BAB 2 STUDI LITERATUR

Bab ini memaparkan teori penunjang penelitian, yang terdiri dari sistem penciuman elektronik, pembelajaran dengan Jaringan Syaraf Tiruan, FNLVQ, *Matriks Similarity Analysis* (MSA) serta Validasi Silang (*cross-validation*). Selain itu bab ini juga menyinggung *Particle Swarm Optimization* (PSO), sebagai teori dasar untuk membentuk metode FNLVQ-PSO.

2.1 Sistem Penciuman Elektronik

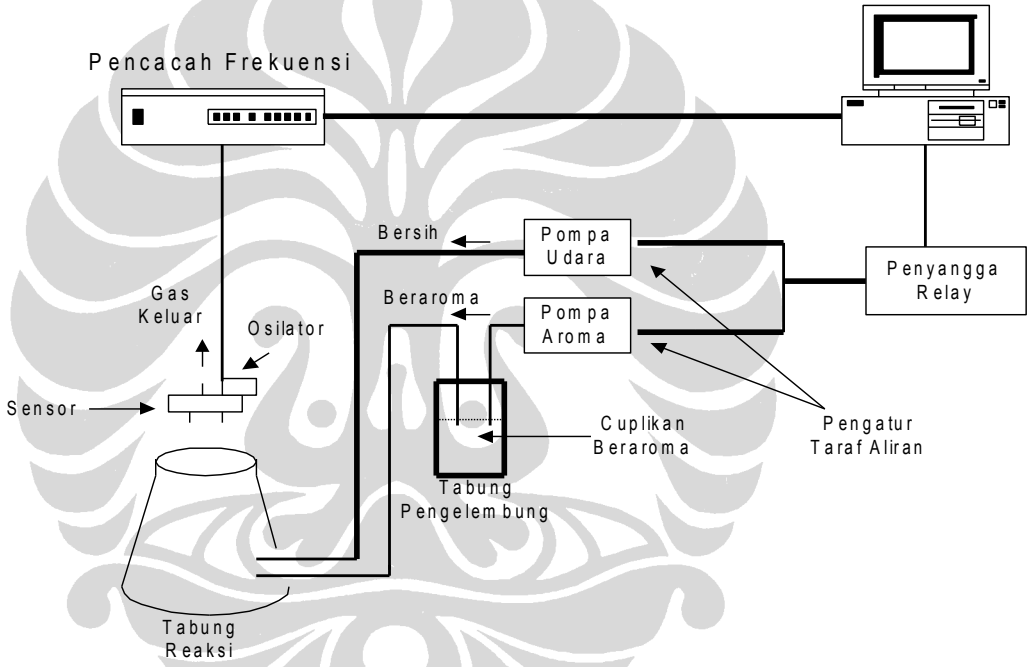
Sistem Penciuman Elektronik (*electronic nose*) [4] dikembangkan dengan menggunakan sensor quartz resonator, kemudian hasil dari sensor tersebut diproses menggunakan algoritma jaringan syaraf tiruan sebagai sistem pengenalan pola. Sistem ini mampu mengenali berbagai macam aroma tunggal, namun untuk aroma campuran sistem ini pengenalannya masih kurang baik. Sub bab berikut akan menjelaskan sedikit mengenai sistem ini.

2.1.1 Sistem Sensor dan Peralatan Penciuman Elektronik

Sistem Penciuman Elektronik yang telah dikembangkan oleh Wisnu et. al. menggunakan teknologi gas sensor yang disebut SAW (*surface acoustic wave*) dan *piezoelectric* (kwarsa). Kedua peralatan ini bertindak sebagai neraca-micron, yang digunakan untuk mengukur perubahan massa dari zat yang akan dianalisis. Perubahan massa zat itu menyebabkan nilai parameter (frekuensi) pada sensor berubah. Perubahan nilai parameter dari setiap sensor tersebut mempunyai karaktersistik yang berbeda, sehingga dapat menimbulkan pola dari setiap zat yang dianalisis. Pola yang dihasilkan tersebut akan digunakan sebagai dasar untuk menganalisis aroma dari gas/zat yang dianalisis. Pada sistem penciuman elektonik pola-pola tersebut akan diklasifikasikan menggunakan metode Jaringan Syaraf Tiruan.

Dalam sistem sensor Sistem Penciuman Elektronik, cairan cuplikan yang disuntikkan haruslah dari bahan yang sangat mudah diuapkan dengan cepat, dan suhu pengukuran harus dijaga tetap tinggi. Untuk lebih jelasnya sistem pengujian pengenal aroma dapat dilihat pada Gambar 2.1.

Sensor yang berjumlah enam belas ditempatkan disebuah mulut tabung uji. Untuk menjaga kestabilan suhu, tekanan dan volume digunakan dua buah pompa udara, yaitu pompa udara bersih dan pompa udara beraroma. Kerja dari kedua pompa udara ini bergantian yang diatur oleh komputer lewat rangkaian antar-muka yang akan menggerakkan sebuah Relay catu daya.



Gambar 2.1 Peralatan sistem pengujian pengenalan aroma.

Sensor yang dipakai adalah sensor resonator kwarsa yang dilapisi membran sensitif. Sensor ini diletakkan di dalam osilator yang bertindak sebagai resonator pada frekuensi tertentu. Prinsip kerja dari sensor hidung elektronik tersebut adalah menghitung nilai/besar penurunan frekuensi. Frekuensi sensor resonator akan menurun saat molekul gas teradsorbsi oleh membran, dan akan kembali normal setelah molekul tersebut mengalami proses deadsorbsi. Fenomena ini disebut efek pembebanan massa (mass loading effect). Perubahan frekuensi ΔF (hz) sebanding dengan massa total molekul gas yang teradsorbsi, diberikan oleh persamaan *Sauerbrey* :

$$\Delta F = -2{,}3 \times 10^{6} \times F^{2} \times \frac{\Delta M}{A} \quad \cdots\cdots (2.1)$$

Dimana F adalah frekuensi resonansi dasar (Mhz), ΔM adalah massa total molekul gas yang terserap (g). dan A adalah luas elektroda (cm^2).

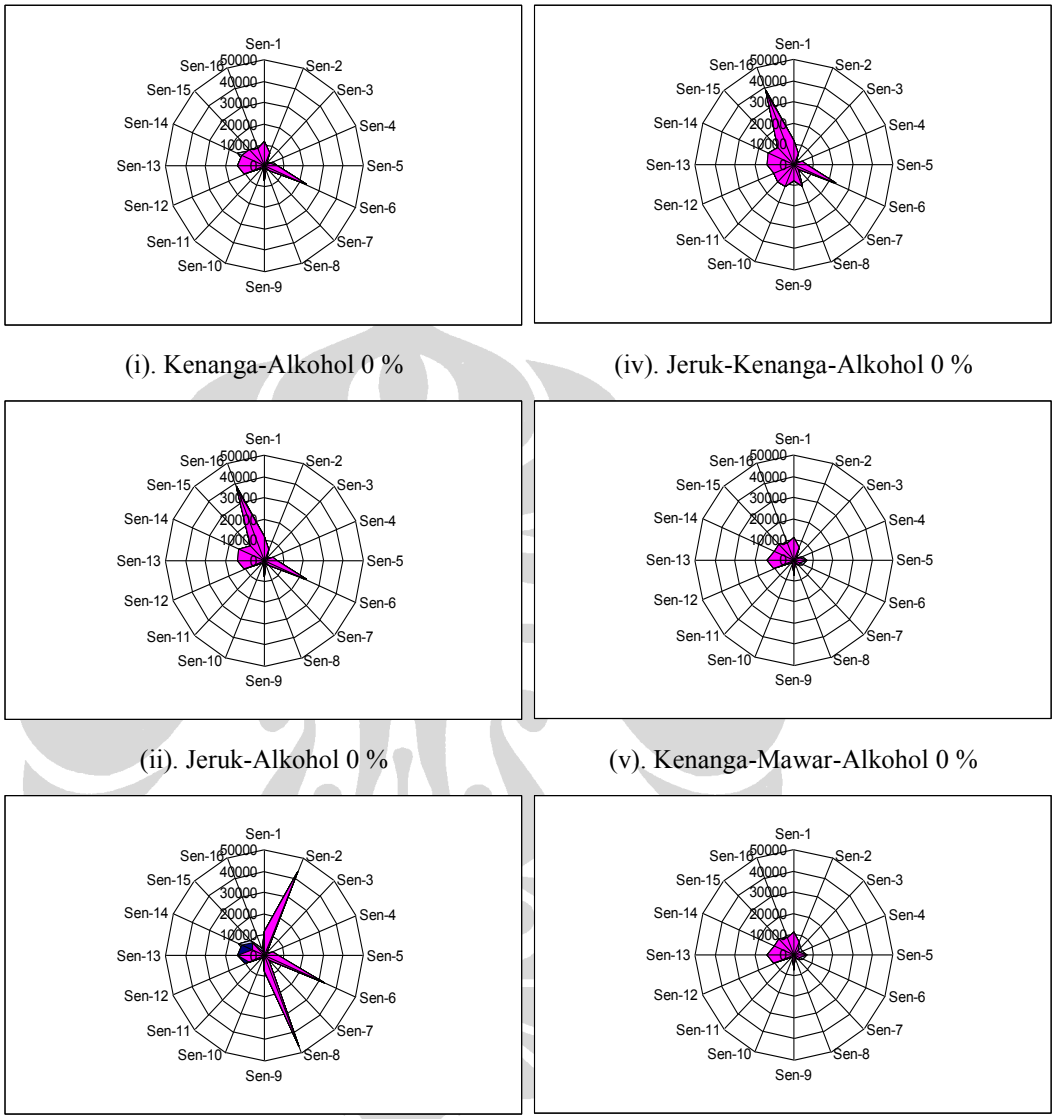
Semua sensor yang berjumlah 16 buah tersebut dilapisi oleh membran-membran sensitif yang mempunyai karakteristik yang berbeda. Perubahan frekuensi masing-masing sensor akan membentuk pola karakteristik tertentu bagi setiap aroma yang berbeda sehingga gas tersebut dapat dibedakan berdasarkan pola yang didapat [4].

