BAB 2
LANDASAN TEORI

2.1 Tinjauan Teori

Penelitian ini didasarkan pada pemahaman teori-teori dasar yang relevan dengan topik utama, yaitu klasifikasi stadium kanker payudara menggunakan data gen dan miRNA. Tinjauan ini mencakup penjelasan mengenai kanker payudara, data ekspresi gen dan miRNA, serta metode-metode analisis data dan algoritma pembelajaran mesin yang digunakan dalam penelitian. Pemahaman terhadap aspek-aspek ini penting untuk menjelaskan dasar ilmiah dari pendekatan yang digunakan dalam studi ini.

2.1.1 Breast Cancer

Kanker payudara disebabkan oleh adanya sel kanker yang bersifat ganas di dalam payudara. Sel-sel kanker ini ditandai dengan pembelahan yang tidak terkendali, yang mengarah pada pertumbuhan abnormal (karsinoma in situ) dan kemampuan untuk menyerang jaringan normal di sekitarnya (kanker invasif). Jenis kanker payudara dapat diklasifikasikan berdasarkan apakah unit kelenjar atau saluran di payudara yang terlibat. Tumor primer dimulai di payudara, tetapi begitu menjadi invasif, dapat menyebar ke jaringan regional di sekitarnya [21].

2.1.2 Breast Cancer Staging

Sistem staging *American Joint Committee on Cancer* (AJCC) didasarkan pada *staging* anatomi, yang mengacu pada tumor primer (T), status kelenjar getah bening regional (N), dan status metastasis (M). *Staging* T mengkategorikan tumor berdasarkan ukuran dan derajat invasi lokal, dari T1 hingga T4. *Staging* N menilai sejauh mana keterlibatan kelenjar getah bening, termasuk aksila, internal *mammary*, dan kelenjar getah bening *supraklavikula ipsilateral*. *Staging* M menilai adanya metastasis jauh dapat dilihat pada Tabel 2.1 [22, 23].

Sistem TNM digunakan untuk menilai sejauh mana kanker telah menyebar di dalam tubuh, dengan mempertimbangkan tiga faktor utama: ukuran tumor, kondisi kelenjar getah bening, dan apakah ada metastasis. Berdasarkan gabungan dari ketiga faktor ini, kanker dikategorikan dalam stadium I-IV, dengan stadium

Tabel 2.1. Klasifikasi Stadium Kanker Payudara berdasarkan Edisi ke-8 dari Sistem Staging *American Joint Committee on Cancer* (AJCC)

Stage	Tumor	Node	Metastasis
0	Tis	N0	M0
IA	T1	N0	M0
IB	T0	N1mi	M0
	T1	N1mi	M0
IIA	T0	N1	M0
	T1	N1	M0
	T2	N0	M0
IIB	T2	N1	M0
	T3	N0	M0
IIIA	T0	N2	M0
	T1	N2	M0
	T2	N2	M0
	T3	N1	M0
	T3	N2	M0
IIIB	T4	N0	M0
	T4	N1	M0
	T4	N2	M0
IIIC	AnyT	N3	M0
IV	AnyT	AnyN	M1

Sumber: [23]

IV sebagai yang paling parah. Stadium 0 mengacu pada *carcinoma* in situ, yaitu kondisi yang belum dianggap kanker tetapi bisa berkembang menjadi kanker di masa depan. Stadium V khusus digunakan untuk tumor *Wilms*, yaitu jenis tumor yang terjadi ketika kedua ginjal terlibat pada saat diagnosis pertama kali.

- **Stage 0** - Menunjukkan carcinoma in situ.
- **Stage I** - Kanker yang terlokalisasi.
- **Stage II** - Kanker yang lokal dan berkembang, tahap awal .
- **Stage III** - Kanker yang lokal dan berkembang, tahap lanjut.
- **Stage IV** - Kanker metastatik.

Staging kanker memiliki ranking terkait keparahan penyakit dan penurunan tingkat kelangsungan hidup [20].

2.2 Dataset

Penelitian ini menggunakan data ekspresi genetik yang diperoleh dari basis data The Cancer Genome Atlas (TCGA) untuk kasus kanker payudara (BRCA). Dataset ini terdiri dari dua jenis data utama, yaitu ekspresi gen dan ekspresi miRNA. Kedua jenis data tersebut diproses dan dianalisis secara terpisah untuk mengevaluasi kontribusi masing-masing terhadap akurasi klasifikasi stadium kanker payudara. Deskripsi detail dari masing-masing dataset disajikan pada subbagian berikut.

2.2.1 Gene Expression RNAseq - STAR - FPKM-UQ

RNA-Seq saat ini menjadi pendekatan yang semakin populer dalam analisis transkriptom karena menawarkan berbagai keunggulan dibandingkan metode lama seperti *mikroarray*. Dengan beragam protokol persiapan, RNA-Seq memungkinkan pemilihan jenis *Ribonucleic Acid* (RNA) tertentu, misalnya RNA polyA+ atau RNA yang sedang aktif ditranskripsi. Dibandingkan *mikroarray*, RNA-Seq memiliki sensitivitas lebih tinggi, cakupan deteksi yang lebih luas, serta kemampuan untuk mengidentifikasi *splicing* alternatif dan transkrip baru hingga ke tingkat nukleotida. Seiring berkembangnya teknologi *sequencing* dan kemudahan dalam analisis data, RNA-Seq diperkirakan akan menggantikan metode berbasis hibridisasi. Proses ini menghasilkan data ekspresi gen dengan menghitung jumlah RNA yang terdeteksi dalam sampel [24].

FPKM-UQ adalah metode untuk menormalkan data ekspresi gen dari RNA-Seq yang merupakan pengembangan dari metode FPKM standar. Cara kerjanya adalah dengan menghitung jumlah pembacaan (reads) yang cocok dengan suatu gen, lalu menyesuaikannya berdasarkan panjang gen dan banyaknya pembacaan pada gen dengan tingkat ekspresi di persentil ke-75. Dengan pendekatan ini, hasil normalisasi jadi lebih seimbang, terutama saat membandingkan ekspresi gen antarsampel.

Rumus:

$$\text{FPKM-UQ} = \frac{RM_g \times 10^9}{RM_{75} \times L} \tag{2.1}$$

Keterangan:

- RM_g : Jumlah reads yang dipetakan ke gen tertentu
- RM_{75} : Jumlah reads pada gen persentil ke-75 (upper quartile)

- L : Panjang gen (dalam satuan pasangan basa)
- 10^9 : Faktor skala untuk menyesuaikan ke satuan kilobase dan juta reads

FPKM-UQ biasanya menghasilkan nilai yang lebih tinggi dibandingkan FPKM standar karena proses normalisasi menggunakan jumlah pembacaan pada satu gen di persentil ke-75, bukan total semua pembacaan, sehingga nilai pembaginya lebih kecil [25].

