BAB 2
LANDASAN TEORI

2.1 Ringkasan

Menurut Kamus Besar Bahasa Indonesia, ringkasan adalah hasil memendekan cerita atau pembicaraan. Menurut van Dijk, ringkasan adalah sebuah teks yang fungsi utamanya adalah untuk mengindikasi dan memprediksi struktur dan isi dari suatu teks [6].

Tujuan utama dari sebuah ringkasan adalah membantu seseorang memahami dan mengetahui isi sebuah buku atau karangan. Dengan membuat ringkasan, seseorang dibimbing dan dituntun untuk membaca karangan asli dengan cermat dan menuliskan kembali dengan tepat.

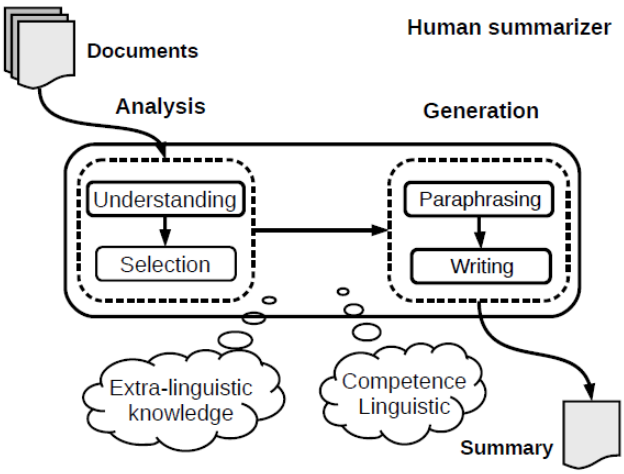
Menurut jenisnya, membuat sebuah ringkasan dapat dikelompokkan sebagai berikut [6].

a. Ekstraksi : Ringkasan yang dibentuk dari menyusun bagian-bagian dari dokumen atau teks.
b. Abstraksi : Ringkasan yang dibentuk dari menulis ulang sebuah teks atau dokumen.
c. Kompresi : Ringkasan yang mempunyai jumlah kalimat yang sama, tetapi dengan panjang kalimat yang lebih berkurang dari dokumen atau teks asli.

Pada peringkasan jenis ekstraksi dan kompresi, ringkasan yang dihasilkan secara umum dibagi ke dalam tiga kelompok, yaitu ringkasan 50%, ringkasan 30% dan ringkasan 20%. Ketiga kelompok tersebut menyatakan jumlah kalimat yang dihasilkan. Sebagai contoh, jika jumlah kalimat pada teks berjumlah 12 kalimat, maka pada ringkasan 50% jumlah kalimat ringkasan yang dihasilkan berjumlah 6 kalimat. Ringkasan 30% dapat dianggap cukup untuk mewakili isi dari keseluruhan teks yang diringkas. Pada penelitian ini ringkasan yang diambil sebanyak 30% dari keseluruhan isi teks.

Pembentukan ringkasan yang dilakukan oleh manusia dapat dilihat pada Gambar 2.1. Pembentukan ringkasan secara abstraksi yang dilakukan secara

manual oleh manusia terbagi ke dalam dua fase, yaitu memahami teks sumber (*analysis*) dan menuliskan kembali hasil pemahaman teks sumber tersebut dalam bentuk sebuah ringkasan (*generation*). Dalam fase *analysis*, seorang peringkas (*summarizer*) terlebih dahulu memahami teks yang akan diringkas (*understanding*), setelah itu melakukan pemilihan kalimat-kalimat mana saja yang akan digunakan dalam ringkasan yang akan dibuat (*selection*). Sedangkan untuk fase *generation*, peringkas akan menuliskan kembali hasil seleksi teks dalam bentuk sebuah ringkasan. Penulisan kembali teks sumber dapat dilakukan dengan dengan cara menuliskan kalimat pilihan secara utuh ataupun dengan menuliskan kembali kalimat dengan menggunakan kata-kata sendiri (*paraphrasing*). Kedua fase tersebut membutuhkan kemampuan dan pengetahuan linguistik sehingga ringkasan yang dibuat akan memiliki hasil yang baik [6].



Gambar 2.1 Proses Pembentukan Ringkasan Hasil Buatan Manusia [6]

2.2 Peringkasan Teks Otomatis

Menurut Juan-Manuel dan Torres-Moreno, ringkasan otomatis adalah sebuah ringkasan yang dibentuk oleh sebuah perangkat lunak, yang mengandung jumlah informasi relevan yang signifikan terhadap teks sumber [6]. Dengan peringkasan teks otomatis pembaca tidak perlu membuat sebuah ringkasan secara manual, tetapi dapat dilakukan dengan menggunakan bantuan komputer.

Secara umum pembentukan ringkasan oleh peringkasan teks otomatis didahului oleh proses *preprocessing*, dimana proses tersebut memecah isi dari dokumen atau teks menjadi bagian-bagian tertentu, seperti kata atau kalimat. *Preprocessing* dalam peringkasan teks otomatis memiliki dua tujuan, yaitu untuk normalisasi kata-kata dalam teks serta pengurangan kosakata yang akan diproses untuk memudahkan proses peringkasan.

Setelah proses *preprocessing* dilakukan, langkah selanjutnya adalah merepresentasi hasil *preprocessing* tersebut menjadi bentuk yang dapat diproses dan dibaca oleh algoritma peringkasan. Salah satu bentuk representasi teks yang sering digunakan adalah representasi dalam bentuk vektor [6]. Vektor tersebut biasanya mengandung nilai yang merepresentasikan jumlah kemunculan (frekuensi) kata-kata dalam sebuah teks atau dokumen yang selanjutnya akan diolah oleh algoritma peringkasan.

2.2.1 Penelitian Tentang Peringkasan Teks Otomatis

Penelitian tentang peringkasan teks otomatis pertama kali dimulai pada tahun 1950-an oleh Hans Peter Luhn untuk meringkas dokumen-dokumen riset dan teknis kimia [6]. Setelah itu pada tahun 1950 hingga 1970-an beberapa riset peringkasan teks otomatis pun mulai dibuat, termasuk karya Edmundson dan Wyllys, Rush, Gerald Francis DeJong, Sparck-Jones, serta Julian Kupiec.

