BAB 2
LANDASAN TEORI

2.1 Pengenalan Tanda Tangan

Tanda tangan merupakan salah satu biometrika yang memiliki karakteristik unik berupa perilaku manusia dalam muat goresan yang memiliki ciri[11]. Tanda tangan menjadi sangat menarik untuk dikembangkan sebagai biometrika karena dapat diterima oleh masyarakat. Penggunaaan tanda tangan dianggap tidak mengganggu kenyamanan manusia dalam proses identifikasi. Kenyamanan dalam pengambilan ciri dari manusia ini diharapkan menjadi solusi dari identifikasi manusia. Metode verifikasi statis dapat mengusahakan untuk mencari satu cara yang memungkinkan verifikasi dapat dilakukan dari keterbatasan-keterbatasan tersebut. Hal ini karena kenyataan yang ada pada masa sekarang, dalam kehidupan sehari-hari masih jauh lebih banyak dilakukan penggoresan tanda tangan pada media kertas, sehingga proses verifikasi statis lebih cocok digunakan dibanding proses verifikasi dinamis, seperti halnya pada dokumen-dokumen resmi pada dunia perbankan, surat-surat berharga dan lain sebagainya [12].

2.2 Citra Digital

Secara umum, istilah pengolahan citra digital menyatakan “pemrosesan gambar berdimensi-dua melalui komputer digital. Foto adalah contoh gambar berdimensi dua yang dapat diolah dengan mudah. Setiap foto dalam bentuk citra digital (misalnya berasal dari kamera digital) dapat diolah melalui perangkat tertentu. Sebagai contoh, apabila hasil bidikan kamera terlihat agak gelap, citra dapat diolah menjadi lebih terang dimungkinkan pula untuk memisahkan foto orang dari latar belakangnya. Gambaran tersebut menunjukkan hal sederhana yang dapat dilakukan melalui pengolahan citra digital. Tentu saja banyak hal pelik lain yang dapat dilakukan melalui pengolahan citra digital [17].

2.3 Jenis Citra Digital

Ada dua jenis citra yang umum digunakan dalam pemrosesan citra. Kedua jenis citra tersebut yaitu citra berwarna dan citra HSV. Jenis citra berwarna merupakan citra digital biasa yang digunakan dalam kamera *handphone* sedangkan untuk citra HSV

(*Hue, Saturation* dan *Value*) biasanya digunakan untuk segmentasi pada warna tertentu [18].

2.3.1 Citra Berwarna

Citra berwarna, atau biasa dinamakan Citra RGB, merupakan jenis citra yang menyajikan warna dalam bentuk komponen R(merah), G(hijau), dan B(biru). Tiap komponen warna menggunakan 8 bit (nilainya berkisar antara 0 sampai dengan 255). Dengan demikian, kemungkinan warna yang dapat disajikan mencapai 255 x 255 x 255 atau 16.581.375 warna. Tabel 2.1 menunjukkan contoh warna R, G dan B [18].

Tabel 2.1 Warna dan nilai penyusunan warna

Warna	R	G	B
Merah	255	0	0
Hijau	0	255	0
Biru	0	0	255
Hitam	0	0	0
Putih	255	255	255
Kuning	0	255	255

2.3.2 Grayscale

Citra *grayscale* menangani gradasi warna hitam dan putih yang menghasilkan efek warna abu-abu. Gradasi warna pada setiap gambar dinyatakan dengan intensitas. Dalam hal ini, intensitas berkisar diantara 0 sampai dengan 255. Nilai 0 menyatakan hitam dan nilai 255 menyatakan putih. Pada *grayscale* nilai intensitas tersebut dapat diseragamkan dengan suatu fungsi. Berikut ini adalah rumus konversi citra berwana (RGB) menjadi nilai intensitas *grayscale*[9].

$$G = wR.Red + wG.Green + wB.Blue \quad \text{.....(2.1)}$$

Keterangan :

$G(x, y)$ = Matriks *grayscale* koordinat piksel.

