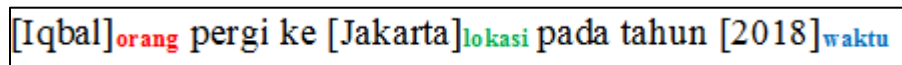


BAB 2

LANDASAN TEORI

2.1 *Named Entity Recognition*

Pengenalan entitas bernama adalah komponen dari ekstraksi informasi untuk klasifikasi teks kedalam kategori data yang sudah ditentukan seperti mengenali entitas nama orang, lokasi, organisasi, dan waktu [1].



[Iqbal]orang pergi ke [Jakarta]lokasi pada tahun [2018]waktu

Gambar 2.1 Contoh NER

2.2 *Preprocessing*

Preprocessing merupakan sebuah tahapan awal yang dilakukan sebelum data masukan di proses dengan algoritma SVM. Dalam penelitian ini *preprocessing* dilakukan dalam beberapa tahap yaitu tokenisasi, *case folding*, ekstraksi fitur, kamus kata dan *one hot encoding*.

2.2.1 **Tokenisasi**

Tokenisasi merupakan proses memotong kalimat berdasarkan tiap kata yang menyusunnya [10]. Pemecahan kalimat menjadi kata-kata tunggal dilakukan berdasarkan kata atau huruf sudah ditentukan sebelumnya.

2.2.2 *Case Folding*

Case Folding merupakan proses mengubah semua kata pada kalimat menjadi suatu bentuk standar (biasanya huruf kecil) [10].

2.2.3 **Ekstraksi Fitur**

Ekstraksi adalah sebuah proses pengambilan ciri pada suatu objek yang dapat menggambarkan objek tersebut [11]. Pada penelitian ini menggunakan ekstraksi fitur ortografi pada penelitian Chang [12]. Fitur ortografi yang digunakan dalam penelitian ini dapat dilihat pada Tabel 2.1.

Tabel 2.1 Fitur Ortografi

Fitur Ortografi	Penjelasan
ALLCAPS	Token yang seluruh hurufnya kapital .
INITCAPS	Token yang berawalan huruf kapital.
LOWERCASE	Token yang terdapat huruf kecil.
ALLDIGIT	Token yang semuanya digit.

2.2.4 Kamus Kata

Kamus kata digunakan untuk menyimpan list kata dari data latih. Kamus kata terdiri dari list kata yang berbeda satu sama lain yang digunakan untuk proses *one hot encoding* nantinya.

2.2.5 One Hot Encoding

One Hot Encoding merupakan salah satu teknik dalam merepresentasikan data dalam bentuk kategori seperti kata untuk diubah kedalam angka. Cara kerja *one hot encoding* adalah dengan memberikan nilai 1 pada kata yang sama dengan kata yang terdapat pada kamus kata [6]. Panjang *one hot encoding* tergantung dari panjang kamus yang dibentuknya.

