BAB 2
LANDASAN TEORI

2.1 Kanker Prostat

Kanker prostat merupakan salah satu jenis kanker yang terjadi pada pria, ditandai dengan pertumbuhan sel abnormal di kelenjar prostat [3]. Kelenjar prostat sendiri merupakan kelenjar eksokrin dalam sistem reproduksi pria yang berperan dalam menghasilkan sebagian cairan semen yang mendukung kelangsungan hidup sperma. Kelenjar ini terletak di bawah kandung kemih dan mengelilingi uretra (saluran kemih). Kanker ini umumnya terjadi pada pria usia lanjut dan sering berkembang secara perlahan. Namun, beberapa tipe kanker prostat bersifat agresif dan dapat tumbuh serta menyebar dengan cepat ke bagian tubuh lain, seperti tulang dan kelenjar getah bening [3]. Perkembangan dan penyebaran dari kanker prostat dapat dibedakan menjadi 4 stadium dengan penjelasan berikut:

1. Stadium I: Kanker masih sangat kecil dan hanya ditemukan di dalam prostat. Umumnya tidak menunjukkan gejala dan terdeteksi secara tidak sengaja melalui tes PSA atau biopsi.

2. Stadium II: Kanker masih berada di dalam prostat tetapi lebih besar ukurannya. Kanker mungkin bisa dirasakan melalui pemeriksaan fisik, namun belum menyebar ke luar prostat.

3. Stadium III: Kanker telah menyebar ke jaringan di sekitar prostat, seperti vesikula seminalis, namun belum mencapai organ yang jauh.

4. Stadium IV: Kanker telah bermetastasis, yaitu menyebar ke organ tubuh lainnya, seperti tulang, kandung kemih, rektum, atau kelenjar getah bening. Pada stadium ini, gejala menjadi lebih parah dan kompleks.

Dalam menentukan stadium kanker prostat, salah satu sistem yang sering digunakan adalah klasifikasi TNM (Tumor, Node, Metastasis), kadar PSA (Prostate-Specific Antigen), dan *Grade Group* dari hasil pemeriksaan histopatologis. Sistem tersebut dikeluarkan oleh *American Joint Committee on Cancer* (AJCC) dan U*nion Internationale Contre le Cancer* (UICC) pada tahun 2010 [14]. Kombinasi dari parameter-parameter ini digunakan untuk mengelompokkan kanker prostat ke

dalam beberapa tahapan, mulai dari stadium I (paling awal) hingga stadium IV (paling lanjut) yang disajikan pada Tabel 2.1.

Tabel 2.1. Staging Kanker Prostat Berdasarkan TNM, PSA, dan Grade Group

Group	T	N	M	PSA (ng/mL)	Grade Group
Stage I	cT1a-c	N0	M0	PSA $<$ 10	1
	cT2a	N0	M0	PSA $<$ 10	1
	pT2	N0	M0	PSA $<$ 10	1
Stage IIA	cT1a-c	N0	M0	PSA $\geq$ 10 and $<$ 20	1
	cT2a	N0	M0	PSA $\geq$ 10 and $<$ 20	1
	pT2	N0	M0	PSA $\geq$ 10 and $<$ 20	1
Stage IIB	cT2b	N0	M0	PSA $<$ 20	2
	cT2c	N0	M0	PSA $<$ 20	2
Stage IIC	T1-2	N0	M0	PSA $<$ 20	3 or 4
Stage IIIA	T1-2	N0	M0	PSA $\geq$ 20	1–4
Stage IIIB	T3-4	N0	M0	Any PSA	1–4
Stage IIIC	Any T	N0	M0	Any PSA	5
Stage IVA	Any T	N1	M0	Any PSA	Any
Stage IVB	Any T	Any N	M1	Any PSA	Any

Sumber: [14]

