam ANN adalah

- Sigmoid

$$f(x) = \frac{1}{1 + e^{-ax}} \quad (5.9)$$

dimana  $a > 0$

- Tangent Hyperbolic

$$f(x) = \operatorname{atanh}(bx) \quad (5.10)$$

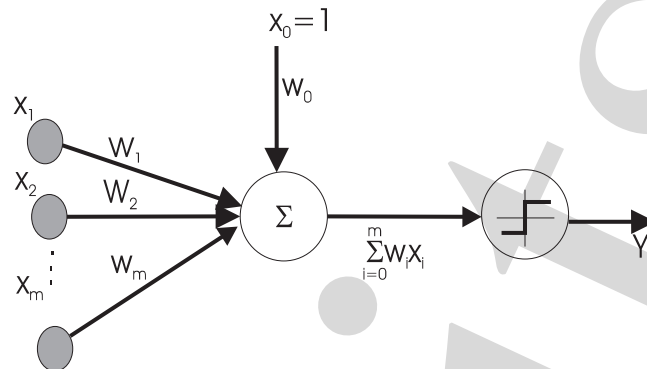
dimana  $(a, b) > 0$

## 5.5 Single-Layer Perceptrons

Perceptron adalah bentuk desain neural network yang paling sederhana yang digunakan untuk mengelompokkan obyek dari dua kelas yang bisa dipisahkan secara linier. Perceptron terdiri dari sebuah neuron dengan synapsis dan bias yang nilainya bisa diatur untuk mendapat solusi yang tepat. Rosenblatt membuktikan bahwa, jika data yang digunakan untuk mentraining perceptron tersebut diambil secara random dari set obyek dari dua kelas yang bisa dipisahkan secara linier, algoritma perceptron akan mencapai konvergensi (didapat solusi) yang memisahkan obyek dua kelas tersebut setelah sekian iterasi. Perceptron dengan satu neuron terbatas untuk menyelesaikan kasus klasifikasi dua kelas saja. Dengan menambah jumlah neuron diharapkan bisa mengatasi kasus dimana obyek berasal dari multi kelas, tetapi terbatas kasus yang linier.

## 5.6 Prosedur Learning

Tujuan dari proses *learning* di sini adalah untuk menemukan bobot  $w$  dan bias,  $b$  atau  $w_0$  sehingga network secara tepat menghasilkan output  $\{-1, +1\}$  untuk setiap data training yang dimasukkan. Ada beberapa algoritma untuk mentraining suatu network. Kita akan membahas dalam bagian ini metoda-metoda optimisasi untuk mendapatkan nilai  $w$  dan  $b$  yang optimal. Ini sangat penting sebagai landasan untuk mentraining ANN dengan banyak neuron.



**Gambar 5.3:** Grafik aliran signal dalam Perceptron

Salah satu cara melatih perceptron adalah dengan mengawali nilai  $w$  dan  $b$  dengan nilai random, lalu secara iteratif memperbarui nilainya untuk setiap titik data bila nilai outputnya tidak sesuai dengan output yang diinginkan. Proses ini diulang sampai ditemukan nilai  $w$  dan  $b$  yang menghasilkan output yang tepat sesuai yang diinginkan untuk semua data training.

### 5.6.1 Perceptron

Menurut *perceptron rule* update  $w_i$  untuk setiap input  $x_i$  dilakukan dengan cara berikut

$$w_{i+1} \leftarrow w_i + \Delta w_i$$

dimana

$$\Delta w_i = \eta(d - y)x_i$$

Di sini  $d$  adalah output dari data  $x_i$  dan  $y$  adalah output dari perceptron untuk data  $x_i$ ,  $\eta$  adalah *learning rate*, suatu bilangan positif. Dengan  $\eta$  positif diharapkan akan mampu membuat perbedaan antara  $d$  dan  $y$  makin lama makin kecil. Secara ringkas algoritma perceptron diberikan sebagai berikut (Haykin, 1999)

#### Algoritma Perceptron

Variabel dan parameter

$x = \{x_0, x_1, x_2, \dots, x_m\}$ : sampel training

$d = \{d_1, \dots, d_m\} \subset \{\pm 1\}$ : output sebenarnya

$\eta$ : learning rate :

$y = g(x)$ : output hasil

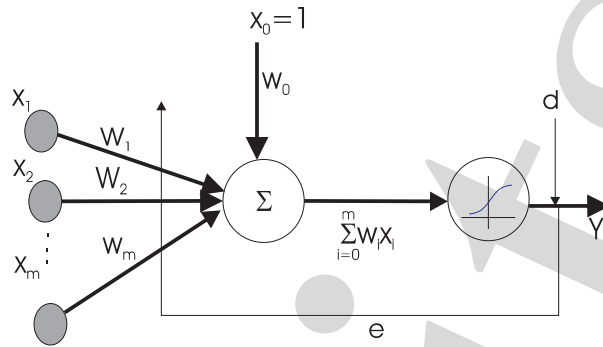
$w = [b, w_1, w_2, \dots, w_m]$ : vektor bobot dan bias  $b$

1. Inisialisasi  $w = 0$
2. Lakukan komputasi berikut untuk semua  $i$  dari  $i = 1, \dots, m$ .  
Mulai proses dengan memasukkan data  $x$  dan respon  $d$
3. Hitung  $g(x_i) = \text{sgn}(wx'_i)$  dimana  $\text{sgn}(\cdot)$  adalah fungsi signum. Semua nilai di atas 0 diberi nilai +1 sebaliknya -1.
4. Jika  $d_i \neq g(x_i)$ , Update  $w$ , menurut  
 $w = w + \eta[d_i - g(x_i)]x_i$
5. sampai semua  $1 \leq i \leq m$  kita peroleh  $g(x_i) = d_i$   
Kita dapat  $f : x \mapsto (w.x)$   
stop

Meskipun *perceptron rule* bisa menemukan bobot dan bias secara tepat untuk memisahkan obyek dari dua kelas, tapi ini hanya berlaku untuk kasus yang linier. Algoritma training yang kedua, *delta rule*, dimaksudkan untuk mengatasi kelemahan ini. Dengan algoritma ini, kasus yang tidak linier masih bisa diatasi. Algoritma ini menggunakan metoda *gradient descent* untuk menemukan bobot yang tepat sehingga output perceptron bisa mendekati output yang sebenarnya.

Contoh

Misalkan kita punya empat titik data sebagai berikut.  $X$  dengan dua variabel  $x_1$  dan  $x_2$  menyatakan input dan vektor  $d$  menyatakan output yang berhubungan untuk setiap titik. Sekarang kita ingin menemukan bobot  $w$  dan  $b$  yang



**Gambar 5.4:** Skema koreksi error dalam perceptron

**Tabel 5.1:** AND Problem

$x_1$	$x_2$	$d$
1	1	1
-1	1	-1
1	-1	-1
-1	-1	-1

bisa memisahkan data ke dalam dua kelas  $+1$  dan  $-1$ . Misalkan vektor

$$w = \begin{bmatrix} w_0 & w_1 & w_2 \end{bmatrix},$$

dimana  $w_0 = b$ . Sedangkan

$$x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix},$$

dimana faktor 1 di baris pertama adalah untuk mengakomodasi variabel  $b$ .

- Inisialisasi  $w$  dengan 0,  $\eta = 1$ ,  $w = [0 \ 0 \ 0]$
- Hitung  $wx'$  untuk titik ke satu  $x = [1 \ 1 \ 1] \rightarrow wx' = [0 \ 0 \ 0][1 \ 1 \ 1]' = 0$ ,  $y = \text{sgn}(0) = -1$ . Output ini tidak sesuai dengan label untuk data ke satu yaitu  $+1$ .

- Update  $w$ ,  $\Delta w = \eta(d - y)x = 1(1 - (-1))[1 \ 1 \ 1] = [2 \ 2 \ 2]$   
 $w = w + \Delta w = [0 \ 0 \ 0] + [2 \ 2 \ 2] = [2 \ 2 \ 2]$
- Hitung untuk titik kedua  $x = [-1 \ 1 \ 1] \rightarrow wx' = [2 \ 2 \ 2][-1 \ 1 \ 1]' = 2$ ,  
 $y = \text{sgn}(2) = +1$ . Output ini tidak sesuai dengan label untuk data ke dua yaitu  $-1$ .
- Update  $w$ ,  $\Delta w = \eta(d - y)x = 1(-1 - 1)[-1 \ 1 \ 1] = [2 \ -2 \ -2]$   
 $w = w + \Delta w = [2 \ 2 \ 2] + [2 \ -2 \ -2] = [4 \ 0 \ 0]$
- Hitung untuk titik kedua  $x = [1 \ -1 \ 1] \rightarrow wx' = [4 \ 0 \ 0][1 \ -1 \ 1]' = 4$ ,  
 $y = \text{sgn}(4) = +1$ . Output ini tidak sesuai dengan label untuk data ke tiga yaitu  $-1$ .
- Update  $w$ ,  $\Delta w = \eta(d - y)x = 1(-1 - 1)[1 \ -1 \ 1] = [-2 \ 2 \ -2]$   
 $w = w + \Delta w = [4 \ 0 \ 0] + [-2 \ 2 \ -2] = [2 \ 2 \ -2]$
- Hitung untuk titik kedua  $x = [-1 \ -1 \ 1] \rightarrow wx' = [2 \ 2 \ -2][-1 \ -1 \ 1]' = -6$ ,  
 $y = \text{sgn}(-6) = -1$ . Output ini sesuai dengan label untuk data ke empat yaitu  $-1$ . Maka set  $w$  yang bisa memisahkan keempat titik di atas ke dalam dua kelas ialah  $w = [2 \ 2 \ -2]$ . Kita bisa mengujinya dengan mengalikan vektor  $w$  dengan matriks  $x'$ .

$$wx' = [2 \ 2 \ -2] \begin{bmatrix} 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & -2 & -2 & -6 \end{bmatrix}.$$

$\text{Sgn}(wx') = [+1 \ -1 \ -1 \ -1]$ , sesuai dengan label  $d$  pada Tabel 5.1.

### 5.6.2 Gradient Descent

Metoda ini menjadi dasar algoritma *Back Propagasi* yang sering digunakan untuk mentraining neural networks dengan banyak neuron. Berikut adalah bagaimana metoda ini bekerja. Misalkan kita definisikan perbedaan antara output sebenarnya dengan output perceptron adalah  $E$ , dimana  $E$  bergantung

pada nilai bobot yang ditemukan, maka bisa kita dapatkan persamaan

$$E(w) = \frac{1}{2} \sum_{t \in T} (d_t - y_t)^2 \quad (5.11)$$

dimana  $T$  adalah set sampel training,  $d_t$  adalah output yang diinginkan untuk titik data  $t$  dan  $y_t$  adalah output yang didapat untuk titik data  $t$ . Dengan metoda *gradient descent* kita perlu mencari derivasi  $E$  untuk melakukan update terhadap vektor bobot  $w$ . Turunan  $E$  terhadap  $w$  adalah

$$\nabla E(w) = \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_m} \right] \quad (5.12)$$

Dengan metoda ini update  $w$  dilakukan dengan cara

$$w \leftarrow w + \Delta w$$

dimana

$$\Delta w = -\eta \nabla E(w)$$

Di sini  $\eta$  adalah bilangan positif. Menurut metode ini  $w$  diupdate dengan mencari arah yang berlawanan dengan gradient. Ini sesuai dengan teori *steepest descent method* dalam optimisasi bahwa arah untuk mencapai titik optimal dari suatu fungsi adalah negatif dari gradient fungsi tersebut di titik tertentu. Dengan cara update seperti ini diharapkan nilai  $E$  makin lama makin kecil. Sehingga ini akan membuat output yang dihasilkan sama dengan output sebenarnya. Untuk setiap komponen  $w$ , bisa dituliskan

$$w_{i+1} \leftarrow w_i + \Delta w_i$$

dimana

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} \quad (5.13)$$

Dari persamaan (5.11) bisa didapatkan turunan parsial dari  $E$  untuk setiap komponen  $w$ .

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{t \in T} (d_t - y_t)^2 \quad (5.14)$$

$$\begin{aligned}
&= \frac{1}{2} \sum_{t \in T} \frac{\partial}{\partial w_i} (d_t - y_t)^2 \\
&= \sum_{t \in T} (d_t - y_t) \frac{\partial}{\partial w_i} (d_t - w \cdot x_t) \\
&= \sum_{t \in T} (d_t - y_t) (-x_{it})
\end{aligned}$$

dimana  $x_t$  adalah vektor dengan komponen  $x_i$ ,  $x_{it}$  adalah komponen input tunggal  $x_i$  untuk titik data training  $t$ . Dengan hasil ini maka persamaan 5.13 bisa dituliskan sebagai

$$\Delta w_i = \eta \sum_{t \in T} (d_t - y_t) (x_{it}) \quad (5.15)$$

#### Gradient Descent

argumen: sampel training ,  $\vec{x} = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_m\} \subset X$ ,  $d = \{d_1, \dots, d_m\} \subset \{\pm 1\}$

learning rate :  $\eta$

hasil: vektor bobot,  $w$

function gradient descent( $\vec{x}, d, \eta$ )

    Inisialisasi  $w$  dengan bilangan random yang cukup kecil

    Ulang

    Inisialisasi  $\Delta w = 0$

        untuk semua  $(\vec{x}, d)$  dari  $i=1, \dots, m$

        Hitung  $g(x_i) = \text{sign}((w \cdot \vec{x}_i))$

        Update  $w$ , menurut

$w_i = w_i + \eta [d_i - g(\vec{x}_i)] x_i$

        stop

sampai semua  $1 \leq i \leq m$  kita peroleh  $g(\vec{x}_i) = y_i$

Kita dapat  $f : x \mapsto (w \cdot x) + b$

stop



### 5.6.3 Metoda Newton

Cara lain untuk mengupdate parameter  $w$  adalah dengan menggunakan metoda Newton. Dengan metode ini, update  $w$  dilakukan sebagai berikut:

$$\begin{aligned}\Delta E(w(n)) &= E(w(n+1)) - E(w(n)) \\ &= g(n)\Delta w(n) + \frac{1}{2}\Delta w(n)H(n)\Delta w(n),\end{aligned}\quad (5.16)$$

dimana  $g(n)$  adalah gradient dari fungsi  $E$ , dan  $H$  adalah matriks Hessian yang didefinisikan sebagai

$$\begin{aligned}H &= \Delta^2 E(w) \\ &= \begin{bmatrix} \frac{\partial^2 E}{\partial w_1^2} & \frac{\partial^2 E}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_1 \partial w_m} \\ \frac{\partial^2 E}{\partial w_2 \partial w_1} & \frac{\partial^2 E}{\partial w_2^2} & \cdots & \frac{\partial^2 E}{\partial w_2 \partial w_m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 E}{\partial w_m \partial w_1} & \frac{\partial^2 E}{\partial w_m \partial w_2} & \cdots & \frac{\partial^2 E}{\partial w_m^2} \end{bmatrix}\end{aligned}\quad (5.17)$$

Nilai  $\Delta E(w)$  mencapai minimum bila

$$\Delta E(w(n)) = 0$$

atau

$$\begin{aligned}g(n) + H(n)\Delta w(n) &= 0 \\ \Delta w(n) &= -H(n)^{-1}g(n)\end{aligned}$$

Dengan metoda ini, update  $w$  dilakukan dengan cara

$$w(n+1) = w(n) - H(n)^{-1}g(n) \quad (5.18)$$

Metoda Newton sangat dipengaruhi oleh pemilihan titik awal ketika iterasi dimulai. Pemilihan titik awal yang kurang tepat bisa menyebabkan metoda ini justru menemukan titik yang semakin jauh dari titik minimum. Selain itu, pemakaian turunan kedua (Hessian) juga menjadi masalah tersendiri. Apabila matriks turunan kedua singular, maka akan ada masalah dengan pencarian matriks inversnya (Reklaitis et al., 1983).