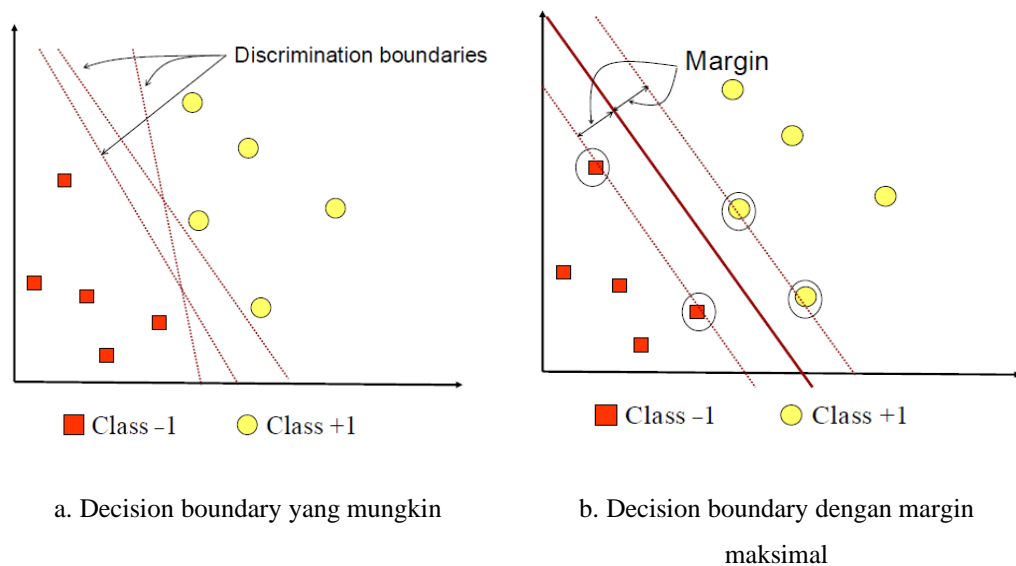
2.3 SVM

Metode klasifikasi SVM adalah salah satu metode diskriminatif yang paling tepat digunakan dalam klasifikasi [13]. Metode ini berakar dari teori pembelajaran statistik yang hasilnya sangat menjanjikan untuk memberikan hasil yang lebih baik daripada metode yang lain. SVM juga bekerja dengan baik pada set data berdimensi tinggi, bahkan SVM yang menggunakan teknik kernel harus memetakan data asli dari dimensi asalnya menjadi dimensi lain yang relatif lebih tinggi. Jika pada ANN semua data latih akan dipelajari selama proses pelatihan, pada SVM tidak semua data latih akan dipandang untuk dilibatkan dalam setiap iterasi pelatihannya. Data-data yang berkontribusi disebut *support vector* sehingga metodenya disebut *Support Vector Machine* [14].

2.3.1 Konsep SVM

Ide dasar SVM adalah memaksimalkan batas hyperplane (*maximal margin hyperplane*), seperti yang diilustrasikan pada Gambar 2.2. Pada Gambar 2.2 (a) ada

sejumlah pilihan hyperplane yang mungkin untuk set data, dan Gambar 2.2 (b) merupakan hyperplane dengan margin yang paling maksimal. Meskipun Gambar 2.2 (a) sebenarnya juga bisa menggunakan hyperplane sembarang, hyperplane dengan margin yang maksimal akan memberikan generalisasi yang lebih baik pada metode klasifikasi [14].



Gambar 2.2 Decision boundary yang mungkin untuk set data

Sumber : Support Vector Machine – Teori dan Aplikasinya dalam Bioinformatika (2003) [15]

Konsep klasifikasi dengan SVM dapat dijelaskan secara sederhana sebagai usaha untuk mencari hyperplane terbaik yang berfungsi sebagai pemisah dua buah kelas data pada urang input. Gambar 2.1 memperlihatkan beberapa pola yang merupakan anggota dari dua buah kelas data : +1 dan -1. Data yang tergabung pada kelas -1 disimbolkan dengan bentuk bujur sangkar, sedangkan data pada kelas +1 disimbolkan dengan bentuk lingkaran.

Hyperplane (batas keputusan) pemisah terbaik antara kedua kelas dapat ditemukan dengan mengukur margin hyperplane tersebut dan mencari titik maksimalnya.

Margin adalah jarak antara hyperplane tersebut dengan data terdekat dari masing-masing kelas. Data yang paling dekat ini disebut *support vector*. Garis solid pada Gambar 2.1 (b) sebelah kanan menunjukkan hyperplane yang terbaik, yaitu terletak tepat pada tengah-tengah kedua kelas, sedangkan data lingkaran dan bujur sangkar yang dilewati garis batas margin (garis putus-putus) adalah *support vector*. Usaha untuk mencari lokasi hyperplane ini merupakan inti dari proses pelatihan pada SVM.

2.3.2 SVM Linier

Setiap data latih dinyatakan oleh (x_i, y_i) , di mana $i = 1, 2, \dots, N$, dan $x_i = \{x_{i1}, x_{i2}, \dots, x_{iq}\}^T$ merupakan atribut (fitur) set untuk data latih ke- i . $y_i \in \{-1, +1\}$ menyatakan label kelas. Hyperplane klasifikasi linier SVM dinotasikan dengan Persamaan 2.1 di bawah ini.

$$w * x_i + b = 0 \quad (2.1)$$

w dan b adalah parameter model. $w * x_i$ merupakan inner-product dalam antara w dan x_i . Data x_i yang masuk ke kelas -1 adalah data yang memenuhi Pertidaksamaan 2.2.

$$w * x_i + b \leq -1 \quad (2.2)$$

Sementara x_i yang masuk ke dalam kelas +1 adalah data yang memenuhi Pertidaksamaan 2.3.

$$w * x_i + b \geq +1 \quad (2.3)$$

Sesuai dengan Gambar 2.2, jika ada data yang dalam kelas -1 (misalnya, x_a) yang bertempat di hyperplane, Persamaan 2.1 akan terpenuhi. Untuk data kelas -1 dinotasikan dengan Persamaan 2.4.

$$w * x_a + b = 0 \quad (2.4)$$

Sementara kelas +1 (misal x_b) akan memenuhi Persamaan 2.5.

$$w * x_b + b = 0 \quad (2.5)$$

Dengan mengurangi Persamaan 2.5 dengan 2.4, didapatkan Persamaan 2.6.