2.2 Ekspresi Gen (RNA-seq)

Ekspresi gen merupakan proses biologis di mana terjadi sintesis produk gen fungsional menggunakan informasi yang disediakan oleh DNA, umumnya berupa protein [15]. Perubahan ekspresi gen menggambarkan respons sel terhadap kondisi biologis tertentu dan dapat ditemukan melalui pengukuran transkrip RNA [15]. Salah satu metode pengukuran dan proses memprofilkan RNA dilakukan menggunakan teknologi *Next-Generation Sequencing* (NGS). Teknik tersebut dinamakan *RNA sequencing* (RNA-seq) yang memungkinkan para peneliti melihat jenis dan jumlah RNA dalam suatu sel atau jaringan pada waktu tertentu [16]. Proses pembacaan dimulai dengan pemecahan RNA menjadi potongan kecil yang diubah menjadi cDNA. cDNA kemudian diperbanyak dan dibaca oleh mesin sekuensing. Hasil sekuensing tersebut berupa *reads* yang dipetakan ke genom. Jumlah *reads* disebut sebagai nilai *counts*, yang mewakili seberapa banyak suatu

gen diekspresikan [17]. Semakin tinggi jumlahnya, maka semakin tinggi pula ekspresi gen tersebut. Data ini umumnya disajikan dalam bentuk matriks dua dimensi, di mana baris mewakili sampel, dan kolom mewakili fitur atau gen. Setiap sel pada matriks berisi nilai numerik *counts* yang diperoleh melalui proses sekuensing tersebut.

2.3 Ekspresi mikroRNA

microRNA atau miRNA merupakan molekul RNA kecil berukuran sekitar 20–24 nukleotida yang tidak dikodekan menjadi protein [18]. miRNA berperan sebagai pengatur ekspresi gen pasca transkripsi, yaitu setelah proses transkripsi dari DNA menjadi RNA . miRNA bekerja dengan cara berikatan secara spesifik dengan mRNA target, yang dapat menyebabkan degradasi mRNA atau menghambat proses translasi protein. Mekanisme ini memungkinkan miRNA untuk secara langsung mengontrol banyak proses biologis seperti proliferasi, apoptosis, dan diferensiasi sel, yang semuanya sangat relevan dalam perkembangan dan progresi kanker [18]. Ekspresi miRNA dalam sel kanker menunjukkan pola yang berbeda secara signifikan dibandingkan dengan jaringan normal. Keunggulan lain dari miRNA adalah sifatnya yang stabil dalam sampel biologis seperti darah dan jaringan, karena dapat terlindungi dalam eksosom atau protein pengikat, menjadikannya kandidat biomarker yang ideal [19]. Faktor-faktor tersebut membuat profil ekspresi miRNA memiliki potensi besar sebagai fitur prediktif dalam sistem klasifikasi kanker, baik untuk tujuan diagnosis dini maupun prognosis.

2.4 Differentially Expressed Genes (Limma)

Differentially Expressed Genes (DEG) merupakan pendekatan bioinformatika yang bertujuan untuk mengidentifikasi gen yang memiliki perbedaan ekspresi yang signifikan antara dua atau lebih kondisi biologis yang dibandingkan [20]. Kondisi tersebut dapat berupa kelompok sampel normal dan kanker, stadium awal dan stadium lanjut, atau kelompok pasien yang merespons dan tidak merespons terapi. Analisis DEG dilakukan untuk menemukan gen-gen yang berperan dalam proses biologis tertentu, sehingga dapat dijadikan kandidat biomarker, target terapi, atau dasar pemahaman mekanisme penyakit secara lebih mendalam [20]. Nilai ekspresi gen dan miRNA dapat digunakan sebagai salah satu sumber data untuk melakukan analisis DEG. Analisis DEG umumnya

dilakukan dengan bantuan alat statistik seperti limma, yang sering digunakan untuk menganalisis data RNA-Seq maupun mikroarray [21].

limma (*Linear Models for Microarray Analysis*) dikembangkan dalam lingkungan perangkat lunak R dan tersedia secara luas melalui Bioconductor. Metode limma menggunakan pendekatan berbasis model linear yang diperkuat oleh teknik statistik empirical Bayes untuk menganalisis data ekspresi gen [21].. Pendekatan tersebut memungkinkan limma untuk mengatasi masalah variabilitas dan jumlah sampel yang kecil, yang sering dijumpai dalam studi ekspresi gen. Limma bekerja dengan terlebih dahulu menerapkan model linear untuk setiap gen guna membandingkan ekspresi antar kondisi atau kelompok. Estimasi varians dari setiap gen kemudian diperbaiki menggunakan teknik *empirical bayes*, yang mengakumulasi informasi dari seluruh gen untuk menghasilkan estimasi varians yang lebih stabil, terutama ketika jumlah sampel terbatas. [21]. Hasil analisis limma akan menghasilkan berbagai nilai statistik seperti *adjusted p-value*, *p-value*, *logFC*, dan nilai *t-test*.