Red = komponen nilai merah *(Red)* dari suatu titik *piksel*

Green = komponen nilai hijau *(Green)* dari suatu titik *piksel*

Blue = komponen nilai bi ru *(Blue)* dari suatu titik *piksel*

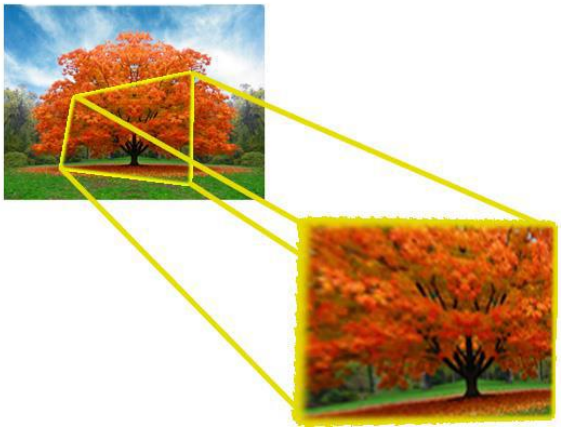
wR,wG,wB = bobot setiap nilai elemen warna.

Persamaan 2.1 di atas merupakan salah satu rumus yang digunakan untuk mengkonversi citra berwarna menjadi *grayscale*. Pada citra *grayscale*, setiap piksel adalah representasi derajat intensitas atau keabuan. Terdapat 256 jenis derajat keabuan pada citra *grayscale*. Mulai dari putih, kemudian semakin gelap sampai warna hitam. Oleh karena terdapat kemungkinan 256 derajat keabuan, maka setiap piksel pada *grayscale* disimpan 1 byte dalam memory (8 bits).

2.4 Segmentasi Citra

Dalam pengolahan citra, terkadang kita menginginkan pengolahan hanya pada obyek tertentu. Oleh sebab itu, perlu dilakukan proses segmentasi citra yang bertujuan untuk memisahkan antara objek (foreground) dengan *background*. Pada umumnya keluaran hasil segmentasi citra adalah berupa citra biner di mana objek (*foreground*) yang dikehendaki berwarna putih (1), sedangkan *background* yang ingin dihilangkan berwarna hitam (0). Sama halnya pada proses perbaikan kualitas citra, proses segmentasi citra juga bersifat eksperimental, subjektif, dan bergantung pada tujuan yang hendak dicapai.

Daerah yang kita inginkan tersebut disebut dengan *Region of Interest* (ROI). Proses untuk mendapatkan ROI salah satunya adalah dengan cara melakukan cropping pada suatu citra [15]. Berikut ini merupakan contoh dari *Region of Interest* (ROI) yang terdapat pada Gambar 2.1



Gambar 2.1 Contoh dari *Region of Interest* (ROI)

2.5 Histogram of oriented Gradients

Histogram of oriented Gradients merupakan cara yang umum dilakukan untuk memperoleh deskriptor untuk deteksi objek tertentu. Sebagai contoh, untuk mendeteksi keberadaan manusia (*Human detection*) seperti penelitian yang pernah dilakukan oleh Dalal dan Trigs [19]. Proses dari algoritma ini dapat dijelaskan sebagai berikut :

1. *Preprocessing*, melakukan intensitas normalisasi atau dengan kata lain mengubah citra gambar menjadi *grayscale* (Citra berskala keabuan).
2. Menghitung *edge map*. Mengestimasi arah x dan y pada gambar lalu menghitung gradien *magnitudes* dan gradien *angle* untuk setiap pixel gambar. Berikut ini merupakan rumus perhitungan untuk gradien *magnitudes* dan gradien *angle* :

   $|G| = \sqrt{I_x^2 + I_y^2}$ .....(2.14)

   Dimana |G| adalah bilangan absolut dari gradien *magnitudes* dan *I* adalah citra gambar yang sudah *grayscale* dari hasil tahapan sebelumnya. Ix merupakan matrik terhadap sumbu-x dan Iy merupakan matrik terhadap sumbu-y. Ix dan Iy dapat dihitung dengan rumus perhitungan sebagai berikut :

   $Ix = I * Dx \; dan \; Iy = I * Dy$ .....(2.15)

D_x adalah *mask* $[-1 \ 0 \ 1]$, sedangkan D_y adalah *mask* $\begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$ masing – masing dihitung dengan cara konvolusi (tanda *). Kemudian gradien *angle* ke dalam koordinat sumbu dengan sudut diantara 0 sampai 180 dapat dihitung dengan rumus perhitungan sebagai berikut :

θ=arctan (Ix/Iy) .....(2.16)