## 5.7 Algoritma Back-Propagasi

Salah satu metoda untuk mentraining multilayer neural networks adalah algoritma back-propagasi yang menggunakan learning rule *gradient descent*. Algoritma ini sangat bermanfaat, cukup handal dan mudah dipahami. Selain itu banyak algoritma yang lain mendasarkan prosesnya pada back-propagasi. Penjelasanannya adalah sebagai berikut. Kita mulai dari unit output. Error adalah selisih antara target yang sebenarnya dan keluaran dari network pada unit output.

$$e_j(n) = d_j(n) - y_j(n) \quad (5.19)$$

Error untuk training adalah jumlah keseluruhan error untuk semua unit output. Dalam hal contoh ini ada  $j$  unit output.

$$E(n) = \frac{1}{2} \sum_j e_j^2(n) \quad (5.20)$$

Sedangkan menurut gradient descent learning rule  $w$  diupdate dengan cara:

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n) \quad (5.21)$$

Sedangkan  $\Delta w_{ji}(n)$  adalah.

$$\Delta w_{ji} = -\eta \frac{\partial E(n)}{\partial w_{ji}} \quad (5.22)$$

dimana  $E$  adalah error training.

$$E(w) = \frac{1}{2} \sum_{k=1}^c (d_k - y_k)^2 \quad (5.23)$$

Di sini output adalah set unit output dalam network,  $d_k$  adalah nilai target untuk unit  $k$ , dan  $y_k$  adalah output sebenarnya untuk unit  $k$ . Untuk memudahkan pembahasan kita, kita pakai beberapa notasi

- $x_{ji}$  = input ke- $i$  ke unit  $j$
- $w_{ji}$  = bobot yang berhubungan dengan input ke- $i$  ke unit  $j$
- $net_j = \sum_i w_{ji} x_{ji}$  jumlah dari semua input dengan bobot untuk unit  $j$

- $y_j = \varphi(\text{net}_j)$  output yang dihitung oleh unit  $j$
- $w_{ji}$  = bobot yang berhubungan dengan input ke- $i$  ke unit  $j$
- $\varphi$  = fungsi aktivasi nonlinear
- $\text{net}_k = \sum_i w_{ki} y_i$  jumlah dari semua input dengan bobot untuk unit  $k$
- $z_k = \varphi(\text{net}_k)$  keluaran yang dihitung oleh unit output

Untuk update  $w$  dengan menggunakan (5.22) ada dua prosedur, dimana hal ini tergantung dimana posisi unit neuron. Dalam hal ini ada dua kemungkinan nilai turunan tersebut bergantung pada posisi unit neuron. Kasus pertama adalah bila unit tersebut adalah unit output. Kasus yang kedua adalah bila unit tersebut adalah hidden.

### 5.7.1 Kasus 1, prosedur training untuk unit output

Keluaran dari neuron  $j$  adalah

$$v_j(n) = \sum_{i=0}^m w_{ji}(n) y_i(n) \quad (5.24)$$

dimana  $m$  adalah total jumlah input termasuk bias,  $y_i(n)$  adalah neuron  $i$  dari layer sebelumnya. Jadi keluaran dari neuron ini setelah menerapkan fungsi aktivasi adalah

$$y_j(n) = \varphi(v_j(n)) \quad (5.25)$$

dimana  $\varphi(\cdot)$  adalah fungsi aktivasi nonlinear. Sedangkan turunan  $E$  (*fungsi cost*) terhadap  $w_{ji}$  atau  $\frac{\partial E(n)}{\partial w_{ji}(n)}$  adalah:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)} \quad (5.26)$$

dimana  $\delta_j(n) = -\frac{\partial E}{\partial e_j(n)} \frac{\partial e_j(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$  adalah local gradient dari neuron  $j$ . Sedangkan

$$\frac{\partial E}{\partial e_j(n)} = e_j(n) \quad (5.27)$$

$$\frac{\partial e_j(n)}{\partial y_j(n)} = -1 \quad (5.28)$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'(v_j(n)) \quad (5.29)$$

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = y_i(n) \quad (5.30)$$

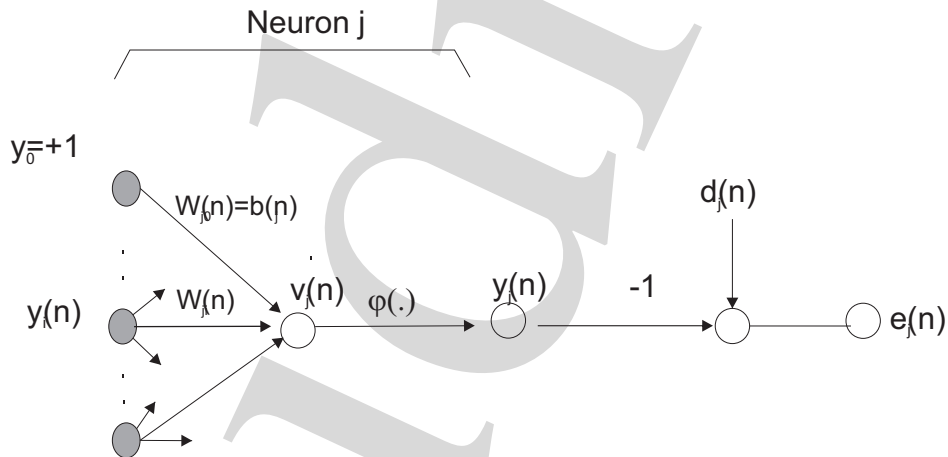
Dengan menggabungkan persamaan-persamaan di atas, kita dapatkan

$$\delta_j(n) = e_j(n)\varphi'(v_j(n)) \quad (5.31)$$

Sehingga bisa diringkas delta rule untuk neuron output  $j$ :

$$\begin{aligned} \Delta w_{ji}(n) &= -\eta \frac{\partial E(n)}{\partial w_{ji}} \\ &= \eta \delta_j(n) y_i(n) \end{aligned} \quad (5.32)$$

atau



**Gambar 5.5:** Mekanisme back-propagasi pada layer output.

### 5.7.2 Kasus 2, prosedur training untuk unit hidden

Kita definisikan lagi local gradient untuk neuron  $j$  adalah

$$\delta_j(n) = -\frac{\partial E}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)} \quad (5.33)$$

dimana  $\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'(v_j(n))$ . Sedangkan fungsi cost didefinisikan sebagai

$$E = \frac{1}{2} \sum_k (e_k)^2(n) \quad (5.34)$$

dimana neuron  $k$  adalah neuron output. Turunan dari persamaan 5.34 terhadap signal  $y_j(n)$

$$\begin{aligned} \frac{\partial E(n)}{\partial y_j(n)} &= \sum_k e_k(n) \frac{\partial e_k(n)}{\partial y_j(n)} \\ &= \sum_k e_k(n) \frac{\partial}{\partial y_j(n)} (d_k(n) - y_k(n)) \\ &= - \sum_k e_k(n) \frac{\partial y_k(n)}{\partial y_j(n)} \\ &= - \sum_k e_k(n) \frac{\partial y_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \end{aligned} \quad (5.35)$$

Selanjutnya

$$v_k(n) = \sum_{j=0}^m w_{kj}(n) y_j(n) \quad (5.36)$$

$$\frac{\partial v_k(n)}{\partial y_j(n)} = w_{kj}(n) \quad (5.37)$$

$$v_k(n) = \varphi(v_k(n)) \quad (5.38)$$

Sedangkan

$$\frac{\partial y_k(n)}{\partial v_k(n)} = \varphi'(v_k(n)) \quad (5.39)$$

Sehingga  $\delta_j$  untuk unit hidden

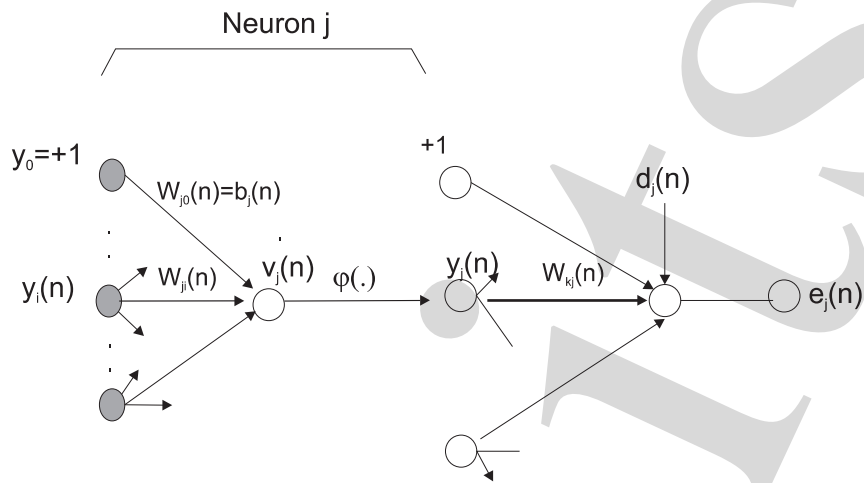
$$\delta_j(n) = \varphi'(v_k(n)) \sum_k e_k(n) \varphi'(v_k(n)) w_{kj}(n) \quad (5.40)$$

Tetapi dari definisi sebelumnya kita tahu bahwa

$$e_k(n) \varphi'(v_k(n)) = \delta_j(n) \quad (5.41)$$

Untuk mengupdate  $w_{ji}$  bisa digunakan

$$\delta_j(n) = \varphi'(v_k(n)) \sum_k \delta_k(n) w_{kj}(n) \quad (5.42)$$



**Gambar 5.6:** Mekanisme back-propagation pada layer hidden.

Berikut adalah ringkasan bagaimana back-propagasi bekerja.

### Feedforward

Hitung signal input

$$z\_in = bias + \sum wx$$

terapkan fungsi aktivasi

$$z_n = \varphi(z\_in)$$

dan hasilnya merupakan input bagi unit untuk layer berikutnya.

Hitung signal input

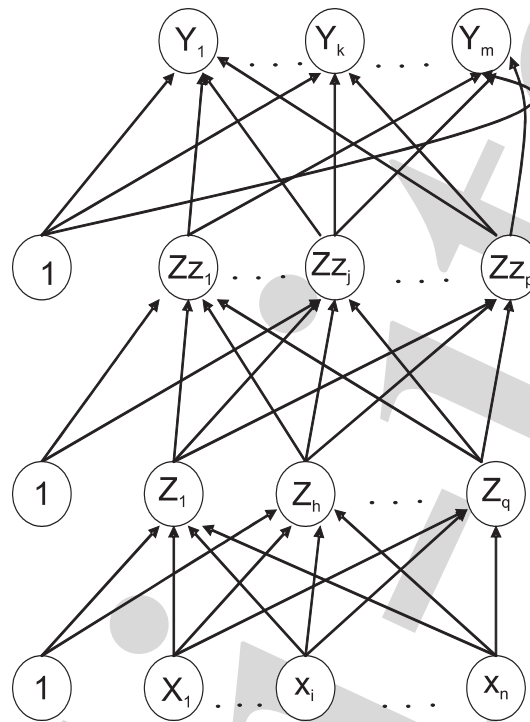
$$zz\_in_j = v_{0j} + \sum_{h=1}^q z_h v_{hj}$$

Terapkan fungsi aktivasi

$$zz_j = \varphi(zz\_in_j)$$

lalu hasilnya digunakan untuk input bagi simpul output Jumlahkan signal input

$$y\_in = w_0 + \sum_{j=1}^p zz_j w_j$$



**Gambar 5.7:** Back-propagasi network dengan 2 hidden layer.

Terapkan fungsi aktivasi

$$y = \varphi(y_{in})$$

### Back-propagasi

Untuk setiap unit  $Y_k$

$$e = d - y$$

Untuk titik data yang sekarang

- kalikan dengan turunan dari fungsi aktivasi

$$\delta = e\varphi'(y_{in})$$

- Hitung perubahan bobot

$$\Delta w_j = \eta \delta z z_j$$

- Hitung perubahan bias

$$\Delta w_0 = \eta \delta$$

- dan  $\delta$  dikirim ke unit hidden  $ZZ_j$

Untuk setiap unit hidden  $ZZ_j, j = 1, ..p$ .

