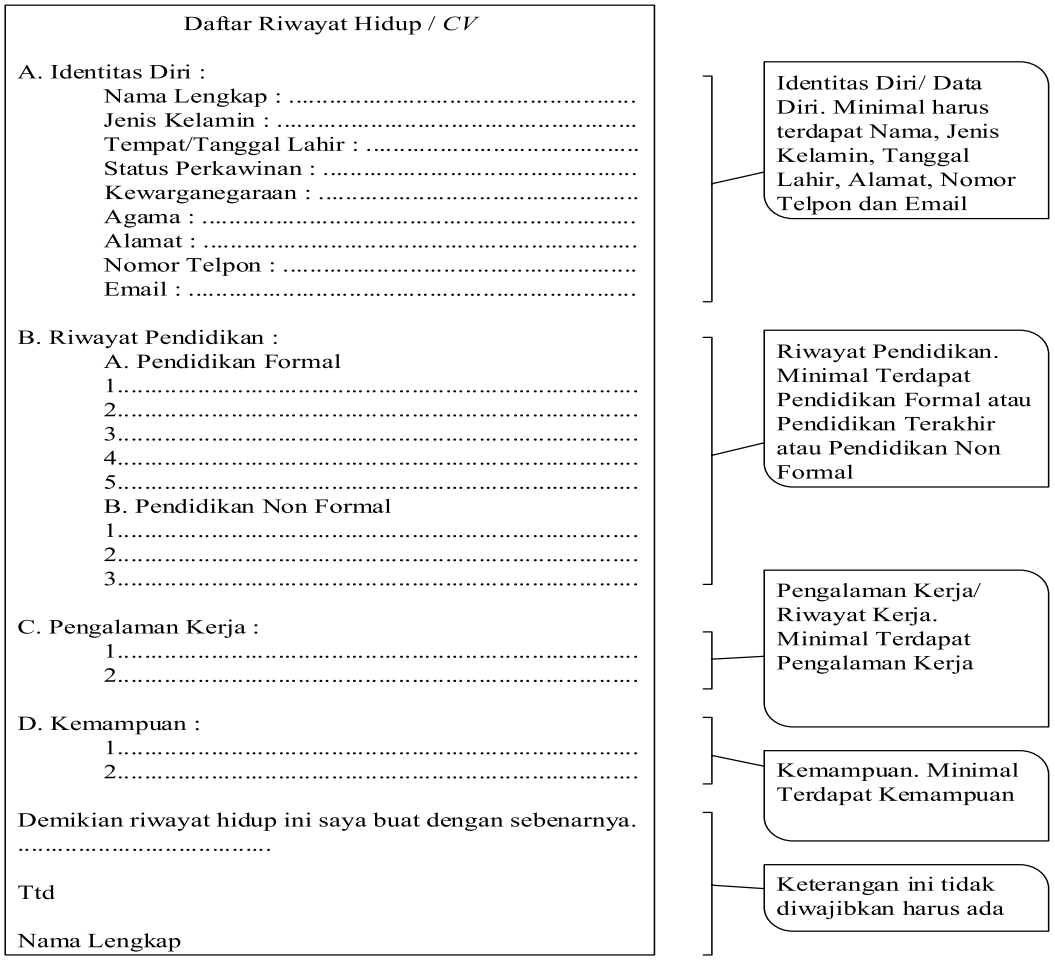
BAB 2
LANDASAN TEORI

2.1 Daftar Riwayat Hidup

Daftar riwayat hidup atau *Curriculum Vitae* (CV) adalah sebuah dokumen yang menerangkan secara detail mengenai diri seseorang, apa yang sudah dilakukannya yang mempunyai hubungan dengan pekerjaan atau kegiatan yang ingin dilakukan, sehingga akan terlihat apakah seseorang dianggap tepat untuk melakukan pekerjaan atau kegiatan tersebut[2].

Biasanya sebuah *CV* memuat berbagai informasi pokok mengenai seorang pelamar pekerjaan diantaranya data diri seperti nama, tempat, tanggal lahir, jenis kelamin, agama, kewarganegaraan, alamat, telepon dan selain data diri juga mencantumkan latar belakang pendidikan, kursus-kursus, *skill* atau kemampuan kerja, pengalaman kerja dan informasi lain yang dibutuhkan[2]. Secara keseluruhan *CV* merupakan informasi lengkap gambaran pribadi seseorang untuk digunakan melamar pekerjaan atau untuk keperluan lainnya seperti melanjutkan studi, diklat dan lain sebagainya. Karena dalam *CV* memiliki penulisan yang berbeda-beda, maka setidaknya harus memiliki isian seperti susunan daftar riwayat hidup pada Gambar 2.1 yang akan digunakan sebagai dasar pengambilan data *CV*.



Gambar 2.1 Susunan Daftar Riwayat Hidup.

2.2 *Text Mining*

Text mining adalah salah satu variasi dari ruang lingkup *data mining*, yang mencoba untuk mencari pola-pola yang unik dari database yang besar. *Text mining* merujuk kepada proses dari mengekstraksi hal-hal yang diinginkan dan informasi serta pengetahuan yang belum terungkap dari teks yang tidak terstruktur[12].

Text mining adalah prosedur untuk mengambil informasi dengan menganalisis hubungan, pola-pola, dan aturan-aturan dari data teks. Salah satu gagasan utama dari *text mining* adalah mentransformasikan teks menjadi data numerik. Metode ini akan mentransformasikan data dari teks menjadi bentuk numerik standar dengan cara membentuk format *spreadsheet* dari teks atau bahasa sederhananya tidak terstruktur menjadi terstruktur[13]. Teks yang sudah

ditransformasikan kemudian akan dilakukan analisis data seperti proses *data mining* yang kebanyakan menggunakan *machine learning*. Dalam *machine learning* terdapat *supervised learning* dan *unsupervised learning*. Perbedaan keduanya adalah jika *supervised learning* memiliki data latih sebelumnya untuk dapat menentukan kelas yang akan diujikan selanjutnya, sementara *unsupervised learning* tidak menggunakan data latih untuk mengelompokan kelas yang akan diujikan selanjutnya.

Kemudian perbedaan dari *data mining* pada umumnya dengan *text mining* adalah, didalam *text mining* pola-polanya di ekstrak dari bahasa alami (*natural language*) bukan dari data yang sudah terstruktur[12]. Jadi, sumber data yang digunakan pada *text mining* adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur. Kesamaan dari *text mining* dan *data mining* terletak setelah proses transformasi kumpulan data numerik dari dokumen yang tidak terstruktur, keduanya sama-sama menggunakan metode statistik dan *machine learning* yang sudah ada untuk memproses informasi[12].

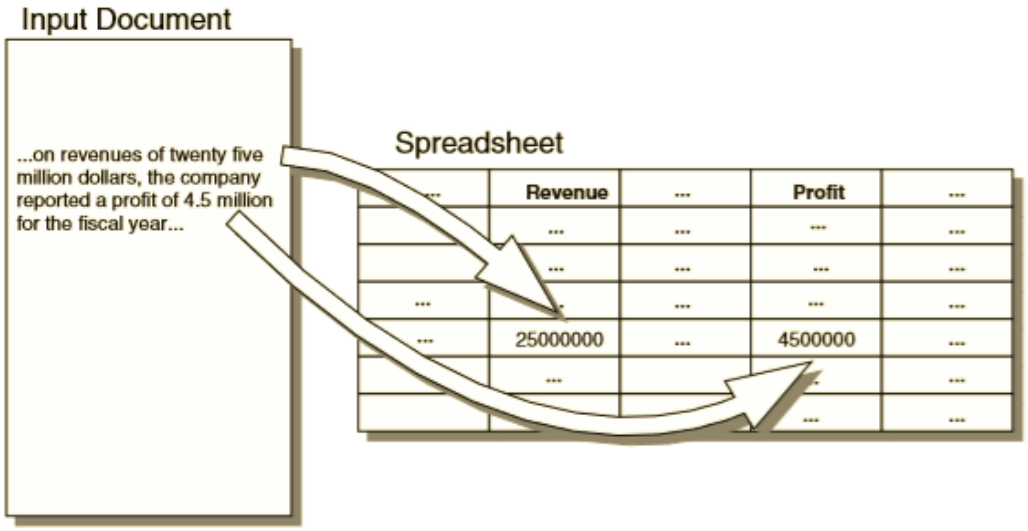
Dari informasi yang didapat dari *text mining* dokumen daftar riwayat hidup inilah akan disusun menjadi informasi yang terstruktur. Dibutuhkannya tahapan proses dalam *text mining* yang tepat agar informasi yang dibutuhkan tetap terjaga. Dalam penelitian ini *text mining* yang dilakukan meliputi beberapa proses, yaitu *text preprocessin*g, pembobotan dan klasifikasi.

2.2.1 Ekstraksi Informasi

Ekstraksi informasi adalah cabang ruang lingkup dari *text mining*, bertujuan untuk mengubah hasil proses *text mining* menjadi akar yang sama dengan dunia data yang terstruktur dalam *data mining*[13]. Metode Ekstraksi Informasi memungkinkan mengambil informasi aktual yang ada dalam teks bukan hanya kategorisasi *tag* teks secara umum. Dengan demikian, teknik *preprocessing* yang melibatkan ekstraksi informasi cenderung akan menciptakan model yang lebih optimal dan fleksibel untuk dokumen dalam sistem *text mining*[14].

Sederhananya proses ekstraksi informasi ini mengecek dokumen dan mengisikannya kedalam sel, proses yang sama seperti membuat tabel dalam

database[13]. Ekstraksi informasi pada penelitian ini akan dilakukan pada dokumen daftar riwayat hidup. Gambaran ekstraksi informasi dapat dilihat pada Gambar 2.2



Gambar 2.2 Contoh Proses Ekstraksi Informasi Dokumen[13]

Pada Gambar 2.2 adalah contoh dari ekstraksi informasi dimana dari sebuah teks masukan diambil informasi pendapatan (*revenue*) sebesar 2,5 juta dan keuntungan sebesar 4,5 juta yang kemudian disusun dalam tabel (*spreadsheet*).

