BAB 2
LANDASAN TEORI

2.1 Ringkasan

Ringkasan merupakan sebuah hasil teks representasi yang singkat dan akurat dari sebuah isi sebuah dokumen. Van Dijk juga juga memberikan definisi mengenai ringkasan yaitu sebuah teks yang mempunyai fungsi utamanya untuk mengindikasikan dan memprediksi struktur dari sebuah dokumen. Definisi ringkasan yang lainnya adalah konten penting dari catatan pengetahuan tertentu, dan itu adalah pengganti dari dokumen sebenarnya [11].

Membuat ringkasan mengharuskan bahwa pembuat ringkasan berupaya untuk memilih, merumuskan kembali, dan membuat teks yang sesuai dengan mengandung segmen paling informatif dari suatu dokumen. Dalam ringkasan dikategorikan dengan berbagai kriteria seperti fungsi, jumlah, genre, tipe, jenis konteks dan target.

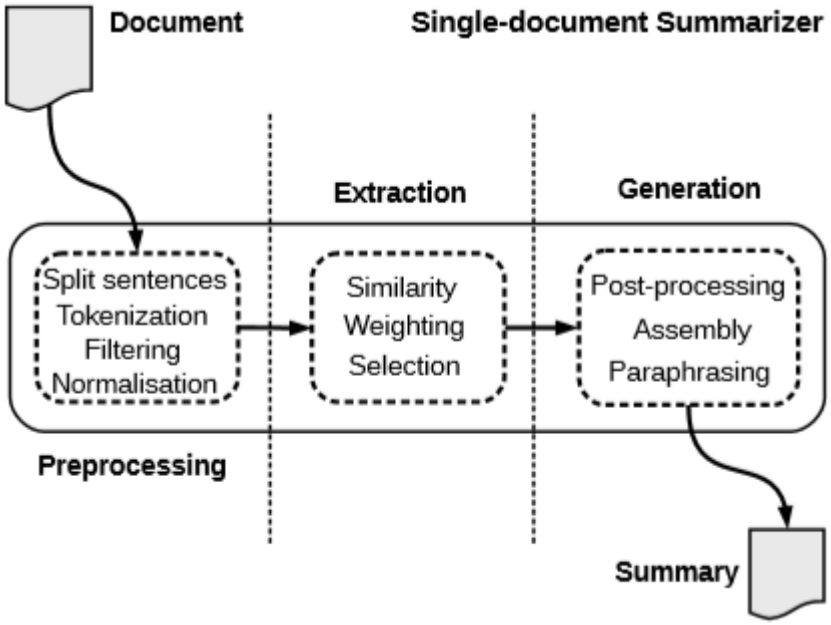
1) Ringkasan berdasarkan fungsi, terbagi menjadi dua yaitu:
   a. Ringkasan *indicative*, adalah memberikan informasi tentang topik yang dibahas dalam dokumen sumber, dimana ringkasan ini menyerupai seperti daftar isi.
   b. Ringkasan *informative,* adalah untuk merefleksikan konteks dari teks sumber, menjelaskan argumen dan seperti versi singkat dokumen.
2) Ringkasan berdasarkan jumlah dokumen, terbagi menjadi dua yaitu:
   a. Dokumen tunggal, adalah ringkasan dengan menggunakan satu dokumen.
   b. Multi dokumen, adalah ringkasan dengan menggunakan lebih dari satu dokumen yang tidak selalu heterogen dan diringkas dengan topik tertentu.
3) Ringkasan berdasarkan genre, terbagi menjadi lima yaitu:
   a. Ringkasan berita, adalah ringkasan dengan menggunakan dokumen artikel berita.

b. Ringkasan khusus, adalah ringkasan dengan meggunakan dokumen dalam bidang tertentu seperti ilmu pengetahuan, teknologi, hukum, dan lain sebagainya.
c. Ringkasan sastra, adalah ringkasan dengan menggunakan dokumen narasi, teks sastra, dan lain sebagainya.
d. Ringkasan ensiklopedis, adalah ringkasan dengan menggunakan dokumen ensiklopedis seperti Wikipedia, dan lain sebagainya.
e. Ringkasan jaringan sosial, adalah ringkasan dengan menggunakan dokumen blog dan dokumen singkat.

4) Ringkasan berdasarkan tipe, terbagi menjadi tiga yaitu:
   a. Ekstrak, adalah ringkasan dengan memilih fragmen dari dokumen sumber.
   b. Abstrak, adalah ringkasan dengan menulis ulang dengan bahasa yang berbeda tetapi dengan maksud yang sama.
   c. Kompresi kalimat, adalah meringkas dengan jumlah kalimat sama tetapi panjang yang berbeda.
5) Ringkasan berdasarkan jenis peringkas, terbagi tiga yaitu:
   a. Ringkasan penulis, adalah ringkasan yang dibuat oleh penulis sendiri dengan sudut pandangnya.
   b. Ringkasan ahli, adalah ringkasan yang dibuat oleh orang lain yang mengkhususkan diri dalam domain tetapi tidak mengkhususkan dalam memproduksi ringkasan.
   c. Ringkasan professional, adalah ringkasan yang dibuat oleh seorang yang ahli dan mengkhususkan dalam memproduksi ringkasan.
6) Ringkasan berdasarkan konteks, terbagi tiga yaitu:
   a. Genetik, adalah ringkasan yang mengabaikan kebutuhan informasi dari pengguna.
   b. Permintaan dipandu, adalah ringkasan yang dipandu dengan kebutuhan dari pengguna.
   c. Pembaharuan, adalah ringkasan yang menampilkan informasi baru yang penting dan menghindari dari pengulangan informasi.
7) Ringkasan berdasarkan target pengguna, terbagi dua yaitu:

a. Tanpa profil, adalah ringkasan yang independent dari kebutuhan profil pengguna.
b. Dengan profil, adalah ringkasan yang ditargetkan pada pengguna dalam domain tertentu seperti politik, ekonomi, olahraga, dan lain sebagainya.