- Jumlahkan input dengan bobot dari layer di atas

$$\delta_{in_j} = \sum_{j=1}^p \delta w_j$$

- lalu dikalikan dengan turunan fungsi aktivasinya hasilnya

$$\delta_j = \delta_{in_j} \varphi'(zz_{in_j})$$

- Hitung koreksi bobot untuk mengupdate  $v_{ij}$

$$\Delta v_{ij} = \eta \delta_j x_i$$

- Hitung koreksi bias untuk mengupdate  $v_{0j}$

$$\Delta v_{0j} = \eta \delta_j$$

- dan masukkan  $\delta_j$  ke unit hidden  $(Z_h, h = 1, .., q)$ .

Untuk setiap unit hidden  $(Z_h, h = 1, .., q)$ :

- jumlahkan input berbobot dari unit-unit di layer di atas untuk mendapatkan

$$\Delta_{in_h} = \sum_{j=1}^p \delta v_{hj}$$

- kalikan dengan turunan fungsi aktivasinya

$$\delta_h = \Delta_{in_h} \varphi'(z_{in_h})$$

- Hitung koreksi bobot untuk mengupdate  $v_{ij}$

$$\Delta u_{ih} = \eta \delta_h x_i$$



- Hitung koreksi bias untuk mengupdate  $v_{0j}$

$$\Delta v_{0h} = \eta \delta_h$$

#### Update untuk bobot dan bias

Untuk setiap unit output ( $j = 0, \dots, p; k = 1, \dots, m$ );

$$w_{jk}(\text{new}) = w_{jk}(\text{old}) + \delta w_{jk}$$

Untuk setiap unit hidden  $ZZ_j(h = 0, \dots, q; j = 1, \dots, p)$ ;

$$v_{hj}(\text{new}) = v_{hj}(\text{old}) + \delta v_{hj}$$

Untuk setiap unit hidden  $Z_h(i = 0, \dots, n; h = 1, \dots, q)$ ;

$$u_{ih}(\text{new}) = u_{ih}(\text{old}) + \delta u_{ih}$$

## 5.8 Implementasi ANN dengan Matlab

Paket software Matlab mempunyai toolbox khusus yang digunakan untuk implementasi neural networks. Dalam paket ini, toolbox neural networks memberikan beberapa pilihan tentang model networks yang ingin dipakai, ukuran error, metoda training, metoda pre-processing untuk data dan lain-lain. Toolbox adalah kumpulan dari fungsi atau subroutine yang dibuat untuk suatu bidang terapan tertentu. Misalkan kita mempunyai data sebagai berikut:

```
>>P = [0 1 2 3 4 5 6 7 8 9 10];
>>T = [0 1 2 3 4 3 2 1 2 3 4];
```

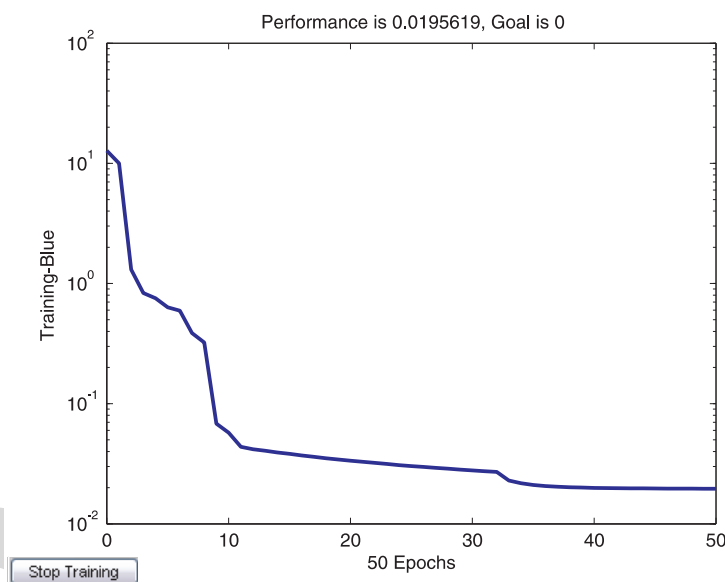
Kita lihat bahwa format yang digunakan sedikit berbeda dengan format yang kita pakai selama ini. Dalam hal ini kita menggunakan format kolom. Kolom menyatakan observasi atau pengamatan. Di sini kita mempunyai 11 observasi dalam satu dimensi dan 11 output ( $T$ ) yang berhubungan dengan setiap input  $P$ . Kita akan memakai desain ANN feedforward back propagasi dengan memakai suatu fungsi dalam Matlab. Perintah berikut adalah untuk menciptakan network feedforward

```
>> net = newff([0 10],[5 1],{'tansig' 'purelin'});
```

Dengan perintah di atas berarti kita membuat network dengan input yang mempunyai nilai minimum 0 dan maksimum 10, jumlah neuron hidden sebanyak 5 dan neuron output 1 buah. Kita menggunakan fungsi aktivasi (fungsi transfer) tansig atau tangent sigmoid untuk neuron hidden dan fungsi aktivasi linier untuk neuron output. Selanjutnya network yang sudah kita buat bisa kita train dengan 50 iterasi atau epoch.

```
>>net.trainParam.epochs = 50;%menentukan jumlah iterasi
>>net = train(net,P,T);%mentrain network dengan
data input P dan output T
```

Sesudah perintah di atas akan dihasilkan plot mean squared error (MSE) yang dihasilkan sebagai fungsi dari jumlah epoch atau iterasi. Gambar 5.8 memperlihatkan hasil tersebut. Setelah kita mentrain network dengan 50 iterasi, kita

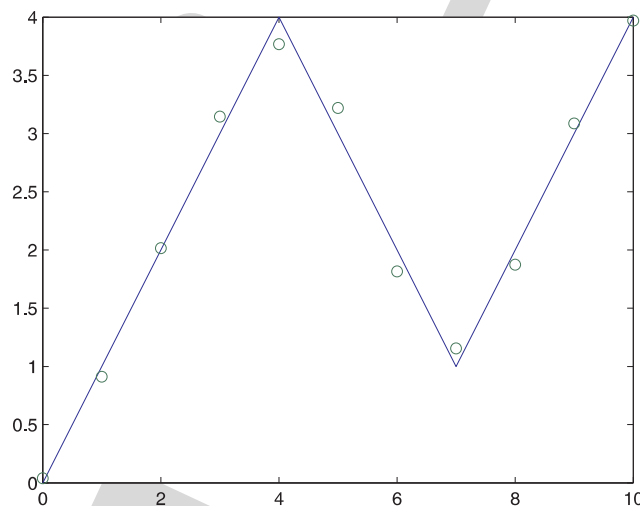


**Gambar 5.8:** Plot dari MSE yang dihasilkan oleh Matlab. Sumbu-x menunjukkan jumlah epoch dan sumbu-y menunjukkan besarnya MSE

simulasikan untuk melihat output yang dihasilkan dengan perintah berikut

```
>>Y = sim(net,P);%mensimulasikan network net  
yang sudah kita train dengan data input P.  
>>plot(P,T,P,Y,'o')% membuat plot untuk melihat  
hasil simulasi Y dan output aktual T
```

Simbol % adalah perintah dalam Matlab untuk tidak mengeksekusi apa pun yang tertulis sesudahnya. Jadi rangkaian kata-kata sesudah tanda % adalah hanya sekedar komentar. Kita menyimpan output hasil prediksi dari network dengan variabel *Y*. Kemudian kita plot hasil simulasi dengan data output sesungguhnya seperti ditunjukkan dalam Gambar 5.9. Perbandingan antara



**Gambar 5.9:** Plot output aktual dan hasil prediksi. Hasil prediksi ditunjukkan dengan lingkaran kecil.

nilai prediksi dan aktual bisa dilihat secara numerik dengan cara memberi perintah berikut

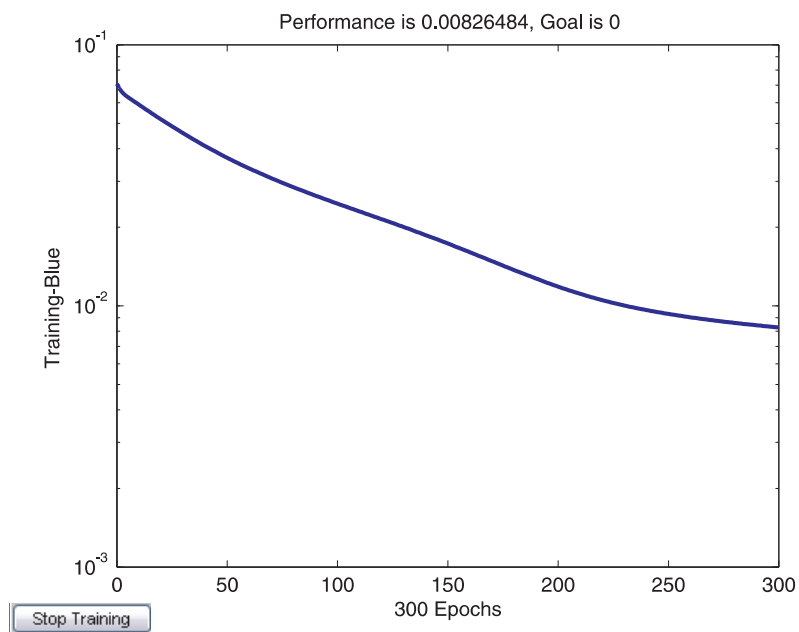
```
>> [Y' T']  
ans =  
0.0402    0
```

0.9115	1.0000
2.0157	2.0000
3.1445	3.0000
3.7667	4.0000
3.2191	3.0000
1.8160	2.0000
1.1554	1.0000
1.8744	2.0000
3.0860	3.0000
3.9690	4.0000

Misalkan kita mempunyai data input baru bernilai 5.5 dan kita ingin melihat berapa nilai outputnya. Kita bisa jalankan perintah berikut ini

```
>> Y = sim(net,5.5)
Y =
    2.5339
```

Jadi output untuk input yang bernilai 5.5 adalah 2.5339. Contoh berikut ini mengenai prediksi data time series. Dalam data time series biasanya variabel outputnya berupa data berurutan yang disusun menurut waktu. Sebagai contoh misalkan kita mempunyai data harga tepung bulanan di suatu kota dari bulan Januari 1999 sampai bulan Januari 2000. Data bulan diwakili angka 1 sampai 13 sebagai variabel prediktor dan harga tepung untuk tiap bulan merupakan data respon atau variabel dependent. Dalam contoh ini kita mempunyai data time series harga tepung di suatu kota di AS. Kita ingin menggunakan 20 data pertama untuk menemukan aproksimasi fungsi. Kemudian 2 data yang tersisa kita pergunakan untuk testing. Pada contoh ini kita lakukan pre-processing data sebelum dimasukkan dalam ANN. Pre-processing yang kita gunakan adalah mengubah data sehingga data terletak antar -1 dan 1. Setelah dilakukan training dan simulasi, maka output yang dihasilkan harus dikembalikan lagi ke skala semula. Pengembalian data ke dalam skala aslinya disebut dengan post-processing. Gambar 5.10 menunjukkan plot MSE untuk tahap training dengan menggunakan 20 data.



**Gambar 5.10:** Perkembangan MSE dalam fase training

X=

- 1 107.1
- 2 113.5
- 3 112.7
- 4 114.7
- 5 123.4
- 6 123.6
- 7 116.3
- 8 118.5
- 9 119.8
- 10 120.3
- 11 127.4
- 12 125.1
- 13 127.6
- 14 129

```

15 124.6
16 134.1
17 146.5
18 171.2
19 178.6
20 172.2
21 171.5
22 163.6

```

```

>>P=X(1:20,1);
>>T=X(1:20,2);
>>s=X(21:22,1);
>>y=X(21:22,2);
>> [pn,minp,maxp,tn,mint,maxt]=premnmx(P',T');%pre-processing
>> [an,mina,maxa,sn,mins,maxs]=premnmx(s',y');%pre-processing
>> net=newff(minmax(pn),[5 1],{'tansig','tansig'},'traingdm');
%menggunakan gradient decent with momentum
>> net.trainParam.epochs=300;
>> net.trainParam.lr=0.3;%learning rate
>> net.trainParam.mc=0.6;%momentum
>> net=train (net,pn,tn);

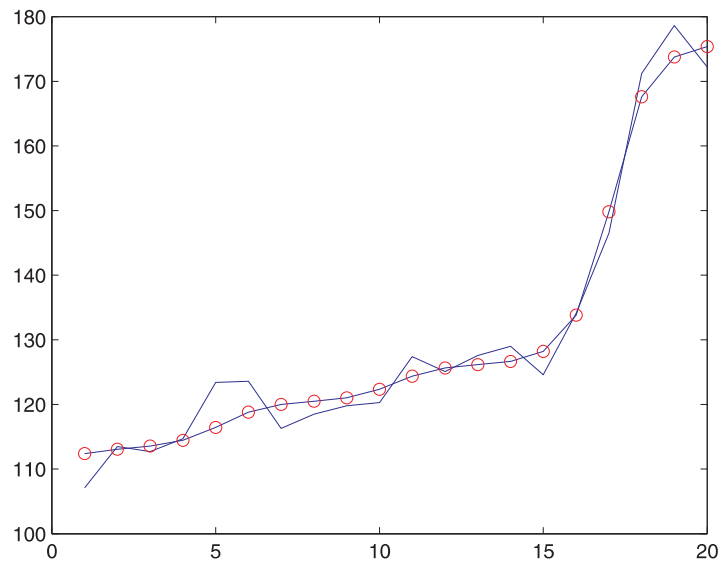
```

Selain itu, kita bisa juga melakukan analisis regresi linier untuk melihat bagaimana hubungan antara hasil prediksi dan data aktualnya dalam bentuk persamaan regresi linier. Jadi di sini kita akan menghitung nilai slope,  $m$  dan nilai intercept,  $b$  dan nilai koefisien korelasi,  $r$ . Berikut ini perintah dalam Matlab untuk melakukan analisis regresi linier terhadap output dari ANN. Gambar 5.12 menggambarkan garis regresi linier antara data akktual dan nilai prediksi.

```

>>yt=sim(net,pn);
>> x=postmnmx(yt',mint,maxt);%post-processing
>> [m,b,r]=postreg(x',T')
m =
    0.9832