2.2.2 *Application Programming Interface* (API)

Web service merupakan sebuah API (*Application Programming Interface*) atau *Web API* yang diakses melalui HTTP (*Hypertext Transfer Protocol*) dan dieksekusi oleh sebuah *remote system* yang menjadi *host* dari *service* tersebut[15].

API yang dimaksud berupa merubah suatu ekstensi *file* menjadi ekstensi baru, dan mengikuti format dari ekstensi baru tersebut. Dalam penelitian ini yang akan dirubah adalah dokumen daftar riwayat hidup yang pada kondisi awalnya berekstensi .pdf menjadi *file* berekstensi baru yaitu .html. Tujuan dari *API* ini adalah untuk menyiapkan dokumen yang lebih mudah dibaca sebagai teks.

Dalam penelitian ini akan menggunakan *Web Service Representational State Transfer* (REST) *API* dari website https://pdftables.com. Keterangan dari masing-masing ekstensi adalah :

a. Ekstensi *file* awal (.pdf) : PDF (*Portable Document Format*) adalah sebuah format berkas yang dibuat oleh Adobe pada tahun 1993 untuk keperluan pertukaran dokumen digital[16]. Alat yang kita gunakan untuk mendigitalkan kertas dokumen kita adalah *scanner*, saat ini ada beberapa jenis teknologi *scanner* yang bisa kita temukan di pasar, seperti flatbed *scanner*, *automatic document feeder* (ADF) *scanner*, *portable scanner*, *book scanner*, yang masing-masing memiliki fungsi yang berbeda-beda[17].
b. Ekstensi *file* baru (.html) : HTML (*Hypertext Markup Language*) bukanlah bahasa pemrograman (*programming language*), tetapi bahasa *markup* (*markup language*)[15]. karena tidak akan ditemukan struktur yang biasa di temukan dalam bahasa pemrograman seperti *IF, LOOP*, maupun variabel. Disebut *Markup Language* karena bahasa *HTML* menggunakan tanda (*mark*), untuk menandai bagian-bagian dari *text*. Misalnya, *text* yang berada di antara tanda tertentu akan menjadi tebal, dan jika berada di antara tanda lainnya akan tampak besar. Tanda ini di kenal sebagai *HTML tag*(< />)[15].

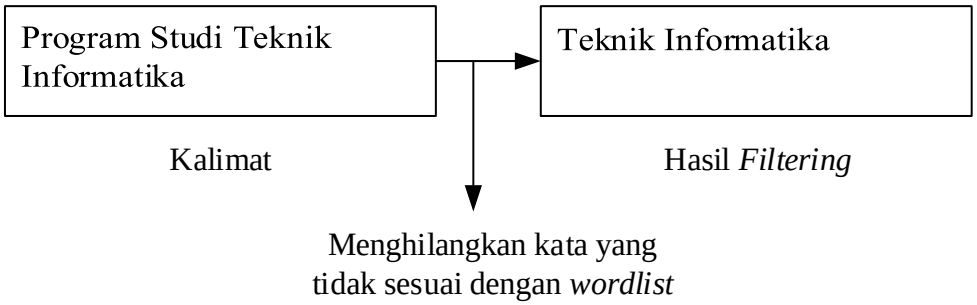
2.2.3 *Text Preprocessing*

Text preprocessing adalah tahapan untuk mempersiapkan teks menjadi data yang akan diolah di tahapan berikutnya. Masukan awal pada proses ini adalah berupa dokumen[14]. Dalam prosesnya *preprocessing* yang digunakan dalam penelitian ini memiliki 3 tahap, yaitu *filtering*, *tagging*, dan *tokenization*. Sebelum melalui 3 tahap tersebut dokumen daftar riwayat hidup harus melalui proses *API* untuk menyiapkan dokumen daftar riwayat hidup yang lebih mudah dibaca sebagai teks.

2.2.3.1 *Filtering*

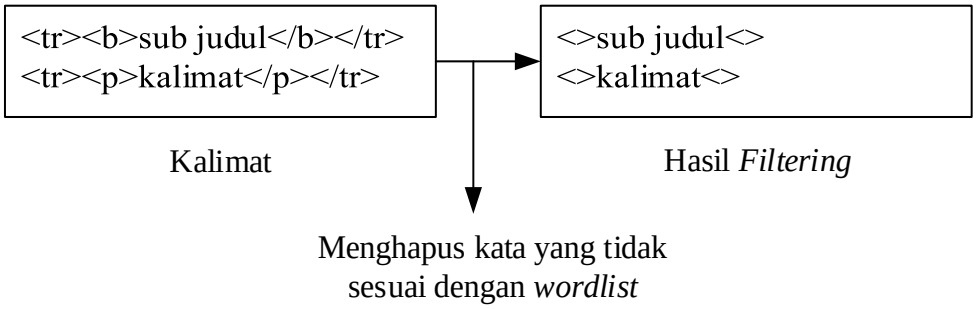
Filtering merupakan proses penghilangan kata yang tidak terdapat pada *wordlist*. *Wordlist* adalah kumpulan kata-kata yang akan di pertahankan. contoh umum *filtering* yang akan digunakan dapat dilihat pada Gambar 2.3. dimana bagian

Kalimat berupa isian yang terdiri dari beberapa kalimat, menjadi hanya beberapa kalimat saja sesuai kumpulan kata pada *wordlist*.



Gambar 2.3 Contoh Umum *Filtering*

Dasar filtering ini digunakan dalam penelitian ini untuk menjadi salah satu metode *preprocessing text*, untuk menghapus kata yang tidak sesuai dengan *wordlist*. Dalam penelitian ini, dapat dilihat proses *filtering* seperti di Gambar 2.4 dimana bagian kalimat akan dihapus jika tidak sesuai dengan *wordlist*. Untuk proses pengambilan *tag HTML* pada sistem menggunakan *library* python yaitu *beautifulsoup,* agar memudahkan dalam mengambil isian *tag HTML* yang sudah terdapat *wordlist* macam-macam *tag HTML*. Selanjutnya untuk menghapus kata didalam *tag* menggunakan *regular expression* atau lebih dikenal sebagai *regex.* Pada bagian hasil *filtering* Gambar 2.4 adalah hasil kata yang dipertahankan sesuai dengan *wordlist* yang ada.

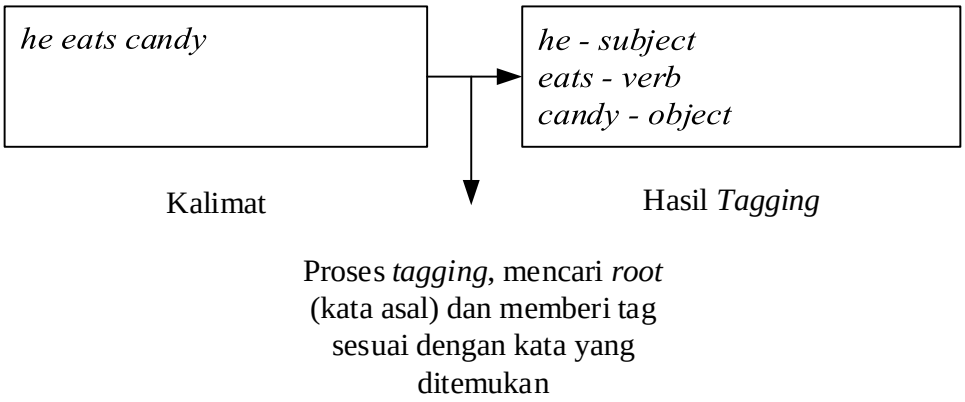


Gambar 2.4 Contoh *Filtering* Dalam Penelitian

2.2.3.2 *Tagging*

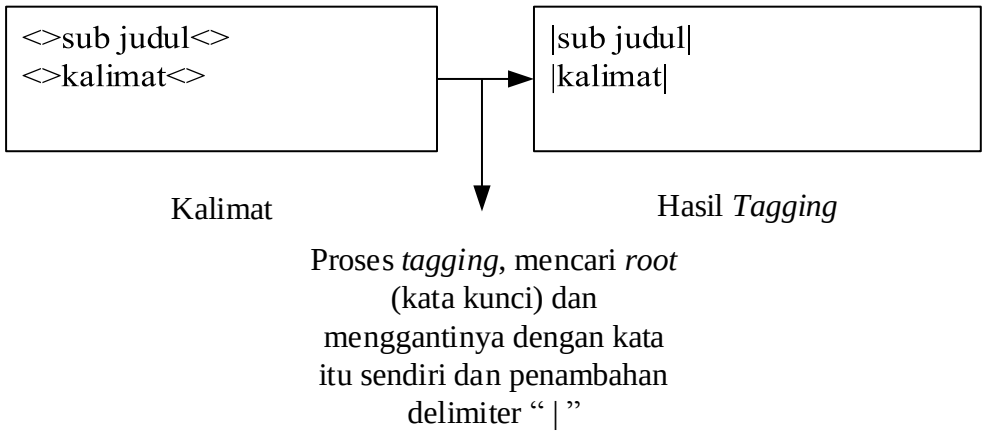
Tagging adalah membagi kata ke dalam kategori-kategori berdasarkan fungsi dari kalimat yang terbentuk dari kata tersebut[14]. Tujuan pemberian *Tag* adalah untuk menentukan kategori dari kemungkinan-kemungkinan kategori yang

terdapat dalam contoh teks tersebut[13]. Contoh penggunaan *tagging* yang paling umum adalah dalam B. Inggris dimana terdapat 5 sampai 7 kategori, dengan kebanyakan memiliki subjek, kata kerja, kata sifat, kata benda, penunjukan, dan kata penghubung. Contoh umum tahap *tagging* dapat dilihat pada Gambar 2.5 dimana bagian kalimat akan diberikan tanda dalam B. Inggris apabila ditemukan kesesuaian dengan kata kunci. Pada bagian hasil *tagging* Gambar 2.5 adalah hasil dari pemberian *tag* dimana token "*he*" sesuai dengan *root* yang memiliki *tag subject* sehingga diberikan tanda *subject*, token "*eats*" sesuai dengan *root* yang memiliki *tag verb* sehingga diberikan tanda *verb*, dan seterusnya.