$$w * (x_b - x_a) = 0 \quad (2.6)$$

$x_b - x_a$ adalah vektor paralel di posisi hyperplane dan diarahkan dari x_a ke x_b . Karena inner-product dalam bernilai nol, arah w harus tegak lurus terhadap hyperplane. Dengan memberikan label -1 untuk kelas pertama dan +1 untuk kelas kedua, prediksi semua data uji akan menggunakan Persamaan 2.7.

$$y = \begin{cases} -1, & \text{jika } w.z + b > 0 \\ +1, & \text{jika } w.z + b < 0 \end{cases} \quad (2.7)$$

Sesuai dengan Gambar 2.3 hyperplane kelas -1 (garis putus-putus) adalah data pada support vector yang memenuhi Persamaan 2.8.

$$w * x_a + b = -1 \quad (2.8)$$

Sementara hyperplane kelas +1 (garis putus-putus) memenuhi Persamaan 2.9.

$$w * x_a + b = +1 \quad (2.9)$$

Dengan demikian, margin dapat dihitung dengan mengurangi Persamaan 2.9 dengan 2.8 didapatkan Persamaan 2.10.

$$w * (x_b + x_a) = 2 \quad (2.10)$$

Margin hyperplane diberikan oleh jarak antara dua hyperplane dari dua kelas tersebut. Notasi diatas diringkas menjadi Persamaan 2.11.

$$\|w\| * d = 2 \text{ atau } d = \frac{2}{\|w\|} \quad (2.11)$$

2.3.3 Hyperplane SVM

Klasifikasi kelas data pada svm pada Persamaan 2.2 dan 2.3 dapat digabungkan menjadi Pertidaksamaan 2.12.

$$y_i(w * x_i + b) \geq 1, i = 1,2,\dots,N \quad (2.12)$$

Margin optimal dihitung dengan memaksimalkan jarak antara hyperplane dan data terdekat. Jarak ini dirumuskan dengan Persamaan 2.11 ($\|w\|$ adalah vektor bobot w). Selanjutnya masalah ini diformulasikan ke dalam *quadratic programming* (QP) problem, dengan meminimalkan invers Persamaan 2.11, $\frac{1}{2} \|w\|^2$, dibawah kendala (syarat), seperti berikut.

Minimalkan :

$$\frac{1}{2} \|w\|^2 \quad (2.13)$$

Syarat

$$y_i(w * x_i + b) \geq 1, i = 1, 2, \dots, N \quad (2.14)$$

Optimisasi ini dapat diselesaikan dengan Lagrange Multiplier.

$$Lp = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i y_i (w * x_i + b) - 1 \quad (2.15)$$

α_i adalah Lagrange Multiplier yang berkorespondensi dengan x_i . Nilai α_i adalah nol atau positif. Untuk meminimalkan Lagrangian, Persamaan 2.15 harus diturunkan pada w dan b , dan diset dengan nilai nol untuk syarat optimalisasi di atas.

Syarat 1:

$$\frac{\partial Lp}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^N \alpha_i y_i x_i \quad (2.16)$$

Syarat 2:

$$\frac{\partial Lp}{\partial b} = 0 \Rightarrow w = \sum_{i=1}^N \alpha_i y_i = 0 \quad (2.17)$$

N adalah jumlah data yang menjadi *support vector*. Karena Lagrange Multiplier (α) tidak diketahui nilainya, persamaan di atas tidak dapat diselesaikan secara langsung untuk mendapatkan w dan b .

Untuk menyelesaikan masalah tersebut, rubah Persamaan 2.15 menjadi kasus pemaksimalan, syarat optimasi untuk dualitasnya menggunakan kendala (*constraint*) Karush-Kuhn-Turcker (KKT) berikut.

Syarat 1:

$$\alpha_i [y_i(w * x_i + b) - 1] = 0 \quad (2.18)$$

Syarat 2:

$$\alpha_i \geq 0, \quad i = 1, 2, \dots, N \quad (2.19)$$

Dengan penerapan kendala pada Persamaan 2.18 dan 2.19, dipastikan bahwa nilai Lagrange Multiplier sama banyaknya dengan data latih, tetapi sebenarnya banyak dari data latih yang Lagrange Multiplier-nya sama dengan nol (karena hanya beberapa saja yang akan menjadi *support vector*) ketika menerapkan syarat pertama. Kendala diatas bahwa Lagrange Multiplier α_i harus nol, kecuali untuk data latih x_i yang memenuhi Persamaan 2.20.

$$y_i(w * x_i + b) = 1 \quad (2.20)$$

Data latih itu, dengan $\alpha_i > 0$, terletak pada hyperplane b_{i1} atau b_{i2} , dan disebut *support vector*. Data latih yang tidak terletak di hyperplane tersebut mempunyai $\alpha_i = 0$. Persamaan 2.16 dan 2.17 juga menyarankan parameter w dan b , yang mendefinisikan hyperplane, hanya tergantung pada *support vector*.