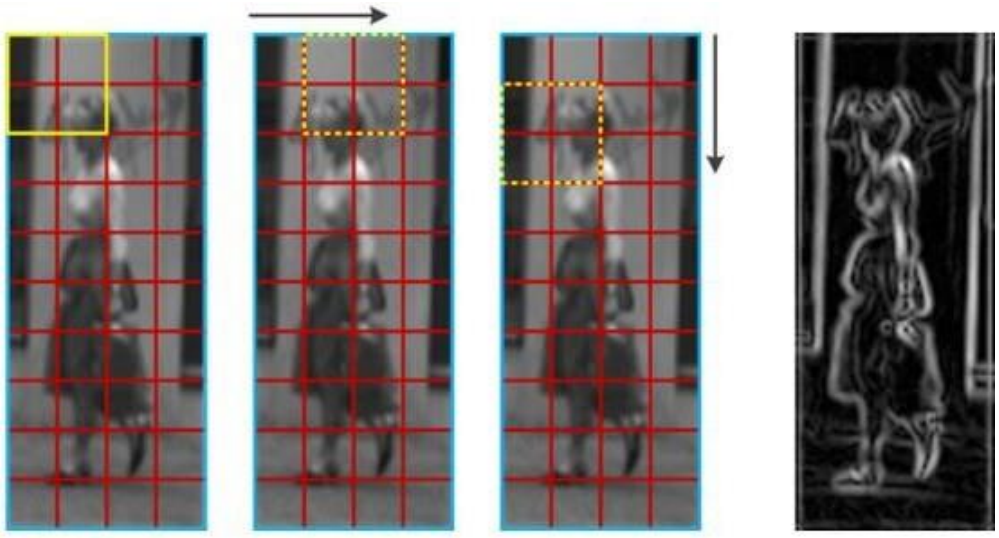
3. *Spatial binning*. Tahapan selanjutnya adalah melakukan perhitungan *histogram* dari gradien *angle* ke tiap-tiap *cell*. Setiap pixel dalam sebuah *cell* mempunyai nilai *histogram* nya sendiri – sendiri berdasarkan nilai yang dihasilkan dalam perhitungan gradien yang kemudian dilakukan normalisasi pada setiap blok. *Cell* memiliki ukuran 8x8 pixel pada sebuah citra. Sedangkan blok memiliki ukuran 2x2 *cell*.
4. *Normalize voting values for generating a descriptor*. Nilai normalisasi fitur blok selanjutnya didapat dengan rumus perhitungan sebagai berikut [19] :

$$norm = \frac{v(n)}{\sqrt{(\sum_{k=1}^{block*l} v(k)^2)+1}} \text{.....(2.17)}$$

Nilai *v* merupakan nilai gradien *magnitudes* sedangkan *n* adalah jumlah *bins* dan *block* (2x2 *cell*) merupakan jumlah *cell* sedangkan *l* merupakan jumlah blok yang tidak *overlap.* Fitur blok dinormalisasi untuk mengurangi efek perubahan kecerahan obyek pada satu blok.

5. *Augment all block vectors consecutively*. Setelah fitur blok dinormalisasi, nilai normalisasi setiap blok akan digabungkan menjadi satu vektor (vektor 1 dimensi) hasil satu vektor inilah bisa disebut sebagai fitur vektor *Histogram of oriented Gradients*.

Berikut ini merupakan ilustrasi dari cara kerja *Histogram of oriented Gradients*terdapat pada Gambar 2.2



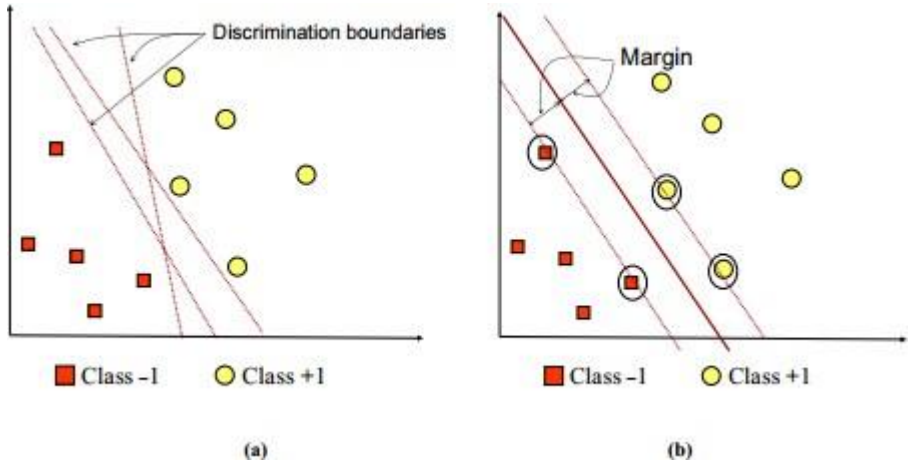
Gambar 2.2 Cara kerja *Histogram of oriented Gradients*