Gambar 2.5 Contoh Umum *Tagging*

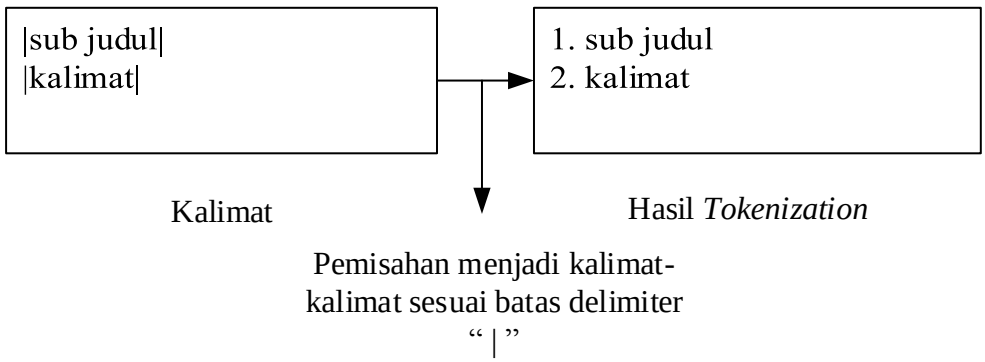
Dasar *tagging* ini digunakan dalam penelitian ini untuk menjadi salah satu metode *preprocessing text*, untuk mencari *tag* "<>" yang kemudian akan diberi *tag* khusus. Dalam penelitian ini, dapat dilihat proses *tagging* seperti di Gambar 2.6 dimana bagian kalimat akan diberikan tanda apabila ditemukan kesesuaian dengan kata kunci. Pada bagian hasil *tagging* Gambar 2.6 adalah hasil dari pemberian tanda dimana ditemukan *tag* "<>" yang bersesuaian sehingga diberikan tanda "|".



Gambar 2.6 Contoh *Tagging* Dalam Penelitian

2.2.3.3 *Tokenization*

Tokenization bertujuan untuk memecah karakter yang kontinyu (teks yang panjang) menjadi bagian-bagian yang lebih bermakna atau detail. Proses ini dapat terjadi dalam beberapa tingkatan. Dokumen dapat dipecah menjadi bab, bagian, paragraf, kalimat, kata, bahkan suku kata dan fonem. Pendekatan proses ini yang paling sering ditemukan dalam sistem *text mining* adalah memecah teks menjadi kalimat atau kata-kata, yang disebut *tokenization*[14]. Proses *tokenization* dapat dilihat pada Gambar 2.7 dimana setiap ditemukan tanda "|" pada bagian kalimat akan dipisahkan menjadi kalimat-kalimat yang terpisah pada bagian hasil *tokenization*.



Gambar 2.7 Contoh *Tokenization* Dalam Penelitian

2.2.4 Pembobotan

Pembobotan menggunakan teknik *information retrieval* (mendapatkan informasi). Secara garis besar, metode pengambilan informasi terbagi dalam dua kategori, yaitu *document selection* dan *document ranking*[18]. Dalam *document selection*, kueri dianggap sebagai batasan spesifik untuk memilih dokumen/informasi yang relevan. Metode khas dari kategori ini adalah *boolean retrieval model*, di mana pengguna menginisialisasi ekspresi kata kunci *Boolean*, seperti "mobil dan bengkel," "teh atau kopi," atau "sistem *database* tetapi bukan Oracle." Sistem pengambilan informasi akan mengambil dan mengembalikan nilai dokumen yang memenuhi ekspresi *boolean* tersebut dengan nilai 1 dan selebihnya adalah 0.

Sederhananya bila memenuhi kueri fungsi bobot *boolean* akan bernilai 1 dan selain itu bernilai 0. Nilai fitur yang digunakan dalam pembobotan diambil dari berbagai penelitian. Pada penelitian oleh Aditya Iftikar Riaddy, dkk[5], ekstraksi fitur yang digunakan dapat dilihat pada Tabel 2.1.

Tabel 2.1 Contoh Ekstraksi Fitur 1

Nama fitur	Deskripsi
Fitur Lokal	
INITCAPS	Dimulai dengan huruf kapital
ALLCAPS	Seluruh karakter adalah huruf kapital
CONTAINSDIGIT	Mengandung angka
ALLDIGITS	Seluruh karakter adalah angka
PHONEORZIP	Nomor telepon atau kode pos
CONTAINSDOTS	Mengandung paling sedikit satu titik
CONTAINSDASH	Mengandung paling sedikit satu garis
ACRO	Akronim / singkatan
LONELYINITIAL	Inisial seperti A.
SINGLECHAR	Hanya mengandung satu karakter
CAPLETTER	Hanya mengandung satu huruf kapital

PUNC	Tanda baca
URL	Alamat URL
EMAIL	Alamat email
Fitur Tata Letak	
LINE_START	Berada di awal baris
LINE_IN	Berada di pertengahan baris
LINE_END	Berada di akhir baris
Fitur *Named Entity*	
LOCATION	Nama lokasi
PERSON	Nama orang
ORGANIZATION	Nama organisasi
MONEY	Nominal uang
PERCENT	Nominal dalam persen
DATE	Waktu dalam penanggalan
TIME	Menunjukan Jam

Selanjutnya pada penelitian yang dilakukan Adam Sulaiman[19], ekstraksi fitur yang digunakan dapat dilihat pada Tabel 2.2.

Tabel 2.2 Contoh Ekstraksi Fitur 2

Nama Fitur	Deskripsi
EMAIL	Memiliki *email/web*
ALLCAPS	Seluruhnya huruf kapital
DIGIT	Memiliki angka
CONTAINDASH	Memiliki tanda garis
CONTAINSLASH	Memiliki garis miring
INITIAL_KEYWORD	Memiliki kata kunci
LOCATION	Memiliki identitas tempat
CONTAINCOLON	Memiliki titik dua
STRING_LENGTH	Panjang Karakter > 250
CONTAINCOMMA	Memiliki tanda koma
DATE	Memiliki identitas tanggal

Sedangkan penelitian yang dilakukan oleh Firdamdam Sasmita[20], ekstraksi fitur yang digunakan dapat dilihat pada Tabel 2.3.

Tabel 2.3 Contoh Ekstraksi Fitur 3

Nama Fitur	Deskripsi
INITCAPS	Mengenali setiap token yang hurufnya diawali dengan kapital.
ALLCAPS	Mengenali setiap token yang semua hurufnya kapital.
CONTAINSDIGIT	Mengenali setiap token yang mengandung angka.
ALLDIGIT	Mengenali setiap token yang semuanya angka.
CONTAINSDOTS	Mengenali setiap token yang mengandung titik.
LOWERCASE	Mengenali setiap token yang semuanya huruf kecil.
PUNCTUATION	Mengenali setiap token yang mengandung tanda tertentu seperti titik, koma, titik dua, titik koma, tanda kurung, dan tanda seru.
EIGHTDIGIT	Fitur tambahan pada penelitian ini, fitur ini dikhususkan untuk mengenali token yang memiliki angka dengan panjang 8 angka.
WORD	Fitur tambahan pada penelitian ini, fitur ini dikhususkan untuk memberikan bobot pada token untuk kelas JENIS_PENELITIAN dan KALIMAT_PENGAJUAN.
LINE_START	Mengenali posisi token pada indeks array awal.
LINE_IN	Mengenali posisi token pada indeks array tengah.
LINE_END	Mengenali posisi token pada indeks array akhir.
PERSON	Mengenali token nama seseorang.
ORGANIZATION	Mengenali token sebuah organisasi.
YEAR	Mengenali ciri token tahun.

Sementara fitur yang digunakan dalam penelitian ini dapat dilihat pada Tabel 2.4.

Tabel 2.4 Ekstraksi Fitur Yang Digunakan

Nama Fitur

INITCAPS
INITDIGIT
ALLCAPS
ALLCHAR
ALLDIGIT
CONTAINCOLON
CONTAINDASH
CONTAINDOT
CONTAINCOMMA
CONTAINSLASH
CONTAINDIGIT
ONEDIGIT
TWODIGIT
FOURDIGIT
MOREFOURDIGIT
FOURDIGITMOREONE
EMAIL
MONTH
PUNCTUATION
WORD
LINE_START
LINE_IN
LINE_END

Untuk lebih jelas akan diberikan penjelasan dari setiap fitur yang digunakan pada penelitian ini, sebagai berikut.

1. *INITCAPS*

   Fitur ini akan memeriksa pada token yang diawali dengan huruf kapital. Sedangkan aturan yang digunakan yaitu menggunakan *regex*. *INITCAPS* diambil dari fitur ketiga peneliti sebelumnya, yang terdapat pada tabel contoh ekstraksi fitur ketiga peneliti tersebut.

2. *INITDIGIT*

   Fitur ini akan memeriksa pada token yang diawali dengan *string* angka. Sedangkan aturan yang digunakan yaitu menggunakan *regex*. *INITDIGIT* pengembangan dari fitur yang ada pada ketiga peneliti sebelumnya yaitu *INITCAPS*.

3. *ALLCAPS*

   Fitur ini memeriksa pada token yang seluruhnya adalah huruf kapital termasuk jika ada spasi dan titik. Dalam penelitian ini menggunakan fungsi *regex*. *ALLCAPS* diambil dari fitur ketiga peneliti sebelumnya, yang terdapat pada tabel contoh ekstraksi fitur ketiga peneliti tersebut.