Berikut jenis sensor yang digunakan dalam Sistem Penciuman Elektronik:

1. *Phosphaticid*
2. *Lecithin*
3. *Cholesterol*
4. *Phospatidyl Inositol*
5. *Phospatidyl Serine*
6. *Phospatidyl Ethanol amine*
7. *Phospatidy Chorine*
8. *Phospatidy Choline 63% Sphingomyelin 37%*
9. *Sphingomyelin*
10. *Lecithin 63% Cholesterol 37%*
11. *Cardioliin*
12. *Ethyl Cellulose*
13. *Silicone OV101*
14. *Silicone OV17*
15. *Silicone 50MB/2.000*
16. *Silicone 75MB/90.000*

Enambelas bahan kimia yang digunakan sebagai sensor pada sistem penciuman elektronik menghasilkan *signature* atau karakteristik pola dari aroma. Dengan menggunakan bermacam bahan kimia kedalam sensor, dari sini dibentuk *database signature*. *Database* ini kemudian dilatih dengan menggunakan sistem pengenalan pola, tujuan pelatihan adalah untuk melakukan pengkonfigurasian

sistem ini sehingga mampu memproduksi sistem pengklasifikasian berbahan masukan kimia, dan mampu memberikan pengenalan secara otomatis.



(i). Kenanga-Alkohol 0 %

(iv). Jeruk-Kenanga-Alkohol 0 %

(ii). Jeruk-Alkohol 0 %

(v). Kenanga-Mawar-Alkohol 0 %

(iv). Mawar-Alkohol 0 %

(vi). Jeruk-Mawar-Alkohol 0 %

(a). Aroma Dua Campuran

(b). Aroma Tiga Campuran

Gambar 2.2 Karakteristik pola (a) aroma dua campuran dan (b) aroma tiga campuran

Pada Gambar 2.2 dapat dilihat database pola dari aroma dua campuran dan aroma tiga campuran. Pola yang dihasilkan saling *overlapping* satu sama lain, misalkan untuk aroma JMAlk0% (jeruk mawar dengan alkohol 0%) dan KMAlk0% (kenanga mawar dengan alkohol 0%) kedua aroma tersebut terlihat hampir sama, akibatnya sulit untuk membedakan diantara aroma tersebut dengan cara konvensional. Teknik mutakhir yang biasa digunakan untuk menganalisis

pola diantaranya *Principal Component Analysis*, *Cluster Analysis* dan Jaringan Syaraf Tiruan.

2.2 Jaringan Syaraf Tiruan

JST dikembangkan atas dasar keingintahuan manusia terhadap proses pembelajaran manusia. JST merupakan suatu sistem yang dibangun berdasarkan cara kerja jaringan syaraf manusia. Jaringan syaraf manusia terdiri dari sel-sel syaraf yang disebut *neuron* yang tersusun dari *dendrite* yang menerima sinyal dari *neuron* lain, *soma* (*cell body*) yang berfungsi menjumlahkan seluruh sinyal ke *neuron* lain dan *axon* yang berfungsi untuk meneruskan sinyal ke *neuron* lain.

Menurut Laurene Fausett [5], JST merupakan sistem pemrosesan informasi yang memiliki karakteristik serupa dengan Jaringan Syaraf Biologis, konsep JST kemudian dimodelkan secara matematis berdasarkan asumsi-asumsi:

- Pemrosesan informasi berlangsung didalam suatu elemen yang disebut *neuron*.
- Sinyal diteruskan melalui koneksi antar *neuron*.
- Setiap koneksi antar *neuron* memiliki bobot masing-masing, yang biasanya bobot tadi dikalikan dengan sinyal masukan.
- Setiap *neuron* menerapkan fungsi aktivasi terhadap jumlah dari perkalian antara sinyal input dengan bobot *neuron* sebelumnya, untuk menentukan nilai output

Paradigma pembelajaran JST secara umum dibagi menjadi dua kelompok utama yaitu pembelajaran dengan pengarahan (*supervised*) dan pembelajaran tidak dengan pengarahan (*unsupervised*), berikut akan dibahas mengenai kedua tipe pembelajaran JST.

2.2.1 Paradigma Pembelajaran JST

Yang dimaksud dengan pembelajaran dengan pengarahan (*supervised learning*) adalah hasil keluaran komputasi dari JST akan dibandingkan dengan hasil keluaran sesungguhnya, sehingga dengan selisih antara keduanya; proses penyesuaian bobot dalam jaringan dapat dilakukan. Untuk itu tipe ini memerlukan suatu data pelatihan yang berisikan data masukan serta target keluaran dari latihan.

JST, tipe ini misalnya *Multi Layer Perceptron*, *Learning Vector Quantization* (LVQ), dll.

Sedangkan yang dimaksud dengan pembelajaran dengan tanpa pelatihan (*unsupervised learning*) adalah pada proses pelatihannya JST tidak memerlukan data target, cara pembelajarannya adalah jaringan akan menyesuaikan bobotnya tanpa campur tangan dari faktor luar dan berusaha menentukan sendiri masuk kedalam kelompok mana. Jaringan macam ini misalnya *Kohonen Self-Organizing Maps* (SOM).

2.2.2 Pembelajaran Berbasis Kompetisi

Pembelajaran berbasis kompetisi dalam JST adalah suatu metode pembelajaran yang dalam memutuskan hasil keluaran-nya ditentukan oleh sebuah *neuron* pemenang, aturan ini dikenal dengan *winner take all*. Ada beberapa metode JST yang mengadopsi aturan ini diantaranya adalah SOM dan LVQ. Untuk kasus LVQ cara pembelajarannya adalah dengan melakukan perbandingan antara vektor masukan dengan semua vektor bobot untuk setiap unit keluaran, kemudian menentukan vektor bobot yang paling mirip dengan vektor masukan yang disebut sebagai vektor pemenang atau *closest vector*. Ada beberapa metode yang sering digunakan dalam menentukan vektor pemenang, dan metode tersebut pada dasarnya mengasumsikan bahwa vektor bobot (*weight vector*) untuk tiap unit keluaran (cluster) berfungsi sebagai pewakil (*reference*) untuk vektor-vektor masukan yang telah diberikan selama proses pembelajaran, sehingga vektor bobot tersebut disebut juga sebagai vektor pewakil (*reference vector*).

Untuk LVQ mekanisme kompetisi untuk menentukan vektor pewakil tersebut ditentukan oleh *Squared Euclidian Distance* yaitu memilih vektor pewakil yang memiliki jarak euclidian paling minimum dari vektor masukan untuk memenangkan kompetisi [5].