Dalam upaya menemukan biomarker dari data RNA-Seq, seperti gen yang menunjukkan perbedaan signifikan dalam tingkat ekspresinya (DEG), pendekatan statistik yang umum digunakan seperti uji-*t* dua sampel bisa menjadi kurang andal, terutama saat data mengandung *outlier*. Rata-rata dan varians yang digunakan dalam metode klasik sangat rentan terhadap nilai-nilai ekstrem ini, yang bisa menyebabkan hasil yang menyesatkan. Untuk mengatasinya, digunakan pendekatan statistik yang lebih *robust*, yakni dengan mengganti perhitungan rata-rata dan varians standar dengan metode berbasis *minimum* β*-divergence*. Pendekatan ini memberikan bobot khusus pada data—melalui parameter β—sehingga pengaruh *outlier* dapat ditekan. Prosesnya dilakukan secara iteratif hingga diperoleh estimasi yang stabil dan lebih akurat. Setelah melalui pengujian, nilai β yang paling optimal ditemukan pada 0,2, berdasarkan hasil *cross-validation*. Pendekatan ini membantu meningkatkan keandalan dalam mendeteksi gen-gen potensial sebagai biomarker [26].

2.2.2 Stem Loop Expression - miRNA Expression Quantification

MicroRNA (miRNA) adalah molekul RNA non-koding rantai tunggal sepanjang 22 nukleotida yang berperan penting dalam berbagai proses biologis seperti perkembangan jaringan, proliferasi, dan diferensiasi, serta dikaitkan dengan berbagai jenis kanker. Analisis ekspresi miRNA matang memerlukan metode khusus untuk dideteksi, salah satunya menggunakan primer *stem-loop*. Primer *stem-loop* menggunakan primer spesifik miRNA dalam transkripsi balik dan amplifikasi. Dengan menggunakan metode *stem-loop* terdapat berbagai macam keuntungan seperti, spesifik terhadap data miRNA matang, meningkatkan efisiensi dan sensitivitas deteksi. Metode *stem-loop* menggunakan primer reverse transkripsi khusus yang berbentuk *stem-loop* dan dirancang spesifik untuk masing-masing miRNA. Struktur primer ini memungkinkan pengikatan yang lebih spesifik dan efisien terhadap miRNA matang. Setelah proses transkripsi balik, amplifikasi

dilakukan menggunakan satu primer yang spesifik terhadap miRNA target dan satu primer lain yang spesifik terhadap bagian dari sekuens *stem-loop* [27].

Metode ini digunakan dalam *quantitative Polymerase Chain Reaction* qPCR yang kerap dijadikan standar validasi hasil *profiling* miRNA karena keunggulan deteksinya. Dari 86 miRNA yang diuji, sebanyak 68 miRNA dianalisis oleh keempat platform dan memiliki nilai CT di bawah 36. Analisis korelasi antara *fold change* dari masing-masing platform dengan hasil qPCR menunjukkan bahwa NGS memiliki kesesuaian tertinggi dengan qPCR (r = 0,86), diikuti oleh *NanoString* (r = 0,72), yang keduanya signifikan secara statistik. Temuan ini menegaskan bahwa NGS merupakan metode paling akurat dalam mendeteksi perubahan ekspresi miRNA dibandingkan platform lainnya [28].

MiRNA dapat menjadi biomarker utama, hal ini dapat dilihat dari penelitian oleh Ji Young Jang. Satu set biomarker yang dikembangkan dalam studi ini menunjukkan akurasi tinggi untuk diagnosis dini kanker payudara. Biomarker ini berasal dari kombinasi dua atau lebih dari sembilan miRNA dalam sampel plasma dan berpotensi menjadi indikator diagnostik yang efektif untuk mendeteksi kanker payudara pada tahap awal [29].

2.3 Differential Expression Gene (LIMMA)

Analisis DEG merupakan teknik yang digunakan dalam biologi molekuler dalam membandingkan tingkat ekspresi gen dua atau lebih kelompok sampel. Sampel dapat berupa jaringan normal atau jaringan tidak normal. Analisis DEG bertujuan mengidentifikasi gen ekspresi berbeda dalam berbagai perbandingan [30]. Limma adalah salah satu paket utama dalam *Bioconductor*, sebuah platform *open-source* berbasis R yang dirancang untuk analisis statistik data genomik. Limma sangat berguna dalam menganalisis data RNA-seq, khususnya dalam identifikasi *Differentially Expressed Genes* (DEG).

Paket limma mencakup berbagai metode statistik yang:

(i) Memfasilitasi peminjaman informasi melalui metode Bayes empiris untuk menghasilkan estimasi variansi posterior (s_g^2),

(ii) Mengintegrasikan bobot pengamatan (w_{gj}, dengan j merujuk pada sampel) untuk mengakomodasi variasi dalam kualitas data,

(iii) Memungkinkan pemodelan variansi untuk menangani heterogenitas teknis atau biologis yang mungkin ada,

(iv) Menyediakan metode pra-pemrosesan, seperti penstabilan variansi, untuk mengurangi kebisingan (noise).

Metode-metode ini berfungsi untuk meningkatkan kualitas inferensi, baik pada tingkat gen individual maupun set gen [31]. Analisis DEG menggunakan paket LIMMA dilakukan pada data ekspresi RNA-seq. Paket ini menerapkan model linear untuk setiap gen dan memanfaatkan pendekatan empirical Bayes guna meningkatkan estimasi variansi [32]. Dalam penelitian ini, data ekspresi dikaitkan dengan status stadium kanker payudara, dengan mempertimbangkan variabel tambahan seperti ras sebagai kovariat dalam desain model. Proses ini menghasilkan daftar gen yang berbeda secara signifikan antara stadium awal dan stadium lanjut.

2.4 Seleksi Fitur

Seleksi fitur (*feature selection*) merupakan strategi praproses data dalam *machine learning*, hal ini bertujuan memilih *subset* fitur relevan dari data asli dalam mengurangi dimensi. *Feature selection* bertujuan dalam meningkatkan akurasi model prediktif, mengurangi biaya komputasi, dan memperbaiki *interpretabilitas* model dengan menyajikan data yang lebih bersih dan mudah dipahami [33].