4. *ALLCHAR*

   Fitur ini akan memeriksa token yang seluruhnya berupa karakter termasuk spasi, titik, koma dan strip. Dalam penelitian ini menggunakan fungsi *regex*. *ALLCHAR* pengembangan dari fitur yang ada pada ketiga peneliti sebelumnya yaitu *ALLCAPS*.

5. *ALLDIGIT*

   Fitur ini memeriksa token yang seluruhnya berupa angka termasuk spasi, garis miring, titik dua, titik, *plus* dan strip. Dalam penelitian ini menggunakan fungsi *regex*. *ALLDIGIT* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar Riaddy, dkk dan Firdamdam Sasmita, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

6. *CONTAINCOLON*

   Fitur ini memeriksa pada token yang mengandung hanya satu titik dua saja. Dalam penelitian ini menggunakan fungsi *regex*. *CONTAINCOLON* diambil dari fitur yang dimiliki oleh peneliti Adam Sulaiman, yang terdapat pada tabel contoh ekstraksi fitur peneliti tersebut.

7. *CONTAINDASH*

   Fitur ini akan memeriksa pada setiap token yang mengandung strip. Dalam penelitian ini menggunakan fungsi *regex*. *CONTAINDASH* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar Riaddy, dkk dan Adam Sulaiman, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

8. *CONTAINDOT*

   Fitur ini memeriksa pada setiap token yang mengandung lebih dari satu titik. Dalam penelitian ini menggunakan fungsi *regex*. *CONTAINDOT* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar Riaddy, dkk dan Firdamdam Sasmita, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

9. *CONTAINCOMMA*

   Fitur ini memeriksa pada token yang mengandung lebih dari satu koma. Dalam penelitian ini menggunakan fungsi *regex*. *CONTAINCOMMA* diambil dari fitur yang dimiliki oleh peneliti Adam Sulaiman, yang terdapat pada tabel contoh ekstraksi fitur peneliti tersebut.

10. *CONTAINSLASH*

    Fitur ini akan memeriksa untuk setiap token yang mengandung *slash*(“/”). Dalam penelitian ini menggunakan fungsi *regex*. *CONTAINSLASH* diambil dari fitur yang dimiliki oleh peneliti Adam Sulaiman, yang terdapat pada tabel contoh ekstraksi fitur peneliti tersebut.

11. *CONTAINDIGIT*

    Fitur ini akan memeriksa pada token yang terdapat sebuah angka atau mengandung angka. Dalam penelitian ini menggunakan fungsi *regex*. *CONTAINDIGIT* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar

Riaddy, dkk dan Firdamdam Sasmita, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

12. *ONEDIGIT*

    Fitur ini mengecek token yang mengandung 1 angka. Dalam penelitian ini digunakan aturan *regex*. *ONEDIGIT* pengembangan dari fitur yang digunakan pada peneliti Firdamdam Sasmita yaitu *EIGHTDIGIT*.

13. *TWODIGIT*

    Fitur ini akan mengecek token yang mengandung 2 angka. Dalam penelitian ini digunakan aturan *regex*. *TWODIGIT* pengembangan dari fitur yang digunakan pada peneliti Firdamdam Sasmita yaitu *EIGHTDIGIT*.

14. *FOURDIGIT*

    Fitur ini akan mengecek pada setiap token yang mengandung 4 angka dan hanya satu saja yang terindikasi sebagai 4 angka. Dalam penelitian ini digunakan aturan *regex*. *FOURDIGIT* pengembangan dari fitur yang digunakan pada peneliti Firdamdam Sasmita yaitu *EIGHTDIGIT*.

15. *MOREFOURDIGIT*

    Fitur ini mengecek untuk token yang mengandung lebih dari 4 angka. Dalam penelitian ini digunakan aturan *regex*. *MOREFOURDIGIT* pengembangan dari fitur yang digunakan pada peneliti Firdamdam Sasmita yaitu *EIGHTDIGIT*.

16. *FOURDIGITMOREONE*

    Fitur ini akan mengecek pada setiap token yang mengandung 4 angka dan lebih dari satu yang terindikasi sebagai 4 angka. Dalam penelitian ini menggunakan fungsi *regex*. *FOURDIGITMOREONE* pengembangan dari fitur yang digunakan pada peneliti Firdamdam Sasmita yaitu *EIGHTDIGIT* dan pengembangan dari fitur sebelumnya yaitu *MOREFOURDIGIT*.

17. *EMAIL*

    Fitur ini akan memeriksa untuk token yang termasuk ke dalam aturan *regex*. Dalam penelitian ini menggunakan fungsi *regex*. *EMAIL* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar Riaddy, dkk dan Adam Sulaiman, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

18. *MONTH*

    Fitur ini akan memeriksa untuk token yang memiliki ciri nama bulan. Dalam penelitian ini menggunakan fungsi *regex*. *MONTH* pengembangan dari fitur yang digunakan oleh peneliti Aditya Iftikar Riaddy, dkk dan Adam Sulaiman yaitu *DATE*. Lalu fitur *MONTH* juga merupakan pengembangan dari fitur yang digunakan oleh peneliti Firdamdam Sasmita yaitu *YEAR*.

19. *PUNCTUATION*

    Fitur *PUNCTUATION* mengecek untuk token yang termasuk tanda baca kurung "( )" baik kurung buka dan kurung tutup, serta kurung "[ ]"siku buka dan tutup. Dalam penelitian ini menggunakan fungsi *regex*. *PUNCTUATION* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar Riaddy, dkk dan Firdamdam Sasmita, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

20. *WORD*

    Fitur *WORD* akan mengecek pada setiap token yang spesifik atau mengandung kata pelatihan dan kursus untuk menambah bobot pada kelas pendidikan non formal. Dalam penelitian ini menggunakan fungsi *regex*. *WORD* diambil dari fitur yang dimiliki oleh peneliti Firdamdam Sasmita, yang terdapat pada tabel contoh ekstraksi fitur peneliti tersebut.

21. *LINE_START*

    Fitur ini akan memeriksa untuk setiap token yang memiliki *index array* diawal baris antara 0 sampai dengan 2. Dalam penelitian digunakan

pencocokan pada *index array*. *LINE_START* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar Riaddy, dkk dan Firdamdam Sasmita, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

22. *LINE_IN*

    Fitur ini akan memeriksa untuk setiap token yang memiliki *index array* pertengahan baris antara 3 sampai dengan 13. Dalam penelitian digunakan pencocokan pada *index array*. *LINE_IN* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar Riaddy, dkk dan Firdamdam Sasmita, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

23. *LINE_END*

    Fitur ini akan memeriksa untuk setiap token yang memiliki *index array* diakhir baris yaitu 4 terbawah dari akhir *index*. Dalam penelitian digunakan pencocokan pada *index array*. *LINE_END* diambil dari fitur yang dimiliki oleh peneliti Aditya Iftikar Riaddy, dkk dan Firdamdam Sasmita, yang terdapat pada tabel contoh ekstraksi fitur kedua peneliti tersebut.

2.2.5 Klasifikasi Teks

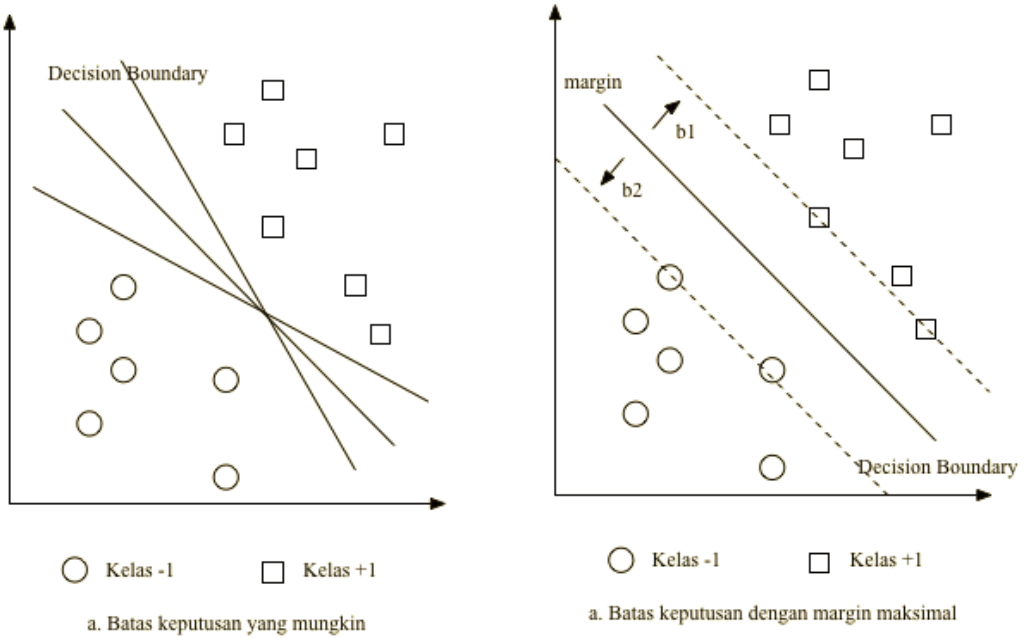
Klasifikasi teks adalah pengelompokan atau pemberian label pada dokumen sesuai dengan kategori yang sesuai dengan konten dokumen tersebut. Kumpulan kategori tersebut sering disebut dengan aturan kosakata (*Controlled Vocabulary*). Klasifikasi teks adalah aksi untuk membagi suatu kumpulan dokumen menjadi dua atau lebih kelas yang setiap dokumen dapat memiliki satu kelas atau kelas ganda[12]. Klasifikasi teks menggunakan *machine learning* sebagai media pembelajaran komputer untuk dapat mengklasifikasi teks.