```



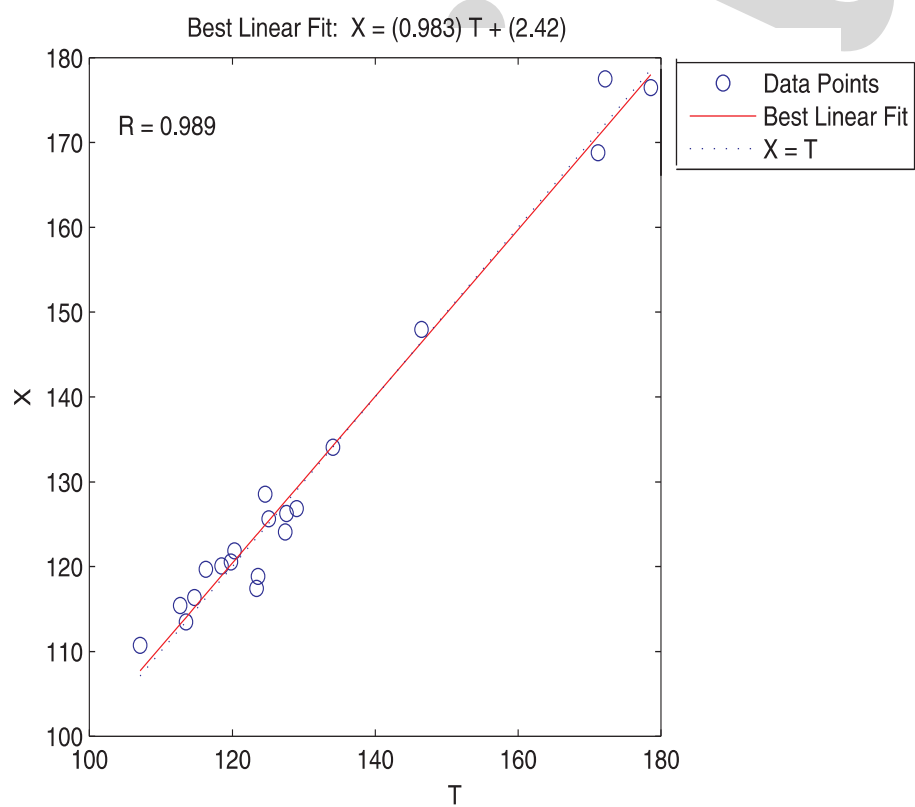
**Gambar 5.11:** Perbandingan data aktual dan hasil prediksi (o) untuk tahap training

```
b =
    2.4175
r =
    0.9894
```

Berikut ini hasil prediksi untuk data testing.

```
>> yt=sim(net,an)
yt =
   -0.8515    0.9098
>> x=postmmx(yt',mins,maxs);%post-processing
x =
   164.1865
   171.1438
```

Yang terakhir di atas adalah hasil prediksi untuk observasi ke 21 dan 22.



**Gambar 5.12:** Analisis regresi linier hasil prediksi dan data aktual



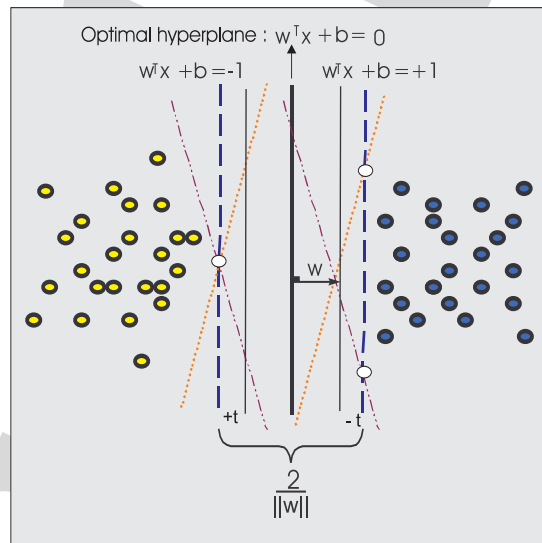
## 6.1 Ide Dasar Support Vector Machine

Support vector machine (SVM) adalah suatu teknik yang relatif baru (1995) untuk melakukan prediksi, baik dalam kasus klasifikasi maupun regresi, yang sangat populer belakangan ini. SVM berada dalam satu kelas dengan ANN dalam hal fungsi dan kondisi permasalahan yang bisa diselesaikan. Keduanya masuk dalam kelas *supervised learning*. Dalam teknik ini, kita berusaha untuk menemukan *fungsi pemisah* (klasifier) yang optimal yang bisa memisahkan dua set data dari dua kelas yang berbeda Vapnik (1995). Teknik ini menarik orang dalam bidang data mining maupun machine learning karena performansinya yang meyakinkan dalam memprediksi kelas suatu data baru. Kita akan memulai pembahasan dengan kasus klasifikasi yang secara linier bisa dipisahkan. Dalam hal ini fungsi pemisah yang kita cari adalah fungsi linier. Fungsi ini bisa didefinisikan sebagai

$$g(x) := \text{sgn}(f(x)) \quad (6.1)$$

dengan  $f(x) = w^T x + b$ ,

dimana  $x, w \in \mathbb{R}^n$  and  $b \in \mathbb{R}$ . Masalah klasifikasi ini bisa dirumuskan sebagai berikut: kita ingin menemukan set parameter  $(w, b)$  sehingga  $f(x_i) = \langle w, x \rangle + b = y_i$  untuk semua  $i$ . Dalam teknik ini kita berusaha menemukan *fungsi pemisah* (klasifier/hyperplane) terbaik diantara fungsi yang tidak terbatas jumlahnya untuk memisahkan dua macam obyek. Hyperplane terbaik adalah hyperplane yang terletak di tengah-tengah antara dua set obyek dari dua kelas. Mencari hyperplane terbaik ekuivalen dengan memaksimalkan margin atau jarak antara dua set obyek dari kelas yang berbeda. Jika  $w x_1 + b = +1$  adalah hyperplane-pendukung (supporting hyperplane) dari kelas  $+1$  ( $w x_1 + b = +1$ ) dan  $w x_2 + b = -1$  hyperplane-pendukung dari kelas  $-1$  ( $w x_2 + b = -1$ ), margin antara dua kelas dapat dihitung dengan mencari jarak antara kedua hyperplane-pendukung dari kedua kelas. Secara spesifik, margin dihitung dengan cara berikut  $(w x_1 + b = +1) - (w x_2 + b = -1) \Rightarrow w(x_1 - x_2) = 2 \Rightarrow \left( \frac{w}{\|w\|} (x_1 - x_2) \right) = \frac{2}{\|w\|}$ . Gambar 6.1 memperlihatkan bagaimana SVM bekerja untuk menemukan suatu fungsi pemisah dengan margin yang maksimal.



**Gambar 6.1:** Mencari fungsi pemisah yang optimal untuk obyek yang bisa dipisahkan secara linier

## 6.2 Formulasi Matematis

Secara matematika, formulasi problem optimisasi SVM untuk kasus klasifikasi linier di dalam *primal space* adalah

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|^2 \\ \text{Subject to} \quad & y_i(wx_i + b) \geq 1, \quad i = 1, \dots, \ell, \end{aligned} \quad (6.2)$$

dimana  $x_i$  adalah data input,  $y_i$  adalah keluaran dari data  $x_i$ ,  $w, b$  adalah parameter-parameter yang kita cari nilainya. Dalam formulasi di atas, kita ingin meminimalkan *fungsi tujuan* (obyektif function)  $\frac{1}{2} \|w\|^2$  atau memaksimalkan kuantitas  $\|w\|^2$  atau  $w^T w$  dengan memperhatikan pembatas  $y_i(wx_i + b) \geq 1$ . Bila output data  $y_i = +1$ , maka pembatas menjadi  $(wx_i + b) \geq 1$ . Sebaliknya bila  $y_i = -1$ , pembatas menjadi  $(wx_i + b) \leq -1$ . Di dalam kasus yang tidak feasible (infeasible) dimana beberapa data mungkin tidak bisa dikelompokkan secara benar, formulasi matematikanya menjadi berikut

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{\ell} t_i \\ \text{Subject to} \quad & y_i(wx_i + b) + t_i \geq 1 \\ & t_i \geq 0, \quad i = 1, \dots, \ell, \end{aligned} \quad (6.3)$$

dimana  $t_i$  adalah variabel *slack*. Dengan formulasi ini kita ingin memaksimalkan margin antara dua kelas dengan meminimalkan  $\|w\|^2$  (Haykin, 1999). Dalam formulasi ini kita berusaha meminimalkan *kesalahan klasifikasi* (misclassification error) yang dinyatakan dengan adanya variabel *slack*  $t_i$ , sementara dalam waktu yang sama kita memaksimalkan margin,  $\frac{1}{\|w\|}$ . Penggunaan variabel slack  $t_i$  adalah untuk mengatasi kasus ketidaklayakan (infeasibility) dari pembatas (constraints)  $y_i(wx_i + b) \geq 1$  dengan cara memberi penalti untuk data yang tidak memenuhi pembatas tersebut. Untuk meminimalkan nilai  $t_i$  ini, kita berikan penalti dengan menerapkan konstanta ongkos  $C$ . Vektor  $w$  tegak lurus terhadap fungsi pemisah:  $wx + b = 0$ . Konstanta  $b$  menentukan lokasi fungsi pemisah relatif terhadap titik asal (origin).

Problem (6.3) adalah *programa nonlinear*. Ini bisa dilihat dari *fungsi tujuan* (objective function) yang berbentuk kuadrat. Untuk menyelesaikannya, secara komputasi agak sulit dan perlu waktu lebih panjang. Untuk membuat masalah ini lebih mudah dan efisien untuk diselesaikan, masalah ini bisa kita transformasikan ke dalam *dual space*. Untuk itu, pertama kita ubah problem (6.3) menjadi *fungsi Lagrangian* :

$$J(w, b, \alpha) = \frac{1}{2}w^T w - \sum_{i=1}^{\ell} \alpha_i [y_i(w^T x_i + b) - 1] \quad (6.4)$$

dimana variabel *non-negatif*  $\alpha_i$ , dinamakan *Lagrange multiplier*. Solusi dari problem optimisasi dengan pembatas seperti di atas ditentukan dengan mencari *saddle point* dari fungsi Lagrangian  $J(w, b, \alpha)$ . Fungsi ini harus diminimalkan terhadap variabel  $w$  dan  $b$  dan harus dimaksimalkan terhadap variabel  $\alpha$ . Kemudian kita cari turunan pertama dari fungsi  $J(w, b, \alpha)$  terhadap variabel  $w$  dan  $b$  dan kita samakan dengan 0. Dengan melakukan proses ini, kita akan mendapatkan dua kondisi optimalitas berikut:

1. kondisi 1:

$$\frac{\partial J(w, b, \alpha)}{\partial w} = 0$$

2. kondisi 2:

$$\frac{\partial J(w, b, \alpha)}{\partial b} = 0$$

Penerapan kondisi optimalitas 1 pada fungsi Lagrangian (6.4) akan menghasilkan

$$w = \sum_{i=1}^{\ell} \alpha_i y_i x_i \quad (6.5)$$

Penerapan kondisi optimalitas 2 pada fungsi Lagrangian (6.4) akan menghasilkan

$$\sum_{i=1}^{\ell} \alpha_i y_i = 0 \quad (6.6)$$

Menurut duality theorem (Bertsekas, 1995):

1. Jika problem primal mempunyai solusi optimal, maka problem dual juga akan mempunyai solusi optimal yang nilainya sama

2. Bila  $w_o$  adalah solusi optimal untuk problem primal dan  $\alpha_o$  untuk problem dual, maka *perlu* dan *cukup* bahwa  $w_o$  solusi layak untuk problem primal dan

$$\Phi(w_o) = J(w_o, b_o, \alpha_o) = \min_w J(w, b, \alpha)$$

Untuk mendapatkan problem dual dari problem kita, kita jabarkan persamaan (6.4) sebagai berikut:

$$J(w, b, \alpha) = \frac{1}{2}w^T w - \sum_{i=1}^{\ell} \alpha_i y_i w^T x_i - b \sum_{i=1}^{\ell} \alpha_i y_i + \sum_{i=1}^{\ell} \alpha_i \quad (6.7)$$

Menurut kondisi optimalitas ke dua dalam (6.6), term ketiga sisi sebelah kanan dalam persamaan di atas sama dengan 0. Dengan memakai nilai-nilai  $w$  di (6.5), kita dapatkan

$$w^T w = \sum_{i=1}^{\ell} \alpha_i y_i w^T x_i = \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j x_i^T x_j \quad (6.8)$$

maka persamaan 6.7 menjadi

$$Q(\alpha) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j x_i^T x_j \quad (6.9)$$

Selanjutnya kita dapatkan formulasi dual dari problem (6.3):

$$\begin{aligned} \max \quad & \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{\ell} y_i y_j \alpha_i \alpha_j x_i^T x_j \\ \text{Subject to} \quad & \sum_{i=1}^{\ell} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i, i = 1, \dots, \ell, \end{aligned} \quad (6.10)$$

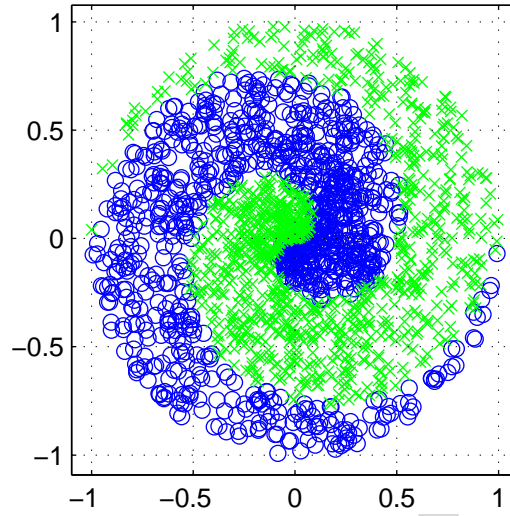
dimana  $k$  adalah fungsi kernel yang dijelaskan dalam bagian 6.3. Formulasi (6.10) adalah quadratic programming (QP) dengan pembatas (constraint) linier. Melatih SVM ekuivalen dengan menyelesaikan problem *convex optimization*. Karena itu solusi dari SVM adalah unik (dengan asumsi bahwa  $k$  adalah positive definite) dan global optimal. Hal ini berbeda dengan solusi neural networks (Haykin, 1999) yang ekuivalen dengan problem *non-convex optimization* dengan akibat solusi yang ditemukan adalah *local optima*. Ambil  $f(x) = \sum_{i=1}^{\ell} y_i \alpha_i^* k(x_i, x) + b^*$ . Fungsi pemisah optimal adalah

$g(x) = \text{sign}(\sum_{i=1}^{\ell} y_i \alpha_i^* k(x, x_i)) + b^*$ , dimana  $\alpha_i^*, i = 1, \dots, \ell$  adalah solusi optimal dari problem (6.10) dan  $b^*$  dipilih sehingga  $y_i f(x_i) = 1$  untuk sembarang  $i$  dengan  $C > \alpha_i^* > 0$  (Cristianini and Shawe-Taylor, 2000). Data  $x_i$  dimana  $\alpha_i^* > 0$  dinamakan *support vector* dan menyatakan data training yang diperlukan untuk mewakili fungsi keputusan yang optimal. Dalam gambar 6.1, sebagai contoh, 3 titik berwarna putih menyatakan *support vector*. Untuk mengatasi masalah ketidaklinieran (nonlinearity) yang sering terjadi dalam kasus nyata, kita bisa menerapkan metoda kernel. Metoda kernel (Schölkopf and Smola, 2002) memberikan pendekatan alternatif dengan cara melakukan mapping data  $x$  dari input space ke *feature space*  $F$  melalui suatu fungsi  $\varphi$  sehingga  $\varphi : x \mapsto \varphi(x)$ . Karena itu suatu titik  $x$  dalam input space menjadi  $\varphi(x)$  dalam feature space.