2.5 Recursive Feature Elimination

Recursive Feature Elimination (RFE) adalah metode seleksi fitur berbasis pendekatan *wrapper* yang secara iteratif mengeliminasi fitur-fitur dengan kontribusi terendah terhadap kinerja model [22]. Proses kerja RFE dimulai dengan melatih model menggunakan seluruh fitur yang tersedia. Setelah model terbentuk, dihitung tingkat kepentingan atau kontribusi masing-masing fitur terhadap hasil prediksi. Hal ini ditentukan berdasarkan bobot atau koefisien yang dihasilkan dari model tersebut. Selanjutnya, fitur dengan nilai penting terendah akan dieliminasi. Proses pelatihan model dan eliminasi fitur ini diulang secara rekursif hingga tersisa sejumlah fitur yang ditentukan atau dianggap optimal [22]. Secara umum, proses seleksi fitur menggunakan metode RFE dapat direpresentasikan melalui pseudocode berikut [23]:

Algorithm 1: Recursive Feature Elimination (RFE)

Input:

Data pelatihan T

Sekumpulan fitur $F = \{f_1, \ldots, f_p\}$

Model pemeringkatan $M(T, F)$

Jumlah fitur yang diinginkan k

Output: Subset fitur terpilih F'

Inisialisasi $F' \leftarrow$ F;

while *jumlah* $F' > k$ **do**

    Latih model $M(T, F)$;

    Hitung skor kepentingan tiap fitur di F'

    Hapus fitur dengan kepentingan terendah dari F'

return F'

Implementasi RFE dapat dilakukan dengan mudah menggunakan pustaka *Python* seperti *scikit-learn*. Pustaka ini menyediakan fungsi RFE yang memungkinkan pengguna untuk menerapkan proses seleksi fitur secara efisien tanpa perlu menulis algoritma dari awal. Fungsi ini merupakan bagian dari modul *sklearn.feature_selection*. Berikut adalah contoh implementasi dari fungsi RFE [24].

```
from sklearn.feature_selection import RFE

# Terapkan RFE untuk memilih n fitur terpenting
rfe = RFE(estimator, n_features_to_select)

# Fit RFE ke data (X: fitur, y: label)
rfe.fit(X, y)
```

Kode 2.1: Contoh Fungsi RFE

Pada Kode 2.1, fungsi RFE digunakan dengan cara melakukan *import* dan menggunakan fungsi RFE secara langsung. Hyperparameter yang wajib didefinisikan adalah jumlah fitur akhir yang ingin didapatkan. Hyperparameter *estimator* juga dapat digunakan untuk menentukan model pemeringkatan yang digunakan RFE. Setelah model RFE telah diinisialisasi, selanjutnya dapat dilakukan *fitting* model dengan menggunakan input fitur dan label.

2.6 Logistic Regression

Logistic Regression merupakan salah satu algoritma klasifikasi yang digunakan dalam pembelajaran mesin untuk memprediksi probabilitas dari suatu kejadian yang memiliki dua kemungkinan. Metode tersebut merupakan metode analisis statistik yang digunakan untuk menggambarkan hubungan antara satu atau lebih variabel independen (fitur) dengan variabel dependen biner (target). Dalam implementasinya, regresi logistik menghitung kombinasi linier dari fitur-fitur yang tersedia. Setiap fitur dikalikan dengan bobot tertentu yang menunjukkan tingkat pengaruhnya terhadap prediksi [25]. Hasil dari kombinasi linier ini dilambangkan dengan simbol z, yang dirumuskan sebagai berikut [26]:

$$z = \alpha + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_k X_k \tag{2.1}$$

Keterangan:

- α merupakan intercept atau bias
- β_k merupakan konstanta untuk setiap fitur/variabel
- X_k merupakan nilai dari setiap variabel/fitur

