BAB 2
LANDASAN TEORI

2.1 Pengertian Jaringan Komputer

Jaringan komputer merupakan salah satu infrastruktur penting dalam sistem komunikasi digital yang memungkinkan pertukaran data antara perangkat-perangkat komputasi melalui media transmisi kabel maupun nirkabel. Fungsi utama dari jaringan komputer adalah mendukung komunikasi, kolaborasi, dan pemrosesan data agar menjadi lebih terdistribusi, baik dalam skala lokal maupun global. Konsep jaringan komputer mulai berkembang sejak tahun 1960 melalui proyek ARPANET, yang kemudian menjadi landasan bagi internet modern [13]. Dalam implementasinya, jaringan komputer dibangun berdasarkan protokol komunikasi seperti TCP/IP yang mengatur pengiriman dan penerimaan data secara andal. Jaringan ini bisa terdiri dari berbagai topologi dan jenis, seperti LAN (*Local Area Network*), MAN (*Metropolitan Area Network*), dan WAN (*Wide Area Network*) [14].

Seiring meningkatnya jumlah perangkat, volume data, dan kompleksitas lalu lintas jaringan, permasalahan seperti kemacetan jaringan (*network congestion*) menjadi tantangan utama dalam pengelolaan infrastruktur jaringan. Kemacetan terjadi ketika kapasitas jalur komunikasi tidak mampu menangani beban data yang melewati jaringan, yang dapat menyebabkan keterlambatan (*latency*), hilangnya paket data (*packet loss*), atau bahkan kegagalan layanan. Dalam hal ini, pengelolaan dan prediksi kemacetan menjadi hal yang krusial untuk menjaga performa dan keandalan jaringan.

Berbagai pendekatan telah digunakan untuk memantau dan menangani kemacetan jaringan, termasuk metode tradisional berbasis aturan serta pendekatan statistik. Namun, seiring dengan perkembangan teknologi, pendekatan berbasis *machine learning* mulai banyak digunakan karena kemampuannya dalam mengidentifikasi pola-pola kompleks pada data jaringan secara otomatis.

2.2 Kemacetan Jaringan atau Network Congestion

Kemacetan *traffic* jaringan merupakan kondisi ketika volume data yang melewati suatu jaringan melebihi kapasitas penanganan saluran transmisi atau *node*, seperti *switch* dan *router*. Hal ini dapat dianalogikan seperti pada kemacetan lalu

lintas di jalan raya, di mana terlalu banyak kendaraan menyebabkan antrean panjang dan keterlambatan. Dalam konteks jaringan komputer, kemacetan menyebabkan keterlambatan pengiriman paket data (*latency*), hilangnya paket (*packet loss*), penurunan *throughput*, serta penurunan kualitas layanan secara keseluruhan [1].

Kemacetan sering terjadi pada titik-titik kritis jaringan seperti *router* atau *buffer switch* yang mengalami antrean panjang akibat *buffer overflow*. Paket yang tidak tertangani tepat waktu akan dibuang atau dikirim ulang, yang pada akhirnya akan menambah beban pada jaringan. Sehingga hal ini menjadi sangat krusial pada layanan *real-time* seperti *video conference*, *autonomous vehicles*, atau layanan perbankan, di mana keterlambatan atau kehilangan data dapat berakibat fatal.

Pada arsitektur jaringan modern seperti *Software-Defined Networking* (SDN), kemacetan bersifat lebih dinamis dan kompleks. Karena pengendalian jalur data dipisahkan dari perangkat keras, lalu lintas dapat dialihkan secara fleksibel. Namun, hal ini juga memungkinkan kemacetan yang bersifat asimetris, misalnya satu jalur *uplink* mengalami beban hingga 95% sementara jalur *downlink* hanya 40% [13]. Dari hal ini maka akan dapat menyebabkan penumpukan lalu lintas secara tidak proporsional, sehingga dapat meningkatkan risiko *bottleneck*.

Dampak yang disebabkan oleh kemacetan jaringan juga berpengaruh pada sistem ekonomi. Menurut laporan AWS pada tahun 2023, satu menit *downtime* pada pusat data akibat kemacetan dapat menyebabkan kerugian rata-rata sebesar $5.600 bagi usaha kecil dan menengah (UMKM) [15]. Oleh karena itu, kebutuhan akan sistem klasifikasi yang dapat membantu mendeteksi dan memprediksi menjadi sangat mendesak.