Peringkasan teks otomatis dalam kamus Bahasa Inggris Oxford adalah pembuatan teks yang telah disingkat oleh program komputer dengan tetap mengandung poin-poin penting dari teks asli. Definisi lain dari peringkasan teks otomatis adalah teks yang dihasilkan oleh perangkat lunak yang berisi sejumlah besar informasi yang relevan dari teks sumber dengan tingkat kompresi kurang dari sepertiga panjang dokumen asli [11]. Skema peringkasan teks otomatis dapat dilihat pada Gambar 2.1 berikut.



Gambar 2.1 Skema Umum Peringkasan Otomatis

2.2 *Preprocessing*

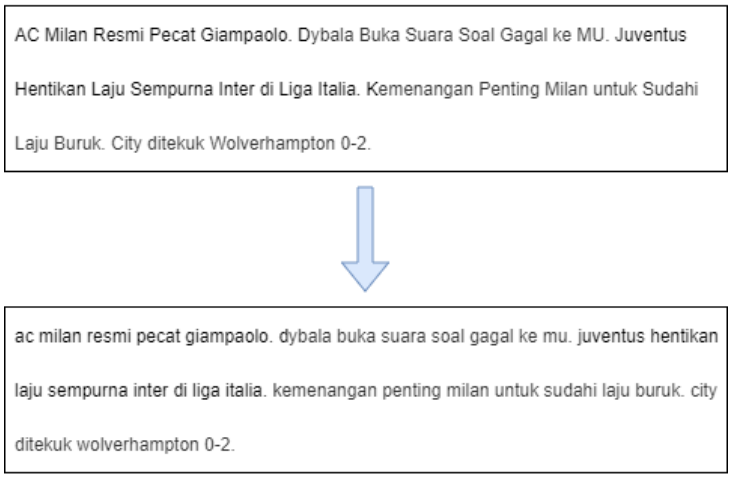
Preprocessing adalah sebuah tahapan untuk mempersiapkan teks agar dapat digunakan atau diolah pada tahapan berikutnya. *Preprocessing* dalam peringkasan teks otomatis bertujuan agar teks yang digunakan nantinya akan bersih dari *noise* atau ciri-ciri yang tidak berpengaruh dalam penghitungan [12]. Pada peringkasan teks otomatis tahap *preprocessing* terdiri dari *case folding, split sentence, filtering,*

tokenization dan *stopword removal*. Adapun tahapan dari *preprocesing* dapat dilihat pada Gambar 2.2 berikut.

Gambar 2.2 Tahapan *Preprocessing*

2.2.1 *Case Folding*

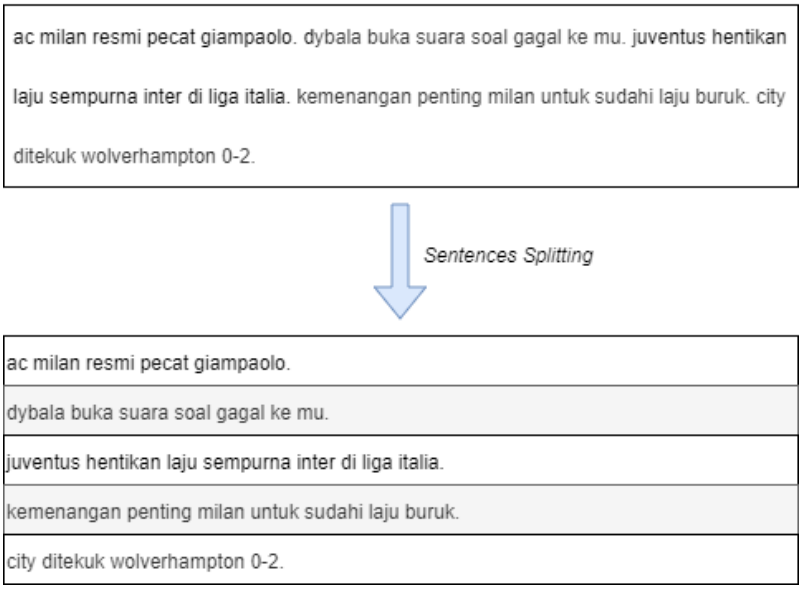
Case Folding adalah proses pemrosesan teks dimana teks dirubah menjadi bentuk yang sama, yaitu menjadi huruf kecil (*lowercase*) atau huruf kapital (*uppercase*). Variasi huruf harus diseragamkan untuk mengurangi *noise* [12]. Pada penelitian ini teks diseragamkan menjadi *lowercase*. Contoh dari *case folding* dapat dilihat pada Gambar 2.3 berikut.