Dari persamaan (2.1), nilai z berada di rentang antara (-∞, ∞). Nilai z pada disebut sebagai logit, yaitu hasil dari kombinasi linier antara fitur-fitur (variabel independen) dan bobotnya. Setelah logit dihitung, nilai tersebut dimasukkan ke dalam fungsi sigmoid untuk mengubahnya menjadi probabilitas antara 0 dan 1 [25]. Fungsi sigmoid yang digunakan dalam *logistic regression* disebut sebagai *Logistic Function* dengan rumus sebagai berikut [26]:

$$\hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.2}$$

Hasil dari fungsi logistik tersebut berbentuk sebuah probabilitas dengan nilai antara 0 dan 1. Pada persamaan (2.2), nilai $\hat{y}$ melambangkan probabilitas bahwa sampel termasuk ke kelas 1. Probabilitas kelas positif ($Y = 1$) dapat dihitung dengan menulis ulang fungsi sigmoid (2.2) menjadi rumus berikut:

$$P(Y = 1|X) = \frac{1}{1 + e^{-z}} = \frac{e^z}{1 + e^z} \tag{2.3}$$

Probabilitas kelas negatif ($Y = 0$) adalah:

$$P(Y = 0|X) = 1 - P(Y = 1|X) \tag{2.4}$$

Secara matematis, proses ini dituliskan sebagai [27]:

$$\hat{y} = \begin{cases} 1, & \text{jika } P(Y = 1 \mid X) \geq 0.5 \\ 0, & \text{jika } P(Y = 1 \mid X) < 0.5 \end{cases} \tag{2.5}$$

Dengan kata lain:

- Jika $P(Y = 1 \mid X) \to 1$, maka model sangat yakin bahwa data X termasuk ke dalam kelas 1 (positif).
- Jika $P(Y = 1 \mid X) \to 0$, maka model sangat yakin bahwa data X termasuk ke dalam kelas 0 (negatif).

Parameter α dan β_k pada model *Logistic Regression* dipelajari menggunakan metode *Maximum Likelihood Estimation* (MLE). Dalam pendekatan ini, digunakan suatu fungsi kerugian atau *log loss* untuk menentukan nilai parameter yang memaksimalkan probabilitas data yang ada. Fungsi kerugian tersebut bekerja dengan cara menghitung perbedaan atau jarak antara label sebenarnya dan label yang diprediksi oleh model. Bentuk fungsi kerugian yang digunakan adalah *Binary Cross-Entropy*, yang dirumuskan sebagai berikut [28]:

$$\mathscr{L}(\beta) = -\sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] \tag{2.6}$$

Keterangan:

- N : jumlah sampel
- y_i : label aktual (0 atau 1)
- $\hat{y}_i$: prediksi probabilitas dari model

Untuk menghindari *overfitting*, *Logistic Regression* sering dikombinasikan dengan teknik regularisasi. Terdapat dua teknik utama yang digunakan, yaitu *L1 Regularization (Lasso)* dan *L2 Regularization (Ridge)*. Teknik tersebut bekerja dengan menambahkan penalti terhadap hasil nilai fungsi kerugian. Fungsi regularisasi *Lasso* dan *Ridge* dapat dilihat pada persamaan (2.7) dan (2.8) [28].

$$\mathscr{L}_{L1}(\beta) = \mathscr{L}(\beta) + \lambda \sum_{j=1}^{p} |\beta_j| \tag{2.7}$$

$$\mathscr{L}_{L2}(\beta) = \mathscr{L}(\beta) + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{2.8}$$

Keterangan:

- λ : parameter regularisasi yang mengontrol kekuatan penalti
- $|\beta_j|$: nilai absolut dari koefisien β_j
- β_j^2 : kuadrat dari koefisien β_j

Implementasi LR dapat dilakukan dengan mudah menggunakan pustaka *Python* seperti *scikit-learn*. Pustaka ini menyediakan fungsi LR yang memungkinkan pengguna untuk memakai model LR secara efisien tanpa perlu menulis algoritma dari awal. Fungsi ini merupakan bagian dari modul *sklearn.linear_model*. Berikut adalah contoh implementasi dari fungsi LR [29].

```
from sklearn.linear_model import LogisticRegression

# inisialisasi LR dengan hyperparameter
logreg = LogisticRegression(random_state)

# Fit LR ke data (X: fitur, y: label)
logreg.fit(X, y)
```

Kode 2.2: Contoh Fungsi LR

Pada Kode 2.1, fungsi LR digunakan dengan cara melakukan *import* dan dapat diinisialisasi secara langsung. Terdapat beberapa hyperparameter yang dapat digunakan, yaitu *penalty*, *C* dan *solver*. *Penalty* digunakan untuk menentukan jenis penalti yang akan diterapkan. *C* merupakan invers dari koefisien penalti. *solver* menentukan algoritma optimasi yang digunakan dalam LR untuk menentukan koefisien fitur. Setelah model LR telah diinisialisasi, selanjutnya dapat dilakukan *fitting* model dengan menggunakan input fitur dan label.