### 6.3 Metoda Kernel

Bila suatu kasus klasifikasi memperlihatkan ketidaklinieran, algoritma seperti perceptron tidak bisa mengatasinya. Secara umum, kasus-kasus di dunia nyata adalah kasus yang tidak linier. Metoda kernel (Schölkopf and Smola, 2002) adalah salah satu untuk mengatasinya. Dengan metoda kernel suatu data  $x$  di input space dimapping ke feature space  $F$  dengan dimensi yang lebih tinggi melalui map  $\varphi$  sebagai berikut  $\varphi : x \mapsto \varphi(x)$ . Karena itu data  $x$  di input space menjadi  $\varphi(x)$  di feature space.

Sering kali fungsi  $\varphi(x)$  tidak tersedia atau tidak bisa dihitung. tetapi *dot product* dari dua vektor dapat dihitung baik di dalam *input space* maupun di *feature space*. Dengan kata lain, sementara  $\varphi(x)$  mungkin tidak diketahui, dot product  $\langle \varphi(x_1), \varphi(x_2) \rangle$  masih bisa dihitung di feature space. Untuk bisa memakai metoda kernel, pembatas (constraint) perlu diekspresikan dalam bentuk dot product dari vektor data  $x_i$ . Sebagai konsekuensi, pembatas yang menjelaskan permasalahan dalam klasifikasi harus diformulasikan kembali sehingga menjadi bentuk dot product. Dalam feature space ini dot product  $\langle . \rangle$  menjadi  $\langle \varphi(x), \varphi(x)' \rangle$ . Suatu fungsi kernel,  $k(x, x')$ , bisa untuk menggantikan dot product  $\langle \varphi(x), \varphi(x)' \rangle$ . Kemudian di feature space, kita



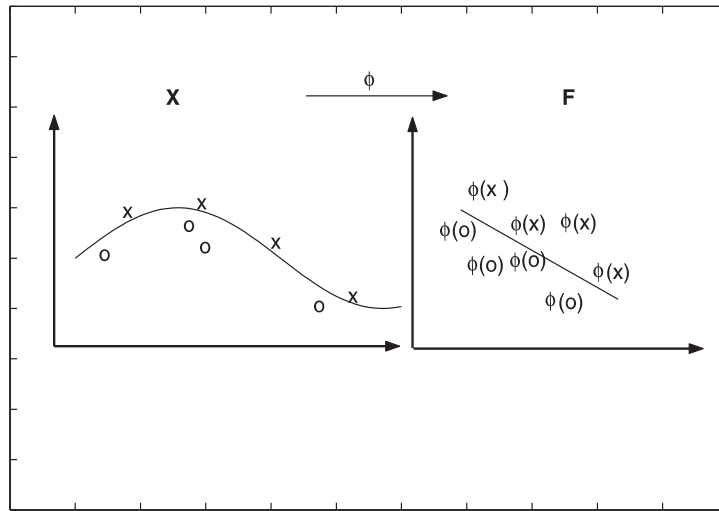
**Gambar 6.2:** Data spiral yang menggambarkan ketidaklinieran

bisa membuat suatu fungsi pemisah yang linier yang mewakili fungsi nonlinear di input space. Gambar 6.3 mendeskripsikan suatu contoh feature mapping dari ruang dua dimensi ke feature space dua dimensi. Dalam input space, data tidak bisa dipisahkan secara linier, tetapi kita bisa memisahkan di feature space. Karena itu dengan memetakan data ke feature space menjadikan tugas klasifikasi menjadi lebih mudah (Schölkopf and Smola, 2002).

Fungsi kernel yang biasanya dipakai dalam literatur SVM (Haykin, 1999):

- linier:  $x^T x$ ,
- Polynomial:  $(x^T x_i + 1)^p$ ,
- Radial basis function (RBF):  $\exp(-\frac{1}{2\sigma^2} \|x - x_i\|^2)$ ,
- Tangent hyperbolic (sigmoid):  $\tanh(\beta x^T x_i + \beta_1)$ , dimana  $\beta, \beta_1 \in \Re$

Fungsi kernel yang legitimate diberikan oleh Teori Mercer (Vapnik, 1995) dimana fungsi itu harus memenuhi syarat: kontinu dan positive definite. Lebih mudah menemukan fungsi kernel daripada mencari map  $\varphi$  seperti apa yang tepat untuk melakukan mapping dari *input space* ke *feature space*.



**Gambar 6.3:** Suatu kernel map mengubah problem yang tidak linier menjadi linier dalam space baru

## 6.4 Implementasi

Contoh yang kedua adalah contoh untuk problem yang tidak bisa dipisahkan secara linier. Untuk contoh ini kita gunakan problem *Exclusive OR* (XOR). Problem AND adalah klasifikasi dua kelas dengan empat data (lihat tabel 6.1). Karena ini problem linier, kernelisasi tidak diperlukan. Menggunakan data di

**Tabel 6.1:** AND Problem

$x_1$	$x_2$	$y$
1	1	1
-1	1	-1
1	-1	-1
-1	-1	-1

Tabel 6.1, kita dapatkan formulasi masalah optimisasi sebagai berikut:

$$\min \frac{1}{2}(w_1^2 + w_2^2) + C(t_1 + t_2 + t_3 + t_4) \quad (6.11)$$



Subject to

$$\begin{aligned} w_1 + w_2 + b + t_1 &\geq 1 \\ w_1 - w_2 - b + t_2 &\geq 1 \\ -w_1 + w_2 - b + t_3 &\geq 1 \\ w_1 + w_2 - b + t_4 &\geq 1 \\ t_1, t_2, t_3, t_4 &\geq 0 \end{aligned}$$

Karena fungsi AND adalah kasus klasifikasi linier, maka bisa dipastikan nilai variable slack  $t_i = 0$ . Jadi Kita bisa masukkan nilai  $C = 0$ . Setelah menyelesaikan problem optimisasi di atas didapat solusi

$$w_1 = 1, w_2 = 1, b = -1$$

Persamaan fungsi pemisahannya adalah

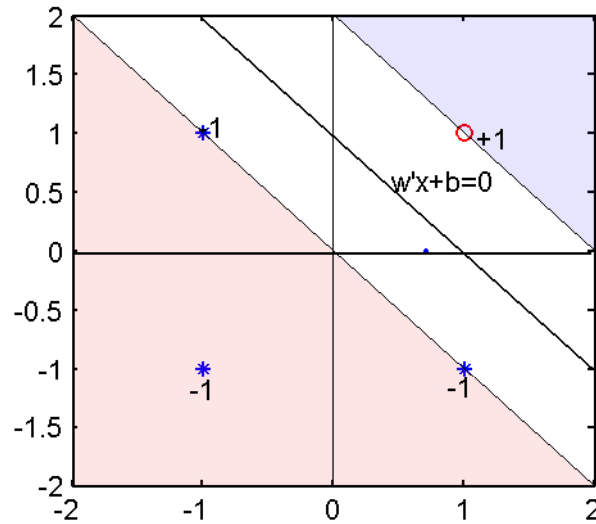
$$f(x) = x_1 + x_2 - 1.$$

Untuk menentukan output atau label dari setiap titik data/obyek kita gunakan fungsi  $g(x) = \text{sign}(x)$ . Dengan fungsi sign ini semua nilai  $f(x) < 0$  diberi label  $-1$  dan lainnya diberi label  $+1$ . Secara grafis fungsi pemisah ini diperlihatkan dalam Gambar 6.4.

**Tabel 6.2:** XOR Problem

$x_1$	$x_2$	$y$
1	1	-1
-1	1	1
1	-1	1
-1	-1	-1

Dalam kasus XOR (lihat Tabel 6.2), data tidak bisa dipisahkan secara linier. Untuk mengatasi masalah ketidaklinieran, kita perlu memformulasi SVM dalam *dual space*. Untuk itu kita perlu mengganti  $w$  dengan  $\sum_{i=1}^{\ell} \alpha_i y_i \varphi(x_i)$ . Kemudian kita perlu melakukan kernelisasi sehingga kita bisa mendapatkan



**Gambar 6.4:** Ilustrasi bagaimana data dipisahkan dalam kasus AND.

fungsi linier di dalam *feature space*. Untuk itu kita gunakan fungsi polynomial kernel pangkat 2 yang didefinisikan sebagai  $K(x_i, x_i) = (x_i x_i' + 1)^2$  (Haykin, 1999). Dengan kernel ini, kita bisa menghitung matriks kernel  $K$  dengan dimensi  $\ell \times \ell$ , dimana  $\ell$  adalah banyaknya data. Memakai data di Tabel 6.1, sebagai contoh, bisa dihitung  $K(1, 1)$  dan  $K(1, 2)$ , sebagai berikut:

$$x_1 = [1 \ 1], \ x_2 = [-1 \ 1]$$

$$x_1 x_1' = [1 \ 1] [1 \ 1]' = 2$$

$$(x_1 x_1' + 1)^2 = 9$$

$$x_1 x_2' = [1 \ 1] [-1 \ 1]' = 0$$

$$(x_1 x_2' + 1)^2 = 1$$

Dengan prosedur yang sama untuk semua nilai  $x_i$ , kita dapatkan nilai matriks  $K$  sebagai berikut:

$$K = \begin{bmatrix} 9 & 1 & 1 & 1 \\ 1 & 9 & 1 & 1 \\ 1 & 1 & 9 & 1 \\ 1 & 1 & 1 & 9 \end{bmatrix} \quad (6.12)$$

Dengan menggunakan matriks  $K$  sebagai pengganti dot product  $x_i x_j$  dalam 6.10 maka kita dapatkan formulasi berikut

$$\begin{aligned} \max \quad & \alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 - \frac{1}{2}(9\alpha_1^2 - 2\alpha_1\alpha_2 - \alpha_1\alpha_3 + \alpha_1\alpha_4 \\ & + 9\alpha_2^2 + 2\alpha_2\alpha_3 - \alpha_2\alpha_4 + 9\alpha_3^2 - 2\alpha_3\alpha_4 + 9\alpha_4^2) \quad (6.13) \\ \text{Subject to} \quad & -\alpha_1 + \alpha_2 + \alpha_3 - \alpha_4 = 0 \\ & \alpha_i \geq 0 \end{aligned}$$

Dalam fungsi tujuan 6.13, term kedua kita kalikan dengan  $y_i y_j$ . Persamaan 6.13 memenuhi bentuk standar *programa kuadratik* (quadratic programming, QP). Sehingga masalah ini bisa diselesaikan dengan solver komersial untuk QP. Penyelesaian problem di atas dengan bantuan software akan memberi hasil sebagai berikut:

$$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = \frac{1}{8}$$

Hasil ini menunjukkan bahwa semua data dalam contoh ini adalah *support vector*.

Setelah kita training SVM, kita bisa menentukan label untuk data testing dengan menggunakan fungsi pemisah sebagai berikut

$$f(x) = y\alpha'K(x_{test}, x_{train}) + b,$$

dimana vektor  $\alpha$  dan konstanta  $b$  diketahui dari hasil training. hasil ini dijelaskan secara grafis dalam Gambar 6.5. Perlu dijelaskan di sini bahwa nilai  $w$  tidak selalu bisa diekspresikan secara eksplisit sebagai kombinasi antara  $\alpha, y$  dan  $\varphi(x)$ , karena dalam banyak kasus  $\varphi(x)$  tidak diketahui atau sulit dihitung, kecuali dalam kasus kernel linier dimana  $\varphi(x) = x$ .