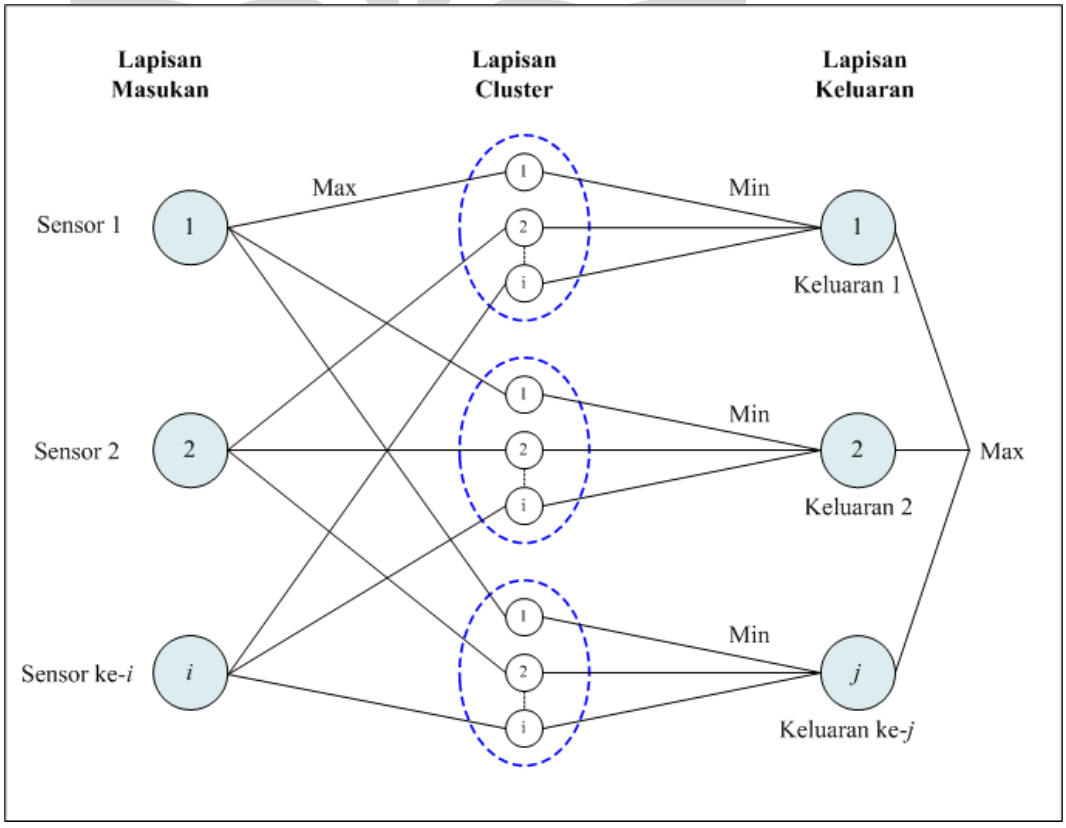
2.3 Fuzzy Neuro Learning Vector Quantization (FNLVQ)

FNLVQ [1,3] merupakan suatu algoritma pengklasifikasian yang dikembangkan dengan menggabungkan pembelajaran LVQ dengan teori *fuzzy*. Dengan mengadopsi dua metode tersebut FNLVQ memiliki keunggulan waktu

komputasi yang cepat layaknya LVQ serta tingkat pengenalan yang lebih tinggi daripada algoritma BPP serta jaringan syaraf probabilistik [3]. Selain itu FNLVQ pun sangat baik terutama dalam hal pengklasifikasian data yang memiliki tingkat kemiripan/similaritas yang tinggi. FNLVQ bekerja dengan paradigma pembelajaran dengan arahan serta dengan prinsip *winner take all*, dalam hal ini diakhir pembelajaran FNLVQ akan terdapat sebuah vektor pewakil yang paling mirip dengan vektor masukan, dan dengan dasar ini dilakukan penyesuaian vektor pewakil mendekati maupun menjauhi vektor pemenang tersebut.

2.3.1 Arsitektur Jaringan FNLVQ

Arsitektur Jaringan FNLVQ memiliki 3 (tiga) lapisan yaitu lapisan masukan, lapisan keluaran ditambah lapisan tersembunyi/lapisan cluster untuk menghitung nilai similaritas vektor masukan terhadap vektor pewakil.



Gambar 2.3 Arsitektur Jaringan FNLVQ

Untuk pengenalan aroma banyaknya *neuron* lapisan masukan adalah sebanyak sensor pada sistem penciuman elektronik, sedangkan banyaknya *neuron* pada lapisan keluaran adalah banyaknya kelas pada data aroma. Banyaknya

neuron pada lapisan cluster adalah sebanyak perkalian antara banyaknya *neuron* pada lapisan masukan dengan banyaknya *neuron* pada lapisan keluaran. Gambar 2.3 adalah arsitektur jaringan FNLVQ dengan *i* buah *neuron* pada lapisan masukan, *j* buah *neuron* pada lapisan keluaran serta *i* x *j* buah *neuron* pada lapisan cluster. Tujuan pelatihan FNLVQ adalah menentukan vektor pewakil pada lapisan cluster yang paling representatif sehingga mampu mengenali seluruh data masukan kedalam kelas aroma.

Pada sistem jaringan FNLVQ bobot dari jaringan adalah fungsi keanggotaan dari vektor pewakil dalam lapisan cluster, sedangkan fungsi aktivasinya adalah merupakan irisan fungsi keanggotaan antara vektor masukan dengan vektor pewakil.

2.3.2 Penentuan Vektor Pewakil Awal

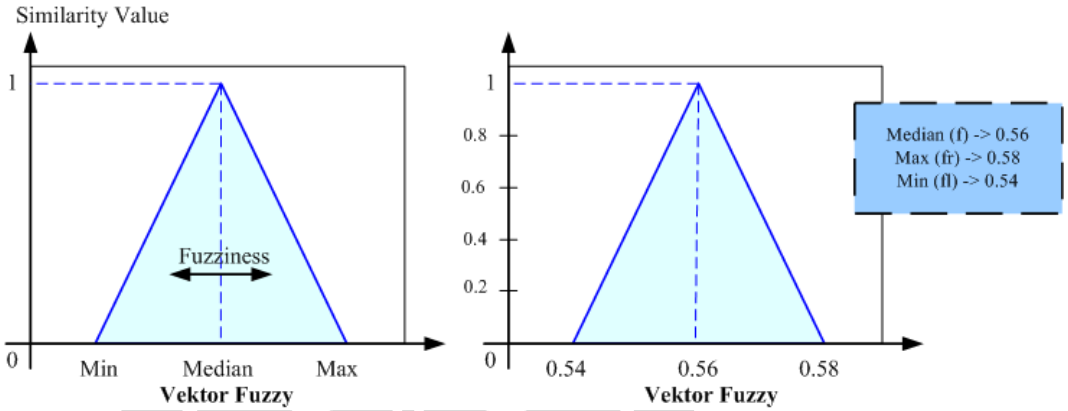
FNLVQ membutuhkan inisialisasi untuk menentukan vektor pewakil dan laju pembelajaran dalam memulai tahap pelatihan. Inisialisasi FNLVQ dilakukan dengan mengambil salah satu vektor sebagai nilai vektor pewakilnya dengan cara acak (inisialisasi acak) maupun dengan menentukan vektor masukan yang terawal sebagai vektor pewakilnya (inisialisasi awal). Selanjutnya penelitian ini menggunakan laju pembelajaran sebesar 0,01 sampai dengan 0,1.

2.3.3 Proses Fuzzifikasi

Sebelum dilakukan pelatihan maupun pengujian seluruh vektor masukan dikonversi kedalam bentuk vektor *fuzzy* dengan cata proses fuzzifikasi, proses ini dilakukan untuk merubah format data *crisp* menjadi berbentuk *fuzzy*. Hal tersebut dilakukan sebagai prasyarat sebelum data aroma dipergunakan dalam jaringan FNLVQ yakni dalam penghitungan nilai similaritas nanti, selain itu penggunaan himpunan *fuzzy* pada vektor masukan bertujuan agar distribusi frekuensi data masukan dapat direpresentasikan. Adapun tahapan-tahapan fuzzifikasi adalah sebagai berikut :

- Data dikelompokkan terlebih dahulu; misalkan 5 vektor data masukan untuk sebuah vektor fuzzy ,

- Setelah itu dihitung nilai maksmum, minimum serta tengah (median) dari kelompok vektor masukan tersebut,
- Bentuk vektor fuzzy dengan nilai maksimum kelompok vektor masukan sebagai f_r, nilai tengah-nya sebagai f dan nilai minimum-nya sebagai f_l.



Gambar 2.4 Ilustrasi Proses Fuzzifikasi

Misal dari 5 data diketahui, nilai min=0.54, max=0.58, nilai tengah=0.56, maka vektor fuzzy-nya : $f_l = 0.54$; $f = 0.56$; $f_r = 0.58$.

2.3.4 Nilai Similaritas

Untuk menentukan vektor pewakil yang paling mirip (*closest vector*) menggunakan nilai kemiripan atau similaritas yang diperoleh dengan prinsip operasi fuzzy.