Salah satu pendekatan teknis untuk mengatasi kemacetan adalah melalui analisis dan pengendalian pola lalu lintas (*traffic shaping* dan *traffic engineering*), yang mengatur aliran data agar lebih seimbang [16]. Namun, pendekatan manual dan berbasis aturan sering kali tidak mampu mengantisipasi dinamika lalu lintas yang sangat cepat dan kompleks pada jaringan modern. Oleh karena itu, pendekatan berbasis *machine learning* menjadi solusi yang semakin relevan untuk mendeteksi dan memprediksi kemacetan secara otomatis dan adaptif.

2.2.1 Algoritma Kontrol Kemacetan dan Estimasi

Untuk mencegah atau mengurangi kemacetan, berbagai algoritma *congestion control* telah dikembangkan. Salah satu algoritma klasik yang banyak digunakan adalah TCP Tahoe, yang menggunakan mekanisme *slow start*,

congestion avoidance, dan *fast retransmit* untuk mendeteksi dan menangani kemacetan. Ketika kehilangan paket terdeteksi (sebagai indikator kemacetan), TCP Tahoe secara drastis menurunkan laju pengiriman (*congestion window*) dan memulai kembali dari fase *slow start* [1]. Seiring berkembangnya jaringan modern, berbagai pendekatan berbasis *machine learning* (ML) telah diaplikasikan untuk mengendalikan kemacetan secara adaptif dan real-time, menggantikan metode tradisional yang berbasis *rule*. Penelitian oleh Tafa dan Milutinovic meninjau beberapa algoritma ML populer dalam *congestion control* yang mampu menyesuaikan strategi kontrol berdasarkan kondisi jaringan secara dinamis [3].

Selain itu, penelitian oleh Toopchinezhad dan Ahmadi mempresentasikan pendekatan ML untuk *Active Queue Management* (AQM), di mana kemacetan diprediksi sebelum terjadi *buffer overflow*, dan kebijakan paket dibuang disesuaikan untuk menjaga kualitas layanan [17].

Pada implementasi TCP BBR Google, Mhaske et al. memanfaatkan ML untuk memperbaiki *fairness* jaringan (*inter-flow fairness*), dengan mengestimasi *bottleneck bandwidth* dan RTT secara adaptif [2]. Ini menggambarkan bagaimana parameter kemacetan dapat diprediksi dari data historis dan lalu lintas jaringan. Penelitian lainnya menyebutkan bahwa algoritma terbaru TCP QtColFair menggunakan teori antrean (*Little's Law*) untuk menghitung laju pengiriman optimal, mengurangi *delay* antrean dan kehilangan paket lebih baik dari TCP BBR maupun CUBIC dalam kondisi *buffer* besar [18]..

Estimasi kemacetan dalam jaringan umumnya dilakukan melalui indikator metrik performa seperti *jitter*, *latency*, *packet loss*, dan *throughput*. Pola kemacetan dapat dikenali dari kombinasi kenaikan *delay* (*latency*), fluktuasi waktu pengiriman (*jitter*), serta penurunan *throughput* secara signifikan. Dalam konteks modern, estimasi ini juga dilakukan dengan bantuan *machine learning*, di mana pola kemacetan dipelajari dari data historis dan fitur jaringan, sehingga model dapat mengenali kondisi "macet" secara otomatis tanpa perlu mendefinisikan aturan eksplisit.

2.3 Parameter dan Threshold Quality of Service

Berdasarkan *Cisco QoS Handbook*, batas ideal untuk parameter QoS pada layanan suara dan video meliputi *latency* maksimum sebesar 150 ms, *jitter* maksimum 30 ms, serta *packet loss* maksimum 1%. Batas ini menjadi acuan dalam klasifikasi kondisi jaringan, termasuk dalam identifikasi kondisi macet atau

tidak macet [19]. Pada *Throughput* sendiri tidak ditunjukkan secara eksplisit pada ITU dan Cisco, namun pada penelitian ini menggunakan indeks *throughput* dari *Telecommunications and Internet Protocol Harmonization Over Networks* (TIPHON) seperti pada penelitian [20] yang mengatakan bahwa jika indeks pada *throughput* kurang dari 2 atau di bawah 50% menunjukkan kualitas layanan yang jelek. Berikut Tabel 2.1 akan menunjukkan batas-batas ideal dari masing-masing parameter.