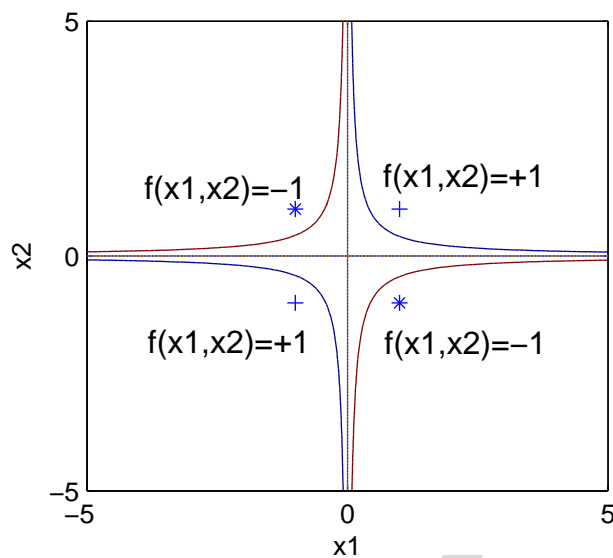
### Algoritma SVM untuk klasifikasi

#### **Variabel dan parameter**

$x = \{x_0, x_1, x_2, \dots, x_m\}$ : sampel training

$y = \{y_1, \dots, y_m\} \subset \{\pm 1\}$ : label data training

kernel: jenis fungsi kernel



**Gambar 6.5:** Ilustrasi bagaimana data dipisahkan dalam kasus XOR.

par: parameter kernel

C: konstanta cost

$\alpha = [\alpha_1, \dots, \alpha_m]$ : Lagrange multiplier dan bias  $b$

1. Hitung matriks kernel  $Q$
2. Tentukan pembatas untuk program kuadrat, termasuk  $A_{eq}$ ,  $b_{eq}$ ,  $A$  dan  $b$
3. Tentukan fungsi tujuan program kuadrat  $\frac{1}{2}xQx + f'x$
4. Selesaikan masalah QP dan temukan solusi  $\alpha$  dan  $b$

## 6.5 Implementasi SVM dengan Matlab

---

## Support Vector Machines untuk Multi-Kelas

---

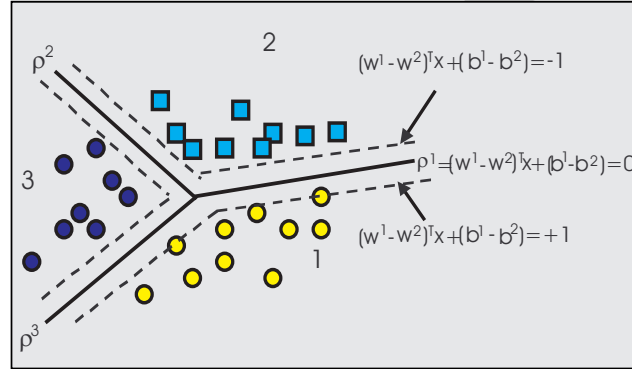
### 7.1 Ide Dasar

Pada awalnya SVM dikembangkan untuk persoalan klasifikasi dua kelas. Ada dua pendekatan utama untuk SVM multi kelas ini. Pertama, kita menemukan dan menggabung beberapa fungsi pemisah persoalan klasifikasi dua kelas untuk menyelesaikan persoalan klasifikasi multi kelas. Yang kedua, secara langsung menggunakan semua data dari semua kelas dalam satu formulasi persoalan optimisasi. Termasuk dalam pendekatan pertama dimana beberapa fungsi untuk problem dua kelas dikembangkan lalu digabung: satu-lawan-semua (One-against-all, OAA), dan satu-lawan-satu (one-against-one, OAO) (Hsu and Lin, 2002; Santosa and Trafalis, 2004). Dalam buku ini kita akan membahas secara mendetail pendekatan yang pertama.

#### 7.1.1 Metoda Satu-lawan-Semua (SLA)

Dengan metoda ini, untuk masalah klasifikasi  $k$ -kelas, kita menemukan  $k$  fungsi pemisah dimana  $k$  adalah banyaknya kelas. Misalkan kita namakan

fungsi pemisah kita dengan  $\rho$ . Dalam metoda ini,  $\rho^i$  ditrain dengan semua data dari kelas- $i$  dengan label +1 dan semua data dari kelas lain dengan label -1. Sebagai ilustrasi, dalam masalah klasifikasi dengan 3 kelas seperti terli-



**Gambar 7.1:** Klasifier SVM for three-class classification problem

hat dalam Gambar 7.1, ketika kita mentraining  $\rho^1$ , semua data dalam kelas 1 diberi label +1 dan data yang lain dari kelas 2 dan 3 diberi label -1. Begitu juga, ketika kita mentraining  $\rho^2$  semua data dalam kelas 2 diberi label +1 dan data yang lain dari kelas 1 dan 3 diberi label -1. Kita lakukan untuk semua  $i = 1, 2, 3$ . Jika kita punya  $\ell$  data untuk training  $(x_1, y_1), \dots, (x_\ell, y_\ell)$  dimana  $x_i \in R^n, i = 1, 2, \dots, \ell$  adalah data input dan  $y_i \in S = \{1, \dots, k\}$  kelas dari  $x_i$  yang bersangkutan, fungsi pemisah ke- $i$  menyelesaikan persoalan optimisasi berikut:

$$\min_{w^i, b^i, t_j} \frac{1}{2} (w^i)^T w^i + C \sum_{j=1}^{\ell} t_j^i \quad (7.1)$$

Subject to

$$w^i x_j + b^i \geq 1 - t_j^i, \text{ jika } y_j = i$$

$$w^i x_j + b^i \leq -1 + t_j^i, \text{ jika } y_j \neq i$$

$$t_j \geq 0, j = 1, \dots, \ell, i = 1, \dots, k$$

Setelah menyelesaikan (7.1), ada  $k$  fungsi pemisah.  $w^1 x + b^1, w^2 x + b^2, \dots, w^k x + b^k$ . kemudian kelas dari suatu data/obyek baru  $x$  ditentukan berdasarkan nilai

terbesar dari fungsi pemisah:

$$j = \text{class of } x = \arg \max_{i=1,\dots,k} w^i x + b^i, \text{ dimana } j \in S \quad (7.2)$$

Secara praktis, kita menyelesaikan bentuk dual dari (7.1) seperti di formulasikan dalam (6.10) dimana banyaknya variabel sama dengan banyaknya data. Karena itu sebanyak  $k$   $l$ -variabel persoalan program kuadrat (QP) harus diselesaikan. Berikut ini suatu contoh untuk memberi ilustrasi bagaimana pendekatan ini bekerja.

#### Contoh 1

Misalkan kita punya persoalan dengan 3-kelas. Setelah semua fungsi pemisah (3 fungsi) kita temukan, ada empat data baru yang harus ditentukan di kelas mana ( $x_1, x_2, x_3, x_4$ ). Nilai prediksi dari setiap fungsi pemisah diberikan dalam matriks berikut. Setiap kolom dari matriks mewakili nilai prediksi dari setiap fungsi pemisah.

$$\hat{y} = \begin{bmatrix} \mathbf{18.5965} & 14.7108 & -33.3073 \\ -12.3010 & 4.6637 & \mathbf{7.6373} \\ -9.4610 & \mathbf{8.4008} & 1.0602 \\ -20.0123 & 2.6552 & \mathbf{17.3571} \end{bmatrix}$$

Nilai maksimum  $\hat{y}$  untuk setiap titik data adalah:

$$\begin{bmatrix} 18.5965 \\ 7.6373 \\ 8.4008 \\ 17.3571 \end{bmatrix}$$

Berdasar nilai maksimum  $\hat{y}$ , kelas dari empat data baru masing-masing adalah 1, 3, 2, dan 3.

Tabel 7.1 adalah algoritma SVM untuk satu lawan semua (SLA).

### 7.1.2 Metoda Satu-lawan-Satu (SLU)

Dengan metoda ini kita perlu menemukan  $k(k-1)/2$  fungsi pemisah dimana setiap fungsi ditrain dengan data dari dua kelas. Misalkan kita punya persoalan dengan 3-kelas, kita harus menemukan 3 fungsi pemisah:  $\rho^{12}$ ,  $\rho^{13}$  and

$\rho^{23}$ . Ketika kita mentraining  $\rho^{12}$ , semua data dari kelas 1 diberi label +1 dan semua data dari kelas 2 diberi label -1. Pendekatan yang sama dipakai untuk mentraining  $\rho^{13}$  and  $\rho^{23}$ . Untuk training data dari kelas ke- $i$  dan ke- $j$ , kita selesaikan persoalan klasifikasi dua-kelas berikut (Hsu and Lin, 2002):

$$\begin{aligned} \min_{w^{ij}, b^{ij}, t^{ij}} \quad & \frac{1}{2}(w^{ij})^T w^{ij} + C \sum_r t_r^{ij} \\ \text{Subject to} \quad & w^{ij} x_r + b^{ij} \geq 1 - t_r^{ij}, \text{ jika } y_r = i \\ & w^{ij} x_r + b^{ij} \leq -1 + t_r^{ij}, \text{ jika } y_r = j \\ & t_r^{ij} \geq 0, \end{aligned} \quad (7.3)$$

dimana  $r$  menunjukkan indeks dari data dari setiap kelas. Setelah semua fungsi pemisah  $k(k-1)/2$  ditemukan, ada beberapa metoda untuk melakukan testing untuk data baru. Salah satu strategi adalah max-voting. Berdasarkan pada strategi ini, untuk pemisah  $\rho^{ij}$ , jika tanda dari suatu data poin  $x$  adalah di kelas  $i$ , kemudian voting untuk kelas ke  $i$  ditambah satu. Sebaliknya, voting untuk kelas ke  $j$  ditambah satu. Kita ulang langkah-langkah ini untuk semua fungsi pemisah. Kemudian, kita prediksi  $x$  berada dalam kelas mana, didasarkan pada nilai voting yang paling tinggi. Dalam kasus dimana ada dua kelas dengan nilai voting yang sama kita pilih yang indeksnya lebih kecil (Santosa and Trafalis, 2005). Secara praktis, dengan metoda ini kita menyelesaikan problem dual (7.3) seperti dirumuskan di (6.10) yang mempunyai jumlah variabel sama dengan banyaknya data yang ada dalam dua kelas. Karena itu bila secara rata-rata setiap kelas mempunyai  $l/k$  titik data, kita harus menyelesaikan  $k(k-1)/2$  program kuadrat dimana setiap program mempunyai  $2l/k$  variabel. *Contoh 2*

Sebagai contoh, kita terapkan metoda ini untuk soal yang sama seperti contoh 1. Setelah menemukan semua fungsi pemisah kita dapatkan nilai  $\hat{y}$  dari fungsi pemisah  $\rho^{12}$  sebagai berikut

$$\hat{y} = \begin{bmatrix} 1.4244 \\ -3.8363 \\ -3.4960 \\ -5.2525 \end{bmatrix}$$



Voting untuk nilai di atas :

Kelas 1	Kelas 2	Kelas 3
1	0	0
0	1	0
0	1	0
0	1	0

Nilai  $\hat{y}$  dari fungsi pemisah  $\rho^{13}$  adalah

$$\hat{y} = \begin{bmatrix} -6.7500 \\ -0.8855 \\ 2.7612 \\ -7.5913 \end{bmatrix}$$

Setelah mempertimbangkan semua fungsi pemisah  $\rho^{12}$  and  $\rho^{13}$  susunan votingnya menjadi:

Kelas 1	Kelas 2	Kelas 3
1	0	1
0	1	1
1	1	0
0	1	1

Nilai  $\hat{y}$  dari pemisah  $\rho^{13}$  adalah

$$\hat{y} = \begin{bmatrix} -24.2676 \\ 1.4833 \\ -3.6762 \\ 7.3307 \end{bmatrix}$$

Setelah mempertimbangkan semua fungsi pemisah  $\rho^{12}$ ,  $\rho^{13}$  dan  $\rho^{23}$  nilai votingnya menjadi:

Kelas 1	Kelas 2	Kelas 3
1	0	<b>2</b>
0	<b>2</b>	1
<b>1</b>	1	1
0	<b>2</b>	1

Berdasar nilai voting di atas, kelas untuk empat data baru ini adalah 3, 2, 1 dan 2.

## 8.1 Pendahuluan

Dengan menggunakan konsep  $\epsilon$ -insensitive loss function, yang diperkenalkan oleh Vapnik, SVM bisa digeneralisasi untuk melakukan pendekatan fungsi (function approximation) atau regresi (Schölkopf and Smola, 2002). Didasarkan pada teori *Structural Risk Minimization* untuk mengestimasi suatu fungsi dengan cara meminimalkan batas atas dari *generalization error*, SVR telah memperlihatkan sebagai metoda yang bisa mengatasi masalah *overfitting*. Sehingga bisa menghasilkan performansi yang lebih bagus dibanding, misalnya, dengan ANN. Dalam bagian ini akan kita bahas bagaimana mengembangkan SVM untuk klasifikasi menjadi SVM untuk regresi.

## 8.2 Ide Dasar

Misalkan kita punya  $\ell$  set data training,  $(x_i, y_i)$ ,  $i = 1, \dots, \ell$  dengan data input  $x = \{x_1, x_2, \dots, x_\ell\} \subseteq \mathbb{R}^N$  dan output yang bersangkutan  $y = \{y_1, \dots, y_\ell\} \subseteq \mathbb{R}$ . Dengan SVR, kita ingin menemukan suatu fungsi  $f(x)$  yang mempunyai de-

viiasi paling besar  $\epsilon$  dari target aktual  $y_i$  untuk semua data training. Maka, dengan SVR kita akan mendapatkan suatu tabung seperti dalam gambar 8.1. Manakala nilai  $\epsilon$  sama dengan 0 maka kita dapatkan suatu regresi yang sempurna. Misalkan kita mempunyai fungsi berikut sebagai garis regresi:

$$f(x) = w^T \varphi(x) + b, \quad (8.1)$$

dimana  $\varphi(x)$  menunjukkan suatu titik di dalam feature space  $F$  hasil pemetaan  $x$  di dalam input space. Koefisien  $w$  dan  $b$  diestimasi dengan cara meminimalkan *fungsi resiko* (risk function) yang didefinisikan dalam persamaan (8.2).

$$\min \frac{1}{2} \|w\|^2 + C \frac{1}{\ell} \sum_{i=1}^{\ell} L_{\epsilon}(y_i, f(x_i)) \quad (8.2)$$

Subject to

$$y_i - w\varphi(x_i) - b \leq \epsilon$$

$$w\varphi(x_i) - y_i + b \leq \epsilon, i = 1, \dots, \ell,$$

dimana

$$L_{\epsilon}(y_i, f(x_i)) = \begin{cases} |y_i - f(x_i)| - \epsilon & |y_i - f(x_i)| \geq \epsilon \\ 0, & \text{otherwise} \end{cases} \quad (8.3)$$