2.4.1 Regularisasi L1 (Lasso) dengan Logistic Regression

Logistic regression merupakan salah satu metode statistik yang sering digunakan untuk menganalisis data dan membuat prediksi, terutama di bidang seperti fisika, ekonomi, kesehatan, dan ilmu sosial. Dalam konteks klasifikasi dua kelas, seperti memprediksi apakah pasien berada pada tahap awal atau lanjut kanker, metode ini membantu memperkirakan kemungkinan dari masing-masing kondisi berdasarkan informasi atau fitur yang tersedia [34]. Metode *logistic regression* menjadi *feature selection* dalam penelitian ini, hal ini tentu untuk menyaring fitur-fitur dengan bobot yang tinggi. Dalam implementasinya, regresi logistik menghitung kombinasi linier dari fitur-fitur yang tersedia. Setiap fitur dikalikan dengan bobot tertentu yang menunjukkan tingkat pengaruhnya terhadap prediksi [35]. Hasil dari kombinasi linier ini disebut sebagai skor z, yang dirumuskan sebagai berikut:

Rumus:

$$z = \mathbf{w} \cdot \mathbf{x} + b \tag{2.2}$$

Keterangan:

- **w** : vektor bobot (*weight vector*) yang merepresentasikan kontribusi tiap fitur
- **x** : vektor fitur (*input feature vector*) dari data yang diamati
- b : bias atau *intercept*, yaitu konstanta yang menggeser fungsi aktivasi
- z : skor linear hasil kombinasi antara fitur dan bobot, yang nantinya digunakan dalam fungsi aktivasi *sigmoid*

Nilai skor z tersebut belum merepresentasikan probabilitas secara langsung karena masih berada pada rentang $(-\infty, +\infty)$. Oleh karena itu, skor ini kemudian diubah ke dalam bentuk probabilitas menggunakan fungsi aktivasi *sigmoid*. Fungsi ini memetakan nilai z ke rentang $(0,1)$, sehingga dapat diinterpretasikan sebagai probabilitas bahwa suatu sampel termasuk ke dalam kelas positif ($y = 1$). Secara matematis, hal ini dituliskan sebagai berikut:

Rumus:

$$p(y = 1 \mid \mathbf{x}; \boldsymbol{\theta}) = \sigma(\boldsymbol{\theta}^\top \mathbf{x}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^\top \mathbf{x})} \tag{2.3}$$

Keterangan:

- y: Label atau kelas target, dengan $y \in \{0,1\}$.
- **x**: Vektor fitur *input*, berukuran $n \times 1$.
- $\boldsymbol{\theta}$: Vektor parameter (koefisien model), berukuran $n \times 1$.
- $\boldsymbol{\theta}^\top \mathbf{x}$: Perkalian dot *product* antara parameter dan fitur, yaitu kombinasi linear dari fitur.
- $\sigma(z)$: Fungsi *sigmoid*, didefinisikan sebagai:

$$\sigma(z) = \frac{1}{1 + \exp(-z)}$$

- $\exp(\cdot)$: Fungsi eksponensial, yaitu $e^{(\cdot)}$, dengan $e \approx 2{,}718$.
- $p(y = 1 \mid \mathbf{x}; \boldsymbol{\theta})$: Probabilitas bahwa kelas y bernilai 1, diberikan *input* **x** dan parameter $\boldsymbol{\theta}$.

Tujuan dalam algoritma ini adalah mencari parameter $\boldsymbol{\theta}$ untuk menghasilkan prediksi terbaik. Namun jika model terlalu kompleks, ini dapat menyebabkan *overfitting*. Solusi untuk masalah ini adalah dengan menambahkan regularisasi. Dengan L1 *Regularization* (*Lasso*) perlu menambahkan penalti $\|\boldsymbol{\theta}\|_1$, yaitu jumlah nilai absolut dari seluruh elemen dalam $\boldsymbol{\theta}$. Ini mendorong model untuk memilih hanya fitur yang paling relevan, karena banyak elemen dalam $\boldsymbol{\theta}$ akan bernilai nol [36].

Rumus:

$$\min_{\boldsymbol{\theta}} \sum_{i=1}^{M} -\log p(y^{(i)} \mid \mathbf{x}^{(i)}; \boldsymbol{\theta}) + \beta \|\boldsymbol{\theta}\|_1 \tag{2.4}$$

Keterangan:

- $\min_{\boldsymbol{\theta}}$: Tujuan optimisasi, mencari parameter $\boldsymbol{\theta}$ yang meminimalkan fungsi keseluruhan.
- $\boldsymbol{\theta}$: Vektor parameter model regresi logistik.
- M: Jumlah total sampel dalam dataset.
- $y^{(i)}$: Label (kelas) dari sampel ke-i, biasanya $y^{(i)} \in \{0, 1\}$.
- $\mathbf{x}^{(i)}$: Vektor fitur dari sampel ke-i.
- $p(y^{(i)} \mid \mathbf{x}^{(i)}; \boldsymbol{\theta})$: Probabilitas prediksi model bahwa $y^{(i)}$ adalah label yang benar untuk $\mathbf{x}^{(i)}$, dihitung menggunakan fungsi sigmoid.
- $-\log p(\cdot)$: Komponen log-loss (negative log-likelihood), yang mengukur kesalahan prediksi; semakin tinggi kesalahan, semakin besar nilai ini.
- $\|\boldsymbol{\theta}\|_1$: Norma L1 dari vektor $\boldsymbol{\theta}$, yaitu $\sum_j |\theta_j|$; ini mendorong sparsity dalam model (mengurangi jumlah parameter aktif).
- β: Koefisien regularisasi, yang mengontrol kekuatan penalti terhadap kompleksitas model.

2.4.2 Recursive Feature Elimination (RFE)

Recursive Feature Elimination (RFE) adalah teknik seleksi fitur yang bekerja dengan cara menghapus fitur yang dianggap kurang penting secara bertahap, hingga jumlah fitur yang diinginkan tercapai. Prosesnya dimulai dengan semua

fitur, kemudian model dilatih dan pentingnya setiap fitur dihitung. Fitur yang memiliki nilai penting terendah akan dihapus, dan langkah ini diulangi hingga hanya tersisa fitur yang paling relevan. Berikut merupakan *Pseudocode* dari RFE [37].

Pseudocode Recursive Feature Elimination (RFE)

Input:
    - Data dengan n fitur
    - Model prediktif
    - Jumlah fitur yang diinginkan (k)
Output: Fitur terpilih dengan jumlah k
Inisialisasi `Fitur_tersisa` $\leftarrow$ semua fitur
while jumlah(`Fitur_tersisa`) $> k$ **do**
    Latih model menggunakan `Fitur_tersisa`
    Hitung pentingnya tiap fitur (importance/koefisien)
    Temukan fitur dengan importance terendah
    Hapus fitur tersebut dari `Fitur_tersisa`
end while
Kembalikan: `Fitur_tersisa`

RFE dapat digunakan bersama berbagai model prediktif, dan karena sifatnya yang rekursif, metode ini memastikan hanya fitur-fitur yang paling informatif yang dipertahankan. RFE dapat digunakan dengan model algoritma lain agar mencapai hasil terbaik, salah satunya adalah *Support Vector Machine - Recursive Feature Elimination* (SVM-RFE). SVM-RFE adalah metode seleksi fitur yang menggabungkan SVM dengan proses eliminasi rekursif. Prosesnya dilakukan dengan menghapus fitur yang memiliki kontribusi (bobot) terkecil secara bertahap. Tiga tahapan utama SVM-RFE:

1. Training Model: Melatih data menggunakan SVM untuk menghitung bobot tiap fitur. Bobot dihitung menggunakan nilai α (alpha) dari hasil pelatihan SVM.