Gambar 2.3 Tahap *Case Folding*

2.2.2 *Split Sentence*

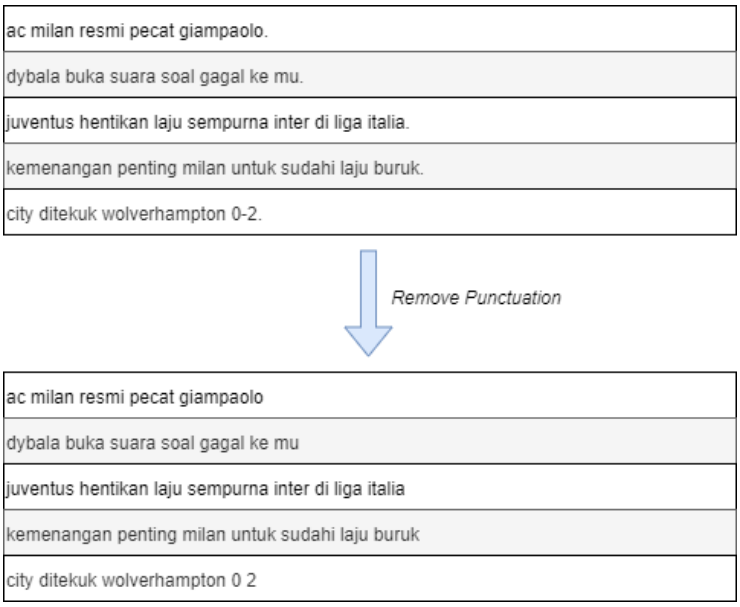
Split Sentence merupakan proses pemecahan dokumen teks menjadi pecahan paragraph atau kalimat. *Split Sentence* dilakukan dengan memecah teks menjadi kalimat dengan berdasarkan pembatas (*delimiter*) yang digunakan [2]. Pada penelitian ini pembatas yang digunakan pada s*plit sentence* adalah tanda titik ".", contoh dari s*plit sentence* dapat dilihat pada Gambar 2.4 berikut.



Gambar 2.4 Tahap *Split Sentence*

2.2.3 *Filtering*

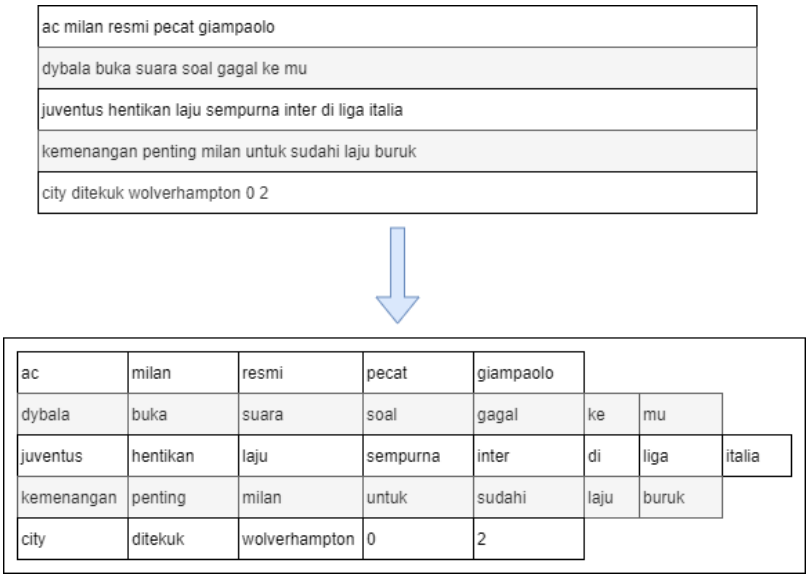
Filtering merupakan proses menghilangkan tanda baca atau simbol dalam kalimat. Proses *filtering* atau *remove punctuation* digunakan agar dapat mengurangi noise pada teks saat melakukan proses selanjutnya [2]. Contoh dari *filtering* dapat dilihat pada Gambar 2.5 berikut.



Gambar 2.5 Tahap *Filtering*

2.2.4 *Tokenization*

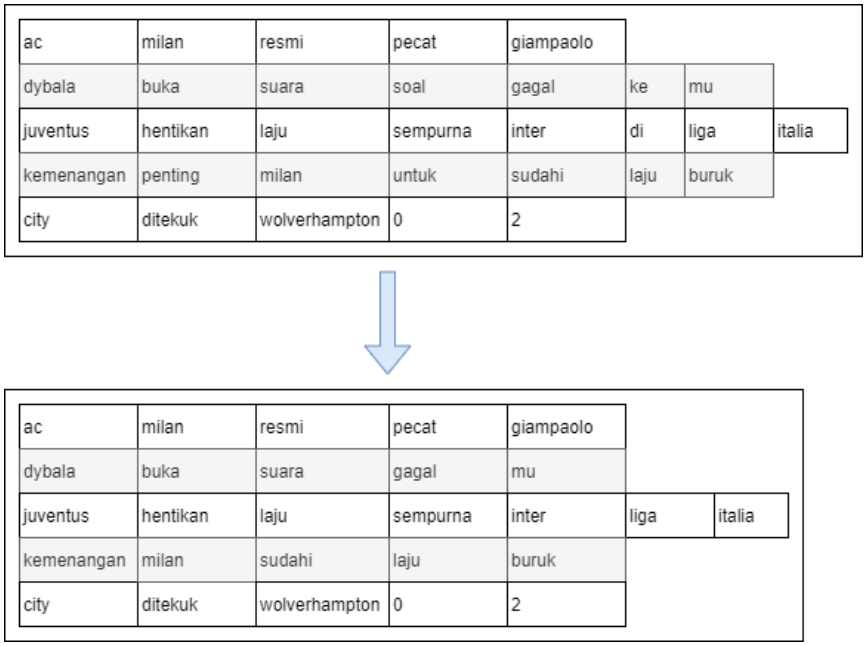
Tokenization merupakan proses tahap pemotongan kalimat berdasarkan setiap kata pada kalimat menjadi kata tunggal yang memiliki arti yang disebut sebagai token. Secara umum pemecahan kalimat menjadi kata tunggal menggunakan pembatas (*delimiter*) seperti spasi, *tab*, dan *newline* [2]. Contoh dari *Tokenization* dapat dilihat pada Gambar 2.6 berikut.