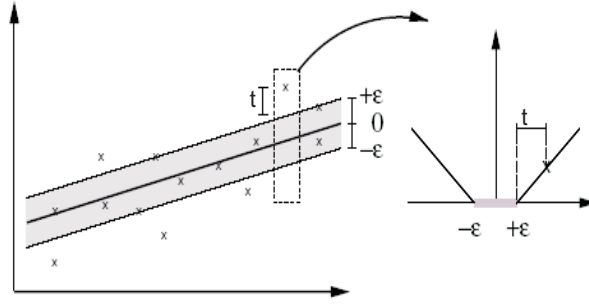
Faktor  $\|w\|^2$  dinamakan *regularisasi*. Meminimalkan  $\|w\|^2$  akan membuat suatu fungsi setipis (flat) mungkin, sehingga bisa mengontrol kapasitas fungsi (function capacity). Faktor kedua dalam fungsi tujuan adalah kesalahan empirik (empirical error) yang diukur dengan  $\epsilon$ -insensitive loss function. Menggunakan ide  $\epsilon$ -insensitive loss function Vapnik (1995), kita harus meminimalkan norm dari  $w$  agar mendapatkan generalisasi yang baik untuk fungsi regresi  $f$ . Karena itu kita perlu menyelesaikan problem optimisasi berikut:

$$\min \frac{1}{2} \|w\|^2 \quad (8.4)$$

Subject to

$$y_i - w\varphi(x_i) - b \leq \epsilon$$

$$w\varphi(x_i) - y_i + b \leq \epsilon, i = 1, \dots, \ell \quad (8.5)$$



**Gambar 8.1:**  $\epsilon$ -insensitive loss function. Semua titik di luar area berwarna dikenai pinalti

Kita asumsikan bahwa ada suatu fungsi  $f$  yang dapat mengaproksimasi semua titik  $(x_i, y_i)$  dengan presisi  $\epsilon$ . Dalam kasus ini kita asumsikan bahwa semua titik ada dalam rentang  $f \pm \epsilon$  (feasible). Dalam hal ketidaklayakan (infeasibility), dimana ada beberapa titik yang mungkin keluar dari rentang  $f \pm \epsilon$ , kita bisa menambahkan variable slack  $t, t^*$  untuk mengatasi masalah pembatas yang tidak layak (infeasible constraints) dalam problem optimisasi. Selanjutnya problem optimisasi di atas bisa diformulasikan sebagai berikut Vapnik (1995):

$$\begin{aligned} \min \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{\ell} (t_i + t_i^*) \\ \text{Subject to} \quad & y_i - w^T \varphi(x_i) - b - t_i \leq \epsilon, i = 1, \dots, \ell \\ & w^T \varphi(x_i) - y_i + b - t_i^* \leq \epsilon, i = 1, \dots, \ell \\ & t_i, t_i^* \geq 0, \end{aligned} \tag{8.6}$$

Konstanta  $C > 0$  menentukan tawar menawar (trade off) antara ketipisan fungsi (flatness of function)  $f$  dan batas atas deviasi lebih dari  $\epsilon$  masih ditoleransi. Semua deviasi lebih besar daripada  $\epsilon$  akan dikenakan pinalti sebesar  $C$ . Gambar 8.1 memperlihatkan situasi ini secara grafis: hanya titik-titik di luar area yang berwarna yang punya kontribusi terhadap ongkos pinalti. Dalam SVR,  $\epsilon$  ekuivalen dengan akurasi dari aproksimasi kita terhadap data training.

Nilai  $\epsilon$  yang kecil terkait dengan nilai yang tinggi pada variabel slack  $t_i^{(*)}$  dan akurasi aproksimasi yang tinggi. Sebaliknya, nilai yang tinggi untuk  $\epsilon$  berkaitan dengan nilai  $t_i^{(*)}$  yang kecil dan akurasi aproksimasi yang rendah. Menurut persamaan (8.6), nilai yang tinggi untuk variabel slack akan membuat kesalahan empirik mempunyai pengaruh yang besar terhadap faktor regularisasi. Dalam SVR, support vectors adalah data training yang terletak pada dan di luar batas  $\epsilon$  dari fungsi keputusan. karena itu jumlah support vectors menurun dengan naiknya nilai  $\epsilon$ . Dalam formulasi dual, problem optimisasi dari SVR adalah sebagai berikut

$$\begin{aligned}
 \max \quad & -\frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} (\alpha_i - \alpha'_i)(\alpha_i - \alpha'_i) K(x_i, x_j) - \\
 & \sum_{i=1}^{\ell} y_i (\alpha_i - \alpha'_i) - \epsilon \sum_{i=1}^{\ell} (\alpha_i + \alpha'_i) \\
 \text{Subject to} \quad & \sum_{i=1}^{\ell} (\alpha_i - \alpha'_i) = 0 \\
 & 0 \leq \alpha_i \leq C, i = 1, \dots, \ell, \\
 & 0 \leq \alpha'_i \leq C, i = 1, \dots, \ell,
 \end{aligned} \tag{8.7}$$

dimana  $C$  didefinisikan oleh user,  $K(x_i, x_j)$  adalah dot-product kernel yang didefinisikan sebagai  $K(x_i, x_j) = \varphi^T(x_i)\varphi(x_j)$  seperti dijelaskan dalam bagian 6.3. Dengan menggunakan *Lagrange multiplier* dan kondisi optimalitas, fungsi regresi secara eksplisit dirumuskan sebagai:

$$f(x) = \sum_{i=1}^{\ell} (\alpha_i - \alpha'_i) K(x_i, x) + b, \tag{8.8}$$

dimana  $K(x_i, x)$  didefinisikan melalui fungsi kernel  $k$ . Persamaan (8.7) adalah program kuadrat (QP). Namun bentuknya tidak sesuai dengan bentuk standar QP.

### 8.3 Formulasi SVR dalam QP Standar

Untuk memenuhi bentuk standar QP sehingga bisa diselesaikan dengan solver QP komersial melalui program komputer bentuk itu perlu dimanipulasi (An-

cona, 1999).

$$\sum_{i=1}^{\ell} \sum_{j=1}^{\ell} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)x_i x_j = \sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*) \left( \sum_{j=1}^{\ell} \alpha_j x_i x_j - \alpha_j^* x_i x_j \right) = \quad (8.9)$$

$$\begin{aligned} & \sum_{i=1}^{\ell} \alpha_i \sum_{j=1}^{\ell} \alpha_j x_i x_j - \sum_{i=1}^{\ell} \alpha_i \sum_{j=1}^{\ell} \alpha_j^* x_i x_j - \sum_{i=1}^{\ell} \alpha_i^* \sum_{j=1}^{\ell} \alpha_j x_i x_j + \sum_{i=1}^{\ell} \alpha_i^* \sum_{j=1}^{\ell} \alpha_j^* x_i x_j = \\ & \sum_{i=1}^{\ell} \alpha_i \left( \sum_{j=1}^{\ell} \alpha_j x_i x_j + \sum_{j=1}^{\ell} \alpha_j^* (-x_i x_j) \right) + \sum_{i=1}^{\ell} \alpha_i^* \left( \sum_{j=1}^{\ell} \alpha_j (-x_i x_j) + \sum_{j=1}^{\ell} \alpha_j^* x_i x_j \right) \end{aligned}$$

Kita definisikan vektor baru  $\lambda$  dengan  $2\ell$  komponen sehingga:

$$\lambda = (\alpha_1, \alpha_2, \dots, \alpha_{\ell}, \alpha_1^*, \alpha_2^*, \dots, \alpha_{\ell}^*)$$

dan  $D$  adalah matriks  $2\ell \times 2\ell$  yang didefinisikan sebagai

$$D_{ij} = \begin{cases} x_i x_j & i = 1, 2, \dots, \ell \\ & j = 1, 2, \dots, \ell \\ -x_i x_{j-\ell} & i = 1, 2, \dots, \ell \\ & j = \ell + 1, \ell + 2, \dots, 2\ell \\ -x_{i-\ell} x_j & i = \ell + 1, \ell + 2, \dots, 2\ell \\ & j = 1, 2, \dots, \ell \\ -x_{i-\ell} x_{j-\ell} & i = \ell + 1, \ell + 2, \dots, 2\ell \\ & j = \ell + 1, \ell + 2, \dots, 2\ell \end{cases}$$

Dengan definisi ini kita bisa ubah term pertama dari fungsi tujuan dalam persamaan 8.7 menjadi

$$\begin{aligned} & \sum_{i=1}^{\ell} \lambda_i \left( \sum_{j=1}^{\ell} \lambda_j D_{ij} + \sum_{j=\ell+1}^{2\ell} \lambda_j D_{ij} \right) + \sum_{j=\ell+1}^{2\ell} \lambda_j \left( \sum_{i=1}^{\ell} \lambda_i D_{ij} + \sum_{i=\ell+1}^{2\ell} \lambda_i D_{ij} \right) = (8.10) \\ & \sum_{i=1}^{\ell} \lambda_i \sum_{j=1}^{2\ell} \lambda_j D_{ij} + \sum_{i=\ell+1}^{2\ell} \lambda_i \sum_{j=1}^{2\ell} \lambda_j D_{ij} = \sum_{i=1}^{2\ell} \lambda_i \sum_{j=1}^{2\ell} \lambda_j D_{ij} = \lambda D \lambda \end{aligned}$$

Untuk term kedua dari fungsi tujuan kita bisa melakukan manipulasi sebagai berikut

$$\begin{aligned}
& \sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*) y_i - \epsilon \sum_{i=1}^{\ell} (\alpha_i + \alpha_i^*) = \quad (8.11) \\
& \sum_{i=1}^{\ell} \alpha_i y_i + \sum_{i=1}^{\ell} \alpha_i^* (-y_i) + \sum_{i=1}^{\ell} \alpha_i (-\epsilon) + \sum_{i=1}^{\ell} \alpha_i^* (-\epsilon) = \\
& \sum_{i=1}^{\ell} \alpha_i (y_i - \epsilon) + \sum_{i=1}^{\ell} \alpha_i^* (-y_i - \epsilon)
\end{aligned}$$

Kita bisa definisikan lagi vektor lain  $c$  dengan ukuran  $2\ell$

$$c = (y_1 - \epsilon, y_2 - \epsilon, \dots, y_{\ell} - \epsilon, -y_1 - \epsilon, -y_2 - \epsilon, \dots, -y_{\ell} - \epsilon)$$

Dari persamaan di atas kita dapatkan

$$\sum_{i=1}^{\ell} \lambda_i c_i + \sum_{i=\ell+1}^{2\ell} \lambda_i c_i = \sum_{i=1}^{2\ell} \lambda_i c_i = \lambda c$$

Lalu kita definisikan vektor  $b$  dengan  $2\ell$  komponen sebagai berikut

$$b = (\underbrace{1, 1, \dots, 1}_{\ell}, \underbrace{-1, -1, \dots, -1}_{\ell})$$

Dengan demikian pembatas linier pada persamaan (8.7) di atas menjadi

$$\sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*) = \sum_{i=1}^{2\ell} \lambda_i b_i = \lambda b = 0 \quad (8.12)$$

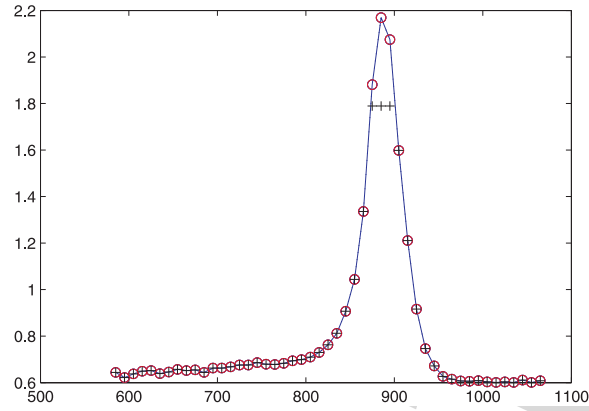
Dengan variabel-variabel baru di atas persamaan (8.7) bisa ditulis dengan bentuk QP standar sebagai berikut

$$\begin{aligned}
& \max_{\lambda} \quad -\frac{1}{2} \lambda D \lambda + \lambda c \\
& \text{Subject to} \quad \lambda b = 0 \\
& \quad \quad \quad 0 \leq \lambda \leq C
\end{aligned} \quad (8.13)$$

Dengan bentuk ini akan mudah diselesaikan dengan solver QP.

Gambar 8.2 mengilustrasikan performansi SVR diterapkan pada data Titanium (Dierckx, 1993) dengan menggunakan software Matlab.





— : actual data, + :  $\sigma = 1, C = 1$ , o :  $\sigma = 1, C = 10$

**Gambar 8.2:** Plot dari data aktual dan hasil dari SVR RBF kernel

## 8.4 Loss Function

Loss function adalah fungsi yang menunjukkan hubungan antara error dengan bagaimana error ini dikenai pinalti. Perbedaan loss function akan menghasilkan formulasi SVR yang berbeda. Gambar 8.3 berikut menunjukkan macam-macam loss function (Gunn, 1998).