Riset tentang peringkasan teks otomatis pun telah berkembang, salah satu risetnya adalah penggunaan *machine learning* dalam kasus peringkasan teks otomatis. Riset peringkasan teks otomatis menggunakan pendekatan *machine learning* pertama kali dibuat oleh Joel Larocca Neto dkk. dengan menggunakan *naïve bayes* dan C45 sebagai algoritmanya. Peringkasan teks otomatis menggunakan metode *machine learning* lainnya pun mulai bermunculan. Salah satu risetnya adalah penggunaan *support vector machine* untuk peringkasan teks otomatis yang dilakukan oleh Fuji Ren dkk.

2.3 *Preprocessing*

Preprocessing adalah sebuah tahapan yang mengubah teks menjadi data yang dapat diolah pada proses berikutnya. *Preprocessing* dalam peringkasan teks otomatis memiliki dua tujuan, yaitu untuk normalisasi kata-kata dalam teks serta pengurangan kosakata yang akan diproses untuk memudahkan proses peringkasan. [6].

Proses-proses yang dikerjakan pada tahap *preprocessing* pada peringkasan teks otomatis meliputi *case folding, splitting, tokenization, stopword removal*, dan *stemming*.

2.3.1 *Case Folding*

Case folding/capitalization [6] adalah proses yang mengubah semua huruf dalam teks dokumen menjadi huruf kecil atau huruf kapital. *Case folding* dilakukan karena dokumen mengandung berbagai variasi dari bentuk huruf. Variasi huruf harus diseragamkan untuk menghilangkan *noise* pada saat pengambilan informasi. Pada penelitian ini, *case folding* merupakan tahapan awal dari *preprocessing*.

2.3.2 *Splitting*

Splitting [6] merupakan pemecahan sebuah teks utuh menjadi pecahan-pecahan paragraf/kalimat. Dalam proses *splitting* secara umum melibatkan dua proses, yaitu *sentences splitting* dan *punctuation*. Perbedaan *split sentences* dan *punctuation* terletak pada fungsinya. *Sentences splitting* memecah teks berdasarkan *delimiter*, sedangkan *punctuation* menghapus *delimiter* yang tersisa dan tanda baca serta simbol lainnya dari hasil *sentence splitting*.

Split sentences (pemecahan kalimat) adalah proses memecah *string* teks dokumen yang panjang menjadi kumpulan kalimat-kalimat. Pemecahan kalimat dalam dokumen menjadi kalimat-kalimat dapat dilakukan dengan cara mendeteksi keberadaan tanda titik ".", tanda tanya "?" dan tanda tanya "!" sebagai *delimiter* untuk memotong *string* dokumen [6].

Setelah dilakukan proses *split sentences*, langkah berikutnya adalah melakukan *punctuation*, yaitu proses penghilangan tanda baca ataupun simbol pada kalimat yang telah dipecah [6]. Pada penelitian ini, *splitting* merupakan bagian dari tahapan *preprocessing* dan akan dilakukan setelah proses *case folding*.

2.3.3 *Tokenization*

Tokenization adalah proses pemotongan teks berupa kalimat untuk mendapatkan bagian-bagian atau *token-token* tertentu. Secara umum pemecahan kalimat menjadi kumpulan *token* dilakukan dengan melakukan pemindaian kalimat dengan pemisah (*delimiter*) serta *white space* (spasi, *tab,* dan *newline*) [8]. Pada penelitian ini, *tokenization* merupakan bagian dari *preprocessing* yang dilakukan setelah proses *splitting*.

2.3.4 *Stopword Removal*

Stopword removal merupakan proses penghilangan *stopword*. *Stopword* adalah kata - kata yang sering kali muncul dalam dokumen namun arti dari kata-kata tersebut tidak deskriptif dan tidak memiliki keterkaitan dengan tema tertentu. *Stopword* dapat berupa kata depan, kata penguhubung, dan kata pengganti [9]. Adapun kamus *stopword* yang digunakan terdapat pada lampiran daftar kamus *stopword*. Kamus *stopword* diambil dari rekomendasi Fadillah Z. Tala dalam penelitiannya yang berjudul "*A Study of Stemming Effects on Information Retrieval in Bahasa Indonesia*". Kamus tersebut tidak hanya digunakan untuk kasus *Information Retrieval* saja, tetapi juga digunakan dalam beberapa kasus peringkasan teks. Selain itu pada kamus tersebut lebih diutamakan untuk dokumen jenis berita, yang juga digunakan sebagai sumber data pada penelitian ini.

2.3.5 *Stemming*

Stemming adalah proses pengembalian kata berimbuhan menjadi kata dasar. Sebuah kata memiliki variasi kombinasi imbuhan kata yang beragam. Variasi imbuhan dapat berupa awalan (*prefix*), akhiran (*suffix*), sisipan (*infix*), dan

kombinasi antara awalan dan akhiran (*confix*) [9]. Proses *stemming* dapat mengurangi variasi kata yang sebenarnya memiliki kata dasar yang sama. Pada penelitian ini, *stemming* merupakan salah satu dari bagian *preprocessing*, yang dijalankan setelah *stopword removal* dilakukan. Salah satu algoritma yang digunakan dalam proses *stemming* untuk Bahasa Indonesia adalah algoritma Nazief-Adriani yang akan dijelaskan pada bagian terpisah.