Berikut langkah untuk mendapatkan nilai similaritas:

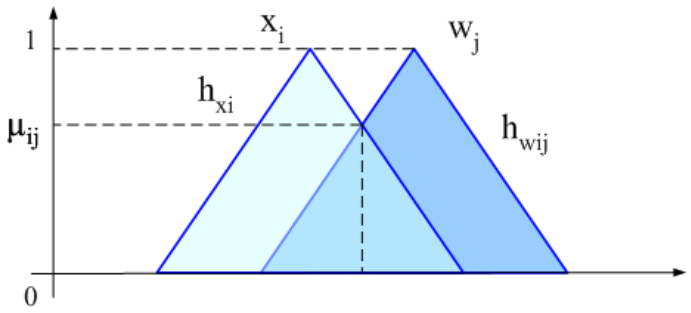
- Misalkan x adalah vektor input dari n buah sensor dan w_i adalah vektor pewakil untuk kelas i maka dapat dinyatakan bahwa:

  $x = (x_1, x_2, x_3, \ldots, x_n)$ ·····(2.2)
- dengan fungsi keanggotaan untuk vektor fuzzy x adalah:

  $h_x = (h_{x1}, h_{x2}, h_{x3}, \ldots, h_{xn})$ ·····(2.3)
- dan untuk vektor pewakil i dapat dinyatakan:

  $w_i = (w_{i1}, w_{i2}, w_{i3}, \ldots, w_{in})$ ·····(2.4)
- dengan fungsi keanggotaan untuk w_i adalah:

  $h_{wi} = (h_{wi1}, h_{wi2}, h_{wi3}, \ldots, h_{win})$ ·····(2.5)
- Maka nilai similaritas (μ_{ij}) antara vektor pewakil dengan vektor masukan dapat dijelaskan pada gambar di bawah ini:



Gambar 2.5 Nilai similaritas vektor pewakil dengan vektor training

Cara menghitung nilai similaritas dua vektor fuzzy adalah mengambil nilai maksimum dari irisan fungsi keanggotaan vektor fuzzy tersebut dengan rumus

$$\mu_{ij} = \max(h_{xj} \Lambda hw_{ij}) \quad \cdots\cdots(2.6)$$

dimana *i* adalah jenis kelas keluaran dari jaringan FNLVQ dan *j* adalah jenis sensor yang menjadi masukan FNLVQ. Nilai similaritas ini dihitung untuk semua vektor pewakil yang ada dengan vektor masukan

2.3.5 Pembelajaran FNLVQ

Setelah semua vektor pewakil dihitung nilai similaritasnya dengan vektor input, maka setiap elemen di lapisan keluaran dicari nilai similaritas terkecilnya. Pencarian nilai similaritas minimum ini menggunakan rumus:

$$\mu_i = \min(\mu_{ij}) \quad \cdots\cdots(2.7)$$

dimana *i* adalah jenis kelas keluaran dari jaringan FNLVQ ini dan *j* adalah jenis sensor yang menjadi masukan FNLVQ.

Penentuan vektor pewakil pemenang dilakukan dengan cara mencari nilai similaritas terbesar yang ada pada tiap-tiap elemen lapisan keluaran tadi. Bila hasil keluarannya mempunyai similaritas bernilai nol artinya vektor masukan tersebut mempunyai jenis aroma yang tidak diketahui.

Setelah diketahui vektor pewakil pemenang, tahapan selanjutnya adalah menentukan perubahan posisi vektor pemenang terhadap vektor masukan serta perubahan *fuzziness* dari vektor pemenang. Terdapat tiga kasus dalam pergeseran vektor pewakil pemenang yaitu:

- Bila lapisan keluaran menyatakan bahwa kelas vektor masukan x sama dengan vektor pewakil pemenang w_c maka posisi vektor pemenang akan digeser mendekati vektor masukan, kemudian vektor pemenang tersebut diperlebar *fuzziness*-nya. Pergeseran vektor fuzzy dilakukan dengan menggunakan rumus berikut:

$$w_{cj}(t+1) = w_{cj}(t) + \alpha(t)[\{1 - \mu_{cj}(t)\} * \{x_j(t)\} - w_{cj}(t)\}] \quad \cdots\cdots(2.8)$$

dimana,

j = 1, 2, 3, ..., n yaitu banyaknya sensor masukan

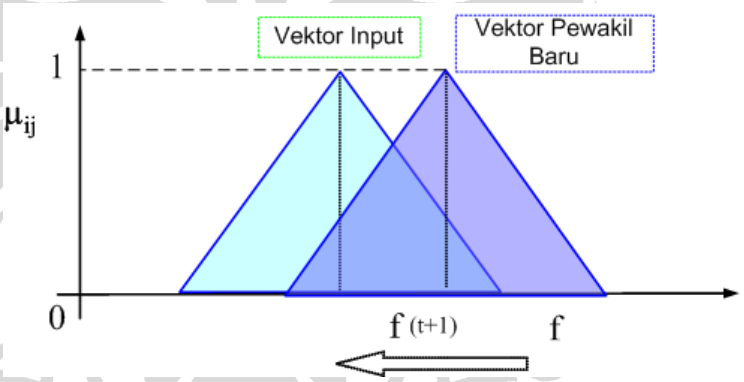
$w_{cj}(t+1)$ = posisi vektor pewakil setelah pergeseran

$w_{cj}(t)$ = posisi vektor pewakil sebelum pergeseran

$\alpha(t)$ = laju pembelajaran

$\mu_{cj}(t)$ = nilai similaritas

$x_j(t)$ = vektor masukan



Gambar 2.6 Vektor Pemenang Pewakil didekatkan kepada Vektor Masukan

Untuk pelebaran fuzziness vektor pemenang pewakil, bisa dilakukan dengan dua cara yang pertama menggunakan faktor konstan digunakan rumus berikut:

$$fl(t+\ ) = fl(t) - (1+\beta) * \{f(t) - fl(t)\} \quad \cdots\cdots(2.9)$$

$$fr(t+\ ) = fr(t) + (1+\beta) * \{fr(t) - f(t)\} \quad \cdots\cdots(2.10)$$

$$f(t+\ ) = w_i(t+\ ) \quad \cdots\cdots(2.11)$$

yang kedua, menggunakan faktor variabel digunakan rumus berikut:

$$fl(t+\ ) = fl(t) - (1-\mu) * \{(1-\kappa)\} * \{f(t) - fl(t)\} \quad \cdots\cdots(2.12)$$

$$fr(t+\ ) = fr(t) + (1-\mu) * \{(1-\kappa)\} * \{fr(t) - f(t)\} \quad \cdots\cdots(2.13)$$

$f(t+\ ) = w_i(t+\ )$ ·····(2.14)

dimana,

$w_i(t+\ )$ = vektor pewakil pemenang setelah dilakukan pergeseran.

$w_i(t)$ = vektor pewakil pemenang sebelum dilakukan pergeseran.

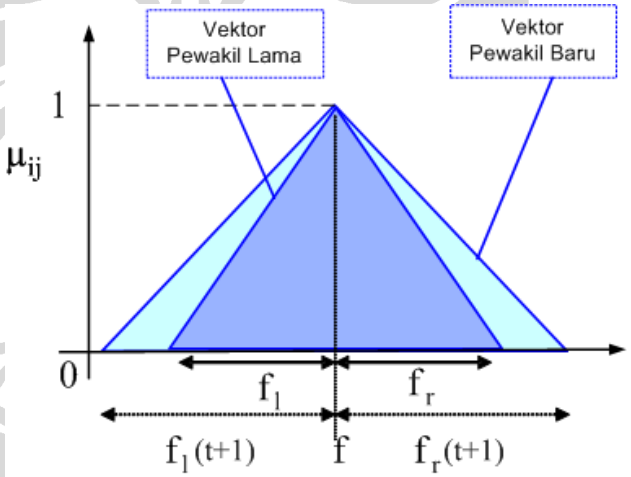
$\alpha_{\ }(t)$ = laju pembelajaran, dengan kisaran 0<α≤1.

β, γ = nilai konstanta untuk memperlebar maupun mempersempit fuzziness, dengan kisaran nilai 0 sampai dengan 1.

η, κ = nilai variabel untuk memperlebar maupun mempersempit fuzziness, dengan rumus:

$$\eta_{\ }(t+\ ) = \frac{1}{100}\{1 - \alpha_{\ }(t+\ )\}$$

$$\kappa_{\ }(t+\ ) = [\ - \alpha_{\ }(t+\ )$$



Gambar 2.7 Vektor Pemenang Pewakil didekatkan kepada Vektor Masukan

- Bila lapisan keluaran menyatakan bahwa kelas vektor masukan tidak sama dengan vektor pewakil pemenang maka posisi vektor pewakil pemenang akan digeser menjauhi vektor masukan kemudian vektor pewakil pemenang tersebut dipersempit *fuzziness*nya. Pergeseran dilakukan dengan menggunakan rumus berikut:

$w_{cj}(t+1) = w_{cj}(t) - \alpha(t)[\{1 - \mu_{cj}(t)\} * \{x_j(t)\} - w_{cj}(t)\}]$·····(2.15)

dengan,

j = 1, 2, 3, ..., n yaitu banyaknya sensor masukan

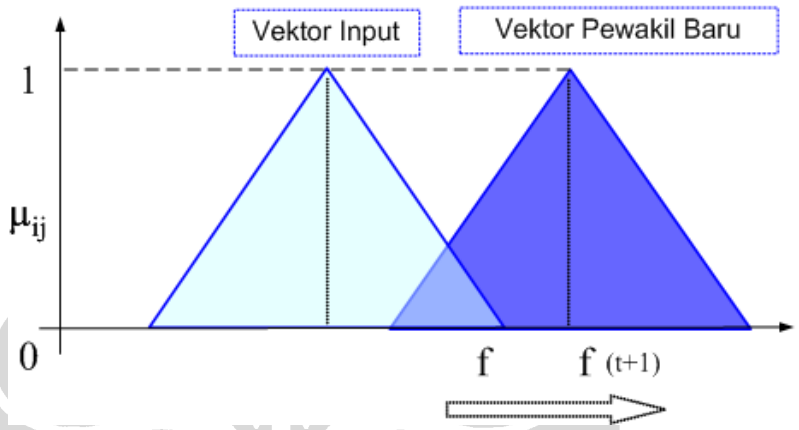
$w_{cj}(t+1)$ = posisi vektor pewakil setelah pergeseran