Gambar 2.6 Tahap *Tokenization*

2.2.5 *Stopword Removal*

Stopword removal merupakan proses menghapus atau menghilangkan kata-kata yang terdapat pada daftar *stoplist.* Kata yang berada dalam daftar *stoplist* adalah kata penghubung, kata pengganti dan kata penunjuk. Daftar *stoplist* yang digunakan berasal dari penelitian Fadilla Z. Tala dengan jumlah kata sebanyak 756 [12]. Contoh dari *Stopword Removal* dapat dilihat pada Gambar 2.7 berikut.



Gambar 2.7 Tahap *Stopword Removal*

2.3 Ekstraksi Fitur

Ringkasan teks yang ekstraktif dirumuskan dengan mengekstraksi teks utama (kalimat atau bagian) berdasarkan skor setiap kalimat. Skor kalimat dihitung berdasarkan fitur eksrtaksi dimana nilai dari setiap fitur berada pada range [0,1] sehingga skor tidak memiliki selisih yang besar [2].

Adapun fitur ekstraksi yang digunakan dalam penelitian ini adalah panjang kalimat (f_1), kalimat mengandung angka (f_2), posisi kalimat (f_3), kalimat menyerupai judul (f_4), kemiripan antar kalimat (f_5), koneksi antar kalimat (f_6), penjumlahan bobot koneksi antar kalimat (f_7), ikatan leksikal dengan kalimat sebelumnya (f_8), ikatan leksikal dengan kalimat setelahnya (f_9), *positive keyword* (f_{10}), *negative keyword* (f_{11}), dan *Term Frequency-Invres Document Frequency* (f_{12}).

2.3.1 Panjang Kalimat (f_1)

Kalimat yang pendek dalam dokumen memungkinkan untuk tidak merepresentasikan topik dokumen, dikarenakan kata yang terkandung didalamnya sedikit, walaupun memilih kalimat yang panjang juga tidak baik untuk peringkasan

teks dokumen [3][13]. Fitur panjang kalimat dapat dihitung menggunakan persamaan 2.1 berikut.

$$Score_{f1}(s_i) = \frac{total\ kata\ dalam\ kalimat\ s_i}{total\ kata\ dalam\ kalimat\ terpanjang} \quad (2.1)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

2.3.2 Kalimat Mengandung Angka (f_2)

Kalimat yang memiliki angka numerik dimungkinkan untuk masuk dalam ringkasan dokumen dikarenakan angka numerik pada kalimat biasannya dianggap penting [2]. Fitur kalimat mengandung angka dapat dihitung menggunakan persamaan 2.2 berikut.

$$Score_{f2}(s_i) = \frac{\#(data\ angka\ dalam\ kalimat\ s_i)}{Length(s_i)} \quad (2.2)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

2.3.3 Posisi Kalimat (f_3)

Posisi kalimat adalah letak dari kalimat dalam sebuah dokumen. Kalimat pertama dalam sebuah dokumen berita dianggap penting, karena dalam berita biasannya representasi dari isi berita berada di awal paragraf. Fitur posisi kalimat dapat dihitung menggunakan persamaan 2.3 berikut.

$$Score_{f3}(s_i) = \frac{N - i}{N} \quad (2.3)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

N = Jumlah kalimat dalam dokumen

i = Indeks kalimat (dimulai dari 1)

2.3.4 Kalimat Menyerupai Judul (f_4)

Kalimat yang menyerupai judul merupakan kumpulan kata yang dapat dicocokan antara judul dengan kalimat satu atau dapat dikatakan bahwa kata yang muncul dalam kalimat sama dengan kata yang ada dalam judul dari dokumen [2]. Kalimat yang menyerupai judul biasanya dianggap penting karena judul dianggap yang menarik pertama kali untuk membaca inti dari berita. Fitur kalimat yang menyerupai judul dapat dihitung menggunakan persamaan 2.4 berikut.

$$Score_{f4}(s_i) = |\frac{keyword\ dalam\ s_i \cap keyword\ dalam\ judul}{keyword\ dalam\ s_i \cup keyword\ dalam\ judul}| \quad (2.4)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

2.3.5 Kemiripan antar Kalimat (f_5)

Kemiripan antar kalimat adalah daftar dari kata-kata yang bisa dicocokan antara kalimat satu dengan kalimat lainnya dalam suatu dokumen atau dapat dikatakan kata yang muncul dalam suatu kalimat sama dengan kata yang muncul dengan kalimat yang lain [2]. Fitur ini berguna karena ringkasan biasanya tidak mengandung kalimat yang memiliki kesamaan dengan kalimat yang telah masuk kedalam ringkasan. Fitur kemiripan antar kalimat dapat dihitung menggunakan persamaan 2.5 berikut.

$$Score_{f5}(s_i) = |\frac{keyword\ dalam\ s_i \cap keyword\ dalam\ antar\ kalimat}{keyword\ dalam\ s_i \cup keyword\ dalam\ antar\ kalimat}| \quad (2.5)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

2.3.6 Koneksi antar Kalimat (f_6)

Koneksi antar kalimat dalam dokumen merupakan banyaknya jumlah *link* dari suatu kalimat yang terhubung dengan kalimat lain atau dapat dikatakan jumlah kalimat yang memiliki kata yang sama dengan kalimat lainnya [2]. Fitur koneksi antar kalimat dapat dihitung menggunakan persamaan 2.6 berikut.