Walaupun bisa diselesaikan, masalah optimalisasi dia atas masih sulit karena banyaknya parameter : w , b , dan α_i . Untuk menyederhanakannya, Persamaan optimalisasi 2.15 harus ditransformasi kedalam fungsi Lagrange Multiplier itu sendiri (disebut dualitas masalah).

Persamaan Langrange Multiplier 2.15 dapat dijabarkan menjadi

$$Lp = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \alpha_i y_i (w * x_i) - b \sum_{i=1}^N \alpha_i y_i + \sum_{i=1}^N \alpha_i \quad (2.21)$$

Syarat optimal Persamaan 2.17 ada dalam suku ketiga di ruas kanan dalam Persamaan 2.21, dan memaksa suku ini menjadi sama dengan 0. Dengan mengganti w dari syarat pada Persamaan 2.16, dan suku $\|w\|^2 = w_i * w_j$, persamaan di atas akan berubah menjadi dualitas Lagrange Multiplier yang berupa Ld, didapatkan.

Maksimalkan :

$$Lp = \sum_{i=1}^N \alpha_i - \frac{1}{2} \sum_{i=1}^N \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \quad (2.22)$$

Syarat 1

$$\sum_{i=1}^N \alpha_i y_i = 0 \quad (2.23)$$

Syarat 2

$$\alpha_i \geq 0, i = 1, 2, \dots, N \quad (2.24)$$

$\mathbf{x}_i \cdot \mathbf{x}_j$ merupakan *dot-product* dua data dalam data latih. Untuk set data yang besar, masalah dualitas optimalisasi tersebut Persamaan 2.22, 2.23 dan 2.24 dapat diselesaikan dengan metode numerik seperti pemrograman kuadratik. Begitu α_i didapatkan, Persamaan 2.16 dan 2.18 bisa digunakan untuk mendapatkan w dan b .

$$w = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \quad (2.25)$$

Hyperplane (batas keputusan) didapatkan dengan Persamaan 2.26.

$$\left(\sum_{i=1}^N \alpha_i y_i \mathbf{x}_i \cdot \mathbf{z} \right) + b = 0 \quad (2.26)$$

N adalah jumlah data yang menjadi *support vektor*. \mathbf{x}_i merupakan support vektor, \mathbf{z} merupakan data uji yang akan diprediksi kelasnya, dan $\mathbf{x}_i \cdot \mathbf{z}$ merupakan inner-product antara \mathbf{x}_i dan \mathbf{z} . Untuk mendapatkan nilai b didapatkan dari Persamaan 2.18 pada *support vector*. Karena α_i dihitung dengan menggunakan metode numerik dan mempunyai error numerik, nilai yang dihitung untuk b tidak sama. Hal ini disebabkan oleh *support vector* yang digunakan dalam Persamaan 2.18, biasanya nilai rata-rata dari b yang didapat untuk menjadi parameter hyperplane. Untuk mendapatkan b , Persamaan 2.18 dapat disederhanakan menjadi.

$$b_i = 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i) \quad (2.27)$$

Penjelasan di atas berdasarkan asumsi bahwa kedua kelas dapat terpisah secara sempurna oleh hyperlane. Akan tetapi pada umumnya, kedua kelas tersebut tidak dapat terpisah secara sempurna. Hal ini menyebabkan proses optimalisasi tidak dapat diselesaikan karena tidak ada w dan b yang memenuhi Pertidaksamaan 2.14. Untuk itu, pertidaksamaan tersebut dimodifikasi dengan memasukan variabel slack ξ_i ($\xi_i \geq 0$). Hasilnya adalah Pertidaksamaan 2.28.

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad (2.28)$$

Demikian juga untuk masalah Persamaan 2.13.

$$\frac{1}{2} \|\mathbf{w}\|^2 + c \sum_{i=1}^N \xi_i \quad (2.29)$$

Parameter c berguna untuk mengontrol pertukaran antara margin dan error klasifikasi. Semakin besar nilai c , semakin besar pula pelanggaran yang dikenakan untuk tiap klasifikasi.

Metode untuk mengoptimisasi hyperplane SVM umumnya dipakai untuk menyelesaikan pemrograman kuadrat dengan kendala yang diterapkan. Beberapa pilihan metode yang bisa digunakan adalah *chunking*, metode dekomposisi, dan *sequential minimal optimization* (SMO).