2.6 Support Vector Machine (SVM)

Support Vector Machine (SVM) diperkenalkan oleh Vapnik pada tahun 1992 sebagai suatu teknik klasifikasi yang efisien untuk masalah nonlinier. SVM berbeda dengan teknik klasifikasi di era 1980-an, seperti *decision tree* dan ANN, yang secara konsep kurang begitu jelas dan seringkali terjebak pada optimum lokal. SVM memiliki konsep yang jauh lebih matang, lebih jelas secara matematis, dibanding teknik – teknik klasifikasi sebelum era 1990-an. SVM berusaha menemukan *hyperplane* dengan memaksimalkan jarak antar kelas. Dengan cara ini, SVM dapat menjamin kemampuan generalisasi yang tinggi untuk data – data yang akan datang [20].

Menurut penelitian yang dilakukan oleh Anto Satriyo Nugroho, Arief Budi Witarto, dan Dwi Handoko [14] konsep SVM dapat dijelaskan secara sederhana sebagai usaha mencari *hyperplane* terbaik yang berfungsi sebagai pemisah dua buah *class* pada *input space*. Gambar 2.3-a memperlihatkan beberapa *pattern* yang

merupakan anggota dari dua buah class : +1 dan –1. *Pattern* yang tergabung pada *class* –1 disimbolkan dengan warna merah (kotak), sedangkan *pattern* pada *class* +1, disimbolkan dengan warna kuning(lingkaran).



Gambar 2.3 SVM mencari Hyperplane terbaik

Masalah klasifikasi dapat diterjemahkan dengan usaha menemukan garis (*hyperplane*) yang memisahkan antara kedua kelompok tersebut. Berbagai alternatif garis pemisah (*discrimination boundaries*) ditunjukkan pada Gambar 2.3-a. *Hyperplane* pemisah terbaik antara kedua *class* dapat ditemukan dengan mengukur *margin hyperplane* tsb. dan mencari titik maksimalnya. *Margin* adalah jarak antara *hyperplane* tersebut dengan pattern terdekat dari masing-masing *class*. *Pattern* yang paling dekat ini disebut sebagai *support vector*.

Garis solid pada Gambar 2.3-b menunjukkan *hyperplane* yang terbaik, yaitu yang terletak tepat pada tengah-tengah kedua *class*, sedangkan titik merah dan kuning yang berada dalam lingkaran hitam adalah *support vector*. Usaha untuk mencari lokasi *hyperplane* ini merupakan inti dari proses pembelajaran pada SVM. Data yang tersedia dinotasikan sebagai $\vec{x_i} \in \Re^d$ sedangkan label masing-masing dinotasikan yi $\in$ {-1,+1}untuk i = 1,2,...n ,yang mana n adalah banyaknya data.

Diasumsikan kedua *class* –1 dan +1 dapat terpisah secara sempurna oleh *hyperplane* berdimensi *d* , yang didefinisikan sebagai berikut :

$\vec{w} \cdot x + b = 0$ .....(2.18)

Pattern $\vec{x_i}$ yang termasuk *class* -1 (sampel negatif) dapat dirumuskan sebagai :

$\vec{w} \cdot x + b \leq -1$ .....(2.19)

Sedangkan *pattern* $\vec{x_i}$ yang termasuk *class* +1 (sampel positif) adalah sebagai berikut :

$\vec{w} \cdot x + b \geq +1$ .....(2.20)

Margin terbesar dapat ditemukan dengan memaksimalkan nilai jarak antara *hyperplane* dan titik terdekatnya, yaitu $1 / \|\vec{w}\|$. Hal ini dapat dirumuskan sebagai *Quadratic Programming (QP) problem*, yaitu mencari titik minimal persamaan (2.21), dengan memperhatikan *constraint* persamaan (2.22)

$$\min_{\vec{w}} \tau(\underline{w}) = \frac{1}{2}\|\vec{w}\|^2 \quad \text{.....(2.21)}$$

$$y_i(\vec{x}_i \cdot \vec{w} + b) - 1 \geq 0, \forall i \quad \text{.....(2.22)}$$

Problem ini dapat dipecahkan dengan berbagai teknik komputasi, di antaranya *Lagrange Multiplier*

$L(\vec{w},b,\alpha) = 1/2\|\vec{w}\|^2 - \Sigma\alpha_i(y_i((x_i \cdot \vec{w} + b) - 1)), i = 1,2 \ldots l_{li=1}$ .....(2.23)