2.7 Random Forest

Random Forest merupakan salah satu algoritma yang sangat populer dalam pembelajaran mesin, terutama untuk menyelesaikan masalah klasifikasi dan regresi. Algoritma ini termasuk dalam kategori metode *ensemble learning* metode *bagging*, yaitu teknik membangun beberapa model dasar, dalam hal ini *decision tree*, dan menggabungkan hasil prediksi dari setiap model tersebut. Hasil prediksi untuk klasifikasi diperoleh dengan *voting* mayoritas dari seluruh *decision tree* yang terbentuk. Jika terdapat T buah pohon, maka prediksi akhir $\hat{y}$ untuk suatu sampel x ditentukan dengan [30]:

$$\hat{y} = \text{mode}\{h_1(x), h_2(x), \ldots, h_T(x)\} \tag{2.9}$$

Dengan:

- $h_t(x)$: prediksi dari pohon ke-t untuk data x
- T : jumlah total pohon dalam hutan
- mode : nilai terbanyak dari hasil prediksi seluruh pohon

Tujuan utama dari pendekatan ini adalah untuk meningkatkan performa dalam prediksi dan mengurangi risiko overfitting yang sering terjadi pada model tunggal. Metode *Random Forest* bekerja dengan membangun sejumlah *decision tree* secara acak dan independen, di mana setiap pohon memberikan prediksi berdasarkan subset fitur yang berbeda [30]. Proses pembentukan pohon keputusan pada algoritma ini dapat dijelaskan melalui tahapan-tahapan berikut [31].

Algorithm 2: Random Forest

Input: Data pelatihan D, jumlah pohon n, jumlah total fitur k, jumlah fitur yang dipilih m ($m \ll k$)

Output: Prediksi akhir berdasarkan pemungutan suara (voting) dari seluruh pohon keputusan

```
for setiap pohon i = 1 hingga n do
    Pilih m fitur secara acak dari total k fitur;
    Bangun pohon keputusan berdasarkan subset data dan fitur terpilih;
    for setiap node dalam pohon do
        Hitung titik pemisah terbaik menggunakan kriteria seperti Gini
         atau Entropi;
        Bagi node menjadi anak berdasarkan pemisahan terbaik;
    Simpan model pohon keputusan yang telah selesai dibangun;
Gabungkan hasil prediksi dari seluruh pohon untuk memperoleh
 prediksi akhir menggunakan voting mayoritas;
```

Decision Tree merupakan struktur model prediktif berbentuk seperti pohon yang terdiri dari beberapa cabang dan simpul (node). Setiap *internal node* mewakili suatu pengujian atau kondisi terhadap sebuah fitur , setiap cabang dari pohon merepresentasikan hasil dari pengujian tersebut, dan setiap daun menandai kelas atau nilai prediksi. Dalam pembangunan sebuah *decision tree*, sebuah pemisahan untuk menjadi cabang seringkali terjadi untuk memisahkan antara dua buah kelas. Salah satu metode paling umum untuk mengukur kualitas suatu pemisahan atau *split* dalam decision tree adalah dengan menggunakan *Gini Index*.

Gini Index mengukur ketidakmurnian dari suatu *node*. Semakin rendah nilai Gini, maka semakin murni *node* tersebut. Kemurnian pada *gini index* mengindikasikan seberapa dominan suatu kelas pada *node* tersebut. *Gini index* dapat dihitung menggunakan rumus berikut [32]:

$$Gini(t) = 1 - \sum_{i=1}^{C} p_i^2 \tag{2.10}$$

Keterangan:

- C adalah jumlah total kelas,
- p_i adalah proporsi elemen dari kelas ke-i dalam node t.