2.4 Algoritma Nazief-Adriani

Algoritma Nazief-Adriani merupakan algoritma yang diciptakan oleh Bobby Nazief dan Mirna Adriani untuk melakukan pencarian kata dasar (*stemming*) pada kata-kata Bahasa Indonesia. Algoritma Nazief-Adriani ini sering digunakan karena diyakini memiliki tingkat keakuratan yang cukup tinggi dibandingkan dengan algoritma-algoritma *stemming* Bahasa Indonesia lainnya [10]. Algoritma ini menggunakan kamus kata dasar sebagai acuan algoritma ini bekerja. Adapun kamus kata dasar yang digunakan, *list-list* kata dasar diambil dari *website* kateglo.com, sebuah *website* yang memiliki data kamus, tesaurus, dan glosarium Bahasa Indonesia. Beberapa kamus kata dasar yang tersebar di *internet* pun bersumber dari *website* tersebut. Adapun jumlah kata dasar yang terdaftar sebanyak 28571 buah kata dasar.

Berikut ini adalah langkah-langkah *stemming* pada algoritma Nazief-Adriani [11]:

a. Di awal proses *stemming* dan setiap langkah yang selanjutnya dilakukan, lakukan pengecekan hasil proses *stemming* kata yang di-input-kan pada langkah tersebut ke kamus kata dasar. Jika kata ditemukan, berarti kata tersebut sudah berbentuk kata dasar dan proses *stemming* dihentikan. Jika tidak ditemukan, maka langkah selanjutnya dilakukan.
b. Hilangkan *Inflection Suffixes* (“-lah”, “-kah”, “-ku”, “-mu”, atau “-nya”). Jika berupa *particles* (“-lah”, “-kah”, “-tah” atau “-pun”) maka langkah ini diulangi lagi untuk menghapus *Possesive Pronouns* (“-ku”, “-mu”, atau “- nya”), jika ada.

c. Hapus *Derivation Suffixes* (“-i”, “-an” atau “-kan”). Jika kata ditemukan di kamus, maka algoritma berhenti. Jika tidak maka ke langkah 1)
    1) Jika “-an” telah dihapus dan huruf terakhir dari kata tersebut adalah “-k”, maka “-k” juga ikut dihapus. Jika kata tersebut ditemukan dalam kamus maka algoritma berhenti. Jika tidak ditemukan maka lakukan langkah 2).
    2) Akhiran yang dihapus (“-i”, “-an” atau “-kan”) dikembalikan, lanjut ke langkah d.
d. Hilangkan *derivation prefixes*.
    1) Langkah d. berhenti jika:
        a) Terjadi kombinasi awalan dan akhiran yang terlarang.
        b) Awalan yang dideteksi saat ini sama dengan awalan yang dihilangkan sebelumnya.
        c) Tiga awalan telah dihilangkan.
    2) Identifikasikan tipe awalan dan hilangkan. Awalan terdiri dari dua tipe :
        a) Standar (“di-”, “ke-”, “se-”) yang dapat langsung dihilangkan dari kata.
        b) Kompleks (“me-”, “be-”, “pe”, “te-”) adalah tipe-tipe awalan yang dapat bermorfologi sesuai kata dasar yang mengikutinya. Oleh karena itu, gunakan aturan pada Tabel 2.1 untuk mendapatkan hasil pemenggalan yang tepat.
e. Cari kata yang telah dihilangkan awalannya pada langkah d. 2) di dalam kamus kata dasar. Apabila tidak ditemukan, maka langkah d. diulangi kembali. Apabila ditemukan, maka keseluruhan proses dihentikan. Apabila setelah langkah 4 kata dasar masih belum ditemukan, maka proses *recoding* dilakukan dengan mengacu pada aturan pada Tabel 2.1. *Recoding* dilakukan dengan menambahkan karakter *recoding* di awal kata yang dipenggal. Pada Tabel 2.1, karakter *recoding* adalah huruf kecil setelah tanda hubung (“-‟) dan terkadang berada sebelum tanda kurung. Sebagai contoh, kata “menangkap” (aturan 15), setelah dipenggal menjadi “nangkap”. Karena tidak valid, maka recoding dilakukan dan menghasilkan kata “tangkap”.

f. Jika semua langkah gagal, maka input kata yang diuji pada algoritma ini dianggap sebagai kata dasar.

Tabel 2.1 Aturan Pemenggalan Algoritma Nazief-Adriani

Aturan	Format Kata	Pemenggalan
1	berV...	ber-V... | be-rV...
2	berCAP...	ber-CAP... dimana C!="r" & P!="er"
3	berCAerV...	ber-CaerV... dimana C!="r"
4	Belajar	bel-ajar
5	beC1erC2...	be-C1erC2... dimana C1!={"r"|"l"}
6	terV...	ter-V... | te-rV...
7	terCerV...	ter-CerV... dimana C!="r"
8	terCP…	ter-CP... dimana C!="r" dan P!="er"
9	teC1erC2...	te-C1erC2... dimana C1!="r"
10	me{l|r|w|y}V...	me-{l|r|w|y}V...
11	mem{b|f|v}...	mem-{b|f|v}...
12	mempe{r|l}...	mem-pe...
13	mem{rV|V}...	me-m{rV|V}... | me-p{rV|V}...
14	men{c|d|j|z}...	men-{c|d|j|z}...
15	menV...	me-nV... | me-tV
16	meng{g|h|q}...	meng-{g|h|q}...
17	mengV...	meng-V... | meng-kV...
18	menyV...	meny-sV…
19	mempV...	mem-pV... dimana V!=„e"
20	pe{w|y}V...	pe-{w|y}V...
21	perV...	per-V... | pe-rV...
22	perCAP	per-CAP... dimana C!="r"danP!="er"
23	perCAerV...	per-CAerV... dimana C!="r"
24	pem{b|f|V}...	pem-{b|f|V}...
25	pem{rV|V}...	pe-m{rV|V}... | pe-p{rV|V}...