$w_{cj}(t)$ = posisi vektor pewakil sebelum pergeseran

$\alpha(t)$ = laju pembelajaran

$\mu_{cj}(t)$ = nilai similaritas

$x_j(t)$ = vektor masukan



Gambar 2.8 Vektor Pemenang Pewakil dijauhkan terhadap Vektor Masukan

Untuk penyempitan fuzziness vektor pewakil pemenang, bisa dilakukan dengan dua cara yang pertama menggunakan faktor konstan digunakan rumus berikut:

$$fl(t+\ ) = fl(t) + (1+\gamma) * \{f(t) - fl(t)\} \quad \cdots\cdots(2.16)$$

$$fr(t+\ ) = fr(t) - (1+\gamma) * \{fr(t) - f(t)\} \quad \cdots\cdots(2.17)$$

$$f(t+\ ) = w_i(t+\ ) \quad \cdots\cdots(2.18)$$

yang kedua, menggunakan faktor variabel digunakan rumus berikut:

$$fl(t+\ ) = fl(t) + (1-\mu) * \{(1-\kappa)\} * \{f(t) - fl(t)\} \quad \cdots\cdots(2.19)$$

$$fr(t+\ ) = fr(t) - (1-\mu) * \{(1-\kappa)\} * \{fr(t) - f(t)\} \quad \cdots\cdots(2.20)$$

$$f(t+\ ) = w_i(t+\ ) \quad \cdots\cdots(2.21)$$

dimana,

$w_i(t+\ )$ = vektor pewakil pemenang setelah dilakukan pergeseran.

$w_i(t)$ = vektor pewakil pemenang sebelum dilakukan pergeseran.

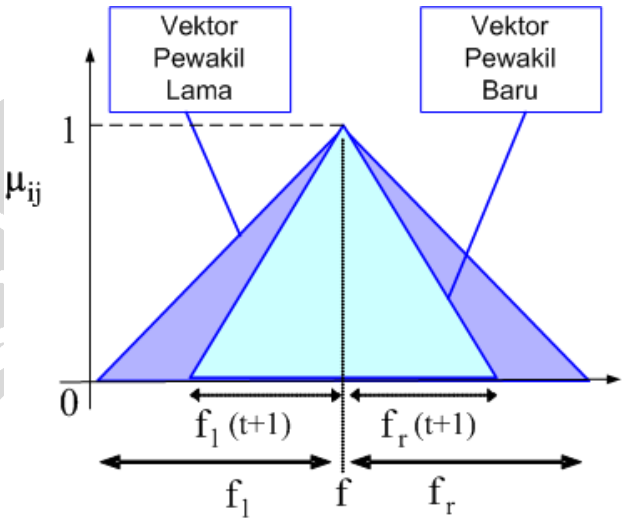
$\alpha(t)$ = laju pembelajaran, dengan kisaran $0<\alpha\leq 1$.

β, γ = nilai konstanta untuk memperlebar maupun mempersempit fuzziness, dengan kisaran nilai 0 sampai dengan 1.

η, κ = nilai variabel untuk memperlebar maupun mempersempit fuzziness, dengan rumus:

$$\eta(t+1) = \frac{1}{100}\{1 - \alpha(t+1)\}$$

$$\kappa(t+1) = 1 - \alpha(t+1)$$



Gambar 2.9 Vektor Pewakil baru diperkecil Fuzziness-nya

- Bila lapisan keluaran menyatakan bahwa similaritas vektor pewakil pemenang dan vektor masukan adalah nol maka semua vektor pewakil pemenang dengan nilai similaritas nol saja yang diperlebar *fuzziness*nya, dengan menggunakan rumus:

$$w_{ij}(t+1) = \delta(t) * w_{ij}(t) \quad \cdots\cdots(2.22)$$

dengan,

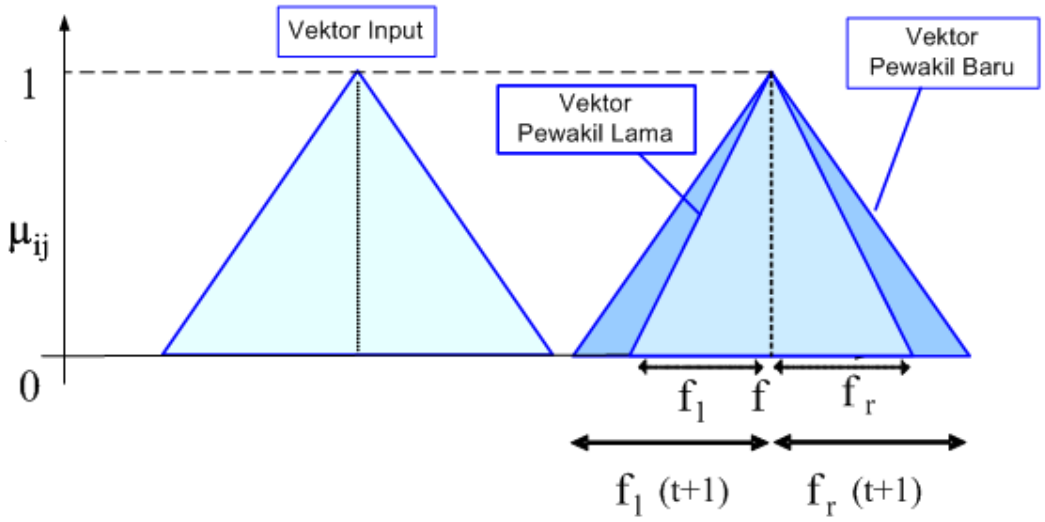
$$\delta(t) = 1.1 \quad \cdots\cdots(2.23)$$

dimana,

$w_i(t+1)$ = vektor pewakil pemenang setelah dilakukan pergeseran.

$w_i(t)$ = vektor pewakil pemenang sebelum dilakukan pergeseran.

ξ = konstanta 1.1



Gambar 2.10 Pembelajaran jika Output FNLVQ sama dengan Target namun Nilai Similaritas-nya nol

2.3.6 Pengujian FNLVQ

Pengujian pada jaringan FNLVQ dilakukan sama halnya dengan proses pelatihan, perbedaannya pada fase pengujian tidak dilakukan penyesuaian vektor pewakil. Hasil dari pengujian berupa vektor pewakil pemenang yang akan dibandingkan dengan target keluaran sesungguhnya. Dengan demikian jika semua data pengujian telah diujicobakan di jaringan FNLVQ maka selanjutnya dilakukan penghitungan tingkat pengenalan (*recognition rate*) keseluruhan data pengujian. Hasil dari pengujian FNLVQ akan dibahas pada sub-bab 4.1.

2.4 Matriks Similarity Analysis (MSA)

MSA merupakan suatu analisis terhadap matriks yang menggambarkan vektor *codebook* (rata-rata vektor pewakil pada sebuah epoh). Dengan analisis matriks tersebut dapat ditentukan kapan suatu pembelajaran akan berhenti, caranya adalah dengan menentukan batasan nilai similaritas vektor pewakil.

2.4.1 FNLVQ dengan MSA

Pembelajaran FNLVQ dihentikan apabila ia telah menyentuh epoh sesuai dengan batasan (*threshold*) yang ditentukan sebelumnya, namun pada saat ini

hasil tingkat pengenalan yang didapatkan dari FNLVQ belum tentu optimal. Oleh karena itu diperlukan suatu alat analisis yang mampu memberi kepastian kapan suatu proses pembelajaran FNLVQ harus berhenti dan meningkatkan pengenalan.

Untuk membentuk matriks similaritas pada suatu epoh, maka setiap keluaran FNLVQ (vektor pewakil pemenang) untuk kelas tertentu dijumlahkan kedalam matriks, setelah epoh selesai maka nilai MSA adalah rata-rata dari penjumlahan tadi. MSA dihitung dengan rumus berikut:

$$m_{mj} = \frac{1}{N}\sum_{k=1}^{N} \max \min \mu_{ij}(k) \quad \cdots\cdots(2.24)$$

dimana,

i = banyak vektor masukan (jumlah sensor)

j = vektor pewakil dalam suatu cluster FNLVQ

m = banyak vektor masukan = i

N = semua vektor pelatihan per-kelas

k = 1,2,3,...N

Berikut adalah gambar dari matriks similaritas pada FNLVQ:

$$M = \begin{bmatrix} m_{11} & m_{21} & . & . & m_{m1} \\ m_{12} & m_{22} & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ m_{1j} & . & . & . & m_{nn} \end{bmatrix}$$

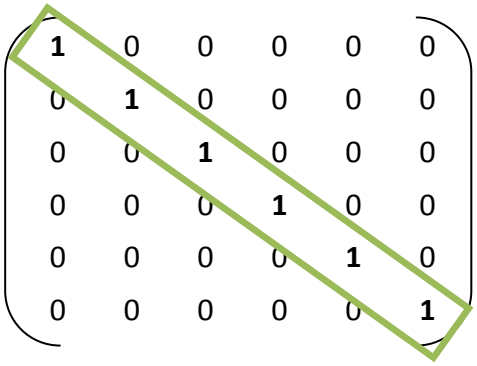
Gambar 2.11 Matriks Similaritas

dimana,

M = matriks similaritas dengan dimensi n x n

n = banyaknya kelas aroma

Kondisi ideal yang diinginkan dari MSA adalah mendapatkan output pengenalan pada FNLVQ yang mendekati matriks identitas. Artinya pengenalan pada tahapan pelatihan terhadap suatu kelas dapat dilakukan 100%, dan pengklasifikasian data yang dilakukan oleh FNLVQ (vektor pewakil) terhadap suatu kelas tidak terganggu oleh posisi vektor pewakil kelas yang lain.