Sebelum terjadi pemisahan, sebuah *decision tree* akan menghitung *gini index* dari setiap node yang dihasilkan. Sebuah *decision tree* akan melakukan *split* atau pemisahan cabang pada hasil perhitungan indeks gini yang paling rendah. Dalam hal ini, masing-masing *node* yang dihasilkan akan memiliki kemurnian yang paling tinggi. Kemurnian yang tinggi menunjukkan bahwa setiap node didominasi oleh satu buah kelas, sehingga cabang yang dihasilkan dapat memisahkan kelas dengan baik.

Implementasi RF dapat dilakukan dengan mudah menggunakan pustaka *Python* seperti *scikit-learn*. Pustaka ini memungkinkan pengguna untuk memakai model RF secara efisien tanpa perlu menulis algoritma dari awal. Fungsi ini merupakan bagian dari modul *sklearn.ensemble*. Berikut adalah contoh implementasi dari fungsi RF [33].

```
from sklearn.ensemble import RandomForestClassifier

# inisialisasi RF dengan hyperparameter
rf_model = RandomForestClassifier(random_state)

# Fit RF ke data (X: fitur, y: label)
rf_model.fit(X, y)
```

Kode 2.3: Contoh Fungsi RF

Pada Kode 2.3, fungsi RF digunakan dengan cara melakukan *import* dan dapat diinisialisasi secara langsung. Terdapat beberapa hyperparameter yang dapat digunakan, yaitu *n_estimators*, *max_depth*, *min_samples_split*, *min_samples_leaf*, dan *max_features*. *n_estimators* merupakan jumlah *decision tree* dalam RF. *max_depth* menentukan kedalaman maksimal dari setiap *tree*. *min_samples_split* menentukan jumlah sampel minimal untuk melakukan *split*. *min_samples_leaf* menentukan jumlah sampel minimal yang harus dimiliki setiap *node*. *max_features* menentukan jumlah fitur maksimum yang dipertimbangkan saat membagi setiap *node*. Setelah model RF telah diinisialisasi, selanjutnya dapat dilakukan *fitting* model dengan menggunakan input fitur dan label.

2.8 Metrik Evaluasi

Metrik evaluasi merupakan sebuah alat ukur yang dapat digunakan untuk menilai kinerja suatu model pembelajaran mesin, khususnya pada tugas klasifikasi. Evaluasi model dapat membantu dalam mengetahui sejauh mana model mampu melakukan prediksi yang akurat terhadap data yang belum pernah dilihat

sebelumnya [34]. Dalam konteks klasifikasi, metrik evaluasi digunakan untuk membandingkan hasil prediksi model terhadap label sebenarnya, sehingga dapat diketahui seberapa baik model mengenali pola dari data yang tersedia. Penggunaan metrik evaluasi yang tepat tidak hanya membantu dalam memilih model terbaik, tetapi juga memberikan wawasan tentang kelemahan model. Dengan demikian, metrik evaluasi memiliki manfaat yang signifikan dalam proses pengembangan, penyempurnaan, serta penerapan model pembelajaran mesin dalam dunia nyata.

2.8.1 Matriks Kebingungan (Confusion Matrix)

Matriks kebingungan merupakan tabel silang yang mencatat jumlah prediksi yang benar dan salah dari model klasifikasi. Baris merepresentasikan kelas sebenarnya, dan kolom mewakili prediksi model. Elemen diagonal utama menunjukkan jumlah klasifikasi yang benar, sementara elemen di luar diagonal menunjukkan kesalahan klasifikasi.

Tabel 2.2. Confusion Matrix

Prediksi / Aktual	**Positif (1)**	**Negatif (0)**
Positif (1)	*True Positive (TP)*	*False Positive (FP)*
Negatif (0)	*False Negative (FN)*	*True Negative (TN)*

Dari *confusion matrix*, dapat diketahui nilai *true positive* (TP), *true negative* (TN), *false positive* (FP), dan *false negative* (FN). Nilai *true positive* (TP) menyatakan jumlah data yang diprediksi positif dan sesungguhnya positif, sedangkan *false positive* (FP) menyatakan jumlah data yang diprediksi positif tetapi sesungguhnya negatif. Nilai *true negative* (TN) menyatakan jumlah data yang diprediksi negatif dan sesungguhnya negatif, sedangkan *false negative* (FN) menyatakan jumlah data yang diprediksi negatif tetapi sesungguhnya positif. Dari nilai-nilai tersebut *accuracy*, *precision*, *recall*, *f1-score* dari suatu permodelan dapat diketahui.