Tabel 2.1. Parameter dan *Threshold Quality of Service*

Parameter	**Notasi**	**Threshold**
DLatency	L	> 150 ms
Jitter	J	> 30 ms
Packet Loss	P	$> 1\%$
Throughput Ratio	$\frac{T}{B}$	< 0.5

2.4 Koefisien Korelasi Pearson

Koefisien korelasi Pearson (r) adalah koefisien korelasi yang paling banyak digunakan dan dikenal dengan banyak nama. Koefisien korelasi Pearson merupakan statistik deskriptif, yang berarti merangkum karakteristik suatu kumpulan data. Secara spesifik, koefisien ini menggambarkan kekuatan dan arah hubungan linear antara dua variabel kuantitatif. Tabel berikut menjelaskan interpretasi umum dari nilai koefisien korelasi Pearson (r) [21]:

Tabel 2.2. Nilai Koefisien Korelasi Pearson

Nilai Korelasi	**Interpretasi**	**Arah**
> 5	Sangat Kuat	Positif
Antara 3 – 5	Sedang	Positif
Antara 0 – 3	Lemah	Positif
0	Tidak Berkolerasi	None
Antara 0 – (-3)	Lemah	Negatif
Antara (-3) – (-5)	Sedang	Negatif
> 5	Kuat	Negatif

2.5 Machine Learning dalam Klasifikasi

Klasifikasi adalah salah satu tugas utama dalam *machine learning*, yaitu proses pemetaan dari data masukan (fitur) ke dalam label kelas yang telah didefinisikan sebelumnya. Dalam konteks jaringan komputer, klasifikasi digunakan untuk mengategorikan kondisi jaringan ke dalam kelas seperti ”macet” atau ”tidak macet” berdasarkan karakteristik metrik lalu lintas, seperti *delay*, *jitter*, *packet loss*, *throughput*, dan jumlah koneksi aktif. Contoh sederhana dari proses ini adalah ketika sistem menerima data masukan seperti `[latency: 150ms, packet loss: 12%, throughput: 2Mbps]` dan memetakan data tersebut ke dalam label ”macet”, namun hal pentingnya adalah bahwa ini hanya melanggar parameter dari *quality of service*, karena untuk memetakan suatu ’macet’ secara tepat lebih kompleks dari contoh di atas. Model *machine learning* akan dilatih menggunakan data historis yang telah memiliki label, sehingga mampu mengenali pola tertentu yang mengindikasikan kemacetan. Teknik klasifikasi yang umum digunakan dalam kasus ini antara lain adalah *decision tree, random forest, naive bayes, support vector machine* (SVM), dan algoritma *ensemble* seperti *gradient boosting*.

Salah satu keunikan dalam penerapan *machine learning* pada jaringan komputer adalah sifat data lalu lintas yang bersifat *temporal-spatial*. Artinya, pola kemacetan tidak hanya tergantung pada nilai metrik saat ini, tetapi juga pada konteks waktu dan lokasi. Selain itu, tantangan yang signifikan dalam klasifikasi lalu lintas jaringan adalah adanya fenomena *concept drift*, yaitu perubahan distribusi data dari waktu ke waktu. Evolusi perangkat jaringan, perubahan kebijakan *routing*, serta perilaku pengguna yang berubah dapat menyebabkan pola lalu lintas berubah juga, sehingga model yang dilatih pada data lama akan menjadi tidak lagi akurat [22]. Oleh karena itu, dalam penerapannya, model klasifikasi perlu dirancang agar adaptif, misalnya dengan menggunakan pembaruan berkala, *online learning*, atau teknik deteksi dan adaptasi terhadap *drift* [22]. Dengan pendekatan klasifikasi yang tepat, sistem *machine learning* dapat berfungsi sebagai alat prediktif yang kuat untuk mendeteksi potensi kemacetan sebelum berdampak nyata dengan fokus pada peningkatan akurasi dan efisiensi komputasi [23].

2.6 Random Forest

Algoritma *random forest* terinspirasi dari konsep ”kebijaksanaan kerumunan” (*wisdom of crowds*) dalam statistik, di mana agregasi banyak

model sederhana (pohon keputusan) menghasilkan prediksi yang lebih akurat dibandingkan model tunggal, dengan kata lain *random forest* adalah sebuah metode gabungan yang menggabungkan beberapa pohon keputusan agar menghasilkan hasil yang lebih akurat [24]. Mekanisme kerja melibatkan dua tingkat acak yakni *bootstrap sampling* untuk data *training* tiap pohon, dan seleksi *subset* fitur secara acak pada setiap *split node*. Dalam implementasi *Python Scikit-learn*, parameter kunci seperti `n_estimators=100` (jumlah pohon) dan `max_depth=10` mengontrol kompleksitas model [24]. *Random forest* adalah algoritma pembelajaran mesin berbasis *ensemble learning* yang dibangun dari banyak *decision tree* untuk menghasilkan hasil prediksi yang lebih akurat dan stabil. Algoritma ini dikembangkan oleh Breiman pada tahun 2001 [25], dan menggabungkan teknik *bagging* (*bootstrap aggregating*) serta seleksi fitur acak untuk meningkatkan performa dan mengurangi *overfitting*.