Rumus:

$$w = \sum_{i=1}^{K} \alpha_k y_k x_k \tag{2.5}$$

keterangan:

- w : vektor bobot dari model SVM
- K : jumlah total data latih (support vectors)

- α_k : koefisien Lagrange untuk data ke-k
- y_k : label kelas (target) untuk data ke-k
- x_k : vektor fitur dari data ke-k

2. Perhitungan Ranking: Menghitung nilai ranking untuk tiap fitur berdasarkan besar kuadrat bobot:

Rumus:

$$C_k = w_k^2, k \quad k = 1, 2, \ldots, |S| \tag{2.6}$$

- C_k : nilai ranking (*criterion ranking*) untuk fitur ke-k
- w_k : bobot fitur ke-k yang dihitung berdasarkan SVM
- $|S|$: jumlah total fitur yang tersedia
- k : indeks fitur yang diurutkan berdasarkan bobotnya

3. Eliminasi Fitur: Mengurutkan fitur berdasarkan nilai ranking dan menghapus fitur dengan bobot terkecil di setiap iterasi.

Metode ini bertujuan untuk menghapus fitur yang tidak relevan dan meningkatkan performa model dengan hanya mempertahankan fitur yang paling informatif [38].

2.5 Algoritma Klasifikasi

Pada tahap klasifikasi, penelitian ini menggunakan beberapa algoritma pembelajaran mesin yang telah terbukti efektif dalam pengenalan pola dan klasifikasi data biologis, khususnya pada data ekspresi genetik. Pemilihan algoritma didasarkan pada kemampuan masing-masing metode dalam menangani data berdimensi tinggi, performa klasifikasi yang baik, serta kemudahan interpretasi. Subbagian berikut menjelaskan prinsip kerja dan karakteristik masing-masing algoritma yang digunakan dalam penelitian ini.

2.5.1 Support Vector Machine (SVM)

Support Vector Machines (SVM), yang pertama kali dikembangkan oleh Vapnik, telah terbukti sangat efektif dalam berbagai masalah pengenalan pola. SVM sering kali memberikan hasil klasifikasi yang lebih baik dibandingkan teknik klasifikasi lainnya [39]. Dalam metode ini, perlu ditentukan *hyperplane* yang berfungsi sebagai batas pemisah antara kelas-kelas yang ada [40]. SVM menggunakan hyperplane linear untuk memisahkan dua kelas, dengan memilih hyperplane yang memiliki jarak terluas (*margin*) antara data dari kedua kelas. *Hyperplane* ini membantu model untuk lebih baik dalam mengklasifikasikan data baru yang belum pernah dilihat sebelumnya, karena memiliki kemampuan generalisasi yang lebih tinggi. Untuk menangani masalah klasifikasi yang tidak dapat dipisahkan secara linear, SVM memanfaatkan teknik *kernel* untuk memetakan data ke ruang fitur yang lebih tinggi. Hal ini menghindari masalah dimensi yang berlebihan dan memungkinkan SVM untuk melakukan klasifikasi linear di ruang fitur yang setara dengan klasifikasi non-linear di ruang data asli. Dengan pendekatan ini, SVM dapat menentukan *hyperplane* pemisah yang optimal di dimensi yang lebih tinggi. Dalam memahami SVM dalam menghitung *hyperplane margin* maksimal dan mendukung klasifikasi non-linier perlu mengetahui *hard margin* terlebih dahulu. *Hard margin* SVM adalah metode di mana tidak ada data pelatihan yang berada di antara batas keputusan. Ini berarti bahwa semua titik data terpisah dengan sempurna oleh dua garis keputusan yang dikenal sebagai batas keputusan (*decision boundaries*) [41].

Dalam mengklasifikasikan data dengan benar, SVM mengharuskan fungsi $F(x)$ menghasilkan nilai positif untuk data kelas positif ($y_i = 1$) dan nilai negatif untuk data kelas negatif ($y_i = -1$). Kondisi ini dapat dirumuskan dalam persamaan 2.7.

Rumus:

$$\begin{cases} \mathbf{w} \cdot \mathbf{x}_i - b > 0 & \text{untuk } y_i = 1 \\ \mathbf{w} \cdot \mathbf{x}_i - b < 0 & \text{untuk } y_i = -1 \end{cases} \tag{2.7}$$

Keterangan:

- **w** adalah *vektor bobot* (weight vector),
- b adalah *bias*,
- **x** adalah *vektor input* atau fitur data.

Formula tersebut dapat digabungkan menjadi bentuk yang lebih umum dalam pertidaksamaan 2.9. Di mana **D** adalah himpunan data pelatihan.

Rumus:

$$y_i\,(\mathbf{w}\cdot\mathbf{x}_i - b) > 0 \quad \forall(\mathbf{x}_i, y_i) \in \mathbf{D} \tag{2.8}$$

Salah satu tujuan utama dari *Support Vector Machine* (SVM) adalah mencari *margin* terbesar, yaitu jarak antara garis pemisah (hyperplane) dan titik data terdekat dari masing-masing kelas. Semakin besar *margin* yang diperoleh, maka model cenderung memiliki kemampuan generalisasi yang lebih baik—artinya, model lebih mampu mengenali atau mengklasifikasikan data baru yang belum pernah dilihat sebelumnya. Untuk mencapai tujuan ini, SVM tidak hanya mempertimbangkan pemisahan kelas, tetapi juga menambahkan batasan tambahan pada kondisi klasifikasi, agar *margin* tersebut dapat dimaksimalkan secara matematis.

Rumus:

$$y_i\,(\mathbf{w}\cdot\mathbf{x}_i - b) > 1 \quad \forall(\mathbf{x}_i, y_i) \in \mathbf{D} \tag{2.9}$$

Saat data tidak dapat dipisahkan secara linear, *Soft-Margin* SVM digunakan. Pendekatan ini memperbolehkan beberapa titik data salah klasifikasi, dengan tetap berusaha memaksimalkan *margin*. Untuk mengakomodasi hal ini, metode *Soft-Margin SVM* memperkenalkan *slack variables* ξ_i, yang digunakan untuk mengukur *sejauh mana pelanggaran margin* yang terjadi. Untuk klasifikasi *soft margin*, masalah primal SVM didefinisikan dalam persamaan 2.10 2.11 [38, 42].