Yang paling sederhana adalah  $\epsilon$ -insensitive. Dengan menggunakan loss function  $\epsilon$ -insensitive, kita peroleh formulasi matematikanya sebagai berikut:

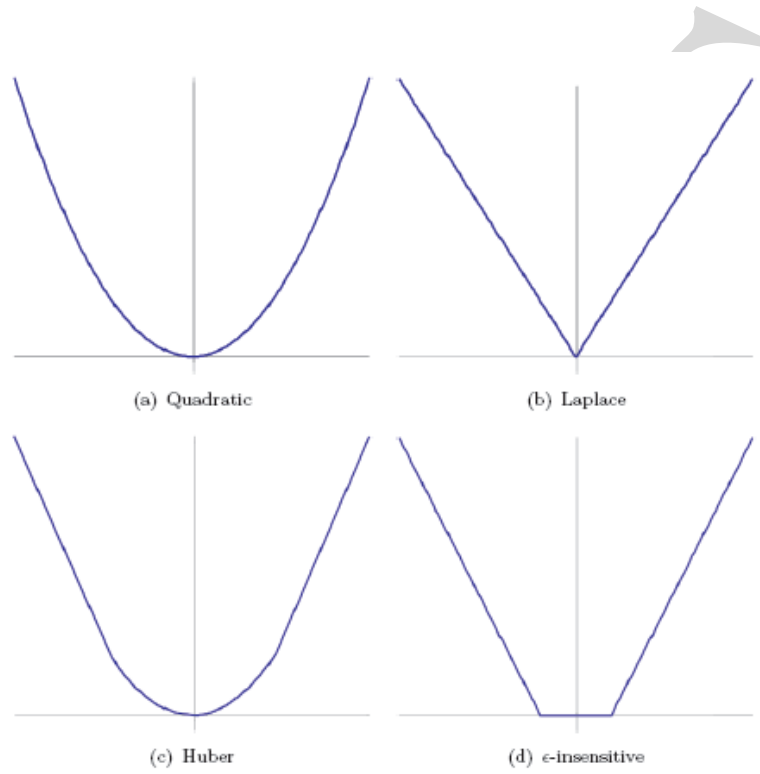
$$L_{\epsilon}(y) = \begin{cases} 0, & \text{untuk } |f(x) - y| < \epsilon \\ |f(x) - y| - \epsilon, & \text{untuk yang lain} \end{cases} \quad (8.14)$$

sehingga solusinya

$$\max -\frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle x_i, x_j \rangle + \sum_{i=1}^{\ell} \alpha_i (y_i - \epsilon) - \alpha_i^* (y_i + \epsilon) \quad (8.15)$$

atau

$$\bar{\alpha} \bar{\alpha}^* = \arg \min_{\alpha, \alpha^*} \max -\frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle x_i, x_j \rangle - \sum_{i=1}^{\ell} \alpha_i - \alpha_i^* y_i + \sum_{i=1}^{\ell} \alpha_i + \alpha_i^* \epsilon \quad (8.16)$$



**Gambar 8.3:** Plot dari data aktual dan hasil dari SVR RBF kernel

$$\begin{aligned} & \text{subject to} \\ & 0 \leq \alpha_i, \alpha_i^* \leq C, i = 1, \dots, \ell \\ & \sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*) = 0 \end{aligned}$$

Atau bisa disederhanakan menjadi

$$\begin{aligned} & \max -\frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \beta_i \beta_j \langle x_i, x_j \rangle - \sum_{i=1}^{\ell} \beta_i y_i \\ & \text{subject to} \\ & -C \leq \beta_i \leq C, i = 1, \dots, \ell \\ & \sum_{j=1}^{\ell} \beta_j = 0, \end{aligned} \tag{8.17}$$

dimana  $\beta = \alpha - \alpha^*$ . Sedangkan quadratic loss function

$$L(f(x) - y) = (f(x) - y)^2 \quad (8.18)$$

menghasilkan solusi

$$\max -\frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} (\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*) \langle x_i, x_j \rangle + \sum_{i=1}^{\ell} (\alpha_i - \alpha_i^*) y_i - \frac{1}{2C} \sum_{j=1}^{\ell} (\alpha_i^2 + \alpha_i^{*2}) \quad (8.19)$$

Atau bisa disederhanakan menjadi

$$\max -\frac{1}{2} \sum_{i=1}^{\ell} \sum_{j=1}^{\ell} \beta_i \beta_j \langle x_i, x_j \rangle + \sum_{i=1}^{\ell} \beta_i y_i - \frac{1}{2C} \sum_{j=1}^{\ell} \beta_i^2 \quad (8.20)$$

subject to

$$\sum_{j=1}^{\ell} \beta_i = 0$$

## 8.5 Implementasi Support Vector Regression Dengan Matlab

Untuk penerapan SVR, mari kita gunakan data harga tepung yang telah kita gunakan dalam neural networks. Kita mempunyai data time series dengan 22 observasi. Kita pakai 20 data pertama sebagai set training dan sisanya sebagai set testing.

```
>> x=
1 107.1
2 113.5
3 112.7
4 114.7
5 123.4
```

```

6 123.6
7 116.3
8 118.5
9 119.8
10 120.3
11 127.4
12 125.1
13 127.6
14 129
15 124.6
16 134.1
17 146.5
18 171.2
19 178.6
20 172.2
21 171.5
22 163.6
>> tr=x(1:20,1);
>> trl=x(1:20,2);
>> ts=x(21:22,1);
>> tsl=x(21:22,2);

```

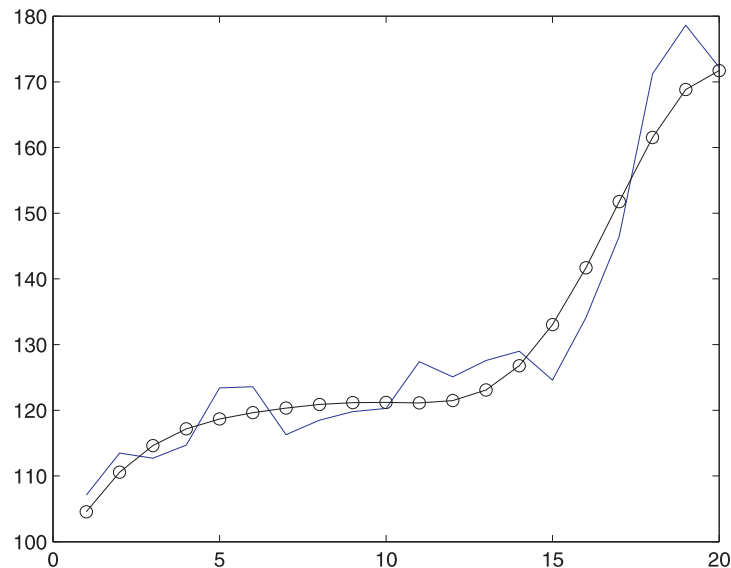
Selanjutnya kita bisa terapkan SVR dengan program komputer untuk menemukan Lagrange multiplier,  $\beta$  dan bias,  $b$  (Gunn, 1998) .

```

>> [beta, b] = svr(tr,trl,'rbf',5,10,'quadratic')
Support Vector Regressing ....
-----
Constructing ...
Optimising ...
Execution time : 0.1 seconds
|w0|^2 : 47895.902313
Sum beta : 325.805073
Support Vectors : 20 (100.0%)

```

```
>> tstY = svroutput(tr,tr,'rbf',5,beta,b)
```



**Gambar 8.4:** Plot output aktual dan hasil prediksi(lingkaran kecil) dengan SVR untuk data harga tepung

Gambar 8.5 menunjukkan plot data harga tepung aktual dan hasil prediksi menggunakan SVR dengan kernel RBF dengan nilai  $\sigma=5$ . Plot ini merupakan plot untuk data training. Jika kita ingin melihat hasil prediksi untuk dua data terakhir bisa kita lakukan perintah berikut

```
>> tstY = svroutput(tr,ts,'rbf',5,beta,b)
tstY =
    168.7707
    159.6169
```

Dari hasil prediksi dan nilai aktual bisa kita hitung error dari hasil prediksi. Misalkan kita ingin menggunakan MSE sebagai ukuran error, maka dalam Matlab bisa kita lakukan dengan cara berikut:

```
>> mse=mean((tsl-tstY).^2)%rata-rata kuadrat selisih
data aktual dan hasil prediksi
mse =
    11.6571
```

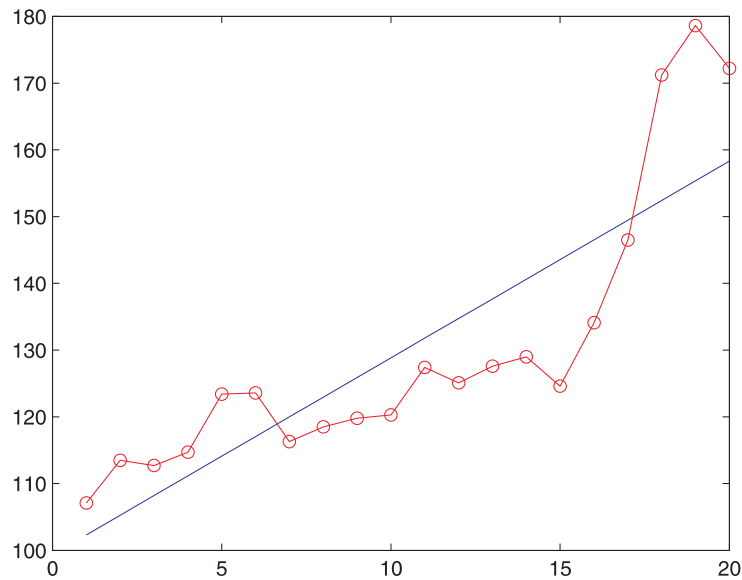
Sedangkan menggunakan regresi linier tradisional bisa dilakukan dengan perintah berikut.

```
>>b=regress(tr1,[ones(20,1) tr])
b =
    99.3737
     2.9463
>> yt=b(1)+b(2)*ts
yt =
    161.2463
    164.1926
>> mse=mean((tsl-yt).^2)
mse =
    52.7446
```

Untuk plot hasil prediksi dan nilai aktual bisa dilihat pada gambar 8.5. Untuk nilai MSE, kita dapatkan bahwa dengan menggunakan regresi linier lainnya lebih besar (52.7446) dibanding nilai MSE dengan menggunakan SVR (11.6571).

Misalkan kita ingin menggunakan regresi kuadrat. Untuk plot hasil prediksi dan nilai aktual bisa dilihat pada gambar 8.5. Sedangkan perintah dalam Matlab adalah sebagai berikut

```
>>stats = regstats(tr1,tr,'quadratic',{'yhat','beta'})
stats =
    source: 'regstats'
      beta: [3x1 double]
     yhat: [20x1 double]
>> b=stats.beta;
>> yt=b(1)+b(2)*ts+b(3)*ts.^2
```



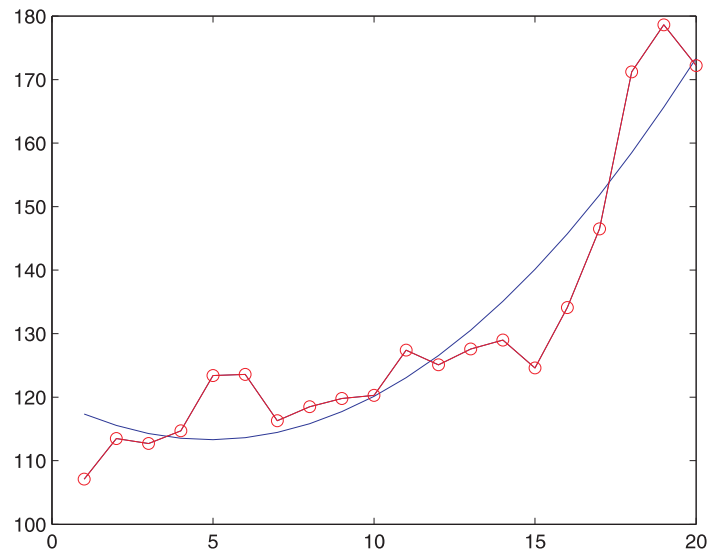
**Gambar 8.5:** Plot output aktual (lingkaran kecil) dan hasil prediksi dengan regresi linier

yt =  
181.5595  
190.3095

Untuk sumber informasi mengenai paper, artikel, software dan buku mengenai metoda kernel bisa dilihat di [www.kernel-machines.org](http://www.kernel-machines.org).

## 8.6 Pemilihan Metoda Prediksi

Untuk prediksi data dengan label diskrit kita bisa menggunakan LDA, SVM, ANN atau teknik lain. Sedangkan untuk data dengan output bilangan kontinu bisa digunakan regresi linier, SVR, atau ANN. Pertanyaan yang sering muncul kemudian, metoda mana yang harus dipilih? Yang perlu ditekankan



**Gambar 8.6:** Plot output aktual (lingkaran kecil) dan hasil prediksi dengan regresi kuadrat

adalah bahwa tidak ada model yang terbaik untuk semua kasus atau data. Untuk suatu kasus biasanya kita terapkan beberapa teknik atau satu teknik dengan beberapa parameter yang berbeda. Selanjutnya sesudah dipakai beberapa teknik, kita bandingkan model-model tersebut dengan melihat *error* yang dihasilkan. Model yang memberi nilai error paling kecil itu yang kita pilih.



---

## Bibliografi

---

- Ancona, N., 1999, *Properties of Support Vector Machines for Regression*, Technical Report RI-IESI/CNR - Nr. 01/99, Tech. rep., Istituto Elaborazioni Segnali ed Immagini, C.N.R., Bari, Italy, <http://www.ba.cnr.it/iesina18/>.
- Bertsekas, D., 1995, *Nonlinear Programming*, Athenas Scientific, Belmont, MA.
- Cristianini, N. and Shawe-Taylor, J., 2000, *An Introduction to Support Vector Machines*, Cambridge University Press.
- Dierckx, P., 1993, *Curve and Surface Fitting with Splines. Monographs on Numerical Analysis*, Clarendon Press, Oxford.
- Fisher, R., 1936, "The use of multiple measurements in taxonomic problems", *Annual Eugenics*, 7, no. II, 179–188.
- Gunn, S., 1998, *Support Vector Machines for Classification and Regression*, Tech. rep., Dept. of Electronics and Computer Science, University of Southampton, <http://www.isis.ecs.soton.ac.uk/resources/svminfo/>.

- Haykin, S., 1999, *Neural Network: A Comprehensive Foundation*, Prentice Hall, New Jersey.
- Hsu, C.-W. and Lin, C.-J., 2002, "A comparison of methods for multi-class support vector machines", *IEEE Transactions on Neural Networks*, 13, 415–425.
- Reklaitis, G., Ravindran, A., and Ragsdell, K., 1983, *Engineering Optimization: Methods and Applications*, John Wiley and Sons, New York.
- Santosa, B., 2006, "Data mining dengan matlab", Draft buku, TI-ITS, Surabaya.
- Santosa, B. and Trafalis, T., 2004, "Multiclass procedure for minimax probability machine", *Intelligent Engineering Systems Through Artificial Neural Networks 14*, (eds.) Dagli, C., Buczak, A., Ghosh, J., Embrechts, M., O.Ersoy, and Kercel, S., ASME Press, pp. 447–452.
- Santosa, B. and Trafalis, T., 2005, "Robust multiclass kernel-based classifiers", *Computational Optimization and Applications*, submitted.
- Schölkopf, B. and Smola, A., 2002, *Learning with Kernels*, The MIT Press, Cambridge, Massachusetts.
- Vapnik, V., 1995, *The Nature of Statistical Learning Theory*, Springer Verlag.