α_i adalah *Langrange multipliers*, yang bernilai nol atau positif ($\alpha_i \geq 0$). Nilai optimal dari persamaan (2.22) dapat dihitung dengan meminimalkan *L* terhadap $\vec{w}$ dan *b*, dan memaksimalkan *L* terhadap α_i. Dengan memperhatikan sifat bahwa pada titik optimal *gradient L* = 0, persamaan (2.22) dapat dimodifikasi sebagai maksimalisasi *problem* yang hanya mengandung α_i, sebagaimana persamaan (2.24) dibawah

Maximize :

$\Sigma\alpha_i - 1/2\,\Sigma\alpha_i\alpha_j y_i y_j x_i.x.j$ .....(2.24)

Dimana $\alpha_i \geq 0$ ($=1,2,....l$) dan $\Sigma \alpha_i y_i = 0 l_{i=1}$. Dari hasil perhitungan ini diperoleh α_i yang kebanyakan bernilai positif. Data yang berkorelasi dengan α_i yang positif inilah yang disebut *support vector*.

Untuk mendapatkan nilai α_i, langkah pertama adalah mengubah setiap *feature* menjadi nilai vektor (*support vector*) = $_{xy}$. Kemudian vektor akan dimasukkan kedalam persamaan *kernel trick phi* φ yaitu sebagai berikut :

$$\varphi \begin{bmatrix} x \\ y \end{bmatrix} = \begin{cases} \sqrt{Xn^2 + Yn^2} > 2, \begin{bmatrix} 4 - x + & |x - y| \\ 4 - x + & |x - y| \end{bmatrix} \\ \sqrt{Xn^2 + Yn^2} \leq 2, maka \begin{bmatrix} x \\ y \end{bmatrix} \end{cases} \quad \text{.....(2.25)}$$

Kemudian untuk mencari nilai α_i didapatkan dari persamaan sebagai berikut :

$$\sum_{i=1,=1}^{n} a_i T^T T_j \ldots\ldots (2.26)$$

Dan selanjutnya menggunakan persamaan sebagai berikut :

$$W = \sum_{i=1}^{n} a_i T_i \quad \text{dan} \quad \sum_{i=1,j=1}^{n} a_i T_i = y_i \ldots\ldots (2.27)$$

Terakhir akan dicari nilai w dan b untuk menemukan *hyperplane* sebagai patokan proses klasifikasi dengan persamaan sebagai berikut :

$y = wx + b$,$w = \Sigma a_i s_i$ ……(2.28)

Nilai s_i merupakan nilai *support vector* yang telah dihitung sebelumnya. Dengan demikian proses klasifikasi selesai dengan memperhatikan *hyperplane* nya.

2.6.1 *Kernel trick*

Feature space dalam prakteknya biasanya memiliki dimensi yang lebih tinggi dari vektor input (*input space*). Hal ini mengakibatkan komputasi pada *feature space* mungkin sangat besar, karena ada kemungkinan *feature space* dapat memiliki jumlah *feature* yang tidak terhingga. Selain itu, sulit mengetahui fungsi

transformasi yang tepat. Untuk mengatasi masalah ini, pada SVM digunakan ”*kernel trick*”. Fungsi kernel yang umum digunakan adalah sebagai berikut [14]:

1. Kernel linier

   $K(x_i, x) = x^T x$ .....(2.29)
   i

2. Polynomial kernel

   $K(x_i, x) = (x_i\ x + c)^d$.....(2.30)

3. Radial basis function (RBF)

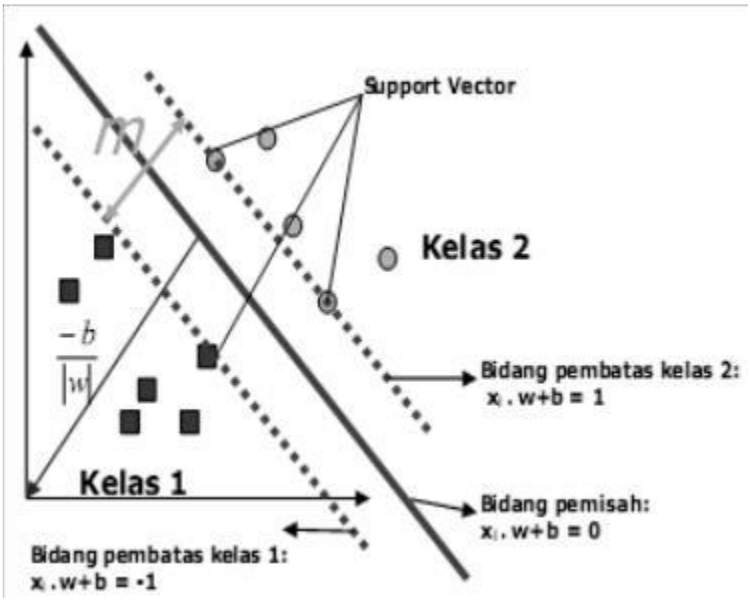
   $(x_i, x) = \exp(-\gamma\ |x_i - x|^2)$, $\gamma > 0$ .....(2.31)