Rumus:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i \tag{2.10}$$

dengan syarat:

$$y_i(w^T x_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \tag{2.11}$$

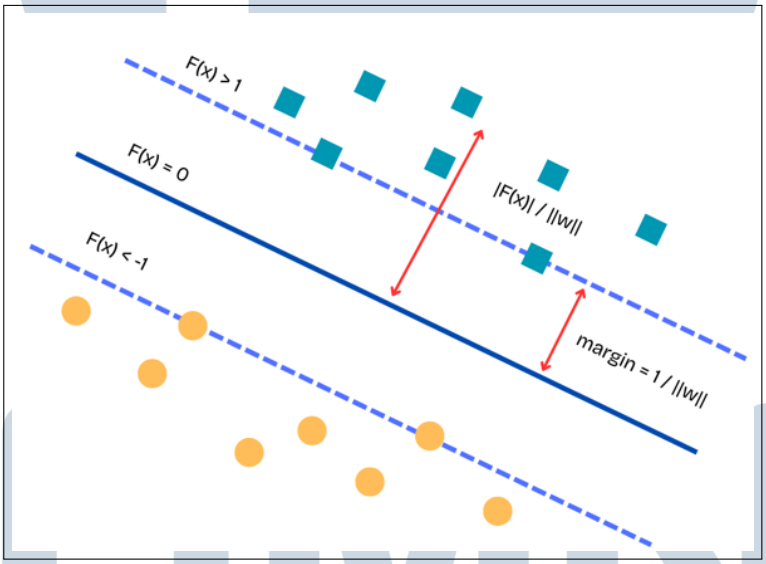
Keterangan:

- $y_i \in \{-1, +1\}$: label kelas
- ξ_i : *slack variable* (jumlah pelanggaran *margin*)
- C : parameter regularisasi (trade-off antara *margin* besar dan kesalahan kecil)

Untuk mendapatkan *hyperplane* dengan *margin* maksimum, SVM harus menghitung jarak *margin* antara dua kelas yang berbeda. Dapat dilihat dalam persamaan 2.12. Visualisasi SVM dalam mencari ukuran *margin* dapat di lihat pada Gambar 2.1. Karena $\mathbf{x}_i$ merupakan vektor yang paling dekat dengan *hyperplane*, maka fungsi keputusan $F(x)$ akan menghasilkan nilai 1 sesuai dengan Persamaan 2.9. Vektor-vektor yang berada paling dekat dan memenuhi Persamaan 2.9 dengan tanda sama dengan disebut sebagai *support vector*.

Rumus:

$$\text{margin} = \frac{1}{\|\mathbf{w}\|} \tag{2.12}$$



Gambar 2.1. Hyperplane SVM yang memaksimalkan margin pemisah antar kelas dalam ruang 2 dimensi [43]

.

Untuk memaksimalkan *margin*, SVM justru berupaya meminimalkan norma $\|\mathbf{w}\|$. Oleh karena itu, proses pelatihan SVM diubah menjadi masalah optimisasi dengan kendala, yang dirumuskan 2.13

Rumus:

$$\text{Minimalkan:} \quad Q(\mathbf{w}) = \frac{1}{2}\|\mathbf{w}\|^2 \tag{2.13}$$

Dalam menyelesaikan permasalahan optimasi, sering kali diperlukan

pendekatan dalam bentuk dual. Salah satu metode yang digunakan adalah metode *Lagrange*, yang diperkenalkan oleh *Joseph Louis Lagrange*. Metode ini bertujuan untuk mentransformasikan permasalahan optimasi yang memiliki kendala menjadi bentuk tanpa kendala, sehingga dapat menyelesaikan persoalan yang melibatkan kendala berupa persamaan maupun pertidaksamaan [44]. Dalam konteks SVM, penting untuk dipahami bahwa variabel slack ξ_i beserta pengali Lagrangenya tidak muncul dalam bentuk dual dari permasalahan. Menariknya, struktur dual untuk kasus data yang tidak sepenuhnya dapat dipisahkan secara linear (*nonseparable*) memiliki bentuk yang hampir sama dengan kasus data yang terpisah secara linear (*separable*). Perbedaan utamanya terletak pada batasan nilai α_i, di mana pada kasus *nonseparable* digunakan batasan yang lebih ketat, yaitu $0 \leq \alpha_i \leq C$, dibandingkan dengan hanya $\alpha_i \geq 0$ pada kasus *separable*. Meskipun demikian, tahapan optimasi dan proses perhitungan bobot model pada kedua kasus tersebut tetap mengikuti prinsip yang serupa.

Rumus:
$$\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \tag{2.14}$$

dengan syarat:
$$\sum_{i=1}^{n} \alpha_i y_i = 0, \tag{2.15}$$
$$0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, n. \tag{2.16}$$

Perlu diperhatikan bahwa dalam permasalahan optimisasi maupun fungsi klasifikasi pada SVM, fungsi pemetaan fitur $\varphi(x)$ selalu muncul dalam bentuk perkalian titik, seperti $\varphi(x_i) \cdot \varphi(x_j)$. Perkalian tersebut merupakan *inner product* antara dua vektor dalam ruang fitur hasil transformasi. Namun, menghitung *inner product* secara eksplisit di ruang berdimensi tinggi tersebut seringkali rumit dan dapat menimbulkan permasalahan kompleksitas yang tinggi, dikenal sebagai *curse of dimensionality*. Untuk mengatasi masalah ini, digunakan pendekatan yang disebut *kernel trick*. *Kernel trick* memungkinkan kita menggantikan $\varphi(x_i) \cdot \varphi(x_j)$ dengan sebuah fungsi *kernel* $K(x_i, x_j)$ yang langsung dihitung di ruang *input* asli, tanpa perlu melakukan transformasi eksplisit ke ruang berdimensi tinggi [45].

Secara formal, fungsi *kernel* $k(x_i, x_j)$ didefinisikan sebagai hasil *inner product* di ruang *Hilbert V* tertentu seperti pada persamaan 2.17.

Rumus:

$$K(x_i, x_j) = \langle \varphi(x_i), \varphi(x_j) \rangle v \tag{2.17}$$

Keterangan:

- x_i, x_j : Vektor *input* (sampel) dalam ruang asli.
- $\varphi(x)$: Fungsi pemetaan dari ruang *input* ke ruang fitur berdimensi lebih tinggi.

Dengan $\varphi : \mathbb{R}^d \to V$ sebagai pemetaan dari ruang input ke ruang fitur berdimensi lebih tinggi V. Dengan mengganti seluruh operasi $\varphi(x_i) \cdot \varphi(x_j)$ menggunakan $k(x_i, x_j)$, maka proses klasifikasi maupun optimisasi dapat dilakukan secara implisit di ruang fitur tersebut. Dengan $\varphi : \mathbb{R}^d \to V$ sebagai pemetaan dari ruang input ke ruang fitur berdimensi lebih tinggi V. Dengan mengganti seluruh operasi $\varphi(x_i) \cdot \varphi(x_j)$ menggunakan $k(x_i, x_j)$, maka proses klasifikasi maupun optimisasi dapat dilakukan secara implisit di ruang fitur tersebut.