$$Score_{f6}(s_i) = \frac{\#jumlah\ koneksi\ antar\ kalimat\ s_i}{\sum_{i=1}^{n} \#jumlah\ koneksi\ antar\ kalimat\ s_i} \quad (2.6)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

n = Jumlah banyaknya kalimat

2.3.7 Penjumlahan Bobot Koneksi antar Kalimat (f_7)

Fitur penjumlahan bobot koneksi antar kalimat berfungsi untuk menjumlahkan kata suatu kalimat dengan kata yang sama dalam kalimat yang lainnya dengan dokumen yang sama [2]. Fitur penjumlahan bobot koneksi antar kalimat dapat dihitung menggunakan persamaan 2.7 berikut.

$$Score_{f7}(s_i) = \frac{\#jumlah\ kata\ sama\ pada\ kalimat\ s_i}{\sum_{i=1}^{n} \#jumlah\ kata\ sama\ pada\ kalimat\ s_i} \quad (2.7)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

n = Jumlah banyaknya data

2.3.8 Ikatan Leksikal dengan Kalimat Sebelumnya (f_8)

Fitur ikatan leksikal antara kalimat dengan kalimat sebelumnya adalah kata yang muncul atau sama dalam kedua kalimat tersebut. Fitur ini berguna untuk menilai kalimat berdasarkan keterhubunganya. Skor diberi nilai 1 jika memiliki hubungan leksikal dan akan diberi skor 0 jika tidak memiliki hubungan leksikal dengan kalimat s_{i-1} [4].

2.3.9 Ikatan Leksikal dengan Kalimat Setelahnya (f_9)

Fitur ikatan leksikal antar kalimat dengan kalimat setelahnya adalah kata yang muncul atau sama dalam kedua kalimat tersebut. Fitur ini berguna untuk menilai kalimat berdasarkan keterhubunganya. Skor diberi nilai 1 jika memiliki hubungan leksikal dan akan diberi skor 0 jika tidak memiliki hubungan leksikal dengan kalimat s_{i+1} [4].

2.3.10 *Positive Keyword* (f_{10})

Fitur *positive keyword* merupakan kata yang paling banyak muncul pada sebuah dokumen [2]. Fitur ini berguna untuk menilai kalimat berdasarkan *keyword* yang paling banyak muncul. Fitur *positive keyword* dapat dihitung menggunakan persamaan 2.8 berikut.

$$Score_{f10}(s_i) = \frac{s_i(positif\ keyword)}{\sum_{i=1}^{n} s_i(positif\ keyword)} \quad (2.8)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

n = Jumlah banyaknya data

2.3.11 *Negative Keyword* (f_{11})

Fitur *negative keyword* merupakan kebalikan dari fitur *positive keyword*, dimana kata yang paling sedikit muncul dalam dokumen [2]. Fitur ini berguna untuk menilai kalimat berdasarkan *keyword* yang paling sedikit muncul dalam dokumen. Fitur *negatif keyword* dapat dihitung menggunakan persamaan 2.9 berikut.

$$Score_{f11}(s_i) = \frac{s_i(negatif\ keyword)}{\sum_{i=1}^{n} s_i(negatif\ keyword)} \quad (2.9)$$

Keterangan:

s_i = Posis kalimat dengan indeks ke-i

n = Jumlah banyaknya data

2.3.12 *Term Frequency-Invers Document Frequency* (f_{12})

Term Frequency-Invers Document Frequency atau Tf-Idf merupakan proses yang digunakan untuk menghitung bobot frekuensi pada setiap kata dalam suatu dokumen [14]. Penghitungan bobot didapat dari frekuensi total kemunculan sebuah kata pada sebuah dokumen disebut sebagai *term frequency* (Tf). Kemudian total kemunculan *term* atau kata didalam koleksi dokumen disebut sebagai *invers document frequency* (Idf). Bobot kata dalam Tf-Idf akan mendapatkan bobot yang besar jika sering muncul dalam satu dokumen dan akan semakin kecil jika kata

tersebut muncul dalam banyak dokumen [15]. Perhitungan nilai Tf dari sebuah dokumen dapat dihitung menggunakan persamaan 2.10 berikut.

$$Tf = f_{td} \tag{2.10}$$

Keterangan:

Tf = *term frequency*

f_{td} = frekuensi kata t dalam dokumen d

Kemudian untuk menghitung nilai Idf sebuah kata (*term*) dapat dihitung menggunakan persamaan 2.11 berikut.

$$idf_t = \log\left(\frac{N}{df}\right) \tag{2.11}$$

Keterangan:

idf_t = total kemuculan *term* didalam koleksi dokumen

N = jumlah dokumen

df = jumlah dokumen yang mengandung setiap kata

Kemudian untuk melakukan perhitungan bobot W dari setiap dokumen terhadap kata (*term*) dapat dihitung menggunakan persamaan 2.12 berikut.

$$W_{dt} = tf_{dt} * idf_t \tag{2.12}$$

Dengan keterangan W_{dt} adalah bobot dokumen ke-d terhadap kata ke-t. Kemudian untuk menghitung skor akhir setiap dokumen dapat dihitung menggunakan rumus 2.13 berikut.

$$Score_{f12}(s_i) = \sum_{i=1}^{Nterm} W_{dt_{ij}} \tag{2.13}$$

Dimana $Nterm$ merupakan jumlah banyaknya *term*, dan $W_{td_{ij}}$ adalah nilai dari Tf-Idf pada *term* ke-i, dokumen ke-j. Dengan keterangan bahwa setiap kalimat dari kumpulan teks dianggap sebagai dokumen dalam perhitungan [16].