4. Sigmoid kernel

   $(x_i, x) = \tanh(\gamma x^T x + r)$ .....(2.32)

2.7 *Smooth Support Vector Machine* (SSVM)

Support Vector Machines (SVM) adalah suatu teknik untuk melakukan prediksi, baik dalam kasus klasifikasi maupun regresi. Pada ruang berdimensi tinggi, akan dicari hyperplane (fungsi pemisah) yang dapat memaksimalkan jarak (margin) antara dua kelas data. Klasifikasi menggunakan SVM dapat dijelaskan secara sederhana yaitu usaha untuk mendapatkan garis sebagai fungsi pemisah terbaik yang dapat memisahkan dua kelas yang berbeda (+1,-1) pada ruang input



Gambar 2.4 Konsep Fungsi Pemisah pada SVM

Pada gambar 2.4 memperlihatkan bahwa beberapa data yang merupakan anggota dari kelas -1 dan +1. Data yang disimbolkan dengan kotak adalah anggota data -1 sedangkan yang bulat menyimbolkan anggota data +1. Fungsi pemisah terbaik adalah fungsi yang mampu memisahkan data dengan nilai margin (m) yang terbesar, dan tepat berada di antara kedua kelas data. Margin merupakan jarak antara fungsi pemisah dengan data terdekat dari masing-masing kelas. Pada gambar 2.4, fungsi pemisah terbaik ditunjukkan dengan garis tebal yang memisahkan kedua kelas. Data yang berada pada bidang pembatas dan terdekat dengan fungsi pemisah terbaiklah yang disebut dengan support vector. Hanya data support vector yang digunakan selama proses mendapatkan fungsi pemisah terbaik. SVM merupakan metode berbasis *machine learning* yang berpotensi untuk dikembangkan lebih jauh karena memiliki performansi tinggi dan dapat diaplikasikan secara luas untuk klasifikasi dan estimasi.

SSVM adalah pengembangan SVM dengan menggunakan teknik smoothing. Metode ini pertama kali diperkenalkan oleh Lee pada tahun 2001. SVM memanfaatkan optimasi dengan quadratic programming, sehingga untuk data berdimensi tinggi dan data jumlah besar SVM menjadi kurang efisien. Oleh karena itu dikembangkan smoothing technique yang menggantikan plus function SVM

dengan integral dari fungsi sigmoid neural network yang selanjutnya dikenal dengan *Smooth Support Vector Machine* (SSVM).

Diberikan masalah klasifikasi dari *n* objek dalam ruang dimensi R^p sehingga susunan data berupa matriks *A* berukuran *n x p* dan keanggotaan tiap titik terhadap kelas {+1} atau {-1} yang didefinisikan pada diagonal matriks D berukuran ,problem optimasinya adalah :

$min_{w,b,\xi} \ \frac{c}{2}\xi'\xi + \frac{1}{2}\ (w'w + b^2)$ .....(2.33)

dengan kendala

$D\ (Aw + eb) + \ \xi \ \geq e, \xi \ \geq 0.$ .....(2.34)

Solusi problem adalah

$\xi \ = \big(e - D(Aw + eb)\big)$ .....(2.35)

Dimana ξ adalah variabel *slack* yang mengukur kesalahan klasifikasi.
Kemudian dilakukan substitusi dan konversi, sehingga persamaan dapat ditulis sebagai berikut :

$min_{w,b} \ \frac{c}{2}||(e - D(Aw - eb)||_2^2 + \frac{1}{2}\ (w'w + b^2)$.....(2.36)

Fungsi objektif dalam persamaan tidak memiliki turunan kedua. Teknik *smoothing* yang diusulkan dilakukan dengan mengganti fungsi plus dengan p(x,a) yaitu integral dari fungsi sigmoid *neural network* atau dapat dituliskan sebagai berikut :

$p\ (x, a) = x + \frac{1}{a}\log(1 + \ \varepsilon^{-ax}), a > 0$ .....(2.37)