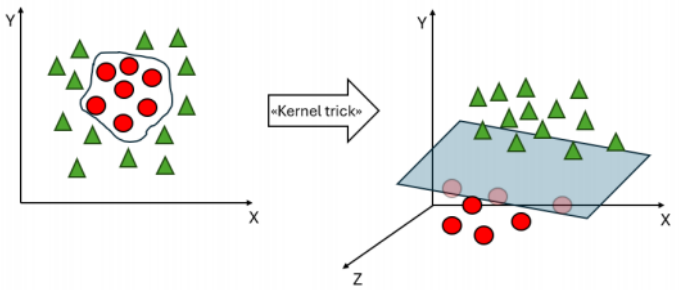
Penggunaan *kernel trick* memungkinkan SVM menyelesaikan permasalahan klasifikasi yang tidak dapat dipisahkan secara linear di ruang asli. Hal ini karena dalam ruang fitur V yang lebih tinggi, data yang semula tidak linear dapat menjadi *separable* oleh suatu *hyperplane*. Meskipun dilakukan di ruang fitur berdimensi tinggi, perhitungan tetap dilakukan di ruang asli melalui fungsi *kernel*, sehingga efisien secara komputasi. Agar suatu fungsi *kernel* dapat digunakan dalam SVM, fungsi tersebut harus memenuhi *Mercer's condition*, yaitu:

Rumus:

$$\int_S \int_S g(x)\, k(x, x')\, g(x')\, dx\, dx' \geq 0 \tag{2.18}$$

Setiap fungsi $g(x)$ yang bersifat kuadrat *integrabel* (*square integrable*). Jika kondisi ini terpenuhi, maka matriks *kernel* M yang dibentuk dengan $M_{ij} = k(x_i, x_j)$ akan memiliki dua sifat penting: simetris ($M = M^T$) dan *positive semi-definite* (PSD), yaitu memenuhi $u^T M u \geq 0$ untuk semua vektor $u \in \mathbb{R}^n$. Dengan kata lain, semua nilai *eigen* dari matriks tersebut tidak bernilai negatif, menjamin konveksitas dari permasalahan optimisasi kuadratik pada SVM. *Kernel trick* tidak hanya menyederhanakan perhitungan di ruang berdimensi tinggi, tetapi juga

memungkinkan SVM menyelesaikan masalah klasifikasi non-linear secara efisien dan optimal [46]. Gambar 2.2 mengilustrasikan transformasi data yang awalnya tidak dapat dipisahkan secara linear menjadi dapat dipisahkan setelah dilakukan pemetaan ke ruang berdimensi lebih tinggi [47].



Gambar 2.2. Ilustrasi *kernel trick* yang memetakan data ke ruang fitur berdimensi lebih tinggi untuk memungkinkan pemisahan non-linear [47].

Terdapat beberapa *kernel* yang umum digunakan dalam metode SVM berupa:

- ***Kernel Linear*:** *Kernel linear* merupakan bentuk paling dasar dari fungsi *kernel*, yang hanya menghitung perkalian titik (*dot product*) antara dua vektor *input*, yaitu x_i dan x_j, di ruang asli. *Kernel* ini digunakan jika data dapat dipisahkan secara linear, sehingga tidak diperlukan pemetaan ke ruang berdimensi lebih tinggi. Pendekatan ini cocok untuk kasus di mana hubungan antar fitur bersifat linier.

  Rumus:

$$K(x_i, x_j) = x_i^T x_j \tag{2.19}$$

- ***Kernel Polinomial*:** *Kernel polinomial* mengukur kemiripan antara dua vektor *input* x_i dan x_j dengan terlebih dahulu menghitung perkalian titik ($x_i^T x_j$), lalu mengubahnya menggunakan parameter γ, r, dan eksponen d. Parameter γ mengatur seberapa besar pengaruh perkalian titik tersebut terhadap *kernel*, sementara r berfungsi sebagai bias yang menambah fleksibilitas. Eksponen d menentukan derajat polinomial, yang memungkinkan *kernel* ini untuk menangkap hubungan non-linier antar

fitur. *Kernel* polinomial sangat berguna untuk masalah yang memerlukan pemisahan data secara non-linier menggunakan kekuatan polinomial.

Rumus:

$$K(x_i, x_j) = (\gamma x_i^T x_j + r)^d \tag{2.20}$$

- ***Radial Basis Function* (RBF):** *Kernel* RBF mengukur kemiripan antara dua vektor *input* dengan menghitung jarak *Euclidean* dan menggunakan fungsi eksponensial. Parameter σ mengatur seberapa sensitif *kernel* terhadap perbedaan antar vektor, di mana semakin kecil σ, semakin besar pengaruh jarak antar vektor. *Kernel* RBF efektif digunakan untuk masalah non-linier dan banyak digunakan dalam algoritma seperti SVM.

  Rumus:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \tag{2.21}$$

Decision function dalam SVM digunakan untuk memprediksi kelas dari data baru berdasarkan model yang telah dilatih. Fungsi ini menghitung kombinasi dari kontribusi data latih yang menjadi *support vector*, dan menentukan pada sisi mana dari *hyperplane* data tersebut berada. Dengan menggunakan fungsi *kernel*, SVM mampu memetakan data yang tidak dapat dipisahkan secara linier ke dalam ruang fitur berdimensi lebih tinggi, sehingga memungkinkan pemisahan kelas yang lebih baik. Nilai dari fungsi keputusan ini kemudian digunakan untuk menentukan label kelas, berdasarkan tanda (*sign*) dari hasil perhitungan [43].

Rumus:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b \tag{2.22}$$

Keterangan:

- $f(x)$: Fungsi keputusan untuk memprediksi kelas dari data uji x
- n : Jumlah data latih

- α_i : Parameter Lagrange multipliers, hanya bernilai non-nol untuk *support vectors*
- y_i : Label kelas dari data latih x_i ($\in \{-1, +1\}$)
- $K(x_i, x)$: Fungsi kernel yang mengukur kemiripan antara data latih x_i dan data uji x
- b : Bias atau konstanta dari hyperplane

2.5.2 Random Forest (RF)

Random Forest (RF) merupakan metode statistik yang banyak digunakan dalam menganalisis data dan membuat berbagai prediksi. *Random Forest* adalah kumpulan dari banyak pohon keputusan (*decision tree*). Untuk *Random Forest*, hasil prediksi $\hat{y}$ dihitung dengan mengambil modus (nilai yang paling sering muncul) dari prediksi semua pohon keputusan [48].