2.4 *Particle Swarm Optimization*

Algoritma *Particle Swarm Optimization* pada awalnya diusulkan oleh J. Kennedy dan R. C. Eberhart sebagai alternatif dari algoritma genetika. Dalam konteks optimasi *multi-variabel*, kawanan diasumsikan mempunyai ukuran tertentu atau tetap dengan setiap partikel posisi awalnya terletak di suatu lokasi yang acak dalam ruang multi dimensi. Setiap partikel diasumsikan memiliki dua karakteristik yaitu posisi dan kecepatan [5]. *Particle Swarm Optimization* atau biasa dikenal dengan PSO mempunyai beberapa komponen utama seperti partikel, komponen kognitif dan komponen social, serta kecepatan partikel. Setiap partikel merepresentasikan solusi penyelesaian. Pengalaman partikel (*cognitive learning*) sebagai *pBest* yaitu posisi terbaik yang pernah dicapai dari sebuah partikel. Kombinasi pembelajaran dari keseluruhan *swarm* (*social learning*) sebagai *gBest* yaitu posisi terbaik dari keseluruhan *swarm*. *pBest* dan *gBest* digunakan untuk menghitung kecepatan partikel, dimana kecepatan digunakan untuk menghitung posisi selanjutnya.

Dalam melakukan pemilihan fitur menggunakan PSO terdapat beberapa komponen dan proses yang dilalui yaitu inisialisasi partikel, menghitung *fitness*, *pBest* dan *gBest*, melakukan update kecepatan dan kondisi berhenti [5].

1) Inisialisais Partikel

   Inisialisais partikel dilakukan untuk membangkitkan himpunan solusi baru secara acak (*random*) yang terdiri sejumlah *string* dimensi partikel. Pada iterasi ke-0 ($t = 0$) dan $v_{ij}(t) = 0$ sebagai kecepatan awal semua partikel.

2) *Fitness*

   Nilai *fitness* dapat dikatakan sebagai posisi partikel. Proses mendapatkan nilai *fitness,* terdapat tiga metode sebagai berikut.

   a. *Filter Base*, metode *filter* menyelesaikan permasalahan pemilihan fitur tanpa menggunakan model pembelajaran (*learning algoritm*) yang menyebabkan waktu komputasi dapat lebih singkat [17].

   b. *Wrapper Base*, metode *wrapper* membutuhkan algoritma pembelajaran sebagai nilai evaluasi terhadapt fitur. Dimana akan dilakukan algoritma

pembelajaran secara berulang untuk memandu proses pemilihan setiap iterasi [17].

c. *Hybrid Base*, metode *hybrid* adalah gabungan antara metode *filter* dan *wrapper*. Gabungan tersebut dimaksudkan untuk mendapatkan performa terbaik dengan algoritma pemeblajaran tetapi dengan waktu komputasi seperti metode *filter base* [17].

3) *pBest* dan *gBest, personal best* atau *pBest* merupakan posisi terbaik yang pernah dicapai partikel dengan membandingkan *fitness* pada posisi partikel sekarang dan sebelumnya, jika nilai *fitness* lebih baik dari sebelumnya maka $pBest = pBest_{ij}$. *Global best* atau *gBest* merupakan posisi terbaik yang pernah dicapai partikel dengan membandingkan nilai *fitness* terbaik dari keseluruhan partikel.

4) Kecepatan, *velocity* atau kecepatan merupakan *vector* yang menentukan arah perpindahan posisi partikel yang dilakukan setiap iterasi dengan tujuan memperbaiki posisi partikel semula. Perhitungan *velocity* dapat dihitung menggunakan rumus 2.14 berikut.

$$V_i(t+1) = V_{ij}(t) + c_1 r_1(t)\left[pBest_{ij}(t) - x_{ij}(t)\right] + c_2 r_2(t)[gBest_{ij}(t) - x_{ij}(t)] \quad (2.14)$$

Keterangan:

c_1, c_2 = nilai koefisien akselerasi

r_1, r_2 = nilai acak dengan range [0,1]

x_{ij} = posisi partikel ke-i dalam dimensi-j dalam waktu ke-t

5) Kondisi Berhenti, menurut Engelbrecht, beberapa kondisi berhenti dalam iterasi PSO adalah sebagai berikut.

a. Berhenti ketika iterasi telah mencapai maksimum iterasi yang diperbolehkan.
b. Berhenti ketika solusi yang diterima telah ditemukan.
c. Berhenti ketika tidak ada lagi perkembangan setelah beberapa iterasi.

2.5 *Support Vector Machine*

Support Vector Machine (SVM) adalah teknik klasifikasi yang efisien dalam masalah nonlinier. SVM berusaha menemukan bidang pemisah (*hyperplane*) dengan memaksimalkan jarak antar kelas (*margin*). Objek data yang berada terluar yang terdekat dengan *hyperplane* disebut *support vector*. Oleh karena itu *support vector* tersebutlah yang nantinya diperhitungkan oleh SVM untuk menemukan *hyperplane* paling optimal. Data himpunan dinotasikan dengan $x_i \in R^d$ sedangkan label kelas dinotasikan sebagi $y_i \in \{-1, +1\}$ untuk $i = 1, 2, \dots, n$, di mana *n* adalah banyaknya data [14]. Jika kedua kelas -1 dan +1 dapat terpisah sempurna oleh *hyperplane* berdimensi d, maka dapat didefinisikan oleh persamaan 2.15 berikut.

$$w.x + b = 0 \tag{2.15}$$

Dengan sebuah data x_i diklasifikasikan sebagai kelas -1 jika:

$$w.x_i + b \leq -1 \tag{2.16}$$

dan akan diklasifikasikan sebagai data +1 jika:

$$w.x_i + b > 1 \tag{2.17}$$

Dimana x_i sebagai data *input*, y_i adalah label yang diberikan, w adalah nilai dari bidang normal dan b adalah posisi bidang relatif terhadap pusat koordinat.