Beberapa metode yang dapat digunakan untuk mengklasifikasikan teks misalnya *Naïve Bayes*, *Support Vector Machine, K-Nearest Neighbour*, dll. Dalam penelitian ini metode yang digunakan adalah *Support Vector Machine*. *SVM* dipilih karena dianggap memiliki reputasi yang baik dalam klasifikasi[21].

2.3 *Support Vector Machine* (SVM)

2.3.1 Konsep *SVM*

Ide dasar *SVM* adalah memaksimalkan batas *hyperplane*, yang diilustrasikan pada Gambar 2.8 Pada gambar (a) ada sejumlah pilihan *hyperplane* yang mungkin untuk set data, sedangkan gambar (b) merupakan *hyperplane* dengan *margin maximum*. Meskipun sebenarnya pada gambar (a) bisa juga menggunakan *hyperplane* sembarang, tetapi *hyperplane* dengan *margin* yang maksimal akan memberikan generalisasi yang lebih baik pada metode klasifikasi[22].

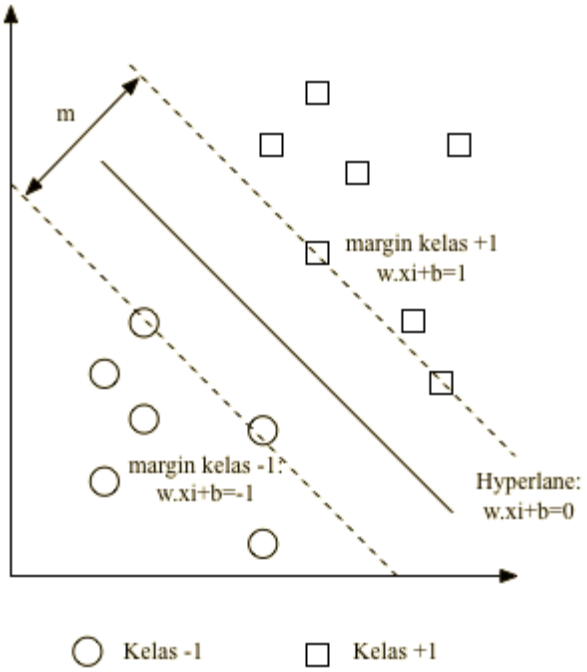


Gambar 2.8 Batas Keputusan Yang Mungkin Untuk Set Data[22]

Konsep klasifikasi dengan *SVM* dapat dijelaskan secara sederhana sebagai usaha mencari *hyperplane* terbaik yang berfungsi sebagai pemisah dua buah kelas data pada *input space* (Nugroho, 2007). Gambar 2.8 memperlihatkan beberapa data yang merupakan anggota dari dua buah kelas data, yaitu +1 dan -1. Data yang tergabung pada kelas -1 disimbolkan dengan bentuk lingkaran, sedangkan data pada kelas +1, disimbolkan dengan bujur sangkar[22].

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 sebagai *support vector*. Garis *solid* pada Gambar 2.8 (b) sebelah kanan menunjukan *hyperplane* yang terbaik, yaitu yang terletak tepat ditengah – 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*[22].



Gambar 2.9 *Margin Hyperplane*

2.3.2 *SVM Linear*

Setiap data latih dinyatakan oleh ($x_i y_i$) dengan i = 1, 2,..., N, dan $x_i = \{x_{i1}, x_{i2}, \ldots, x_{iq})^T$ merupakan atribut (fitur) set untuk data latih ke-i[22]. Untuk $y_i \in \{-1, +1\}$ menyatakan label kelas[22]. *Hyperplane* klasifikasi *linear SVM* seperti pada Gambar 2.9, dinotasikan:

$$\boldsymbol{w}.\boldsymbol{x_i} + b = 0 \tag{2.1}$$

w dan b adalah parameter model. $\boldsymbol{w}.\boldsymbol{x_i}$ merupakan *inner-product* antara w dan x_i.

Data x_i yang masuk ke dalam kelas -1 adalah data yang memenuhi pertidaksamaan linier.

$$\boldsymbol{w}.\boldsymbol{x_i} + b \leq -1 \tag{2.2}$$

Sementara x_i data yang masuk ke dalam kelas +1 adalah data yang memenuhi pertidaksamaan berikut:

$$\boldsymbol{w}.\boldsymbol{x_i} + b \geq +1 \tag{2.3}$$

Sesuai dengan Gambar 2.9, jika ada dalam kelas -1 (misalnya, x_a) bertempat di *hyperplane* akan memenuhi persamaan 2.1. Untuk data kelas -1 dinotasikan:

$$\boldsymbol{w}.\boldsymbol{x_a} + b = 0 \tag{2.4}$$

Sementara kelas +1 (misalnya, x_b) akan memenuhi persamaan:

$$\boldsymbol{w}.\boldsymbol{x_b} + b = 0 \tag{2.5}$$

Dengan mengurangkan persamaan (2.5) dengan (2.4), didapatkan:

$$\boldsymbol{w}.(\boldsymbol{x_b} - \boldsymbol{x_a}) = 0 \tag{2.6}$$

$x_b - x_a$ adalah vektor paralel di posisi *hyperplane* dan diarahkan dari x_a ke x_b.

Karena *inner-product* bernilai nol, arah w harus tegak lurus terhadap *hyperplane* (Gambar 2.9).

Dengan memberikan label -1 untuk kelas pertama dan +1 untuk kelas kedua, maka untuk prediksi semua data uji akan menggunakan formula:

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

Sesuai dengan Gambar 2.9, *hyperplane* untuk kelas -1 (garis putus-putus) adalah data pada *support vector* yang memenuhi persamaan:

$$\boldsymbol{w}.\boldsymbol{x_a} + b = -1 \tag{2.8}$$

Sementara *hyperplane* untuk kelas +1 (garis putus-putus) adalah data yang memenuhi persamaan:

$$\boldsymbol{w}.\boldsymbol{x_b} + b = +1 \quad (2.9)$$

Dengan demikian maka *margin* dapat dihitung dengan mengurangkan persamaan 2.9 dengan 2.8, didapatkan:

$$\boldsymbol{w}.(\boldsymbol{x_b} - \boldsymbol{x_a}) = 2 \quad (2.10)$$

Margin hyperplane diberikan oleh jarak antara dua *hyperplane* dari dua kelas tersebut. Notasi di atas diringkas menjadi:

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

2.3.3 *Hyperplane SVM*

Klasifikasi kelas data pada *SVM* pada Persamaan 2.2 dan 2.3 dapat digabungkan dengan notasi:

$$y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) \geq 1, i = 1, 2, \ldots, \text{N} \quad (2.12)$$

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

Minimalkan:

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

Syarat:

$$y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) \geq 1, i = 1, 2, \ldots, \text{N} \quad (2.14)$$

Optimalisasi ini dapat diselesaikan dengan *lagrange multiplier*:

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

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 terhadap w dan b, dan diubah dengan nilai nol untuk syarat optimasi diatas:

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:

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

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, modifikasi Persamaan 2.15 diatas menjadi kasus memaksimalkan dengan syarat optimal untuk dualitasnya menggunakan konstrain *Karush-Kuhn-Tucker* (KKT) sebagai berikut:

Syarat 1:

$$\alpha_i[y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) - 1] = 0 \quad (2.18)$$

Syarat 2:

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

Dengan menerapkan konstrain pada Persamaan (2.18) dan (2.19) maka dipastikan bahwa nilai *Lagrange Multiplier* sama banyaknya dengan data latih, meskipun sebenarnya banyak dari data latih yang *Lagrange Multiplier* sama dengan nol (karena hanya beberpa saja yang akan menjadi *support vector*) ketika menerapkan syarat pertama. Konstrain diatas menyatakan bahwa *Lagrange Multiplier* α_i harus nol kecuali untuk data latih x_i yang memenuhi persamaan:

$$y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) = 0 \tag{2.20}$$

Data latih tersebut, 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 *support vector*.

Masalah optimasi di atas masih sulit diselesaikan karena banyaknya parameter (w, b dan α_i). Untuk menyederhanakannya, persamaan optimasi 2.15 diatas harus ditransformasi ke dalam fungsi *Lagrange Multiplier* itu sendiri (disebut dualitas masalah).

Persamaan *Lagrange Multiplier* 2.15 dapat dijabarkan menjadi:

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

Syarat optimal (2.17) ada dalam suku ketiga di ruas kanan dalam persamaan (2.21), dan memaksa suku ini menjadi sama dengan nol. Dengan mengganti *w* dari syarat (2.16), dan suku $\|w\|^2 = \boldsymbol{w_i}.\boldsymbol{w_j}$, maka persamaan di atas akan berubah menjadi dualitas *Lagrange Multiplier* berupa Ld dan didapatkan:

Maksimalkan:

$$Ld = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x_i}.\boldsymbol{x_j} \tag{2.22}$$

$\boldsymbol{x_i}.\boldsymbol{x_j}$ merupakan *dot-product* dua data dalam data latih.

Syarat 1:

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \tag{2.23}$$

Syarat 2:

$$\alpha_i > 0, \qquad i = 1,2,\dots,N \tag{2.24}$$

Untuk set data yang besar, masalah dualitas optimasi tersebut (2.22, 2.23, 2.24) dapat diselesaikan dengan metode numerik seperti *Quadratic Programming*.

Sekali α_i didapatkan, persamaan (2.16) dan (2.17) bisa digunakan untuk mendapatkan solusi layak untuk w dan b.