Rumus:

$$\hat{y} = \text{mode}\,(h_1(x), h_2(x), \ldots, h_T(x)) \tag{2.23}$$

Keterangan:

- $h_t(x)$ adalah prediksi dari pohon ke-t,
- T adalah jumlah total pohon dalam Random Forest,
- mode adalah fungsi yang menghasilkan nilai yang paling sering muncul (modus).

Metode *Random Forest* bekerja dengan membangun sejumlah pohon keputusan secara acak dan independen, di mana setiap pohon memberikan prediksi berdasarkan *subset* fitur yang berbeda. Proses pembuatan pohon keputusan ini dapat digambarkan melalui langkah-langkah berikut [49]:

Pseudocode untuk Random Forest

Input: Data training D, jumlah pohon n, jumlah fitur yang dipilih k, jumlah fitur yang digunakan m
Output: Prediksi akhir berdasarkan pemungutan suara dari pohon-pohon keputusan
for setiap pohon $i = 1$ hingga n **do**
    Secara acak pilih m fitur dari total k fitur ($m \ll k$)
    Bangun pohon keputusan berdasarkan subset data dan fitur terpilih
    for setiap node dalam pohon **do**
        Hitung node dengan menggunakan titik pemisah terbaik berdasarkan kriteria tertentu (misal, Gini atau Entropi)
        Kategorikan node menjadi node anak berdasarkan pemisahan terbaik
    end for
    Setelah pohon selesai, simpan pohon keputusan
end for
Gabungkan hasil prediksi dari semua pohon untuk mendapatkan prediksi akhir (*moda* atau prediksi mayoritas)

Dalam menghitung prediksi, implementasi metode dalam konteks analisis data sangat penting, di mana hasil prediksi dari setiap pohon keputusan digabungkan untuk menghasilkan prediksi akhir yang lebih akurat. Setiap pohon, data dibagi berdasarkan kondisi tertentu dari suatu fitur. Pada klasifikasi, pemilihan pembagi di setiap *node* didasarkan pada seberapa besar pengurangan Gini *impurity* yang dihasilkan. Setelah semua pohon terbentuk, pentingnya setiap fitur dihitung dari rata-rata pengurangan Gini *impurity* yang disumbangkannya di seluruh pohon dalam model.

Rumus:

$$GI_m = 1 - \sum_{k=1}^{|k|} p_{mk}^2 \tag{2.24}$$

Keterangan:

- GI_m: Gini impurity pada node m
- $|k|$: Jumlah total kelas dalam dataset
- p_{mk}: Proporsi data pada node m yang termasuk dalam kelas ke-k
- $\sum_{k=1}^{|k|} p_{mk}^2$: Jumlah kuadrat dari proporsi setiap kelas pada node m

Gini impurity mengukur ketidakhomogenan suatu *node*. Nilai 0 menunjukkan bahwa semua data dalam *node* berasal dari satu kelas (murni),

sedangkan nilai yang lebih tinggi menunjukkan distribusi kelas yang lebih merata (tidak murni) [50]. Selain Gini *impurity*, metode lain yang umum digunakan untuk mengukur ketidakhomogenan data dalam sebuah node adalah entropi, yang berasal dari teori informasi dan mempertimbangkan tingkat ketidakpastian dalam distribusi kelas. Entropi adalah metode kuantitatif untuk mengukur seberapa besar informasi yang dibawa oleh suatu data. Semakin merata distribusi data tersebut, maka semakin tinggi nilai entropinya. Misalnya, suatu node yang akan dibagi (*split*) terdiri dari sekumpulan data S, di mana S memiliki s sampel dan terdapat n kelas di dalamnya [51]. Maka, nilai entropi dari node tersebut dapat dihitung dengan rumus berikut:

Rumus:

$$\text{Entropy}(D) = -\sum_{k=1}^{n} p_k \log_2(p_k) \tag{2.25}$$

- D : Sekumpulan data atau node yang sedang dianalisis.
- n : Jumlah kelas yang terdapat dalam data D.
- p_k : Proporsi (probabilitas) dari kelas ke-k dalam data D, yaitu $\frac{|D_k|}{|D|}$.
- $\log_2$: Logaritma basis 2, digunakan untuk mengukur jumlah bit informasi.

Setelah membahas konsep entropi sebagai ukuran ketidakhomogenan data dalam suatu *node*, dapat dilanjutkan dengan metode lain yang berbasis pada entropi, yaitu *Information Gain* (IG). IG digunakan untuk memilih fitur dengan menghitung pengurangan entropi yang terjadi setelah data dikelompokkan berdasarkan nilai fitur tertentu. Nilai dari fitur A akan mempengaruhi pemisahan data yang lebih baik, yang dinyatakan sebagai gain (y,A) [52]. Formula *Information Gain* adalah sebagai berikut:

Rumus:

$$IG(D,A) = \text{Entropy}(D) - \sum_{v \in \text{Values}(A)} \frac{|D_v|}{|D|} \text{Entropy}(D_v) \tag{2.26}$$

Keterangan:

- $IG(D,A)$: Information Gain yang dihitung berdasarkan pembagian data D menurut fitur A. Ini mengukur seberapa besar penurunan ketidakpastian (entropy) yang dihasilkan setelah membagi data berdasarkan fitur A.

- Entropy(D) : Entropi dari data D sebelum pembagian, yang menunjukkan tingkat ketidakpastian atau ketidakteraturan dalam dataset secara keseluruhan.
- $v \in \text{Values}(A)$: Menyatakan bahwa v adalah elemen dari himpunan nilai yang mungkin dari fitur A.
- D_v : Subset data yang diperoleh dengan membagi dataset D berdasarkan nilai v pada fitur A.
- $|D_v|$: Jumlah sampel dalam subset D_v, yaitu banyaknya data yang memiliki nilai v pada fitur A.
- $|D|$: Jumlah total sampel dalam dataset D.
- $\frac{|D_v|}{|D|}$: Proporsi sampel dalam subset D_v terhadap seluruh dataset D.
- Entropy(D_v) : Entropi dari subset D_v, yang mengukur ketidakpastian dalam subset data yang terbentuk setelah pembagian berdasarkan nilai v.

2.5.3 Voting Classifier

Voting Classifier adalah metode *ensemble learning* yang menggabungkan beberapa model *machine learning* berbeda (*classifier*) untuk membuat prediksi akhir berdasarkan suara mayoritas (*hard voting*) atau rata-rata probabilitas (*soft voting*). Metode ini digunakan dalam masalah klasifikasi dan regresi, dan biasanya memberikan hasil yang lebih baik dibandingkan dengan model individu. Prediksi yang digunakan pada penelitian ini berupa rata-rata probabilitas (*soft voting*). *Soft Voting* adalah metode di mana prediksi akhir ditentukan dengan menghitung rata-rata probabilitas prediksi dari tiap kelas oleh semua *classifier*, lalu memilih kelas dengan rata-rata probabilitas tertinggi. *Soft Voting Classifier* dengan Tiga Model (SVM, *Random Forest*, dan *Logistic Regression*) [53].