Dalam implementasinya, *random forest* membuat banyak pohon keputusan dari *subset* data latih yang berbeda (dihasilkan dari *bootstrap sampling*), lalu melakukan *voting* mayoritas (untuk klasifikasi) atau rata-rata (untuk regresi) dari seluruh pohon [12]. Secara umum, prediksi *random forest* ($\hat{f}(x)$) pada input x dirumuskan sebagai berikut:

$$\hat{f}(x) = \frac{1}{B} \sum_{b=1}^{B} T_b(x) \tag{2.1}$$

dengan:

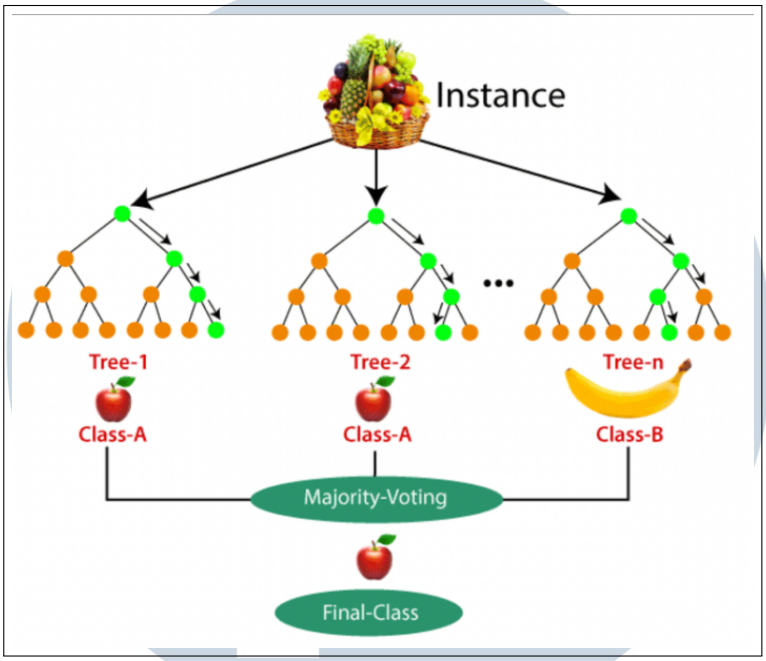
1. B adalah jumlah pohon dalam hutan,
2. $T_b(x)$ adalah hasil prediksi dari pohon ke-b,
3. Untuk klasifikasi, digunakan voting mayoritas:

$$\hat{y} = \text{mode}\{T_1(x), T_2(x), \ldots, T_B(x)\} \tag{2.2}$$

4. Untuk regresi, digunakan rata-rata seperti pada rumus di atas.

Setiap pohon dibangun berdasarkan *subset* acak dari data pelatihan, dan pada setiap *node* dilakukan pemilihan *subset* acak dari fitur (fitur acak) untuk proses

pemisahan terbaik. Hal ini meningkatkan keragaman antar pohon dan membantu mengurangi korelasi antar pohon [25].



Gambar 2.1. Contoh *Random Forest Tree*
Sumber: [26]

2.7 Unsupervised Learning (K-means)

Machine Learning memiliki dua pendekatan utama dalam klasifikasi, yakni *supervised learning* dan *unsupervised learning*. *Random forest* merupakan algoritma *ensemble* untuk *supervised learning*, sedangkan *K-Means* adalah metode *clustering* yang umum digunakan dalam *unsupervised learning* [27]. Pada penelitian ini menggunakan *unsupervised learning* untuk membantu pengecekan pada *dataset* yang bertujuan untuk mengetahui apakah *dataset* mudah untuk dipisahkan menjadi dua kelompok 'macet' dan 'tidak macet', karena *unsupervised learning* adalah salah satu pendekatan *machine learning* yang dapat belajar dari data yang tidak memiliki label, yang bertujuan untuk menemukan pola tersembunyi pada data tanpa adanya informasi eksplisit [28].

Unsupervised learning adalah pendekatan pembelajaran mesin yang digunakan ketika label *output* tidak tersedia. Tujuan utamanya adalah untuk menemukan pola atau struktur dalam data tanpa informasi sebelumnya tentang kelas atau kategori data tersebut. Salah satu metode *unsupervised learning* yang populer

digunakan adalah *k-means*. *k-means* adalah algoritma partisi yang bertujuan untuk membagi data ke dalam K klaster berdasarkan kedekatan pada fiturnya, tujuan dari *k-means* adalah untuk meminimalkan *intra-cluster variance* [29].