Tabel 2.1 Aturan Pemenggalan Algoritma Nazief-Adriani (Lanjutan)

Aturan	Format Kata	Pemenggalan
26	pen{c|d|j|z}...	pen-{c|d|j|z}...
27	penV...	pe-nV... | pe-tV...
28	peng{g|h|q}...	peng-{g|h|q}...
29	pengV...	peng-V... | peng-kV...
30	penyV...	peny-sV…
31	pelV...	pe-lV... kecuali “pelajar” yang menghasilkan “ajar”
32	peCerV...	per-erV... dimana C!={r|w|y|l|m|n}
33	peCP...	pe-CP... dimana C!={r|w|y|l|m|n} dan P!=“er“

Keterangan simbol huruf:

C: huruf konsonan

V: huruf vokal

A: huruf vokal atau konsonan

P: partikel atau fragmen dari suatu kata, misalnya “er”

2.5 *Term Frequency-Inverse Document Frequency* (*tf-idf*)

Metode *Term Frequency-Inverse Document Frequency* (*tf-idf*) adalah cara pemberian bobot hubungan suatu kata (*term*) terhadap dokumen. Untuk dokumen tunggal tiap kalimat dianggap sebagai dokumen. Metode ini menggabungkan dua konsep untuk perhitungan bobot, yaitu *term frequency* (*tf*) merupakan frekuensi kemunculan *term j* pada dokumen *i*. *Document frequency* (*df*) adalah banyaknya kalimat dimana suatu *term j* muncul [6].

Frekuensi kemunculan *term* di dalam dokumen yang diberikan menunjukkan seberapa penting *term* itu di dalam dokumen tersebut. Frekuensi dokumen yang mengandung *term* tersebut menunjukkan seberapa umum *term* tersebut. Bobot *term* semakin besar jika sering muncul dalam suatu dokumen dan semakin kecil jika muncul dalam banyak dokumen [8]. Pada algoritma *tf-idf*

digunakan rumus untuk menghitung bobot *term* masing masing dokumen dengan rumus sebagai berikut [6].

$$tf.idf_{i,j} = tf_{i,j} \cdot \log_2 \frac{N}{n} \tag{2.1}$$

Dimana $tf_{i,j}$ merupakan frekuensi kemunculan *term j* dalam dokumen *i*; *N* merupakan jumlah keseluruhan dokumen; dan *n* merupakan jumlah dokumen dimana *term j* muncul.

Selain versi *tf-idf* di atas, terdapat pula versi *tf-idf* yang telah dinormalisasi. Versi *tf-idf* yang telah dinormalisasi ini sudah termasuk menghitung kesamaan (*similiarities*) dari dokumen. Adapun persamaan dari tf-idf yang telah dinormalisasi adalah sebagai berikut [6].

$$w_{i,j} = \frac{\mathrm{tf}_{i,j} \cdot \mathrm{idf}_j}{\| \vec{D}_i \|} = \frac{\mathrm{tf}_{i,j} \cdot \log_2 \frac{N}{n_j}}{\sqrt{\sum_{s=1}^{k} (\mathrm{tf}_{i,s} \cdot \log_2 \frac{N}{n_j})^2}} \tag{2.2}$$

Dimana $tf_{i,j}$ merupakan frekuensi kemunculan *term j* dalam dokumen *i*; dan n_j merupakan jumlah dokumen dimana *term j* muncul.

2.6 *Relevance Vector Machine*

Pada *supervised learning*, sekumpulan data latih yang terdiri dari himpunan vektor $\{x_n\}_{n=1}^{N}$ digunakan bersamaan dengan target yang sesuai $\{t_n\}_{n=1}^{N}$, dimana pada dasarnya target merupakan nilai sebenarnya dalam hal regresi dan label kelas pada proses klasifikasi. Tujuan dari *supervised learning* adalah membuat sebuah model dari target dengan masukan dari *training set*, sehingga prediksi yang akurat terhadap target t dapat dicapai untuk nilai x yang belum diketahui sebelumnya [12]. Secara umum prediksi tersebut dapat dihitung dengan menggunakan persamaan y(x), yaitu:

$$y(x; w) = \sum_{i=1}^{M} w_i \phi_i(x) + w_0 = \mathbf{w}^T \varphi(x) + w_0 \quad (2.3)$$

dimana $w = (w_1, w_2, \dots, w_m)$ merupakan vektor bobot (*weight vector*) dan $\phi_i(x)$ merupakan fungsi kernel terhadap data x, dan w_0 merupakan *bias*.

Proses belajar atau pelatihan pada dasarnya merupakan sebuah proses untuk menentukan parameter dari fungsi y(x), dalam konteks ini menentukan parameter bobot $\mathbf{w} = (w_1, w_2, \dots, w_m)$. Untuk himpunan dari pasangan data latih $\{x_n, t_n\}_{n=1}^{N}$, tugas dari proses pelatihan ini adalah untuk mencari nilai dari bobot $\mathbf{w} = (w_1, w_2, \dots, w_m)^T$, sehingga fungsi y(x) dapat digeneralisasi secara cukup baik terhadap data yang baru, dan bobot yang dihasilkan memiliki elemen nilai bukan nol (*nonzero*) yang sedikit. Memiliki beberapa elemen nilai yang bukan nol dapat menghasilkan representasi vektor yang *sparse*, dengan keuntungan dapat menyediakan implementasi yang cepat [12].

Relevance Vector Machine adalah sebuah metode pembelajaran mesin yang diperkenalkan oleh Mike Tipping pada tahun 2001, yang diadaptasi dari *Bayesian Framework* dan memiliki bentuk model fungsi yang mirip dengan *Support Vector Machine* [1]. Sama halnya dengan *Support Vector Machine*, Relevance Vector Machine digunakan dalam proses klasifikasi dan regresi. *Relevance Vector Machine* diperkenalkan untuk menutupi beberapa kelemahan yang dimiliki oleh *Support Vector Machine*. Kelemahan tersebut antara lain [1].

a. Walaupun *Support Vector Machine* menghasilkan vektor yang relatif *sparse*, namun jumlah *Support Vector* yang ada akan secara linear bertambah seiring dengan ukuran himpunan data latih.
b. Prediksi yang tidak bersifat probabilistik, dan *Support Vector Machine* tidak cocok untuk prose klasifikasi dimana peluang *posterior* dari keanggotaan kelas dianggap penting.
c. *Support Vector Machine* memerlukan proses estimasi parameter *error/margin* C, yang secara umum memerlukan prosedur *cross-validation* yang dapat memperlambat proses komputasi.

d. Fungsi *kernel* yang digunakan dalam *Support Vector Machine* harus memenuhi kondisi *Mercer*.