Gambar 2.12 Kondisi Ideal MSA adalah mendekati matriks identitas

Bilamana jaringan FNLVQ-MSA sudah mencapai kondisi matriks identitas, maka proses pelatihan dapat dihentikan, demikian pula jika nilai diagonal MSA sudah mencapai nilai batasan (*threshold*) yang ditentukan maka pelatihan dapat dihentikan.

2.4.2 Penyesuaian Vektor Pewakil dengan MSA

Dengan MSA maka ditambahkan algoritma penyesuaian vektor pewakil baru kedalam algoritma FNLVQ. Algoritma-nya adalah sebagai berikut:

- Pada awal pelatihan FNLVQ buat MSA awal,
- Pada pelatihan selanjutnya bandingkan tiap diagonal pada MSA sebelumnya dengan MSA sekarang.
- Bilamana nilai diagonal dari MSA sebelumnya lebih besar daripada nilai diagonal MSA sekarang, maka vektor pewakil pada lapisan cluster yang nilai diagonal-nya lebih besar tersebut digantikan dengan vektor pewakil yang lama. Dengan asumsi nilai vektor pewakil sebelumnya lebih baik daripada nilai vektor pewakil sekarang, maka vektor pewakil perlu dipertahankan.

Untuk lebih jelasnya, berikut adalah ilustrasi penyesuaian vektor pewakil pada MSA:

Tabel 2.1 Ilustrasi Penyesuaian Vektor Pewakil dengan MSA

MSA pada Epoch ke-1

Similaritas	Kategori Output					
	I	II	III	IV	V	VI
Custer I	0.4412736	0.0013117	0	0	0	0
Custer II	0	0.2614505	0.0494303	0	0	0
Custer III	0	0	0.4042655	0	0.0001374	0
Custer IV	0	0	0	0.5174772	0	0
Custer V	0	0	0	0	0.504989	0
Custer VI	0	0	0	0	0	0.4132916
Total	0.4412736	0.2601388	0.3548352	0.5174772	0.5048516	0.4132916

(a)

MSA pada Epoch ke-2

Similaritas	Kategori Output					
	I	II	III	IV	V	VI
Custer I	0.5664314	0	0	0	0	0
Custer II	0	0.591884	0.0644975	0	0	0
Custer III	0	0.335887	0.7693601	0	0	0
Custer IV	0	0	0	0.813216	0	0
Custer V	0	0	0	0	0.8189164	0
Custer VI	0	0	0.062193	0	0	0.8123799
	0.5664314	0.255997	0.6426696	0.813216	0.8189164	0.8123799

(b)

Dapat dilihat di Tabel 2.1 bahwa pengenalan untuk kelas output ke-2 pada epoh ke-1 lebih baik daripada epoh ke-2; Oleh karena itu vektor pewakil yang berada pada cluster ke-2 dipertahankan untuk pelatihan epoh berikutnya.

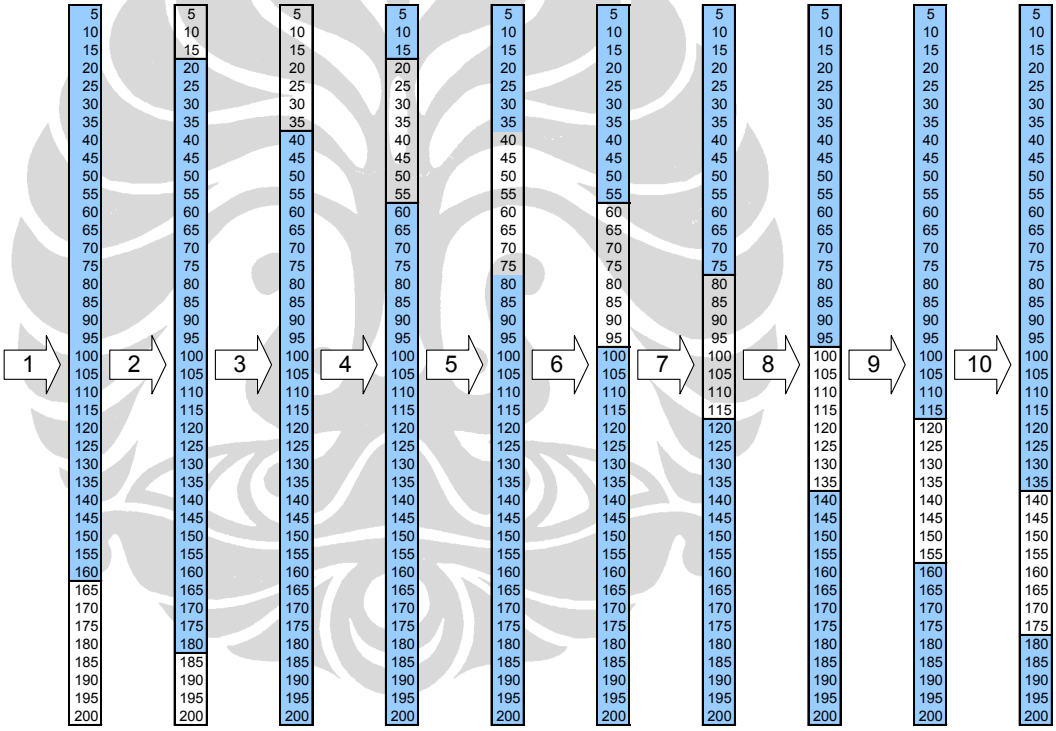
2.5 Validasi Silang

Validasi silang (*cross-validation*) adalah suatu metode statistik yang digunakan untuk melakukan pemisahan/partisi dari suatu data menjadi sub-bagian, dengan demikian analisis data dapat dilakukan terhadap sub-bagian tersebut, sisa dari sub-bagian tadi dipergunakan untuk melakukan konfirmasi kebenaran serta validasi sub-bagian awal tadi.

Terdapat beberapa macam validasi-silang diantaranya adalah validasi *Hold Out*, validasi sub-sampling acak berulang, validasi silang (*Leave One Out Cross-validation*/LOOCV) serta validasi silang K-Fold [5].

Yang dipergunakan dalam penelitan ini adalah validasi silang K-Fold, dimana data hasil observasi di pisah/partisi menjadi sebanyak K sub-bagian. Dari K sub-bagian tersebut, sebuah sub-bagian dipergunakan untuk validasi/pengujian dan sisanya K – 1 digunakan untuk pelatihan. Proses validasi silang kemudian diulangi sebanyak K kali, dengan setiap sub-bagian dipergunakan hanya sekali saja. Hasil dari validasi tersebut kemudian diambil nilai rata-rata-nya untuk mendapatkan nilai akhir. Keuntungan metode ini adalah seluruh data observasi digunakan baik pada tahapan pelatihan maupun pengujian. Dalam penelitian ini digunakan validasi silang 10-Fold.

Berikut adalah ilustrasi validasi silang untuk pelatihan menggunakan 80% data latih dan 20% data uji:



Gambar 2.13 Ilustrasi validasi silang

Misalkan terdapat 200 data awal, maka dengan validasi 10-Fold dibuat 10 sub-bagian data sehingga semua data awal dapat digunakan sebagai pelatihan dan pengujian. Dari tiap sub-bagian tersebut dihitung tingkat pengenalannya, setelah semua sub-bagian didapatkan tingkat pengenalannya maka tingkat pengenalan validasi silang diperoleh melalui rata-rata seluruh pengenalan pada tiap sub-bagian validasi silang.

2.6 Particle Swarm Optimization (PSO)

Sejak pertama kali diperkenalkan oleh Kennedy dan Eberhart [7] [8] pada tahun 1995, PSO telah mengalami kemajuan yang sangat pesat dan telah banyak diterapkan dalam berbagai masalah pencarian. Pada umumnya peningkatan algoritma PSO bertujuan untuk meningkatkan sifat konvergen dan divergen dari algoritma tersebut. Sebelum melihat modifikasi FNLVQ-PSO, akan dibahas terlebih dahulu algoritma PSO standar, yaitu algoritma PSO yang pertama kali diperkenalkan oleh mereka pada tahun 1995.