2.3.4 SVM Nonlinier

SVM sebenarnya adalah hyperlane linier yang hanya bekerja pada data yang dapat dipisahkan secara linier. Untuk data yang distribusi kelasnya tidak linier biasanya digunakan pendekatan kernel pada fitur data awal set data. Kernel dapat didefinisikan sebagai suatu fungsi yang memetakan fitur data dari dimensi awal ke fitur lain yang berdimensi lebih tinggi. Pendekatan ini berbeda dengan metode klasifikasi pada umumnya yang justru mengurangi dimensi awal untuk menyederhanakan proses komputasi dan memberikan akurasi prediksi yang lebih baik. Algoritma pemetaan kernel ditunjukkan pada Persamaan 2.30.

$$\Phi: D^q \rightarrow D^r, x \rightarrow \Phi(x) \quad (2.30)$$

Φ merupakan fungsi kernel yang dapat digunakan untuk pemetaan, D merupakan data latih, q merupakan set fitur dalam satu data lama, dan r merupakan set fitur yang baru sebagai hasil pemetaan untuk data latih. Sementara x merupakan

data latih, dimana $x_1, x_2, \dots, x_n \in D^q$ merupakan fitur-fitur yang akan dipetakan ke fitur berdimensi tinggi r , jadi untuk set data yang digunakan sebagai pelatihan dengan algoritma yang ada dari dimensi fitur yang lama D ke dimensi baru r . Misalnya, untuk n sampel data.

$$(\Phi(x_1), y_1, \Phi(x_2), y_2, \dots, \Phi(x_n), y_n) \in D^r \quad (2.31)$$

Pemetaan fitur lama pada set data ke fitur baru merupakan analogi dari layer tersembunyi pada ANN dimana jumlah neuron dalam layer tersembunyi biasanya lebih banyak dari pada jumlah vektor masukan.

Selanjutnya dilakukan proses pelatihan yang sama sebagaimana pada SVM linier. Proses pemetaan pada fase ini memerlukan perhitungan dot-product dua buah data pada ruang fitur baru. Dot-product kedua buah vektor (x_i) dan (x_j) dinotasikan sebagai $\Phi(x_i) \cdot \Phi(x_j)$. nilai dot-product kedua buah vektor ini dapat dihitung secara tidak langsung, yaitu tanpa mengetahui fungsi transformasi Φ . Teknik komputasi seperti ini kemudian disebut trik kernel, yaitu menghitung dot-product dua buah vektor di ruang dimensi baru dengan memakai komponen kedua buah ketor tersebut di ruang asal, seperti berikut :

$$K(x_i, x_j) = \Phi(x_i) \cdot \Phi(x_j) \quad (2.32)$$

Dan prediksi pada set data dengan dimensi fitur baru yang diformulasikan dengan Persamaan 2.33.

$$f(z) = \sum_{i=1}^N \alpha_i y_i K(x_i, z) + b \quad (2.33)$$

N adalah jumlah data yang menjadi *support vector*, x_i adalah *support vector*, dan z adalah data uji yang akan diprediksi kelasnya.

Beberapa kernel yang terdapat pada svm meliputi.

a. Kernel linier

$$K(x_i, x) = x \cdot x_i^T \quad (2.34)$$

b. Polynomial

$$K(x_i, x) = (Y x_i^T x + r)^p, Y > 0 \quad (2.35)$$

c. Radial basis function (RBF)

$$K(x_i, x) = \exp(-Y|x_i - x|^2), Y > 0 \quad (2.36)$$

d. Sigmoid kernel

$$K(x_i, x) = \tanh(Y x_i^T x + r) \quad (2.37)$$

2.3.5 SMO

SMO merupakan salah satu metode untuk menyelesaikan masalah optimasi pada SVM dengan menyelesaikan permasalahan dalam mencari nilai *lagrange multiplier* dengan kondisi KKT berikut.

$$\begin{aligned} \alpha_i = 0 &\Leftrightarrow y_i f(x) \geq 1 \\ 0 < \alpha_i < c &\Leftrightarrow y_i f(x) = 1 \\ \alpha_i = c &\Leftrightarrow y_i f(x) \leq 1 \end{aligned} \quad (2.38)$$

Pengupdate-an nilai dari *alpha* dibutuhkan nilai dari *L* dan *H* untuk batas atas dan bawah. Nilai *L* dan *H* didapatkan jika $y_i \neq y_j$.

$$L = \max(0, \alpha_j - \alpha_i) \quad (2.39)$$

$$H = \min(c, c + \alpha_j - \alpha_i) \quad (2.40)$$

Dan jika $y_i = y_j$, maka persamaan berikut digunakan.

$$L = \max(0, \alpha_j + \alpha_i - c) \quad (2.41)$$

$$H = \min(c, \alpha_j + \alpha_i) \quad (2.42)$$

Sehingga nilai dari *alpha* baru bisa didapatkan dari Persamaan 2.43.