Hyperplane (batas keputusan) didapatkan dengan formula:

$$\left(\sum_{i=1}^{N} \alpha_i y_i \boldsymbol{x_i}.\boldsymbol{z}\right) + b = 0 \tag{2.25}$$

N adalah jumlah data yang menjadi *support vector*, x_i merupakan *support vector*, z merupakan data uji yang akan diprediksi kelasnya, dan $\boldsymbol{x_i}.\boldsymbol{z}$ merupakan *inner-product* antara x_i dan z. Untuk nilai b didapatkan dari persamaan (2.18) pada *support vector*. Karena α_i dihitung dengan teknik metode numerik dan mempunyai *error* numerik, nilai yang dihitung untuk b bisa jadi tidak sama. Hal ini disebabkan oleh *support vector* yang digunakan dalam persamaan (2.18), biasanya diambil nilai rata-rata dari b yang didapat untuk menjadi parameter *hyperplane*. Untuk persamaan (2.18) dalam mendapat dapat b dapat disederhanakan menjadi:

$$b_i = 1 - y_i(\boldsymbol{w}.\boldsymbol{x_i}) \tag{2.26}$$

Penjelasan di atas berdasarkan asumsi bahwa kedua kelas dapat terpisah secara sempurna oleh *hyperplane*. 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 ($\xi_i \geq 0$), Menjadi:

$$y_i(\boldsymbol{w}.\boldsymbol{x_i} + b) \geq 1 - \xi_i \tag{2.27}$$

Demikian juga untuk masalah persamaan(2.13):

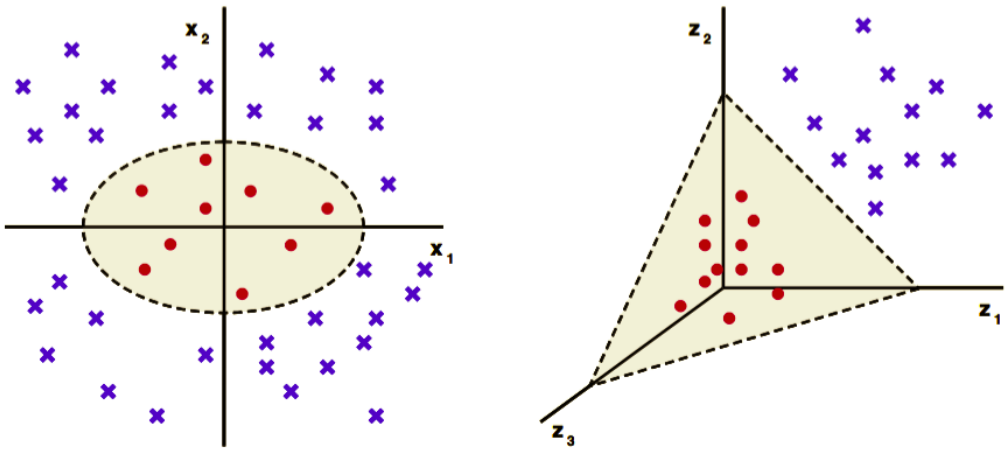
$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i \tag{2.28}$$

Parameter *C* berguna untuk mengontrol *trade-off* antara *margin* dan *error* klasifikasi. Semakin besar nilai *C* maka semakin besar pula pelanggaran yang dikenakan untuk tiap klasifikasi[22].

Metode untuk mengoptimalisasi *hyperplane* SVM umumnya dipakai untuk menyelesaikan *Quadratic Programming* dengan konstrain yang ditetapkan. Beberapa pilihan metode yang bisa digunakan adalah *chunking* (Vapnik, 1982), metode dekomposisi (Osuna *et al*, 1997), dan *Sequential Minimal Optimization* (SMO)(Plat, 1999).

2.3.4 *SVM Non Linear*

SVM sebenarnya adalah *hyperplane linear* yang hanya bekerja pada data yang dapat dipisahkan secara *linear*[22]. Untuk data yang distribusi kelasnya tidak *linear* biasanya digunakan pendekatan *kernel* pada fitur data dari awal set data. *Kernel* dapat di definisikan sebagai suatu fungsi yang memetakan fitur data dari dimensi awal (rendah) ke fitur lain yang berdimensi lain yang lebih tinggi (bahkan jauh lebih tinggi)[22]. Pendekatan ini berbeda dengan metode klasifikasi pada umumnya yang justru mengurangi dimensi awal untuk menyederhanakan proses komputasi dan memberikan akurasi prediksi yang lebih baik[22].



Gambar 2.10 Dimensi Data

Proses pelatihan yang sama sebagaimana pada *SVM linear*. 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 $\Phi(x_i)$.

$\Phi(x_j)$. Nilai *dot-product* kedua buah vektor ini dapat dihitung secara tidak langsung, yaitu tanpa mengetahui fungsi transformasi Φ. Teknik komputasi seperti ini kemudian disebut *kernel trick*, yaitu menghitung *dot–product* dua buah vektor di ruang dimensi baru dengan memakai komponen kedua buah vektor tersebut di ruang dimensi asal, seperti berikut:

$$\mathrm{K}(x_i, x_j) = \Phi(x_i) \,.\, \Phi(x_j) \tag{2.29}$$

Dan prediksi pada set data dengan dimensi fitur yang baru diformulasikan:

$$\mathrm{f}(\Phi(x)) = \mathrm{sign}(\mathrm{w}.\,\Phi(\mathrm{z}) + \mathrm{b}) = \mathrm{sign}\left(\sum_{\mathrm{i}=1}^{\mathrm{N}} \alpha_\mathrm{i} \mathrm{y}_\mathrm{i} \Phi(\mathrm{x}_\mathrm{i}).\,\Phi(\mathrm{z}) + b\right) \tag{2.30}$$

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

Berikut beberapa pilihan fungsi *kernel* yang banyak digunakan dalam aplikasi:

Tabel 2.5 Fungsi *Kernel*

Nama *Kernel*	**Definisi Fungsi**
Linear	$\mathrm{K}(\mathrm{x}, \mathrm{y}) = \mathrm{x}.\,\mathrm{y}$
Polynomial	$\mathrm{K}(\mathrm{x}, \mathrm{y}) = (\mathrm{x}.\,\mathrm{y} + \mathrm{c})^{\mathrm{d}}$
Gaussian RBF	$\mathrm{K}(\mathrm{x}, \mathrm{y}) = \exp(-\gamma\|\mathrm{x} - \mathrm{y}\|^2), \gamma > 0$
Sigmoid (tangen hiperbolik)	$\mathrm{K}(\mathrm{x}, \mathrm{y}) = \tanh(\sigma(\mathrm{x}.\,\mathrm{y}) + \mathrm{c})$
Invers Multiquadric	$\mathrm{K}(\mathrm{x}, \mathrm{y}) = \frac{1}{\sqrt{\|\mathrm{x} - \mathrm{y}\|^2 + \mathrm{c}^2}}$

Keterangan:

x dan y adalah pasangan dua data dari semua bagian data latih. Parameter σ, c, d, γ > 0, merupakan konstanta. $\|\mathrm{x} - \mathrm{y}\|^2$ merupakan kuadrat jarak antara vektor x dan y.

2.3.5 *SVM* Multikelas

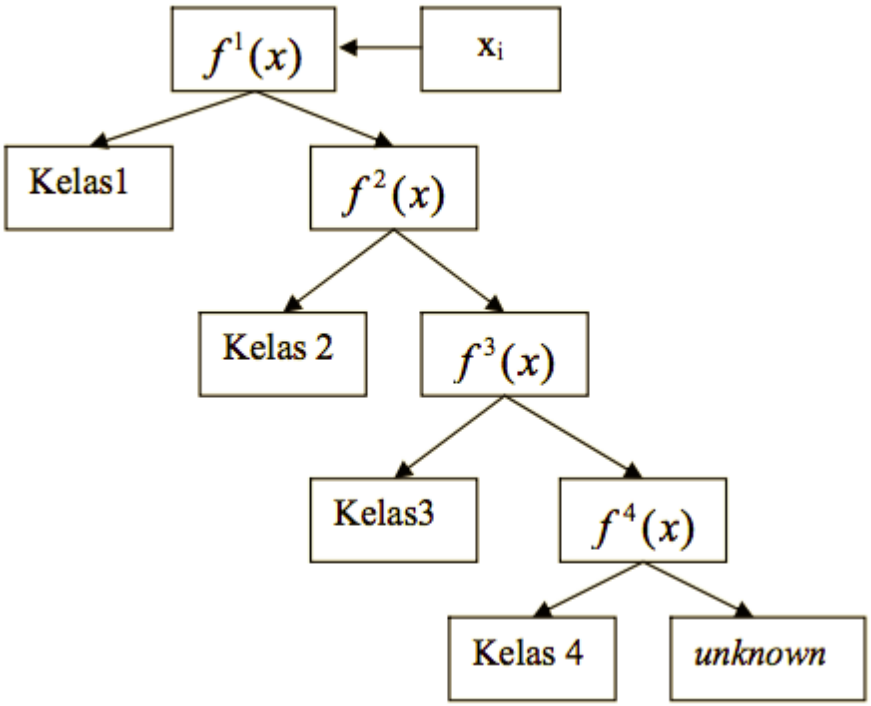
Metode-metode klasifikasi seperti *Decision Tree*, *Artficial Neural Network*, *Nearest Neighbor* di desain untuk dapat melakukan klasifikasi multikelas sekaligus, tetapi *SVM* tidak. *SVM* hanya dapat melakukan klasifikasi biner (dua kelas). Sementara masalah di dunia nyata umumnya mempunyai banyak kelas seperti pengenalan karakter, pengenalan wajah, atau diagnosis pasien, di mana data masukan terbagi menjadi lebih dari dua kelas[22]. Ada 3 Pendekatan *SVM* Multikelas yaitu *one-against-all* (OAA), *one-against-one* (OAO) dan *error correcting output code* (ECOC) (Tan *et al*, 2005). Tetapi yang akan dijelaskan pada penelitian ini adalah *SVM* Multikelas *one-against-all* (OAA).

2.3.5.1 Metode *One Against All* (OAA)

Dengan metode ini, dibangun k buah model *SVM* biner (k adalah jumlah kelas). Setiap model klasifikasi ke-i dilatih dengan menggunakan keseluruhan data, untuk mencari solusi permasalahan[22]. Contohnya, terdapat permalasahan klasifikasi dengan 4 buah kelas. Untuk pelatihan digunakan 4 buah *SVM* biner.