PSO terdiri dari kumpulan-kumpulan partikel yang bergerak pada ruang pencarian. Partikel-partikel ini bergerak mencari solusi. Posisi partikel merupakan hasil dari posisi partikel sebelumnya ditambah dengan kecepatan pergerakannya. Kecepatan partikel ditentukan dengan rumus sebagai berikut:

$$\mathbf{v}_i(t+\ ) = \chi\ \mathbf{v}_i(t) + c_1 Rand\ ()(\mathbf{p}_i(t) - \mathbf{x}_i(t)) + c_2 Rand\ ()(\mathbf{p}_g(t) - \mathbf{x}_i(t))) \quad \cdots\cdots\cdots(2.25)$$

$\mathbf{x}_i(t)$ merupakan vektor posisi partikel ke-i (i = 1,2,3,...) pada iterasi ke-t (t = 0,1,2,...). $\mathbf{v}_i(t)$ merupakan kecepatan partikel ke-i pada iterasi ke-t, χ merupakan faktor konstriksi yang bernilai kurang dari 1, c_1 dan c_2 adalah faktor akselerasi yang mengatur distribusi dari pengaruh komponen kognitif dan sosial, $\mathbf{p}_i$ dan $\mathbf{p}_g$ adalah *local best* dan *global best*, $\mathbf{x}_i(t)$ adalah posisi partikel ke-i pada iterasi ke-t dan *Rand*() adalah bilangan acak antara 0 sampai 1.

Selanjutnya posisi partikel adalah posisi.partikel sebelumnya ditambah dengan kecepatan pergerakannya.

$$\mathbf{x}_i(t+\ ) = \mathbf{x}_i(t) + \mathbf{v}_i(t+\ ) \quad \cdots\cdots\cdots(2.26)$$

Faktor kecepatan yang dihitung dengan rumus (2.25) menentukan pergerakan partikel selanjutnya. Kecepatan ini diperoleh dengan menggunakan informasi dari partikel itu sendiri dan partikel-partikel lainnya. Informasi dari partikel itu sendiri disebut dengan komponen kognitif, di mana nilainya merupakan jarak dari posisi partikel tersebut dengan posisi terbaik yang telah

diperoleh partikel tersebut selama pencarian. Posisi terbaik partikel tersebut dinamakan *local best* ($\mathbf{p}_i$). Setiap partikel memiliki informasi *local best*-nya masing-masing. Dari *local best* yang telah diperoleh dari setiap partikel, tentu ada lokasi yang memiliki posisi terbaik di antara semua *local best* lainnya. Posisi terbaik ini disebut dengan nama *global best* ($\mathbf{p}_g$). *Global best* inilah yang disebut dengan komponen sosial dari PSO.

Untuk mengetahui seberapa baik posisi yang telah ditemukan, maka diperlukan suatu fungsi yang dapat mengukurnya. Fungsi ini disebut dengan fungsi *fitness* $f(\mathbf{x})$. fungsi ini memiliki domain vektor posisi $\mathbf{x}$ dengan keluaran berupa nilai tertentu yang menunjukan seberapa baik vektor posisi $\mathbf{x}$. Makin baik nilainya, makin dekat posisi tersebut dengan solusi. Dengan fungsi ini, posisi-posisi yang telah ditemukan bisa dibandingkan kedekatannya dengan solusi yang dicari. Fungsi *fitness* berbeda pada tiap-tiap permasalahan bergantung pada masalah yang akan dihadapi.

Pada parameter c_1 dan c_2, jika c_1 lebih besar maka pengaruh komponen kognitif akan lebih besar daripada komponen sosial dan begitu pula sebaliknya. Komponen kognitif yang lebih besar akan membuat partikel bergerak lebih menuju kepada *local best*-nya sendiri, sementara pengaruh komponen sosial yang besar akan membuat partikel bergerak lebih menuju kepada *global best*. Untuk permasalahan dengan solusi tunggal, pengaruh komponen sosial yang lebih besar akan memberikan hasil yang lebih efisien. Sementara pengaruh komponen kognitif yang lebih besar mungkin akan memberikan keuntungan pada permasalahan dengan banyak solusi [9].

Setelah kecepatan setiap partikel dihitung dengan rumus (2.25), perubahan posisi dari tiap partikel juga akan dihitung dengan menggunakan rumus (2.26). Hal ini akan dilakukan pada setiap iterasi. Secara keseluruhan algoritma PSO adalah sebagai berikut.

Algoritma PSO

```
repeat
   for each particle i
      // set local best
      if f(xi) is better than f(lbest) then
            lbest = xi
      end if
      // set global best
      if f(xi) is better than f(gbest) then
            gbest = xi
      end if
   end for
   for each particle i
      update velocity using equation (3.1)
      update position using equation (3.2)
   end for
until stopping condition is true
```

Algoritma PSO bersifat konvergen dimana pada iterasi tertentu seluruh partikel akan menuju ke satu titik *global best*. Jika hal ini terjadi, maka kemungkinan pergerakan partikel-partikel tersebut tidak akan menjadi terlalu signifikan yang membuat perhitungan PSO menjadi tidak efisien. Pada saat inilah akan ditentukan apakah pencarian akan dilanjutkan atau tidak. Sifat konvergen ini menentukan apakah solusi yang ditemukan sudah dapat dianggap berhasil atau tidak.

Bagian terakhir dari algoritma PSO adalah kondisi tertentu yang dapat menyebabkan pencarian berhenti. Terdapat beberapa kondisi yang dapat dijadikan parameter untuk membuat PSO berhenti [8].

1. **Iterasi yang terjadi sudah melebihi batas waktu yang ditentukan**. Batas waktu ini menjadi hal yang penting dalam kondisi ini. Jika batas waktu terlalu singkat, pencarian akan berhenti sebelum solusi ditemukan. Jika batas waktu terlalu panjang, pada pencarian yang gagal PSO akan terus berjalan walaupun PSO sudah mencapai titik konvergen dimana pergerakan menjadi tidak signifikan.

2. **Solusi yang bisa diterima sudah ditemukan**. Jika dalam pencarian ternyata ditemukan solusi yang dapat diterima, maka pencarian akan dihentikan. Batas diterima di sini adalah ambang batas dimana solusi yang ditemukan sudah sesuai dengan apa yang diinginkan. Jika ambang batas terlalu jauh, solusi yang ditemukan mungkin tidak cukup sesuai dengan apa yang diinginkan. Sementara ambang batas yang terlalu kecil akan membuat PSO cukup kesulitan untuk menemukan solusi yang dapat memenuhinya.
3. **Tidak ada peningkatan dalam beberapa iterasi**. Terdapat beberapa cara untuk mengetahui apakah peningkatan masih cukup signifikan untuk dapat dilanjutkan atau tidak. Jika perubahan posisi partikel cukup kecil, maka dapat dipastikan PSO sudah menuju ke titik konvergen. Cara lain, jika perhitungan kecepatan pada PSO memberikan hasil yang mendekati nol maka partikel hampir tidak bergerak, dan perhitungan PSO dapat dihentikan.
4. **Radius normal dari kumpulan partikel mendekati nol**. Radius normal dihitung sebagai berikut.

$$R_{norm} = \frac{R_{\max}}{diameter\ \ (S)} \qquad \text{.....(2.27)}$$

diameter(S) merupakan diameter dari kumpulan partikel awal dan R_{max} merupakan jarak maksimum partikel dari *global best*.

$$R_{\max} = \left\| \mathbf{x}_m - \mathbf{x}_g \right\| \qquad m = 1,2,...,n(partikel) \qquad \text{.....(2.28)}$$

dan

$$R_{\max} \geq \left\| \mathbf{x}_i - \mathbf{x}_g \right\| \qquad \forall i = 1,2,...,n(partikel) \qquad \text{.....(2.29)}$$

jika R_{max} mendekati nol, maka kumpulan partikel memiliki kemungkinan kecil untuk melakukan peningkatan. Perlu didefinisikan batasan cukup dekat dengan nol. Jika batasan terlalu besar, maka PSO akan berhenti sebelum solusi optimum ditemukan. Sebaliknya, jika batasan terlalu kecil, akan dibutuhkan iterasi yang lebih panjang untuk dapat mencapat batasan tersebut.

5. **Ketika perubahan solusi yang ditemukan mendekati nol**. Kondisi ini terjadi dengan melihat perubahan solusi yang telah ditemukan. Perubahan solusi dihitung dengan rumus berikut.

$$f' = \frac{f(\mathbf{p}_g(t)) - f(\mathbf{p}_g(t-))}{f(\mathbf{p}_g(t))} \quad \text{..... (2.30)}$$

f(**x**) adalah fungsi *fitness* yang menghitung seberapa besar solusi telah ditemukan pada posisi **x**.

Jika f' terlalu kecil, maka kumpulan partikel dapat diasumsikan sudah konvergen menuju satu titik. Di sini juga dibutuhkan batasan seberapa dekat perubahan yang terjadi dengan nol. Seperti pada kondisi-kondisi sebelumnya, jika ambang batas terlalu besar, maka PSO akan berhenti sebelum solusi yang optimal ditemukan. Begitu pula sebaliknya.

Konvergen dalam PSO seperti yang telah dibahas di atas belum tentu berarti partikel sudah menemukan titik optimum (lokal atau global). Konvergen di sini berarti PSO telah mencapai titik keseimbangan (*equilibrium*). Partikel-partikel bergerak hanya menuju ke satu titik, bisa itu berupa titik optimum atau bukan.