Pada klasifikasi untuk dua label kelas (*binary classification*), semua target dapat diklasifikasikan kedalam dua kelas, yang dapat direpresntasikan dengan nilai 0 dan 1, seperti $t_n \in \{0,1\}$. Distribusi *bernoulli* dapat diadopsi untuk peluang $\mathrm{p(t|x)}$ dalam *framework* probabilistik karena hanya dua nilai kelas yang mungkin, yaitu 0 dan 1 [12]. Dalam kasus klasifikasi, model prediksi mengambil bentuk kombinasi linear dari fungsi basis/kernel yang diubah oleh fungsi *logistic sigmoid* [13].

$$\mathrm{y}(\mathbf{x}, \mathbf{w}) = \sigma(\mathrm{x; w}) = \sigma(\mathbf{w}^{\mathrm{T}}\varphi(\mathrm{x})) \tag{2.4}$$

Dimana $\sigma(.)$ Merupakan fungsi *logistic sigmoid* yang didefiniskan dengan persamaaan

$$\sigma(\mathrm{y}) = \frac{1}{1 + \exp(-\mathrm{y})} \tag{2.5}$$

Berdasarkan definisi dari distribusi *Bernoulli*, *likelihood* terdefinisikan sebagai berikut

$$\mathrm{p(t|w)} = \prod_{\mathrm{t}=1}^{\mathrm{N}} \sigma\{\mathrm{y}(\mathrm{x_n}; \mathbf{w})\}^{\mathrm{t_n}}[1 - \sigma\{\mathrm{y}(\mathrm{x_n}; \mathbf{w})\}^{1-\mathrm{t_n}} \tag{2.6}$$

untuk target $t_n \in \{0,1\}$.

Persamaan *likelihood* dilengkapi dengan sebuah *prior* terhadap parameter (bobot) dalam bentuk

$$\mathrm{p}(\mathbf{w}|\boldsymbol{\alpha}) = \prod_{\mathrm{t}=1}^{\mathrm{N}} \frac{\sqrt{\alpha_\mathrm{i}}}{\sqrt{2\pi}} \exp\left(-\frac{\alpha_\mathrm{i} {w_\mathrm{i}}^2}{2}\right) \tag{2.7}$$

dimana $\alpha = (\alpha_1, \alpha_2, \dots, \alpha_n)^T$ merupakan *hyperparameter* yang diperkenalkan untuk mengontrol kekuatan dari *prior* terhadap parameter bobot yang diasosiasikannya.

Untuk nilai α tertentu, distribusi *posterior* bobot terhadap data dapat dihitung menggunakan aturan *Bayes*, dengan persamaan

$$\mathrm{p}(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) = \frac{\mathrm{p}(\mathbf{t}|\mathbf{w})\mathrm{p}(\mathbf{w}|\boldsymbol{\alpha})}{\mathrm{p}(\mathbf{t}|\boldsymbol{\alpha})} \tag{2.8}$$

dimana $\mathrm{p}(\mathbf{t}|\mathbf{w})$ adalah *likelihood*, $\mathrm{p}(\mathbf{w}|\boldsymbol{\alpha})$ adalah *prior*, dan $\mathrm{p}(\mathbf{t}|\boldsymbol{\alpha})$ adalah *evidence*.

Parameter bobot model tidak dapat diperoleh dengan cara analitik, sehingga aproksimasi *Laplacian* digunakan. Sejak $\mathrm{p}(\mathbf{w}|\mathbf{t},\boldsymbol{\alpha})$ secara linear proporsional untuk $\mathrm{p}(\mathbf{t}|\mathbf{w})$ x $\mathrm{p}(\mathbf{w}|\boldsymbol{\alpha})$, dapat dimungkinkan untuk mencari maksimum dari persamaan

$$\begin{aligned} \ln \mathrm{p}(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) &= \ln \{\mathrm{p}(\mathrm{t}|\mathrm{w})\mathrm{p}(\mathrm{w}|\alpha)\} + \ln \mathrm{p}(\mathbf{t}|\alpha) \\ &= \sum_{\mathrm{n}=1}^{\mathrm{N}} \{\mathrm{t_n} \ln \mathrm{y_n} + (1 - \mathrm{t_n}) \ln(1 - \mathrm{y_n})\} + \frac{1}{2}\mathrm{w^T A w} \end{aligned} \tag{2.9}$$

untuk parameter bobot yang paling mungkin $\mathbf{w}'$, dengan $\mathrm{y_n} = \sigma\{y(x_n; \boldsymbol{w})\}^{t_n}$ dan $\mathbf{A}$ = diag $(\alpha_1, \alpha_2, \dots, \alpha_n)$ yang terdiri dari nilai α yang diinisialisasi. Untuk mencari parameter $\mathbf{w}'$ pada proses reestimasi, metode *Iteratively Reweight Least-Square* dapat digunakan.

Fungsi *logistic log-likelihood* dapat diturunkan sebanyak dua kali untuk memperoleh matriks *Hessian* dengan persamaan

$$\mathbf{H} = \nabla\nabla \ln \mathrm{p}(\mathbf{w}|\mathbf{t}, \boldsymbol{\alpha}) = \boldsymbol{\varphi}_{\mathrm{n}}^{\mathrm{T}} \mathbf{B} \boldsymbol{\varphi}_{\mathrm{n}} + \mathbf{A} \tag{2.10}$$

dimana $\boldsymbol{\varphi}$ merupakan matriks kernel, dan $\mathbf{B} = \mathrm{diag}(\beta_1, \beta_2, \dots, \beta_n)$ merupakan matriks diagonal yang elemen matriksnya diperoleh dengan persamaan

$$\beta_n = \sigma\{y(x_n; \boldsymbol{w}')\}[1 - \sigma\{y(x_n; \boldsymbol{w}')\} \tag{2.11}$$

dimana **w'** merupakan parameter bobot yang akan direestimasi. Proses reestimasi parameter bobot **w** didapat dengan persamaan

$$\mathbf{w}' = \mathbf{w}_{\mathrm{n}} + \lambda\Delta \tag{2.12}$$

dimana λ merupakan koefisien estimasi, dan Δ merupakan vektor yang diperoleh dari persamaan

$$\Delta = \mathbf{H}^{-1}.\mathbf{g} \tag{2.13}$$

dimana $\mathbf{H}^{-1}$ adalah invers dari matriks hessian dan **g** adalah gradien negatif yang diperoleh dari penurunan fungsi *logistic log-likelihood* sebanyak satu kali dengan persamaan

$$\mathbf{g} = -\nabla \ln p\,(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\alpha}) = \varphi_{\mathrm{n}}^{\mathrm{T}}\mathrm{e} - \mathbf{A}\mathrm{w}_{\mathrm{n}} \tag{2.14}$$