Margin terbesar dapat dicari dengan cara memaksimalkan jarak antar bidang pembatas kedua kelas dan titik terdekatnya, yaitu $\frac{1}{||w||}$. Hal ini dirumuskan sebagai permasalahan *quadratic programming* (QP) *problem* yaitu mencari titik minimal persamaan 2.18 dengan memperhatikan persamaan 2.19 berikut.

$$min\ \tau(w) = \frac{1}{2}||w||^2 \tag{2.18}$$

$$y_i(w * x_i + b) - 1 \geq 0, (i = 1, \dots, n) \tag{2.19}$$

Kemudian solusi untuk mengoptimasi yang dilakukan oleh Vapnik (1995) diselesaikan menggunakan metode *lagrange multiplier* sebagai berikut.

$$L(w,b,a) = \frac{1}{2}w^T w - \sum_{i=1}^{n} a_i\{y_i[(w^T * x_i) + b] - 1\} \quad (2.20)$$

Dimana a_i adalah pengganda fungsi *lagrange multiplier* dan i = 1,2, ... ,n.

Nilai optimal dapat dihitung dengan meminimalkan L terhadap w, b dan a. Hal ini layaknya kasus dual problem.

$$\frac{\partial L}{\partial b} = w - \sum_{i=1}^{n} a_i y_i = 0 \quad (2.21)$$

$$\frac{\partial L}{\partial w} = \sum_{i=1}^{n} a_i x_i y_i = 0 \quad (2.22)$$

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} a_i y_i (w^T x_i + b) - \sum_{i=1}^{n} a_i = 0 \quad (2.23)$$

Dimana n adalah jumlah data yang menjadi *support vector*. Karena *lagrange multiplier* (a) tidak diketahui nilainya, persamaan diatas tidak dapat diselesaikan secara langsung. Maka masalah *lagrange* untuk klasifikasi dapat dinyatakan pada persamaan 2.26 dengan memodifikasi persamaan diatas menjadi maksimasi dengan syarat optimal untuk dualitas menggunakan konstrain KKT (*Karush-Kuhn-Tucker*) yang dinyatakan sebagai persamaan 2.24 dan persamaan 2.25 berikut.

Syarat 1: $a_i\ [y_i(w.x_i + b) - 1] = 0$ (2.24)

Syarat 2: $a_i \geq 0, i = 1,2,\dots,n$ (2.25)

$$\min r(w) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i y_i (w^T x_i + b) - \sum_{i=1}^{n} a_i \quad (2.26)$$

Model persamaan diatas adalah model *primal lagrange*, sedangkan untuk memaksimalkan L terhadap a_i, persamaanya berubah menjadi persamaan (2.27) berikut.

$$L_D = \sum_{i=1}^{n} a_i - \frac{1}{2}\sum_{i=1,j=1}^{n} a_i a_j y_i y_j^T x_i x_j^T \quad (2.27)$$

Dengan memperhatikan syarat persamaan 2.28 dan 2.29 berikut.

Syarat 1: $\sum_{i=1}^{n} a_i y_i = 0$ (2.28)

Syarat 2: $a_i \geq 0, i = 1,2, \ldots, n$ (2.29)

Nilai *x* didapatkan dari persamaan 2.30 kernel linier untuk *x* berikut.

$$\sum_{i=1,j=1}^{n} x_i x_j^T = (i, j = 1 \ldots, n) \quad (2.30)$$

Nilai *y* didapatkan dari persamaan 2.31 kernel linier untuk *y* berikut.

$$\sum_{i=1,j=1}^{n} y_i y_j^T = (i, j = 1 \ldots, n) \quad (2.31)$$

Dengan $x_i x_j$ adalah *dot-product* dari dua data dalam suatu data latih. Batas pemisah atau *hyperplane* didapatkan dengan Persamaan 2.32 berikut.

$$\sum_{i=1}^{n} a_i y_i x_i z + b = 0 \quad (2.32)$$

Dimana *n* adalah jumlah data yang menjadi *support vector*, x_i merupakan *suport vector*, dan z adalah data uji yang kemudian akan diprediksi, $x_i z$ adalah *inner-product* antara x_i dan z, kemudian dalam penelitian ini menggunakan parameter b sebagai bias yang diberi nilai 0 [18]. Terdapat empat jenis fungsi kernel yang dapat digunakan adalah sebagai berikut.

1) Kernel linier
$$K(x, x_k) = x_k^T x \quad (2.33)$$
2) Kernel polinomial
$$K(x, x_k) = (x_k^T x + 1)^d \quad (2.34)$$
3) Kernel Gaussian (radial basis function, RBF)
$$K(x, x_k) = \exp\{-\|x - x_2\|_2^2\} / \sigma^2 \quad (2.35)$$
4) Kernel sigmoid