K-Means adalah metode *exclusive clustering* yang digunakan untuk membagi data menjadi (*k*) *cluster* berdasarkan jarak ke pusat *cluster* (*centroid*). Proses algoritma ini adalah sebagai berikut [27, 12]:

1. Menentukan jumlah *cluster* (*k*).
2. Menginisialisasi titik *centroid* secara acak.
3. Mengelompokkan setiap data ke *cluster* dengan jarak terdekat ke *centroid*, biasanya menggunakan *Euclidean Distance*:

$$D(x,c) = \sqrt{(x_1 - c_1)^2 + (x_2 - c_2)^2 + \cdots + (x_n - c_n)^2} \tag{2.3}$$

4. Memperbarui *centroid* tiap *cluster* menggunakan rata-rata:

$$v_{ij} = \frac{1}{N_i} \sum_{k=1}^{N_i} x_{kj} \tag{2.4}$$

di mana:

(a) v_{ij} = *centroid* untuk *cluster* ke-i dan variabel ke-j

(b) N_i = jumlah data pada *cluster* ke-i

(c) x_{kj} = nilai variabel ke-j dari data ke-k di *cluster* ke-i

Kelebihan *K-Means* adalah kesederhanaan dan kecepatan komputasinya, namun memiliki kekurangan seperti kepekaan terhadap pemilihan *centroid* awal dan harus menentukan (*k*) secara eksplisit [27].

2.8 Hyperparameter Tuning

Hyperparameter tuning adalah sebuah proses untuk mengatur parameter yang tidak dipelajari oleh model *machine learning* namun akan mempengaruhi performa di akhir, contoh analogi sederhana, proses ini mirip saat menyetel mesin pada mobil balap, di mana diperlukannya usaha untuk menemukan kombinasi

optimal antara kompresi mesin (`max_depth`), jumlah silinder (`n_estimators`), dan rasio bahan bakar (`max_features`) agar mencapai performa maksimal. Dari analogi di atas, dapat dikatakan dalam konteks *random forest*, parameter seperti `n_estimators` (jumlah pohon), `max_depth` (kedalaman maksimal pohon), dan `max_features` (jumlah fitur yang dipertimbangkan dalam split) sangat penting. Sehingga jika terdapat suatu kesalahan dalam penyesuaian maka akan dapat menyebabkan *overfitting* (model terlalu kompleks) atau *underfitting* (model terlalu sederhana). Teknik optimasi seperti *grid search* melakukan eksplorasi sistematis, sedangkan *random search* lebih efisien untuk ruang parameter berdimensi tinggi [30].

2.9 GridSearchCV

GridSearchCV adalah implementasi Scikit-learn untuk melakukan pencarian parameter optimal dengan melakukan *brute-force*. Metode ini mencoba seluruh kombinasi dari nilai parameter yang didefinisikan oleh pengguna dan mengevaluasinya menggunakan *stratified k-fold cross-validation* untuk menghindari *overfitting* dan meningkatkan generalisasi. Setiap kombinasi divalidasi dengan *stratified 5-fold cross validation* untuk mengurangi variansi evaluasi. Salah satu kelemahan dari metode ini adalah meningkatkan waktu komputasi lebih lama, hal ini dikarenakan waktu komputasi meningkat secara eksponensial terhadap jumlah kombinasi dari parameter, misalnya jika adanya penambahan satu parameter dengan 5 nilai baru, maka akan terjadi peningkatan sebanyak 5 kali lipat [31].

2.10 Normalisasi Data

Dalam proses pemodelan *machine learning*, perbedaan skala antar fitur dapat menyebabkan ketidakseimbangan kontribusi fitur terhadap hasil pelatihan. Oleh karena itu, dilakukan pra-pemrosesan berupa normalisasi atau standarisasi untuk menyelaraskan skala fitur agar dapat memberikan kontribusi yang setara dalam pembelajaran model.

MinMaxScaler adalah sebuah metode normalisasi data yang menghitung skala fitur ke dalam rentang [0, 1]. Normalisasi ini penting untuk memastikan bahwa semua fitur memiliki kontribusi yang seimbang dalam proses pelatihan model. Pada konteks penelitian ini, digunakan metode *MinMaxScaler*, salah

satu teknik normalisasi yang paling umum digunakan. Rumus normalisasi *MinMaxScaler* dituliskan sebagai berikut: 2.5:

$$X_{\text{norm}} = \frac{X - X_{\text{min}}}{X_{\text{max}} - X_{\text{min}}} \tag{2.5}$$

2.11 Standarisasi

Standarisasi bertujuan untuk mengubah distribusi data sehingga memiliki rata-rata 0 dan standar deviasi 1, berdasarkan prinsip *Z-Score scaling* dirumuskan sebagai berikut:

$$X_{\text{std}} = \frac{X - \mu}{\sigma} \tag{2.6}$$

Teknik ini lebih cocok untuk fitur yang memiliki distribusi normal (*Gaussian*) dan tidak memiliki *outlier* ekstrem [32]. Namun, jika digunakan pada data yang *skewed* atau mengandung *outlier*, standarisasi dapat menghasilkan hasil yang bias karena rata-rata dan standar deviasi dapat terpengaruh secara signifikan.