Tabel 2.6 Contoh 4 *SVM* Biner Dengan Metode *One Against All*

Yi = +1	Yi = -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	$f^4(x) = (w^4)x + b^4$



Gambar 2.11 Contoh Klasifikasi Dengan Metode *One Against All*

Penelitian lebih lanjut untuk mengembangkan *SVM* sehingga dapat melakukan klasifikasi lebih dari dua kelas yaitu *multiclass SVM*. Dalam klasifikasi kasus multikelas *SVM*, *hyperplane* yang terbentuk adalah lebih dari satu. Yang umum digunakan untuk mengimplementasikan multikelas *SVM* adalah pendekatan metode *One Against All*(OAA).

Konsep pada *OAA* yaitu dimisalkan pada kasus lima kelas, kelas 1, 2, 3, 4 dan 5. Bila akan diujikan ρ(1), semua data dalam kelas 1 diberi label +1 dan data dari kelas 2, 3, 4 dan 5 diberi label -1. Pada ρ(2), semua data dalam kelas 2 diberi label +1 dan data dari kelas 1, 3, 4 dan 5 diberi label -1. Pada ρ(3), semua data dalam kelas 3 diberi label +1 dan data dari kelas 1, 2, 3 dan 4 diberi label -1. Pada ρ(4), semua data dalam kelas 4 diberi label +1 dan data dari kelas 1, 2, 3, 4 diberi label -1. Begitu juga untuk ρ(5), semua data dalam kelas 5 diberi label +1 dan data dari kelas 1, 2, 3 dan 4 diberi label -1. Kemudian dicari *hyperplane* dengan algoritma *SVM* dua kelas. Maka akan didapat *hyperplane* untuk masing-masing kelas di atas. Kemudian kelas dari suatu data baru x ditentukan berdasarkan nilai terbesar dari *hyperplane*.

$$kelas\ x = \max_{\ell=1....k} \left(\left(w^{(\ell)} \right)^T . \Phi(x) + b^{(\ell)} \right) \quad (2.31)$$

2.4 Pemrograman Berorientasi Objek

Pemrograman Berorientasi Objek adalah paradigma pemrograman yang berorientasikan kepada objek yang merupakan suatu metode dalam pembuatan program, dengan tujuan untuk menyelesaikan kompleksnya berbagai masalah program yang terus meningkat[23]. Objek adalah entitas yang memiliki atribut, karakter (*behavour*) dan kadang kala disertai kondisi (*state*)[23].

Konsep dasar pemrograman berorientasi objek:

1. Objek (*Object*)

   Objek merupakan suatu entitas yang mampu menyimpan informasi yang dapt diterapkan atau dapat berpengaruh pada status objeknya.

2. Kelas (*Class*)

   Kelas merupakan penggambaran satu kumpulan objek yang memiliki atribut yang sama. Kelas mirip dengan tipe data ada pemrograman non objek, akan tetapi lebih komprehensif karena terdapat struktur sekaligus karakteristiknya. Kelas baru dapat dibentuk lebih spesifik dari kelas ada umumnya. Kelas merupakan jantung dalam pemrograman berorientasi objek.

3. Pembungkusan (*Encapsulation*)

   Enkapsulasi adalah proses memastikan pengguna sebuah objek tidak dapat menggantikan keadaan dari sebuah objek dengan cara yang tidak sesuai prosedur. Artinya, hanya metode yang terdapat dalam objek tersebut yang diberi izin untuk mengakses keadaan yang diinginkan. Setiap objek mengakses *interface* yang menyabutkan bagaimana objek lainnya dapat berintegrasi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

4. Pewarisan (Inheritance)

   Konsep inheritas mempunyai fungsi mengatur *polimorfisme* dan enkapsulasi dengan mengizinkan objek didefinisikan dan diciptakan

dengan jenis khusus dari objek yang sudah ada. Objek-objek ini dapat membagi dan memperluas perilaku mereka tanpa mengimplementasikan perilaku tersebut.

5. *Polimorfisme*

   Polimorfisme merupakan suatu fungsionalitas yang diimplikasikan dengan berbagai cara yang berbeda. Pada program berorientasi objek, pembuat program dapat memiliki berbagai implementasi untuk sebagian fungsi tertentu.

2.5 *Unified Modeling Language* (UML)

UML (*Unified Modeling Language*) adalah sebuah bahasa yang berdasarkan grafik/gambar untuk memvisualisasi, untuk menspesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pembangun *software* berbasis OO (*Object-Oriented*)[23]. *UML* merupakan sebuah bahasa standar pemodelan untuk pengembangan perangkat lunak[23]. *UML* juga memberi standar penulisan sebuah sistem *blue print*, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa pemrograman yang spesifik, skema *database*, dan komponen-komponen yang diperlukan dalam sistem *software*. Ada beberapa hal yang sangat mendasar dalam model *UML*, juga merupakan bagian paling statistik dari sebuah model, diantaranya sebagai berikut:

1. *Classes*

   Classes merupakan sekelompok dari objek yang mempunyai atribut, operasi, hubungan yang semantik. Sebuah kelas mengimplementasikan 1 atau lebih *interfaces*.

2. *Interfaces*

   Interfaces merupakan sebuah antar-muka yang menghubungkan dan melayani antar kelas dan atau elemen. *Interface* mendefinisikan sebuah kelompok dari spesifikasi pengoperasian, uumnya digambarkan dengan sebuah lingkaran yang disertai dengan namanya.

3. *Collaboration*

Collaboration merupakan interaksi sebuah kumpulan/kelompok dari kelas-kelas yang bekerja secara bersama-sama.

4. *Use Cases*

   Use Cases merupakan suatu rangkaian sekelompok yang saling terkait dan membentuk sistem secara teratur yang dilakukan atau diawasi oleh sebuah aktor.

5. *Nodes*

   Nodes merupakan fisik dari elemen-elemen yang ada pada saat dijalankannya sebuah sistem.

Ada 4 macam hubungan didalam penggunaan *UML*, yaitu:

1. *Dependency*

   Dependency adalah hubungan semantik antara dua benda yang mana sebuah benda berubah mengakibatkan benda satunya akan berubah pula.

2. *Association*

   Association merupakan hubungan antar benda struktural yang terhubung diantara obyek. Kesatuan obyek yang tehubung merupakan hubungan khusus, yang menggambarkan sebuah hubungan struktural diantara seluruh atau sebagian.

3. *Generalization*

   Generalization menggambarkan hubungan khusus dalam objek anak/*child* yang menggantikan objek *parent*/induk.

4. *Realization*

   Realization merupakan hubungan semantik antara pengelompokan yang menjamin adanya ikatan diantaranya.

Dalam menggambarkan sistem dari aplikasi yang akan dibangun, penelitian ini menggunakan *UML*. Karena, aplikasi yang dibangun berorientasi objek. Sehingga dapat dengan mudah dalam menggambarkan setiap aktivitas dari pengguna.

2.6 Python

Python adalah bahasa pemrograman komputer, sama layaknya seperti bahasa pemrograman lain, misalnya C, C++, Pascal, Java, PHP, Perl dan lain-lain. Sebagai bahasa pemrograman, Python tentu memiliki varian, kosakata atau kata kunci, dan aturan tersendiri yang jelas berbeda dengan bahasa pemrograman lainnya[24].

Bahasa pemrograman Python disusun di akhir tahun 1980-an dan implementasinya baru dimulai pada Desember 1989 oleh Guido Van Rossum di *Centrum Wiskunde & Information* (CWI), sebuah pusat riset di bidang matematika dan *sains*, Amsterdam – Belanda; sebagai suksesor atau pengganti dari bahasa pemrograman pendahulunya, bahasa pemrograman ABC, yang juga dikembangkan di *CWI* oleh Leo Geurts, Lambert Meertens, dan Steven Pemberton[24].

Secara umum, para *programer* banyak yang menjatuhkan pilihannya ke bahasa Python karena alasan – alasan berikut[24].

a. Python memiliki konsep desain 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.
b. Python dapat meningkatkan produktivitas dan menghemat waktu bagi para *programer*. Untuk memperoleh hasil program yang sama, kode Python jauh lebih sedikit dibandingkan dengan kode yang ditulis menggunakan bahasa–bahasa pemrograman lain.
c. Program yang ditulis menggunakan Python dapat dijalankan di hampir semua sistem operasi (*Linux, Windows, Mac*, dan lain-lain), termasuk untuk perangkat-perangkat *mobile*.
d. Python bersifat gratis atau bebas (*free*) dan *open-source*, meskipun digunakan untuk kepentingan komersial.

2.7 Django *Framework*

Django adalah *web framework open source* yang ditulis dengan bahasa pemrograman Python yang dikembangkan pertama kali pada tahun 2003 oleh *programer* koran *Lawrence Journal World* Adrian Holovaty dan Simon Willison[25]. Tahun 2005 Django dirilis sebagai proyek *open source*, hingga saat ini Django dikembangkan dan dimaintain oleh Django *Software Foundation* di bawah lisensi *BSD license*.

Sejak Django dirilis sebagai proyek *open source* hingga saat ini terdapat ribuan perusahaan dan organisasi di dunia yang menggunakannya dalam proyek besar maupun kecil, di antaranya: *The Washington Post, The Lawrence Journal World, Google, EveryBlock, Newsvine, Curse Gaming, Tabblo* dan *Pownce*[25].

Django menyediakan *high level framework* yang dapat digunakan untuk membangun aplikasi *web* dengan sedikit baris kode, simpel, kuat, fleksibel dan mudah.

2.8 PostgreSQL

PostgreSQL adalah sistem manajemen basis data relasional (*ORDBMS*). *ORDBMS* adalah ekstensi dari sistem manajemen basis data relasional (*RDBMS*) yang lebih tradisional. *RDBMS* memungkinkan pengguna untuk menyimpan potongan data terkait dalam struktur data dua dimensi yang disebut tabel[26]. Berikut merupakan penjelasan mengenai kemampuan PostgreSQL:

1. Skalabilitas. PostgreSQL memiliki kemampuan dalam menangani penambahan beban. Dengan penambahan beban yang sedikit, tidak akan ada dampak berlebih yang diterima oleh PostgreSQL.