$$\alpha_j^{baru} = \alpha_j - \frac{y_j(E_i - E_j)}{\eta} \quad (2.43)$$

Dimana nilai dari η , E_i , dan E_j didapatkan dari Persamaan 2.44 dan 2.45.

$$E_i = f(x_i) - y_i, E_j = f(x_j) - y_j \quad (2.44)$$

$$\eta = 2K(x_i, x_j) - K(x_i, x_i) - K(x_j, x_j) \quad (2.45)$$

Setelah nilai α baru ditemukan, maka langkah selanjutnya adalah mengatur nilai α agar tidak keluar dari $boundary$. Pembatasan tersebut dilakukan dengan menggunakan Persamaan 2.46. Nilai α_j^{baru} dari Persamaan 2.43.

$$\alpha_j^{baru} = \begin{cases} H & , \alpha_j^{baru} > H \\ \alpha_j^{baru} & , L \leq \alpha_j^{baru} \leq H \\ L & , \alpha_j^{baru} < L \end{cases} \quad (2.46)$$

Sehingga nilai α yang satunya lagi bisa didapatkan dengan persamaan berikut.

$$\alpha_i^{baru} = \alpha_i + y_i \times y_j (\alpha_j - \alpha_j^{baru}) \quad (2.47)$$

Setelah nilai α_i dan α_j didapat, maka selanjutnya mencari nilai $b\alpha_i$ dan $b\alpha_j$ dengan Persamaan 2.48 dan 2.49.

$$b_1 = b - E_i - y_i(\alpha_i^{baru} - \alpha_i)K(x_i, x_i) - y_j(\alpha_j^{baru} - \alpha_j)K(x_i, x_j) \quad (2.48)$$

$$b_2 = b - E_j - y_i(\alpha_i^{baru} - \alpha_i)K(x_i, x_j) - y_j(\alpha_j^{baru} - \alpha_j)K(x_j, x_j) \quad (2.49)$$

Nilai b akan dipilih jika memenuhi salah satu kondisi berikut.

$$b = \begin{cases} b_1 & , 0 < \alpha_i^{baru} < c \\ b_2 & , 0 < \alpha_j^{baru} < c \\ (b_1 + b_2)/2 & , otherwise \end{cases} \quad (2.50)$$

Untuk lebih jelasnya setiap langkah yang dikerjakan dengan metode SMO dapat dilihat pada Tabel 2.2 di bawah ini

Tabel 2.2 Algoritma SMO

Sumber : The Simplified SMO Algorithm [16]

Algoritma: Simplified SMO	
Input : $[x_1, \dots, x_m]$ (data latih), $[y_1, \dots, y_m]$ (class data latih), tol , c , max_pass	
Output : α, b	
1	$\alpha_i := 0, \forall_i$
2	$b := 0$
3	$pass := 0$
4	while ($pass < max_pass$) do
5	ganti_alpha := 0
6	for $i := 1$ to m begin
7	Hitung $E_i := f(x_i) - y_i$ (2.32)
8	if ($(y_i \times E_i < -tol$ and $\alpha_i < c$) or ($y_i \times E_i > tol$ and $\alpha_i > 0$)) then
9	pilih $j \neq i$ secara acak
10	Hitung $E_j := f(x_j) - y_j$
11	Simpan $\alpha_i^{lama} := \alpha_i, \alpha_j^{lama} := \alpha_j$
12	Hitung L dan H

```

13     if (L==H) then
14         continue
15     endif
16     Hitung  $\eta$  (2.45)
17     if ( $\eta \geq 0$ ) then
18         continue
19     end if
20     Hitung dan simpan nilai  $\alpha_j$  baru (2.43) dan (2.46)
21     if ( $|\alpha_j - \alpha_j^{lama}| < 10^{-5}$ ) then
22         continue
23     end if
24     hitung dan simpan nilai  $\alpha_i$  baru (2.47)
25     hitung  $b_1$  dan  $b_2$  (2.48) dan (2.49)
26     hitung nilai  $b$  (2.50)
27     ganti_alpha := ganti_alpha+1
28 end if
29 end for
30 if (ganti_alpha==0) then
31     pass := pass +1
32 else
33     pass := 0
34 end if
35 end while

```

Berdasarkan algoritma SMO pada Tabel 2.2 kondisi berhenti untuk mengoptimalkan nilai alpha terjadi jika kondisi pada variabel *max_pass* sudah terpenuhi. Namun jika nilai alpha yang akan dioptimasi cukup banyak maka kondisi optimal akan sulit tercapai, maka ditambahkan satu parameter untuk kondisi berhenti dengan variabel *max_iter* yang berguna untuk membatasi banyaknya iterasi yang sudah dilakukan untuk mengoptimalkan nilai alpha, jika kondisi pada *max_pass* tidak tercapai sebanyak iterasi yang di tentukan. Algoritma SMO pada Tabel 2.2 ditambahkan dengan variabel *max_iter* untuk kondisi berhenti yang kedua. Untuk lebih jelasnya dapat dilihat pada Tabel 2.3 di bawah ini.