Rumus:

$$P_{\text{Vote}}(y = k \mid x) = \frac{1}{3}\left(P_{\text{SVM}}(y = k \mid x) + P_{\text{RF}}(y = k \mid x) + P_{\text{LR}}(y = k \mid x)\right) \quad (2.27)$$

Keterangan:

- $P_{\text{Vote}}(y = k \mid x)$: Probabilitas akhir bahwa sampel x termasuk kelas k, berdasarkan rata-rata dari semua model.
- $P_{\text{SVM}}(y = k \mid x)$: Probabilitas bahwa x termasuk kelas k menurut model *Support Vector Machine*.
- $P_{\text{RF}}(y = k \mid x)$: Probabilitas bahwa x termasuk kelas k menurut model *Random Forest*.
- $P_{\text{LR}}(y = k \mid x)$: Probabilitas bahwa x termasuk kelas k menurut model *Logistic Regression*.
- Faktor $\frac{1}{3}$: Menunjukkan pengambilan rata-rata dari tiga model (jumlah model = 3).

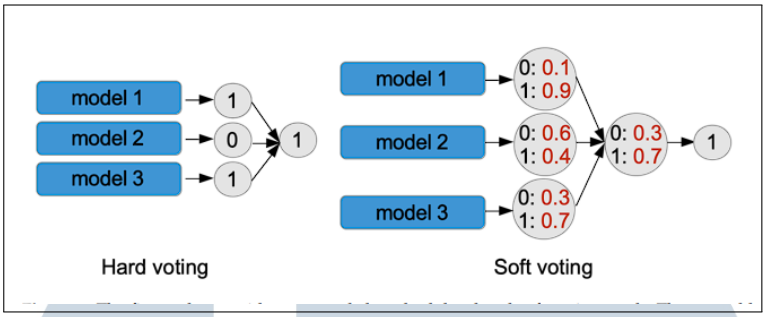
Soft voting adalah metode untuk menggabungkan prediksi dari beberapa model dengan cara menghitung rata-rata probabilitas yang diberikan oleh setiap model untuk masing-masing kelas. Dalam metode ini, alih-alih memilih kelas berdasarkan hasil prediksi keras (seperti pada *hard voting*), setiap model memberikan nilai probabilitas untuk setiap kelas, seperti yang dapat dilihat pada Gambar 2.3. Kemudian, kelas dengan rata-rata probabilitas tertinggi dari semua model dipilih sebagai hasil akhir. Jadi, *soft voting* mempertimbangkan tingkat keyakinan setiap model terhadap setiap kelas, dan memilih kelas dengan probabilitas terbesar sebagai prediksi akhir [54].

Rumus:

$$P(y = k \mid x) = \frac{1}{M} \sum_{m=1}^{M} P_m(y = k \mid x) \tag{2.28}$$

Keterangan:

- $P(y = k \mid x)$: Probabilitas akhir untuk kelas k diberikan *input* x, yang dihitung dengan rata-rata probabilitas yang diberikan oleh semua model dalam ensemble.
- M: Jumlah total model dalam ensemble.
- $P_m(y = k \mid x)$: Probabilitas yang diprediksi oleh model ke-m untuk kelas k pada *input* x.



Gambar 2.3. Mekanisme *Hard Voting* dan *Soft Voting* [54]

2.6 Evaluasi

Dalam menunjukkan model memprediksi secara akurat atau tidak, dapat dilihat dari metrik penilaian *accuracy, precision, recall*, dan *f1-score*.

2.6.1 Confusion Matrix

Confusion matrix adalah sebuah matriks yang digunakan untuk menggambarkan kinerja suatu algoritma klasifikasi, dengan menunjukkan seberapa baik model dalam memprediksi kelas yang benar. Matriks ini mencerminkan jumlah prediksi yang benar maupun prediksi yang salah, dan memberikan gambaran menyeluruh mengenai seberapa akurat model dalam mengklasifikasikan data ke dalam kelas yang tepat. Dapat dilihat *confusion matrix* pada Tabel 2.2.

Tabel 2.2. Confusion Matrix

	Positive (1)	**Negative (0)**
Positive (1)	True Positive (TP)	False Negative (FN)
Negative (0)	False Positive (FP)	True Negative (TN)

Sumber: [55]

2.6.2 Accuracy

Accuracy mengukur seberapa sering model klasifikasi membuat prediksi yang benar. Dengan menghitung proporsi jumlah prediksi yang benar (baik positif maupun negatif) dibandingkan dengan seluruh jumlah prediksi. *Accuracy* dapat dihitung dengan:

Rumus:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.29}$$

Keterangan:

- **TP (True Positives)**: Kasus positif yang diprediksi benar.
- **TN (True Negatives)**: Kasus negatif yang diprediksi benar.
- **FP (False Positives)**: Kasus negatif yang diprediksi sebagai positif (salah).
- **FN (False Negatives)**: Kasus positif yang diprediksi sebagai negatif (salah).

2.6.3 Precision

Precision mengukur seberapa akurat model dalam memprediksi kelas positif. Artinya, dari semua yang diprediksi positif oleh model, berapa banyak yang benar-benar positif. *Precision* tinggi menunjukkan bahwa model jarang memberikan prediksi positif yang salah (*False Positive* rendah), yang sangat penting dalam situasi di mana kesalahan positif berisiko besar, seperti dalam diagnosis penyakit. *Precision* dapat dihitung dengan:

Rumus:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.30}$$

2.6.4 Recall

Recall menggambarkan sejauh mana model mampu mendeteksi seluruh kasus yang benar-benar positif. Dengan kata lain, *recall* mengukur proporsi data positif yang berhasil diidentifikasi oleh model dari keseluruhan data yang sebenarnya positif. *Recall* yang tinggi menunjukkan bahwa model jarang melewatkan kasus positif, yang sangat penting dalam situasi seperti deteksi kanker, di mana kehilangan satu kasus positif bisa memiliki dampak yang sangat serius. *Recall* dapat dihitung dengan:

Rumus:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.31}$$

2.6.5 F1-Score

F1-score menggabungkan *precision* dan *recall* menjadi satu metrik untuk memberikan gambaran yang lebih menyeluruh tentang kinerja model. Metrik ini berguna ketika kita ingin memastikan model tidak hanya akurat dalam memprediksi kasus positif, tetapi juga mampu mendeteksi sebanyak mungkin kasus positif yang sebenarnya ada. *F1-score* yang tinggi menunjukkan bahwa model memiliki performa yang baik dalam kedua aspek tersebut, baik dalam hal keakuratan prediksi maupun kemampuan untuk menemukan kasus positif [55]. *F1-Score* dapat dihitung dengan:

Rumus:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.32}$$