2.12 Data Skewness

Skewness adalah ukuran asimetri distribusi data terhadap nilai rata-ratanya. Distribusi disebut:

1. *Right-skewed* jika banyak nilai rendah dan sedikit nilai tinggi (contoh: latency tinggi),
2. *Left-skewed* jika banyak nilai tinggi dan sedikit nilai rendah.

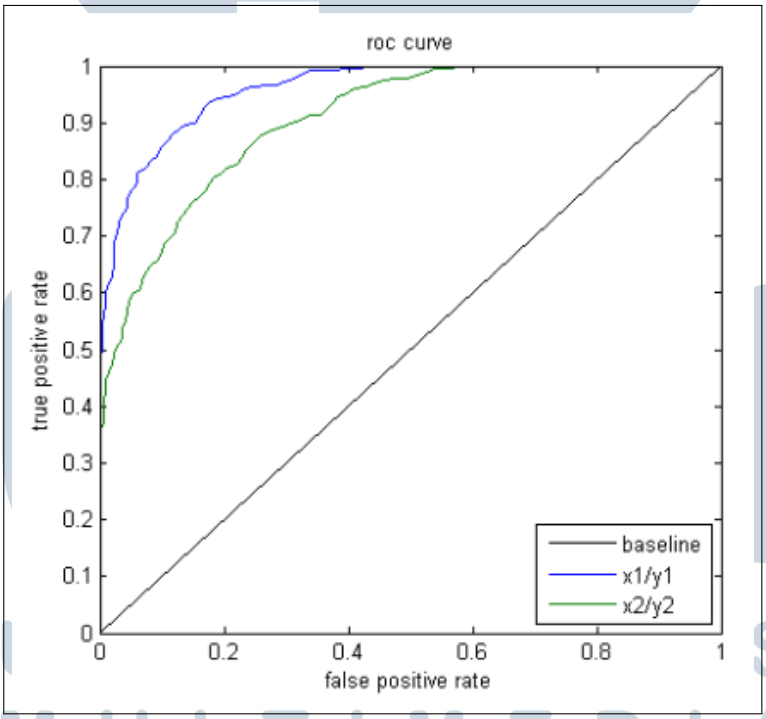
Distribusi *skewed* umum ditemukan dalam data jaringan, karena hal tersebut jika terdapat data yang *skewed* tidak cocok untuk menggunakan standarisasi [33]. Sehingga pilihan yang tepat adalah menggunakan normalisasi *MinMaxScaler*, atau menggunakan transformasi seperti log atau *Box-Cox* terlebih dahulu.

2.13 Threshold Tuning dan ROC Curve

Threshold tuning adalah penyesuaian ambang batas probabilitas untuk memutuskan prediksi kelas. Dengan mengubah *threshold*, kita bisa mengatur *trade-off* antara *precision* dan *recall*. ROC *Curve* (*Receiver Operating Characteristic*)

adalah plot antara *true positive rate* dan *false positive rate* untuk berbagai *threshold*, sedangkan AUC (*Area Under Curve*) menunjukkan seberapa baik model dapat membedakan antar kelas. Dalam beberapa studi literatur, *threshold* klasifikasi *default* 0.5 tidak selalu optimal untuk semua kasus. Pada aplikasi deteksi kemacetan yang mengutamakan keamanan (*fail-safe*), *threshold* dapat diturunkan ke 0.3 untuk meningkatkan *recall* meski akan mengorbankan *precision* menjadi turun. ROC *curve* memvisualisasikan *trade-off*, di mana area di bawah kurva (AUC) di atas 0.9 menunjukkan model dapat membedakan kelas dengan baik di berbagai *threshold* [34].

ROC *Curve* adalah kurva yang memetakan hubungan antara *True Positive Rate* (TPR atau *Recall*) dan *False Positive Rate* (FPR) di berbagai nilai *threshold*. Semakin tinggi area di bawah kurva ROC (*Area Under Curve* AUC), semakin baik model dalam membedakan kelas positif dan negatif. ROC sangat cocok digunakan pada *dataset* yang relatif seimbang [35].