Tabel 2.3 Algoritma SMO

Algoritma:Simplified SMO	
Input : $[x_1, \dots, x_m]$ (data latih), $[y_1, \dots, y_m]$ (class data latih), tol , c , max_pass , max_iter	
Output : α, b	
1	$\alpha_i := 0, \forall_i$
2	$b := 0$
3	$pass := 0$
4	$iter := 0$
5	while ($pass < max_pass$ and $iter < max_iter$) do

```

6  ganti_alpha = 0
7  for i := 1 to m begin
8    Hitung  $E_i := f(x_i) - y_i$  (2.32)
9    if (( $y_i \times E_i < -tol$  and  $\alpha_i < c$ ) or ( $y_i \times E_i > tol$  and  $\alpha_i > 0$ )) then
10   pilih  $j \neq i$  secara acak
11   Hitung  $E_j := f(x_j) - y_j$  (2.32)
12   Simpan  $\alpha_i^{lama} := \alpha_i, \alpha_j^{lama} := \alpha_j$ 
13   Hitung  $L$  dan  $H$ 
14   if ( $L=H$ ) then
15     continue
16   endif
17   Hitung  $\eta$  (2.45)
18   if ( $\eta \geq 0$ ) then
19     continue
20   end if
21   Hitung dan simpan nilai  $\alpha_j$  baru (2.43) dan (2.46)
22   if ( $|\alpha_j - \alpha_j^{lama}| < 10^{-5}$ ) then
23     continue
24   end if
25   hitung dan simpan nilai  $\alpha_i$  baru (2.47)
26   hitung  $b_1$  dan  $b_2$  (2.48) dan (2.49)
27   hitung nilai  $b$  (2.50)
28   ganti_alpha := ganti_alpha+1
29   end if
30 end for
31 if (ganti_alpha==0) then
32   pass := pass +1
33 else
34   pass := 0
35 end if
36 iter := iter+1
37 end while

```

Contoh implementasi metode SMO dengan menggunakan bahasa pemrograman Python dapat dilihat pada [17].

2.4 OOP

OOP (Object Oriented Programming) merupakan teknik pemrograman dengan menggunakan konsep objek. Tujuan dari OOP adalah untuk memudahkan programmer dalam pembuatan program dengan menggunakan konsep objek yang ada dalam kehidupan sehari-hari. Jadi setiap bagian permasalahan adalah objek, dan objek itu sendiri merupakan gabungan dari beberapa objek yang lebih kecil [18]. Dengan orientasi OOP pada sebuah obyek dalam pembangunan aplikasi

berikut merupakan kemampuan yang dapat dilakukan oleh pemrograman berbasis obyek ini.

1. *Class*

Class merupakan gambaran dari sebuah obyek. *Class* dapat didefinisikan dengan penampung nilai *property* dan *methode* yang dimana *methode* merupakan tingkah laku yang dapat dilakukan oleh sebuah obyek.

2. Obyek

Obyek merupakan hasil inisiasi dari sebuah *class*, mengandung *resource* yang telah didefinisikan pada *class*. Dengan kata lain obyek dapat didefinisikan sebagai hasil cetakan dari *class*.

3. *Encapsulation*

Encapsulation merupakan mekanisme untuk “membungkus” sebuah *property* maupun *method*. Dalam istilah lain seringkali disebut dengan “*Information Hiding*”. Terdapat 3 jenis “pembungkusan” yang dapat dilakukan yaitu *private*, *protected*, dan *public*. Jenis “pembungkusan” tersebut dapat dikatakan sebagai *modifier* yang digunakan untuk mendefinisikan tingkat visibilitas sebuah *property* maupun *method* di dalam sebuah *class*.

4. *Polymorphism*

Polymorphism membuat obyek-obyek yang berasal dari *subclass* yang berbeda. Hal ini terjadi ketika memilih *method* yang sesuai untuk diimplementasikan ke obyek tertentu berdasarkan pada *subclass* yang memiliki *method* bersangkutan. Kondisi yang harus dipenuhi untuk dapat mengimplementasikan *polymorphism* adalah :

- a. *Method* yang dipanggil harus melalui *variable* dari basis *class* atau *superclass*
- b. *Method* yang dipanggil harus juga menjadi *method* dari basis *class*.
- c. Parameter *method* harus sama baik pada *superclass* maupun *subclass*.
- d. *Access modifier attribute* pada *subclass* tidak boleh lebih terbatas dari basic *class*.