2.7 Modifikasi PSO untuk masalah dengan banyak solusi

Pada dasarnya algoritma PSO standar merupakan algoritma untuk melakukan pencarian solusi tunggal. Hal ini terlihat dari sifat algoritma PSO yang konvergen menuju satu titik *global best*. Namun, bukan berarti algoritma PSO tidak cocok digunakan untuk melakukan pencarian banyak solusi. Beberapa modifikasi PSO telah dibuat untuk menangani masalah dengan banyak solusi.

Dalam bidang *Evolutionary Computation*, algoritma yang menangani masalah dengan banyak solusi disebut dengan algoritma *niching*. Proses yang menemukan banyak solusi tersebut, atau *niche*, disebut dengan *speciation*. Model dari algoritma *niching* berasal dari proses alami makhluk hidup, ketika sejumlah individu bersaing untuk mendapatkan sumber makanan yang jumlahnya terbatas

di alam bebas. Persaingan ini menghasilkan sifat-sifat alami dari makhluk hidup dimana individu-individu yang ada membentuk suatu kelompok sendiri-sendiri sesuai dengan jenis kebutuhannya. Horn [10] mendefinisikan *niching* sebagai berikut.

"form of cooperation around finite, limited resources, resulting in the lack of competition between such areas, and causing the formation of species for each niche"

Dari definisi ini, *niche* adalah bagian dari ruang pencarian. Sementara, *species* adalah bagian dari kumpulan partikel yang bersaing dalam lingkungan tersebut.

Dalam istilah *computational optimization*, sebuah *niche* menggambarkan satu solusi dari suatu masalah, sementara *species* adalah kelompok dari individu-individu yang bersifat konvergen ke satu *niche*.

Niching pada PSO disebut dengan nama *PSO niching*. Dengan ini, algoritma PSO dimodifikasi hingga mampu menangani permasalahan dengan banyak solusi. *PSO niching* dibagi menjadi tiga macam sesuai dengan sifatnya yaitu *sequential PSO niching*, *parallel PSO niching* dan *quasi-sequential PSO niching*.

2.7.1 Sequential PSO Niching

Sequential PSO niching adalah algoritma PSO yang dapat melakukan pencarian solusi dengan cara sekuensial. Ketika algoritma PSO sudah menemukan salah satu solusi, maka solusi tersebut ditandai dan kemudian dihapus dalam ruang pencarian sehingga PSO bisa mencari solusi lainnya. Proses ini terus berulang hingga mencapai kondisi yang dapat membuat PSO berhenti.

Permasalahan pada *sequential PSO niching* adalah bagaimana menghilangkan solusi yang telah ditemukan dari area pencarian. Jika solusi tersebut tidak dihilangkan, maka terdapat kemungkinan partikel akan kembali mencari solusi tersebut. Hal ini sangat tidak efisien karena hanya mengulang pencarian yang sama dengan sebelumnya.

Parsopoulos *et al*. [11] [12] mengembangkan pendekatan *sequential PSO niching* yang menemukan semua titik optimum global dalam fungsi kontinu. Titik optimum lokal dihilangkan dengan fungsi *stretching*. Fungsi *stretching* adalah transformasi dua-tahap yang menghilangkan titik optimum lokal dan mempertahankan global optimum. Parsopoulos *et al.* [11] [12] menunjukan bahwa fungsi *stretching* mampu menyelesaikan permasalahan konvergen pada titik optimum lokal.

2.7.2 Parallel PSO Niching

Sesuai dengan namanya, algoritma ini bekerja secara paralel. Setiap *niche* dicari secara paralel dan penanganannya dilakukan selama dalam proses pencarian. Salah satu algoritma dari *parallel PSO niching* adalah *NichePSO*.

NichePSO dikembangkan untuk mencari masalah dengan banyak solusi [13-14]. Dasar dari *NichePSO* adalah sifat *self-organization* dalam membentuk subkelompok-subkelompok baru. Tidak ada informasi yang mengalir antara subkelompok yang satu dengan subkelompok lainnya. Sifat independen ini membuat subkelompok yang ada dapat mencari *niche-niche* yang ada.

NichePSO berawal dari seluruh kelompok partikel yang bernama kelompok utama. Setiap partikel dalam kelompok ini bergerak sendiri-sendiri tanpa menggunakan komponen sosial dalam pergerakannya. Hal ini menyebabkan terjadinya pencarian lokal. Ketika salah satu partikel mulai menemukan *niche* yang potensial, sebuah subkelompok partikel dibentuk dengan partikel lain yang berada pada jarak tertentu dari partikel yang pertama kali menemukan *niche* tersebut. Selanjutnya partikel-partikel yang berada pada subkelompok tersebut dihapus dari kelompok utama. Subkelompok ini lalu menelusuri *niche* tersebut hingga menemukan solusinya dengan menggunakan model PSO standar.

Seiring dengan berjalannya waktu, anggota kelompok utama akan berkurang dengan terbentuknya subkelompok-subkelompok baru. Jika suatu partikel dari kelompok utama bergerak mendekati area suatu subkelompok, maka partikel tersebut akan menjadi anggota subkelompok dan dihapus dari kelompok utama.

Setiap subkelompok memiliki informasi *global best* dan posisinya masing-masing. Setiap subkelompok bisa mencari niche yang berbeda-beda. Namun tidak ada jaminan setiap subkelompok mencari *niche* yang berbeda. Bisa saja ada beberapa subkelompok yang mencari *niche* yang sama. Hal ini terjadi karena tidak ada komunikasi antara subkelompok yang satu dengan yang lain. Jika hal ini terjadi, maka subkelompok-subkelompok tersebut akan berkompetisi dalam mencari niche tersebut. Hal ini menyebabkan pencarian dilakukan berulang dan tidak efisien. Untuk mencegah hal ini, maka subkelompok-subkelompok tersebut digabung menjadi subkelompok yang lebih besar. Dengan penggabungan ini, pencarian akan berjalan lebih divergen dengan tidak melakukan pencarian yang berulang-ulang.

Beberapa studi mengenai performa dari *NichePSO* telah dilakukan. Brits *et al*. menunjukan bahwa dengan jumlah partikel yang cukup, *NichePSO* menemukan hampir seluruh titik optimum pada masalah dengan dimensi yang cukup besar [13] [14] [15].

2.7.3 Quasi-Sequential Niching

Algoritma ini merupakan gabungan dari *sequential* dan *parallel PSO niching*. Di sini *niche* ditelusuri secara sekuensial, namun penanganannya dilakukan secara paralel. Ketika sebuah *niche* ditemukan, *niche* tersebut tidak langsung dihilangkan. Algoritma ini melanjutkan pencarian *niche* yang lainnya, sementara *niche-niche* yang telah ditemukan ditangani secara paralel.

Salah satu algoritma *Quasi-Sequential Niching* adalah *Vector-Based Niching PSO*. Proses identifikasi *niche* berdasarkan asumsi bahwa *dot product* antara vektor kecepatan yang menuju ke arah yang berbeda akan memberikan hasil negatif, sementara vektor dengan arah yang sama akan memberikan hasil *dot product* yang positif Untuk setiap partikel *i*, dua vektor kecepatan akan dihitung sebagai berikut.

- Kecepatan vektor kognitif

$$\mathbf{v}_i(t) = \mathbf{p}_i(t) - \mathbf{x}_i(t) \quad \cdots\cdots(2.31)$$

- Kecepatan vektor sosial

$$\mathbf{v}_i(t) = \mathbf{p}_g(t) - \mathbf{x}_i(t) \quad \cdots\cdots(2.32)$$

$\mathbf{p}_g$ merupakan *global best* dari seluruh partikel yang belum menuju *niche*.

Jika *dot product*, $\mathbf{v}_i(t) \bullet \mathbf{v}_i(t)$, dari kedua vektor di atas positif, maka partikel *i* akan dimasukan ke dalam pencarian *niche* tersebut. Jika hasilnya negatif maka partikel *i* tidak akan dimasukan ke dalam pencarian *niche* tersebut. Seluruh partikel yang telah masuk ke dalam pencarian *niche* tersebut akan dikeluarkan dari kelompok utama partikel untuk membentuk subkelompok baru. *Global best* baru kemudian diambil dari sisa partikel di kelompok utama. Proses ini terus berulang hingga semua partikel bergerak mencari *niche*. Dengan ini, *niche* dicari secara sekuensial dan ditangani secara paralel.

Algoritma *Vector-Based Niching PSO* menunjukan kinerja yang cukup baik [15], namun memiliki masalah jika lebih dari satu subkelompok menuju *niche* yang sama. Seperti pada algoritma *NichePSO*, subkelompok-subkelompok yang ada mungkin menuju *niche* yang sama. Hal ini tidak efisien karena melakukan pencarian secara berulang. Untuk menangani masalah ini, maka solusi yang sama pada *NichePSO* diterapkan. Subkelompok-subkelompok yang menuju niche yang sama akan digabung menjadi subkelompok yang lebih besar [16].