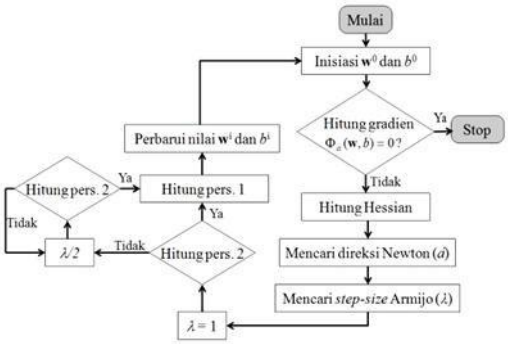
dimana α adalah parameter *smoothing*. Dengan menggantikan fungsi plus dengan p (x,a) maka diperoleh model SSVM sebagai berikut :

$min_{(w,b)\in Rp+1}\phi_a(w, b) = min_{(w,b)\in Rp+1} \ \frac{c}{2}||p(e - D(Aw - eb)||_2^2 + \frac{1}{2}\ (w'w + b^2)$ .....(2.38)

Secara umum, problem optimasi SSVM dapat ditulis sebagai berikut :

$$min_{(w,b)\in Rp+1}\phi a(w,b) = min_{(w,b)\in Rp+1}\ \frac{c}{2}||p(e - D\big(K\big(x_i, x_j\big)Dw - eb\big), a)||_2^2 + \frac{1}{2}\ (w'w + b^2)$$ .....(2.39)

Yang diselesaikan dengan iterasi Newton Armijo (Gambar 2.5) dan *K(Xi,Xj)* merupakan fungsi kernel yang dalam penelitian ini digunakan kernel Gaussian atau bisa dirumuskan berikut *K(Xi,Xj) = exp (-y(*||Xi – Xj||^2) dengan parameter kernel *y*.



Gambar 2.5 Diagram Alir Algoritma Newton-Armijo

Persamaan 1 :

$$\phi_a(w_i, b_i) - \phi_a((w_i, b_i) + (\lambda_i d_i) \geq\ - \delta\lambda_i \nabla\phi_a(w_i, b_i)d_i$$.....(2.40)

Persamaan 2 :

$$w_{i+1,}b_{i+1} = (w_i, b_i) +\ (\lambda_i d_i)$$ .....(2.41)

Saat iterasi pada algoritma Newton-Armijo berhenti, diperoleh nilai *w* dan *b* yang konvergen. Dengan demikian fungsi pemisah yang diperoleh untuk kasus klasifikasi linier adalah

F(x) = sign (w'x + b) .....(2.42)

Sedangkan fungsi permisah untuk kasus klasifikasi nonlinier adalah sebagai berikut

F(x) = *sign* (w'x + b) = *sign* (u'*D'K*(Xi,Xj) + b) .....(2.43)

Perumusan program linier SVM 1-norm adalah salah satu cara untuk memilih atribut (*feature selection*) di antara varian-varian norm SVM, problem linier tersebut adalah sebagai berikut

$$\min_{(w,b,s,\xi)\in R^{(2p)+1+n}} Ce'\xi + e's \quad \text{.....(2.44)}$$

Dengan kendala

$D(Aw + eb) + \xi \geq e$ .....(2.45)

$-s \leq w \leq s$

$\xi \geq 0$ .....(2.46)

Solusi dari ***w*** mampu menghasilkan model yang parsimoni dan bersifat *sparsity*. Jika nilai dari elemen vektor ***wp*** = 0, maka variabel *p* tidak berkontribusi dalam penentuan kelas. Kontribusi atribut atau variabel prediktor dapat dinilai dari besarnya nilai ***wi*** untuk masing-masing atribut, dengan *l*=1,2, ..., *p* [5].

2.8 Metode one-against-all

Dengan menggunakan metode ini, akan dibangun k buah model SVM biner (k adalah jumlah kelas). Contohnya, terdapat permasalahan klasifikasi dengan 4 buah kelas. Untuk pelatihan digunakan 4 buah SVM biner seperti pada Tabel 2.2 dan penggunanya dalam mengklasifikasi kelas pada data baru dapat dilihat pada persamaan sebagai berikut [22] :

$$Kelas\ x = \arg\max_{i=1.k}\left(\left(w^{(i)}\right).\varphi(x) + b^{(i)}\right) \text{.....(2.47)}$$

Dengan menentukan *hyperplane* terbesar pada nilai x maka akan mengklasifikasikan kelas tersebut.