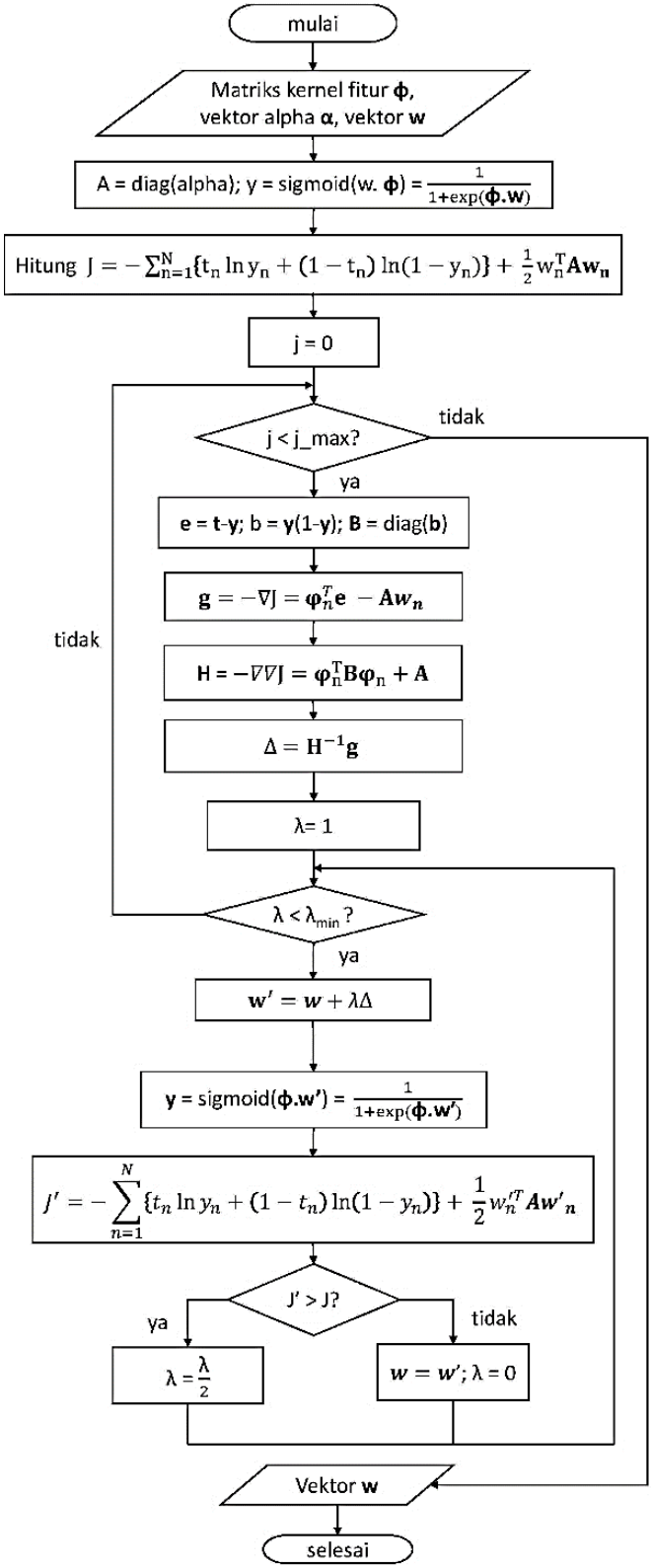
Penjelasan proses estimasi parameter bobot **w** secara rinci dalam bentuk diagram *flowchart* dapat dilihat pada Gambar 2.2.

Setelah estimasi parameter bobot **w** dilakukan, selanjutnya *hyperparameter* α_{j} dilakukan reestimasi dengan menggunakan persamaan

$$\alpha_{\mathrm{j}}{}^{new} = \frac{\gamma_{\mathrm{j}}}{\mathrm{w}_{\mathrm{j}}^2} \tag{2.15}$$

dimana γ_j didefinisikan dengan persamaan

$$\gamma_j = 1 - \alpha_{\mathrm{j}}\Sigma_{\mathrm{jj}} \tag{2.16}$$



Gambar 2.2 *Flowchart* Estimasi Parameter Bobot w

dimana Σ_{jj} merupakan elemen diagonal matriks kovarians yang didapat dari persamaan

$$\boldsymbol{\Sigma} = -\left(\nabla\nabla \ln p\left(\boldsymbol{w}|\boldsymbol{t}, \boldsymbol{\alpha}\right)\right)^{-1} = \mathbf{H}^{-1} \tag{2.17}$$

dimana $\mathbf{H}^{-1}$ merupakan invers dari matriks *Hessian*. Selain dengan cara menggunakan persamaan (2.15), reestimasi *hyperparameter* α_j juga dapat dilakukan dengan cara "*hybrid update*" [14] dengan menggunakan persamaan