2.8.2 Akurasi (Accuracy)

Akurasi adalah metrik yang paling umum digunakan, dan didefinisikan sebagai rasio jumlah prediksi yang benar (*True Positive* + *True Negative*) terhadap total prediksi. Akurasi merupakan indikator yang menunjukkan keakuratan model

untuk memprediksi label secara keseluruhan. Dalam *confusion matrix*, akurasi dapat dihitung dengan menjumlahkan semua elemen diagonal utama dan kemudian dibagi dengan jumlah total seluruh elemen. Akurasi dihitung menggunakan rumus [34]:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.11}$$

2.8.3 Precision

Precision merupakan salah satu metrik evaluasi yang digunakan untuk mengukur ketepatan prediksi positif dari suatu model klasifikasi. Metrik ini menunjukkan proporsi dari prediksi positif yang benar-benar relevan atau tepat, yaitu berapa banyak dari semua sampel yang diprediksi sebagai kelas positif oleh model yang memang benar-benar termasuk dalam kelas tersebut. *Precision* dihitung dengan rumus [34]:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.12}$$

2.8.4 Recall

Recall juga dikenal sebagai sensitivitas atau true positive rate. *Recall* merupakan metrik evaluasi yang digunakan untuk mengukur seberapa baik model klasifikasi mampu mendeteksi seluruh data yang benar-benar termasuk dalam kelas positif. Metrik ini menunjukkan proporsi dari total kasus positif yang berhasil diprediksi dengan benar oleh model. *Recall* dihitung dengan rumus sebagai berikut [34]:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.13}$$

2.8.5 F1-Score

F1-Score adalah metrik evaluasi yang digunakan untuk mengukur kinerja model klasifikasi dengan mempertimbangkan keseimbangan antara *precision* dan *recall*. Metrik ini sangat berguna ketika terdapat ketidakseimbangan kelas dalam data, yaitu ketika jumlah sampel pada satu kelas jauh lebih banyak dibandingkan kelas lainnya. F1-*Score* dihitung sebagai rata-rata harmonik dari *precision* dan *recall*, dengan rumus sebagai berikut [34]:

$$\text{F1-Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.14}$$

2.8.6 Receiver Operating Characteristic (ROC)

Receiver Operating Characteristic (ROC) adalah kurva yang menggambarkan performa model klasifikasi pada ambang batas keputusan. Kurva ROC dibuat dengan memplot nilai *True Positive Rate* (TPR) terhadap *False Positive Rate* (FPR) pada berbagai nilai threshold [35]. Metrik ini memungkinkan evaluasi sensitivitas model terhadap perubahan threshold dan sangat berguna untuk analisis performa pada masalah klasifikasi biner. Model yang ideal akan menghasilkan kurva ROC yang melengkung tajam ke kiri atas (mendekati TPR = 1 dan FPR = 0), menunjukkan deteksi positif yang tinggi dan kesalahan prediksi positif yang rendah.Nilai TPR dan FPR dapat dihitung menggunakan rumus berikut [35]:

$$\text{TPR} = \frac{TP}{TP + FN} \tag{2.15}$$

$$\text{FPR} = \frac{FP}{FP + TN} \tag{2.16}$$

2.8.7 Area Under Curve (AUC)

Area Under Curve (AUC) adalah skor yang merepresentasikan luas di bawah kurva ROC. Nilai AUC berkisar antara 0 hingga 1, dengan nilai yang lebih

mendekati 1 menunjukkan performa model yang lebih baik dalam membedakan antara kelas positif dan negatif. Metrik AUC-ROC sangat berguna ketika data tidak seimbang, karena metrik ini tidak hanya mempertimbangkan prediksi benar, tetapi juga seberapa baik model membedakan kelas. Berikut adalah tabel interpretasi nilai AUC [36].

Tabel 2.3. Interpretasi Nilai AUC

Nilai AUC	**Interpretasi**
$0.9 \leq \text{AUC} \leq 1.0$	Sangat baik
$0.8 \leq \text{AUC} < 0.9$	Baik
$0.7 \leq \text{AUC} < 0.8$	Cukup
$0.6 \leq \text{AUC} < 0.7$	Kurang
$0.5 \leq \text{AUC} < 0.6$	Gagal