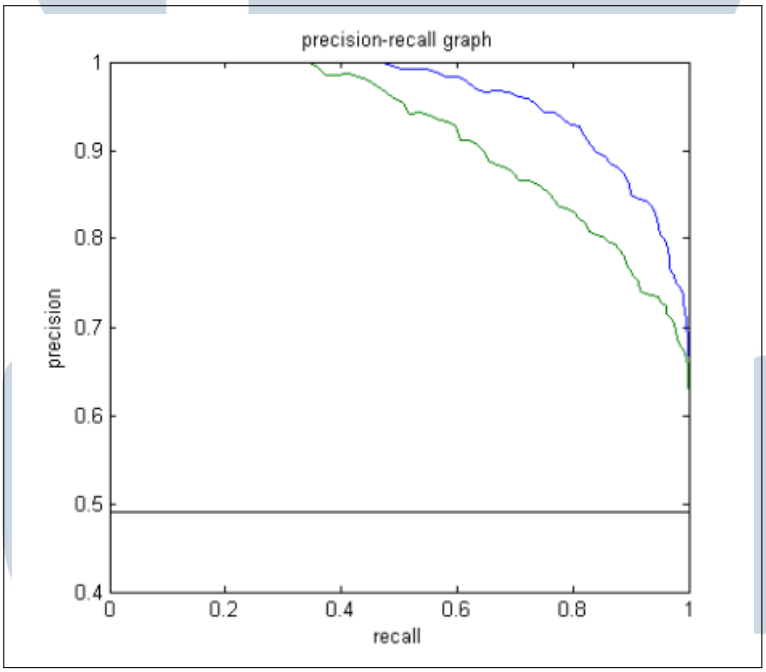


Gambar 2.2. Contoh Gambar Visualisasi ROC *Curve*

Sumber: [36]

2.14 Precision-Recall Curve

Precision-Recall Curve menggambarkan hubungan antara *precision* dan *recall* untuk berbagai *threshold*. *Precision* mengukur proporsi prediksi positif yang benar, sedangkan *recall* mengukur proporsi data positif yang terdeteksi. PR *Curve* lebih informatif dibanding ROC pada *dataset* yang tidak seimbang, karena tidak memperhitungkan *True Negatives* [37]. Ini berguna terutama dalam kasus data yang tidak seimbang, karena lebih fokus pada performa terhadap kelas minoritas. Berbeda dengan ROC *curve* yang bisa mengakibatkan bias pada data tidak seimbang, *Precision-Recall curve* lebih informatif dengan memfokuskan pada performa kelas minoritas. Pada studi lanjutan, *Under* PR *Curve* (AUPRC) di atas 0.75 menunjukkan bahwa di berbagai *threshold*, model mempertahankan *precision* dan *recall* yang seimbang [34].



Gambar 2.3. Contoh Gambar Visualisasi PR *Curve*
Sumber: [36]

2.15 Overfitting dalam Machine Learning

Overfitting merupakan salah satu masalah utama dalam pengembangan model *machine learning*. *Overfitting* terjadi ketika model terlalu menyesuaikan diri terhadap data pelatihan, sehingga ia kehilangan kemampuan generalisasi terhadap data baru yang tidak dikenalnya. *Overfitting* terjadi ketika model terlalu

"menghafal" data pelatihan sehingga gagal memprediksi data uji atau data dari dunia nyata secara akurat. Akibatnya, model memiliki akurasi tinggi pada data latih namun menunjukkan performa rendah pada data baru [38].

2.15.1 Penyebab Overfitting

Beberapa faktor umum penyebab *overfitting* antara lain:

1. Model terlalu kompleks, misalnya *random forest* dengan jumlah pohon yang terlalu banyak atau kedalaman pohon yang besar [39, 40].
2. Ukuran *dataset* terlalu kecil, sehingga model cenderung mempelajari *noise* atau pola kebetulan yang tidak merepresentasikan populasi secara umum [40].
3. Adanya *noise* pada data, yang dianggap sebagai informasi penting oleh model padahal sebetulnya merupakan gangguan [39].

2.15.2 Bias–Variance Trade-off

Overfitting berkaitan erat dengan konsep bias–*variance trade-off*. Model yang terlalu kompleks cenderung memiliki *variance* tinggi dan *bias* rendah. Hal ini membuat model sangat fleksibel namun sensitif terhadap fluktuasi data pelatihan, sehingga menghasilkan performa buruk pada data baru [39].

2.15.3 Dampak Overfitting

Overfitting memiliki dampak sebagai berikut:

1. Error pada data latih sangat rendah, namun *error* pada data uji tinggi.
2. Kemampuan generalisasi menurun, sehingga kurang dapat diandalkan pada data dunia nyata.
3. Metrik evaluasi tidak dapat dipercaya, karena model terlihat sangat baik di *training* set, tetapi buruk saat *deployment*.