2. Kecepatan. Untuk kebutuhan aplikasi yang dibangun, kecepatan yang diperhatikan adalah kecepatan ketika proses *insert* dan *read*. Berdasarkan penelitian perbandingan *database* PostgreSQL cukup unggul dalam hal kecepatan jika dibandingkan dengan *database* lainnya, sehingga PostgreSQL dapat memenuhi kebutuhan aplikasi yang dibangun dalam segi kecepatan.

3. Konsistensi data. Konsep *RDBMS* memiliki kemampuan dalam menjaga konsistensi data. Dengan konsep relasi yang dimiliki, akan menjaga data yang

saling berhubungan pada setiap entitas yang memiliki *primary key* dan *foreign key*. Jika ada data yang salah relasi, maka *RDMBS* akan menolak data tersebut sehingga data akan tetap konsisten dan jika terdapat suatu perubahan maka semua data yang saling berhubungan memiliki data yang sudah diubah.

4. Kemampuan dalam menangani transaksi yang dilaksanakan secara bersamaan. Konsep *RDBMS* terutama pada PostgreSQL mendukung konsep *isolation*, pada aplikasi yang dibangun, dibutuhkan konsep *isolation* sehingga setiap proses transaksi yang dilakukan secara bersamaan jangan sampai mempengaruhi satu sama lain.

5. Kapasitas penyimpanan data. Data pada aplikasi yang dibangun akan terus bertambah seiring waktu. *Database* PostgreSQL memiliki kemampuan untuk menyimpan baris data (*row*) tak terbatas. Adapun batasan penyimpanan data pada PostgreSQL terbatas pada ruang memori yang tersedia. Tabel 2.7 menunjukkan kapasitas penyimpanan data pada *database engine* PostgreSQL.

Tabel 2.7 Kapasitas Penyimpanan Data PostgreSQL[26]

Batasan	Nilai
Ukuran maksimum *database*	*Unlimited*
Ukuran maksimum tabel	32 TB
Ukuran maksimum baris	1,6 TB
Ukuran maksimum *field*	1 GB
Maksimum baris setiap tabel	*Unlimited*
Maksimum kolom setiap tabel	250 – 1600 *depending on column types*
Maksimum *index* setiap tabel	*Unlimited*

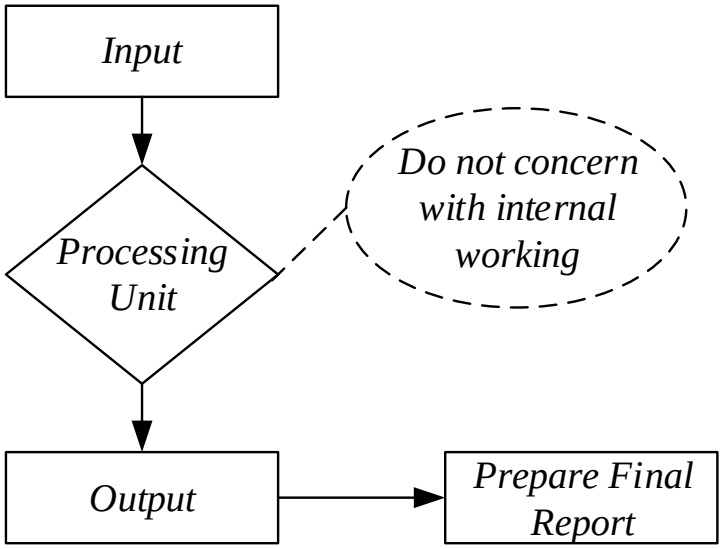
2.9 *Black Box Testing*

Black Box Testing merupakan pengujian yang berfokus pada spesifikasi fungsional dari perangkat lunak, *tester* dapat mendefinisikan kumpulan kondisi input dan melakukan pengetesan pada spesifikasi fungsional program[27]. Dalam *black box testing* cenderung untuk menemukan hal-hal berikut.

a. Fungsi yang tidak benar atau tidak ada.
b. Kesalahan antarmuka (*interface errors*).

c. Kesalahan pada struktur data dan akses basis data.
d. Kesalahan performa (*performance errors*).
e. Kesalahan inisialisasi dan terminasi.

Berikut pada Gambar 2.12 gambaran dari *black box testing*, dimana *user* menyatakan masukan dan keluaran pada setiap unit proses dan didapatkan hasil dari setiap proses tersebut apakah sesuai dengan yang dinyatakan atau tidak.



Gambar 2.12 Gambaran Pengujian *Black Box*[27]

2.10 *Confusion Matrix*

Confusion matrix merupakan sebuah metode perhitungan yang digunakan untuk mencari keakuratan pada hasil klasifikasi[18]. Pada dasarnya *confusion matrix* mengandung informasi yang membandingkan hasil klasifikasi yang dilakukan oleh sistem dengan klasifikasi yang seharusnya. *Precision* adalah tingkat ketepatan antara informasi yang diminta oleh pengguna dengan jawaban yang diberikan oleh sistem. Sedangkan *recall* adalah tingkat keberhasilan sistem dalam menemukan kembali sebuah informasi. *F1-score* merupakan salah satu perhitungan evaluasi dalam informasi temu kembali yang mengkombinasikan *recall* dan *precision*.

Pada pengukuran kinerja menggunakan *Confusion Matrix*, terdapat empat istilah sebagai representasi hasil proses klasifikasi. Keempat istilah tersebut adalah *True Positive* (TP), *True Negative* (TN), *False Positive* (FP), dan *False Negative*

(FN). Nilai *TP* merupakan data positif yang terdeteksi dengan benar, nilai *TN* merupakan jumlah data negatif yang terdeteksi benar, nilai *FP* merupakan data negatif namun terdeteksi sebagai data positif, dan *FN* merupakan data positif terdeteksi sebagai data negatif. Berikut contoh *confusion matrix* dapat dilihat pada Tabel 2.8 berikut.

Tabel 2.8 Aturan Nilai Dalam *Confusion Matrix*

Kelas		**Target**	
		P	***N***
Hasil Klasifikasi	***P***	*TP*	*FP*
	N	*FN*	*TN*

Keterangan:

a. *True Positive* (TP), merupakan jumlah dokumen dari kelas 1 yang benar diklasifikasikan sebagai kelas 1.
b. *False Positive* (FP), merupakan jumlah dokumen dari kelas 0 yang salah diklasifikasikan sebagai kelas 1.
c. *False Negative* (FN), merupakan jumlah dokumen dari kelas 1 yang benar diklasifikasikan sebagai kelas 0.
d. *True Negative* (TN), merupakan jumlah dokumen dari kelas 0 yang salah diklasifikasikan sebagai kelas 0.

Untuk menghitung akurasi digunakan perhitungan *Overall Accuracy* dengan membagi jumlah hasil klasifikasi yang bernilai benar (Jumlah dari diagonal di *confusion matrix*) dengan total dari seluruh baris dan kolom dari *confusion matrix*, atau dapat dilihat pada rumus 2.32.

$$Accuracy = \frac{Keseluruhan\ data\ terklasifikasi\ benar}{Keseluruhan\ testing\ data} * 100\% \quad (2.32)$$

$$Error\ Rate = \frac{Keseluruhan\ data\ tidak\ klasifikasi\ benar}{Keseluruhan\ testing\ data} * 100\% \quad (2.33)$$

Kemudian rumus untuk menghitung *precision, recall,* dan *f1-score* jika jumlah kelas lebih dari 1 adalah menggunakan rumus rata-ratanya. Formula yang digunakan untuk klasifikasi multikelas adalah sebagai berikut.

$$Precision = \frac{\sum_{i=1}^{l} \frac{TP_i}{TP_i + FP_i}}{l} * 100\% \quad (2.34)$$

$$Recall = \frac{\sum_{i=1}^{l} \frac{TP_i}{TP_i + FN_i}}{l} * 100\% \quad (2.35)$$

$$F1 - Score = \frac{\sum_{i=1}^{l} \frac{2\ x\ precision_i\ x\ recall_i}{precision_i + recall_i}}{l} * 100\% \quad (2.36)$$

Dengan keterangan tambahan :

l = Jumlah Kelas.

2.11 *Synthetic Minority Oversampling Technique* (SMOTE)

Synthetic Minority Oversampling Technique (SMOTE) adalah salah satu turunan dari metode *oversampling*[28]. Proses *oversampling* dilakukan setelah proses pemisahan data *fold* selesai. Proses *oversampling* hanya dilakukan pada *fold* data latih dan tidak dilakukan pada *fold* data uji[29]. Hal ini dilakukan untuk menghindari keadaan *overfitting* pada saat proses evaluasi kinerja klasifikasi. *SMOTE* pertama kali diperkenalkan oleh Nithes V. Chawla, Pendekatan ini bekerja dengan membuat replikasi dari data minoritas[28]. Replikasi tersebut dikenal dengan data sintetis (*syntetic data*). Metode *SMOTE* bekerja dengan mencari k *nearest neighbors* (yaitu ketetanggaan terdekat data sebanyak k) untuk setiap data di kelas minoritas, setelah itu dibuat data sintetis sebanyak persentase duplikasi data minor (*percentage oversampling*, N%) yang diinginkan dan *k-nearest neighbors* yang dipilih secara acak ((jumlah data kelas mayoritas/jumlah data kelas minoritas) x 100%)[28]. Proses *oversampling* dengan menggunakan teknik *SMOTE* pada

penelitian ini menggunakan *library* bahasa pemrograman *python* yang diambil dari *scikitlearn*, sehingga dapat menangani data yang tidak seimbang menjadi *balanced*.