$$K(x, x_k) = \tanh\left[\kappa\, x_k^T x + \theta\right] \tag{2.36}$$

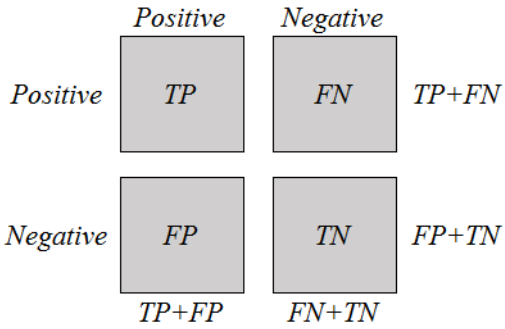
2.6 Python

Python merupakan bahasa pemrograman komputer, sama layaknya seperti bahasa pemrograman lain, misalkan Pascal,C,C++, Java, PHP dan lain-lain. sebagai bahasa pemrograman, python tentu memili dialek, kosakata atau kata kunci (keyword) dan aturan tersendiri yang jelas berbeda dengan bahasa pemrograman lainnya. Bahasa pemrograman phyton disusun pada akhir tahun 1980-an dan baru di implementasikan pada Desember 1989 opeh Guido Van Rossum di Centrum Wiskunde & Informatica (CWI), sebuah riset pada bidang matematika dan sains, Amsterdam-Belanda. Sebagai suksesor atas pengganti bahasa pemrograman sebelumnya, bahasa ABC, yang juga dikembangkan di CWI oleh Leo Guerts, Lambert Martens, dan Steven Pemberton [19]. Banyak para programmer menggunakan bahasa Python dikarenakan beberapa alasan sebagai berikut.

1. Python memiliki konsep design yang bagus dan sederhana, yang berfokus pada kemudahan dalam penggunaan. Kode Python dirancang untuk mudah dibaca, dipelajari, digunakan ulang, dan dirawat. Selain itu, Python juga mendukung pemrograman berorientasi objek dan pemrograman fungsional.
2. Python dapat meningkatkan produktivitas dan menghemat waktu bagi para programmer. Untuk memperoleh hasil program yang sama, kode Python jauh lebih sedikit dibandingkan dengan kode yang ditulis menggunakan bahasa–bahasa pemrograman lain.
3. Program yang ditulis menggunakan Python dapat dijalankan di hamper semua sistem operasi (Linux, Windows, Mac, dan lain – lain), termasuk untuk perangkat–perangkat mobile.
4. Python bersifat gratis atau bebas (free) dan open-source, meskipun digunakan untuk kepentingan komersil.

2.7 Pengujian Kinerja Klasifikasi

Kinerja dari klasifikasi dapat menggunakan *matrix confussion*. *Matrix confussion* merupakan tabel yang mencatat hasil dari kerja klasifikasi. Ketepatan dari klasifikasi dapat dilihat dari nilai akurasi klasifikasi [20]. Akurasi klasifikasi menunjukkan performasi model klasifikasi secara keseluruhan, dimana semakin tinggi akurasi klasifikasi hal ini semakin baik performansi model klasifikasi.



Gambar 2.8 *Confussion Matrix*

Adapun untuk menghitung akurasi dapat menggunakan persamaan berikut:

$$Akurasi = \frac{TP + TN}{TP + FN + FP + TN} \; x \; 100\% \tag{2.37}$$

Keterangan :

TP : *True positive*

FN : *False Negative*

FP : *False Positive*

TN : *True Negative*

Selain akurasi, nilai presisi juga dapat digunakan untuk mengukur ketepatan klasifikasi. Presisi digunakan untuk mengukur ketepatan informasi yang diminta terhadap klasifikasi:

$$Precission = \frac{TP}{TP + FP} \; x \; 100\% \tag{2.38}$$

2.8 Diagram Konteks

Diagram konteks atau *context diagram* adalah tingkatan tertinggi dalam diagram aliran data, dikarenakan dalam diagram ini hanya memuat satu proses, Dalam diagram konteks hanya ada satu proses yang menggambarkan sistem secara keseluruhan dan tidak boleh terdapat *store*, sistem juga dibatasi oleh *boundary* (dapat digambarkan dengan garis putus). Proses tersebut diberi nomor nol, didalamnya terdapat semua entitas eksternal yang ditunjukan pada diagram konteks berikut aliran data-data utama menjuju dari sistem. Diagram konteks merupakan gambaran awal yang mencakup dasar, sistem, umum, dan keluaran [21].

2.9 Data Flow Diagram (DFD)

Data Flow Diagram (DFD) merupakan suatu cara atau metode untuk membuat rancangan sebuah sistem yang mana berorientasi pada alur data yang bergerak pada sebuah sistem nantinya. Dalam pembuatan Sistem Informasi, DFD sering digunakan. DFD dibuat oleh para analis untuk membuat sebuah sistem yang baik. Dimana DFD ini nantinya diberikan kepada para programmer untuk melakukan proses coding. Dimana para programmer melakukan sebuah coding sesuai dengan DFD yang dibuat oleh para analis sebelumnya. Dan juga DFD digunakan untuk menggambarkan suatu sistem yang telah ada atau sistem baru yang akan dikembangkan secara logika tapa mempertimbangkan lingkungan fisik dimana data tersebut mengalir atau lingkungan fisik dimana data tersebut akan disimpan. Data flow diagram juga digunakan pada metodologi pengembangan sistem yang terstruktur [21].

2.10 Pemrograman Terstruktur

Pemrograman terstruktur adalah teknik pemrograman dalam merangkai instruksi-instruksi dalam bahasa komputer agar bisa tersusun secara logis dan sistematis agar mudah dimengerti, diuji, dan dimodifikasi. Pemrograman terstruktur membuat sebuah program sebagai kumpulan prosedur. Dimana prosedur-prosedur tersebut dapat saling memanggil dan dipanggil dari manapun di dalam program serta dapat mengunakan parameter yang berbeda-beda untuk setiap

pemanggilan. Bisa dibilang, pemrograman terstruktur ini merupakan implementasi dari fungsi flowchart dalam pemrograman [22].