2.15.4 Cara Mendeteksi Overfitting

Tanda-tanda *overfitting* antara lain:

1. Performa model pada *training* set jauh lebih tinggi dibanding *test* set.
2. Kurva *loss* pada data validasi mulai meningkat sementara kurva *loss* data latih terus menurun (divergen) [38, 41].

2.15.5 Upaya Pencegahan Overfitting

Beberapa strategi umum untuk mencegah *overfitting* meliputi:

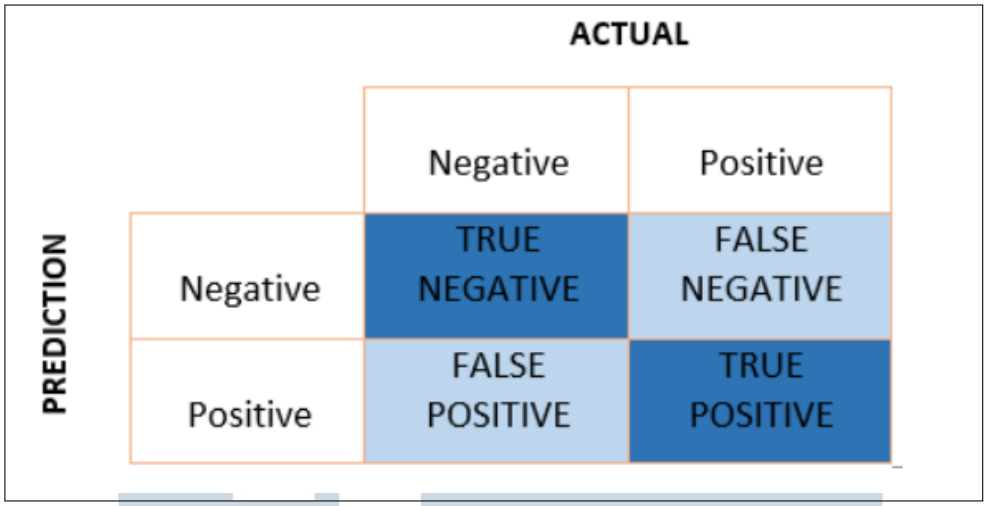
1. Regularisasi, seperti L1 dan L2, atau penggunaan metode *early stopping* [41].
2. Cross-validation, untuk memastikan bahwa model mampu bekerja baik pada berbagai subset data [40].
3. Pruning atau pembatasan kedalaman pohon pada model *decision tree* seperti *random forest*.
4. Penambahan data pelatihan, atau penggunaan *ensemble learning* seperti *bagging* yang mampu mengurangi variansi, dan juga pengujian lebih lanjut menggunakan *unseen dataset* [42].

2.16 Confusion Matrix

Confusion matrix adalah *matrix* 2×2 untuk klasifikasi biner dengan nilai aktual pada satu sumbu dan nilai yang diprediksi pada sumbu lainnya [43].

1. TP (*True Positive*) adalah jumlah data positif yang diprediksi sebagai positif,
2. TN (*True Negative*) adalah jumlah data negatif yang diprediksi sebagai negatif,
3. FP (*False Positive*) adalah jumlah data negatif yang diprediksi sebagai positif,
4. FN (*False Negative*) adalah jumlah data positif yang diprediksi sebagai negatif.

Gambar di bawah ini menunjukkan contoh *confusion matrix*.



Gambar 2.4. Contoh *Confusion Matrix*

2.17 Metrik Evaluasi

Dalam penelitian ini, untuk mengukur performa model digunakan beberapa metrik evaluasi yang umum pada tugas klasifikasi, yaitu *accuracy*, *precision*, *recall*, dan *F1-score* [43]. Metrik ini penting untuk mengevaluasi kualitas dari sebuah model yang dibangun, terutama jika terdapat kasus *dataset* yang tidak seimbang.

1. *Accuracy* menggambarkan seberapa akurat model dalam mengklasifikasikan dengan benar dengan rumus sebagai berikut:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.7}$$

2. *Precision* menggambar semua prediksi positif, berapa persen yang benar-benar positif dibandingkan dengan seluruh prediksi positif dan dapat dirumuskan sebagai berikut:

$$Precision = \frac{TP}{TP + FP} \tag{2.8}$$

3. *Recall*, atau sensitivitas, menggambarkan dari total positif, berapa persen yang diprediksi positif atau keberhasilan model dalam menemukan kembali sebuah informasi. *Recall* dapat dirumuskan sebagai berikut:

$$Recall = \frac{TP}{TP + FN} \tag{2.9}$$

4. Sementara itu, *F1-score* menggambarkan rata-rata harmonik dari *Precision* dan *Recall* bertujuan untuk memberikan keseimbangan antara keduanya. *F1-score* dapat dirumuskan sebagai berikut:

$$F1score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.10}$$