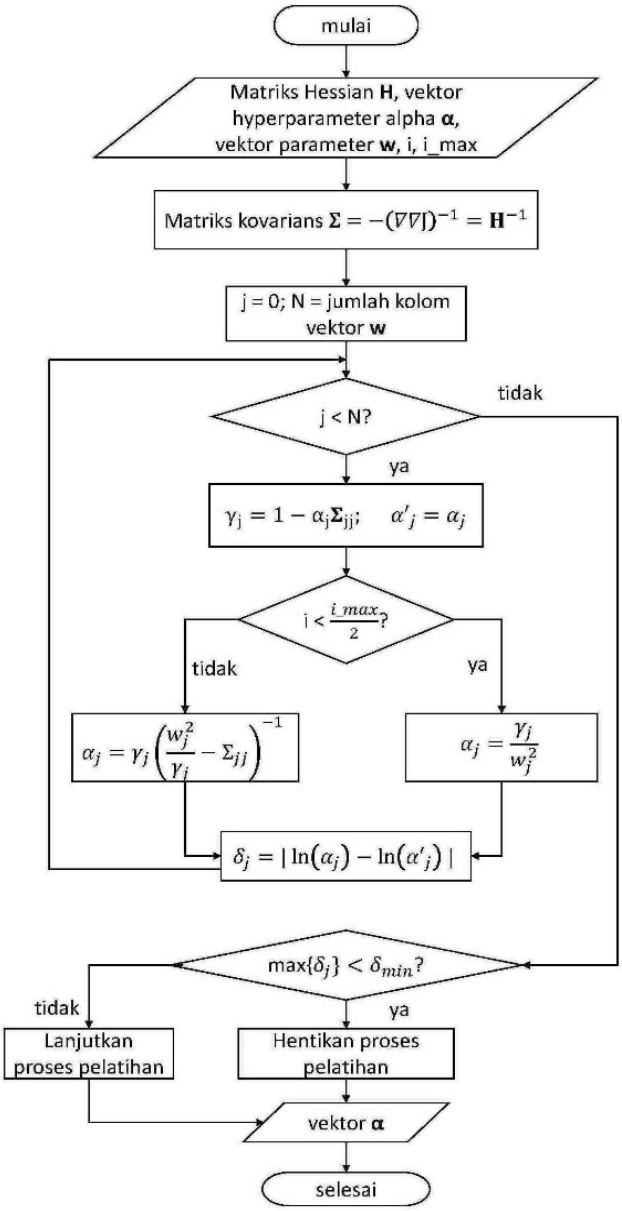
$$\begin{aligned} \alpha_j &= \frac{\gamma_j}{w_j^2}; \text{jika } i < \frac{i_{max}}{2} \\ \alpha_j &= \gamma_j \left(\frac{w_j^2}{\gamma_j} - \Sigma_{jj}\right)^{-1}; \text{jika } i \geq \frac{i_{max}}{2} \end{aligned} \tag{2.18}$$

dimana i_{max} merupakan jumlah iterasi maksimum pada pelatihan.
Penjelasan proses estimasi parameter *hyperparameter alpha* secara rinci dalam bentuk *flowchart* dapat dilihat pada Gambar 2.3.

Pada saat proses optimasi, banyak *hyperparameter* α_j yang akan memiliki nilai yang sangat besar, dan parameter bobot model yang berkoresponden dengan *hyperparameter* tersebut akan dipangkas (*pruned*), karena telah dalam keadaan *sparsity*. Proses optimasi terus berlanjut hingga perubahan maksimum dari α_j dibawah koefisien tertentu atau jumlah maksimum iterasi telah dicapai.

2.7 Evaluasi Ringkasan

Evaluasi ringkasan merupakan cara untuk mengetahui apakah ringkasan yang dihasilkan baik oleh peringkasan secara manual ataupun peringkasan teks otomatis memiliki tingkat akurasi yang baik atau tidak. Pada dasarnya mengevaluasi sebuah ringkasan dapat dikatakan sebuah kegiatan yang sangat sulit, karena mengevaluasi sebuah ringkasan merupakan sebuah kegiatan yang subjektif.



Gambar 2.3 *Flowchart* Estimasi *Hyperparameter Alpha*

Sebuah ringkasan menurut sebagian orang sudah tepat, tetapi menurut sebagian orang lagi tidak demikian [6].

Secara umum terdapat dua jenis metode evaluasi ringkasan, yaitu [6]:

a. Ekstrinsik yaitu kualitas ringkasan diukur berdasarkan bagaimana ringkasan tersebut dapat membantu penyelesaian tugas orang yang membaca ringkasan tersebut.

b. Intrinstik yaitu hanya diukur dari kualitas hasil keluaran (*output*) ringkasan yang dihasilkan.

Evaluasi sistem peringkas yang akan digunakan dalam penelitian ini adalah evaluasi intrinsik. Pengevaluasi menciptakan sebuah ringkasan yang manual, untuk menguji teks. Kemudian membandingkan hasil ringkasan sistem dengan ringkasan ideal. Evaluasi yang diukur adalah *overlap* dari isi, seringkali disebut dengan *recall* dan *precision* kalimat. Pengukuran *precision* dan *recall* ini sangat dipengaruhi oleh panjang ringkasan manual dan juga panjang ringkasan yang dievaluasi. Akurasi menurun sejalan dengan bertambahnya panjang ringkasan. Sulit untuk mengambil kesimpulan terhadap *performance* sistem dari nilai *precision* dan *recall*. Untuk standarisasi proses evaluasi ringkasan sama sekali belum dieksplorasi. Masalah utama dari evaluasi ini sangat nyata, yaitu tidak ada satupun ringkasan yang benar [6]. Kombinasi antara nilai *recall* dan *precision* menghasilkan *f-measure*.

2.7.1 *Recall*

Recall (r) merupakan istilah yang digunakan untuk kalimat terpanggil yang relevan dengan pernyataan atau vektor *query*. Perbandingan jumlah informasi relevan yang didapatkan sistem dengan jumlah seluruh informasi relevan yang ada dalam koleksi informasi. *Recall* berhubungan dengan kemampuan dalam menemukan kalimat yang relevan [15]. Hal ini berarti *recall* adalah bagian dari proses evaluasi yang dapat digunakan sebagai alat ukur relevannya sebuah ringkasan. Adapun rumus untuk menghitung recall dapat dilihat pada persamaan berikut.

$$recall = \frac{\sum Kalimat\ Relevan}{\sum Kalimat\ Ringkasan\ Manual} \tag{2.19}$$

Dimana ΣKalimat Relevan merupakan jumlah kalimat ringkasan yang terdapat pada sistem dan juga terdapat pada ringkasan manual, dan ΣKalimat Ringkasan Manual merupakan jumlah kalimat ringkasan yang dihasilkan secara manual.