Tabel 2.2 Contoh 4 SVM biner dengan metode One-against-all

$y_i = 1$	$y_i = -1$	Hipotesis
Kelas 1	Bukan kelas 1	$f^1(x) = (w^1)x + b^1$
Kelas 2	Bukan kelas 2	$f^2(x) = (w^2)x + b^2$

Kelas 3	Bukan kelas 3	$f^3(x) = (w^3)x + b^3$
Kelas 4	Bukan kelas 4	$f^4(x) = (w^4)x + b^4$

2.9 NetBeans IDE

Bagi pengembang aplikasi berbasis Java maupun PHP mungkin sudah familiar dengan menggunakan NetBeans IDE. Biasanya NetBeans IDE ini digunakan oleh pengembang aplikasi untuk melakukan pemrograman, kompilasi, mencari kesalahan, dan menjalankan aplikasi yang telah dibuat. NetBeans IDE sendiri dibuat dengan menggunakan bahasa pemrograman Java, namun untuk membuat aplikasi yang dapat digunakan dalam sebuah perangkat komputer, maupun mobile, IDE ini pun mampu mendukung bahasa lain. Beberapa bahasa yang didukung oleh NetBeans IDE ini terdiri dari Java, C/C++, PHP, XML, HTML, Javadoc, Javascript, JSP, dan masih banyak lagi. Selain itu, pembaca dapat memasang *plugin*, maupun modul yang bisa didapatkan di komunitas untuk mendukung bahasa lain agar dapat dijalankan di NetBeans IDE ini.

Tampilan antarmuka dari NetBeans IDE ini bisa dibilang cukup memudahkan pengembang aplikasi dalam melakukan interaksi secara grafikal yang tidak hanya berupa kode saja sehingga dapat dilihat hasil kodenya secara langsung dalam bentuk grafis. Selain itu, library yang disediakan dalam NetBeans ini bisa dibilang cukup lengkap untuk beberapa bahasa pemrograman sehingga pengembang aplikasi pemula pun dapat mempelajari langsung *library* yang dibutuhkan dalam membuat aplikasi.

NetBeans IDE terbaru telah memasuki versi 8.0, dan telah mendukung Java 8.0 Selain itu, IDE ini bisa berjalan di sistem operasi Windows, Mac OS, dan Linux 32/64 bit. Untuk mengunduh NetBeans IDE ini pengguna dapat langsung mengunjungi halaman resmi unduh NetBeans yang nantinya akan diberikan pilihan paket unduhan NetBeans IDE yang terdiri dari paket JAVA SE, JAVA EE, C/C++, HTML5 & PHP, atau paket keseluruhan [16].

2.10 OpenCV (Open Source Computer Vision Library)

OpenCV (*Open Source Computer Vision Library*) adalah sebuah *library* perangkat lunak yang ditujukan untuk pengolahan citra yang biasa digunakan secara *real-time* (pada waktu itu juga). *Library* ini dibuat oleh Intel dan merupakan *library* yang bebas digunakan dan berada dalam naungan sumber terbuka (*Open source*) dari lisensi. *Library* ini juga bisa digunakan diberbagai *platform* dan didedikasikan sebagian besar untuk pengolahan citra secara *real-time*. Umumnya *library* ini menggunakan bahasa pemrograman C/C++, tetapi akhir – akhir ini sudah dikembangkan keberbagai bahasa pemrograman seperti Phyton, Javascript dan Java.

Secara garis besar OpenCV mempunyai modul/subrutin yang ada pada *library* yang akan dijelaskan sebagai berikut [13]:

1. *Core* – sebuah modul/subrutin dasar dari struktur data, sudah termasuk array multi dimensi dan matriks.

2. *Imgproc* – sebuah modul/subrutin untuk pemprosesan citra seperti *image filtering*, *geometrical image*, *image transformation* dan *color space conversion*.

3. *Video* – sebuah modul/subrutin untuk analisis video termasuk *motion*, *background subtraction* dan *object tracking algorithm*.

4. *Calib3d* – sebuah modul/subrutin untuk *geometry algorithm*, kalibrasi kamera dan elemen untuk membangun gambar 3D (3-*Dimensional*).

5. *Features2d* – sebuah modul/subrutin untuk perhitungan konvolusi dan ekstraksi fitur.

6. *Objdetect* – sebuah modul/subrutin untuk deteksi objek dari kelas yang sudah ditentukan.

7. *Highgui* – sebuah modul/subrutin untuk menangkap kamera *webcam* dan *image and video codecs.*

8. *Ml* – sebuah modul/subrutin tambahan untuk melakukan perhitungan *Machine learning* seperti *K-nearest Neighbors*, *Support Vector Machine* dan *Decision tree*.