5. *Constructor* dan *Destructor*

OOP memungkinkan untuk menyatakan *method* ketika sebuah obyek diinisiasikan yang otomatis akan terpanggil ketika obyek dibuat yang disebut dengan *constructor*. Begitu pula dengan *destructor* yang otomatis akan panggil ketika tidak ada referensi lain.

6. *Inheritance*

Inheritance merupakan cara untuk menggunakan kembali sebuah *class* atau untuk mendirikan *subtype* dari obyek yang sudah ada. *Inheritance* memungkinkan untuk *subclass* menggunakan *method* maupun *attribute* yang sudah didefinisikan oleh *class* dasarnya.

2.5 Python

Python adalah Bahasa pemrograman bersifat umum. Python diciptakan pada tahun 1990 oleh Guido van Rossum. Bahasa level tinggi, stabil, dinamis, orientasi objek dan *cross platform* adalah karakteristik yang membuat Bahasa python disukai oleh banyak pengembang. Tidak seperti bahasa lain yang susah untuk dibaca dan dipahami, python lebih menekankan pada keterbacaan kode agar lebih mudah untuk memahami sintaks. Bahasa pemrograman Python saat ini dikembangkan dan dikelola oleh suatu tim relawan dengan nama *Python Software Foundation* [19].

2.6 *Confusion Matrix*

Confusion matrix adalah suatu metode yang biasanya digunakan untuk melakukan perhitungan akurasi pada konsep data mining. *Confusion matrix* digambarkan dengan tabel yang menyatakan jumlah data uji yang benar diklasifikasikan dan jumlah data uji yang salah diklasifikasikan [20]. Tabel *confusion matrix* dapat dilihat pada Tabel 2.4.

Tabel 2.4 *Confusion Matrix*

<i>Prediction</i>	<i>Actual</i>	
	<i>Positive</i>	<i>Negative</i>
<i>Positive</i>	TP	FP
<i>Negative</i>	FN	TN

Keterangan :

1. *True Positive* (TP) adalah jumlah data pada kelas positif yang diklasifikasikan sebagai nilai positif.
2. *False Positive* (FP) adalah jumlah data pada kelas negatif yang diklasifikasikan sebagai kelas positif.
3. *False Negative* (FN) adalah jumlah data pada kelas positif yang diklasifikasikan sebagai kelas negatif.
4. *True Negative* (TN) adalah jumlah data pada kelas negatif yang diklasifikasikan sebagai kelas negatif.

$$Akurasi = \frac{TP+TN}{TP+FP+FN+TN} \quad (2.49)$$

$$Precision = \frac{TP}{TP+FP} \quad (2.50)$$

$$Recall = \frac{TP}{TP+FN} \quad (2.51)$$

$$F_1 \text{ Score} = 2 \times \frac{Recall \times Precision}{Recall + Precision} \quad (2.52)$$

2.7 UML

UML (*Unified Modeling Language*) adalah salah satu bentuk bahasa visual yang digunakan untuk menjelaskan, memberikan spesifikasi, merancang, membuat model, dan mendokumentasikan aspek-aspek dari sebuah sistem. Bahasa visual yang digunakan lebih mengedepankan pada penggunaan diagram untuk menggambarkan aspek dari sistem yang dimodelkan. UML merupakan salah satu alat bantu dalam bidang pengembangan sistem berorientasi objek [21].

2.7.1 Use Case Diagram

User Case Diagram merupakan pemodelan untuk menggambarkan kelakuan sistem yang akan dibuat. *Use Case Diagram* mendeskripsikan sebuah interaksi antara satu atau lebih dengan sistem yang akan dibuat serta untuk mengetahui fungsi apa saja yang ada didalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut [21].

2.7.2 Class Diagram

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Dalam sebuah diagram kelas terdapat atribut dan metode didalamnya. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas sedangkan metode merupakan fungsi-fungsi yang dimiliki oleh suatu kelas [21].

2.7.3 Activity Diagram

Activity diagram digunakan untuk menggambarkan aliran kerja atau aktivitas dari sebuah sistem atau proses bisnis. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi namun lebih menggambarkan proses dan jalur aktivitas secara umum. Sebuah aktivitas dapat digambarkan oleh satu *use case* atau lebih. Aktivitas menggambarkan suatu proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas [21].

2.7.4 Sequence Diagram

Sequence Diagram menggambarkan perilaku atau kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan pesan yang dikirimkan dan diterima antar objek. Untuk menggambar *sequence diagram* harus diketahui dulu objek-objek yang terlibat dalam *use case* beserta metode-metode yang dimiliki kelas yang diinisialisasi menjadi objek [21].