2.7.2 *Precision*

Precision (p) biasanya menjadi salah satu ukuran yang digunakan untuk menilai efektivitas kalimat yang di hasilkan dalam sebuah rangkuman. *Precision* adalah jumlah kalimat relevan dari total jumlah kalimat yang ditemukan oleh sistem [15]. Perbandingan jumlah informasi relevan yang didapatkan sistem dengan jumlah seluruh informasi yang terambil oleh sistem baik yang relevan maupun tidak. Adapun, pengukuran tingkat ketepatan (*precision*) dirumuskan pada persamaan berikut.

$$precision = \frac{\sum Kalimat\ Relevan}{\sum Kalimat\ Ringkasan\ Sistem} \tag{2.20}$$

Dimana ΣKalimat Relevan merupakan jumlah kalimat ringkasan yang terdapat pada sistem dan juga terdapat pada ringkasan manual, dan ΣKalimat Ringkasan Sistem merupakan jumlah kalimat ringkasan yang dihasilkan oleh sistem.

2.7.3 *F-Measure*

F-Measure (F_1) merupakan salah satu perhitungan evaluasi yang mengkombinasikan *recall* dan *precision*. Hubungan antara *recall* dan *precision* yang mempresentasikan akurasi sistem. Nilai *recall* dan *precision* pada suatu keadaan dapat memiliki bobot yang berbeda. Ukuran yang menampilkan timbal balik antara *recall* dan *precision* adalah *f-measure* yang merupakan bobot *harmonic mean* dari *recall* dan *precision* [15]. Perhitungan *f-measure* dapat dirumuskan pada persamaan berikut.

$$F - measure = \frac{2 * precision * recall}{precision + recall} \tag{2.21}$$

Dimana parameter *precision* merupakan hasil dari perhitungan *precision* dan parameter *recall* merupakan hasil dari perhitungan *recall.*

2.8 *Unified Model Language* (*UML*)

Unified Modeling Language (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (*OOP*) serta aplikasinya. [16]. *UML* adalah metodologi untuk mengembangkan sistem *OOP* dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar *OOP* sejak tahun 1980-an. Sekarang *UML* sudah mulai banyak digunakan oleh para praktisi *OOP*. *UML* merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari *IBM*.

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. *UML* dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian *UML* dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan *UML* dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri perangkat lunak dan pengembangan sistem.

UML menyediakan beberapa macam diagram untuk memodelkan aplikasi berorientasi objek. Adapun beberapa diagram yang sering digunakan adalah *Use Case, Activity Diagram, Sequence Diagram,* serta *Class Diagram*.

2.8.1 *Use Case Diagram*

Use case diagram digunakan untuk memodelkan proses bisnis berdasarkan perspektif pengguna sistem [16] . *Use case diagram* terdiri atas diagram untuk *use case* dan *actor*. *Actor* merepresentasikan orang yang akan mengoperasikan atau orang yang berinteraksi dengan sistem aplikasi.

Use case merepresentasikan operasi-operasi yang dilakukan oleh *actor*. *Use case* digambarkan berbentuk elips dengan nama operasi dituliskan di dalamnya. *Actor* yang melakukan operasi dihubungkan dengan garis lurus ke *use case*.

2.8.2 *Activity Diagram*

Activity diagram adalah representasi grafis dari seluruh tahapan alur kerja. Diagram ini mengandung aktivitas, pilihan tindakan, perulangan dan hasil dari aktivitas tersebut. Pada pemodelan UML, diagram ini dapat digunakan untuk menjelaskan proses bisnis dan alur kerja operasional secara langkah demi langkah dari komponen suatu sistem.

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir [16]. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. *Activity diagram* merupakan state diagram khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour internal* sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. Struktur diagram ini mirip *flowchart* atau *Data Flow Diagram* (*DFD*) pada perancangan terstruktur. Sangat bermanfaat apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan. *Activity diagram* dibuat berdasarkan sebuah atau beberapa *use case* pada *use case diagram.*

2.8.3 *Sequence Diagram*

Diagram sequence merupakan salah satu diagram interaksi (*interaction diagram*) yang menjelaskan bagaimana suatu operasi itu dilakukan dan *message* (pesan) apa yang dikirim dan kapan pelaksanaannya [16]. Diagram ini diatur berdasarkan waktu. Obyek-obyek yang berkaitan dengan proses berjalannya operasi diurutkan dari kiri ke kanan berdasarkan waktu terjadinya dalam pesan yang terurut.

2.8.4 *Class Diagram*

Class diagram menggambarkan struktur statis *class* di dalam sistem. *Class* merepresentasikan sesuatu yang ditangani oleh sistem. *Class* dapat berhubungan dengan yang lain melalui berbagai cara: *associated* (terhubung satu sama lain), *dependent* (satu *class* tergantung/menggunakan *class* yang lain), *specialed* (satu *class* merupakan spesialisasi dari class lainnya), atau *package* (*group* bersama sebagai satu unit) [16]. Sebuah sistem biasanya mempunyai beberapa *class diagram*.

2.9 Pengujian *Black Box*

Pengujian *black box* disebut juga dengan pengujian perilaku, merupakan metode pengujian perangkat lunak yang berfokus pada persyaratan fungsional perangkat lunak. Pengujian *black box* berupaya untuk menemukan kesalahan dalam kategori seperti berikut [17]:

1. Fungsi yang salah atau hilang.
2. Kesalahan antarmuka.
3. Kesalahan dalam struktur data atau akses basis data eksternal.
4. Kesalahan perilaku atau kinerja.
5. Kesalahan inisialisasi dan penghentian.