BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terdahulu

Sari (2024) membandingkan akurasi algoritma *Random Forest* dan *K-Nearest Neighbors (KNN)* dalam memprediksi kelulusan mahasiswa Fakultas Teknik. Penelitian ini memanfaatkan variabel jumlah IPS, SKS semester 1–8, total SKS, dan IPK. Hasil analisis menunjukkan bahwa *Random Forest* memiliki akurasi tertinggi, yakni 100%, sementara *KNN* mencapai 96,42%, sehingga *Random Forest* menjadi model paling akurat dalam menentukan kelulusan mahasiswa.

Labib Mu'tashim dkk. (2023) meneliti klasifikasi ketepatan lama studi mahasiswa Fakultas Ilmu Komputer UPN Veteran Jakarta menggunakan algoritma *Random Forest* dan *Gradient Boosting*. Variabel yang digunakan meliputi program studi, jenis kelamin, jenis sekolah, provinsi sekolah, jalur masuk, beasiswa, IPK, IPS, UKT kuliah, dan status mahasiswa. Hasil penelitian menunjukkan bahwa *Random Forest* memiliki akurasi 82,64%, lebih unggul dibandingkan *Gradient Boosting* yang mencapai 79,66%.

Alkaff dkk. (2020) mengembangkan sistem peringatan dini untuk mendeteksi keterlambatan masa studi mahasiswa menggunakan algoritma *Support Vector Machines (SVM)*. Studi ini menganalisis faktor-faktor utama seperti jumlah SKS, jenis kelamin, IPK, dan program studi. Dengan akurasi sebesar 88,2%, model yang dikembangkan terbukti efektif dalam mengidentifikasi mahasiswa yang berisiko tidak lulus tepat waktu.

Novianto dkk. (2024) membandingkan performa algoritma *Random Forest* dan *Support Vector Machine (SVM)* dalam memprediksi capaian studi mahasiswa. Variabel yang digunakan mencakup jenis kelamin, asal sekolah, jalur sekolah, asal daerah, IPS semester 1–4, SKS semester 1–4, total SKS, IPK, dan status mahasiswa. Hasil analisis menunjukkan bahwa baik *Random Forest* maupun *SVM* setelah seleksi fitur memiliki akurasi yang sama, yaitu 97,67%, dengan total SKS sebagai faktor utama yang memengaruhi ketepatan kelulusan mahasiswa.

Ricodia dkk. (2024) mengkaji efektivitas lima algoritma dalam memprediksi kelulusan tepat waktu mahasiswa, yaitu *Decision Tree*, *Naïve Bayes*, *Logistic Regression*, *KNN*, dan *Random Forest*. Penelitian ini menggunakan dataset mahasiswa angkatan 2015–2019 dengan total 26 atribut, seperti tahun masuk, waktu kuliah, jenis kelamin, tipe sekolah, jurusan, IPS semester 1–10, SKS semester 1–10, dan status mahasiswa. Hasil penelitian menunjukkan bahwa *Random Forest* menjadi model paling akurat dengan tingkat akurasi 90,88%, diikuti oleh *Logistic Regression* (89,47%) dan *Decision Tree* (89,12%), dengan AUC sebesar 97,2% dan *F1-Score* 89,9%.

Tabel 2.1 Penelitian Terdahulu

Peneliti	**Judul**	**Variabel**	**Model dan Akurasi**	**Hasil**
Sari (2024)	Perbandingan Akurasi *Random Forest* dan KNN (*K-Nearest Neighbors*) pada Studi Kasus Kelulusan Mahasiswa Fakultas Teknik	Nim, Nama, jumlah_IPS, SKS semester 1-8, IPS semester 1-8, SKS_total, total_IPK.	*Random Forest* (100%), KNN (96,42%).	Penelitian ini menunjukkan bahwa mahasiswa dengan performa akademik rendah cenderung tidak lulus tepat waktu. *Random Forest*

				mencapai akurasi 100%, sementara *K-Nearest Neighbors* mencapai 96,42%, menjadikan *Random Forest* sebagai model paling akurat.
Labib Mu'tashim dkk. (2023)	Klasifikasi Ketepatan Lama Studi Mahasiswa Dengan Algoritma *Random Forest* Dan *Gradient Boosting* (Studi Kasus Fakultas Ilmu Komputer	Program Studi, Jenis Kelamin, Jenis Sekolah, Provinsi Sekolah, Jalur Masuk, Beasiswa/	*Random Forest* (82,64%), *Gradient Boosting* (79,66%).	Penelitian ini memprediksi kelulusan mahasiswa S1 FIK UPN Veteran Jakarta menggunakan *Random*

	Universitas Pembangunan Nasional Veteran Jakarta)	Non, Indeks Prestasi Kumulatif (IPK), Indeks Prestasi Semester (IPS), UKT Kuliah, Status Mahasiswa.		*Forest* (akurasi 82,64%) dan *Gradient Boosting* (79,66%), dengan *Random Forest* menunjukkan performa lebih baik.
Alkaff dkk. (2020)	Sistem Peringatan Dini Keterlambatan Masa Studi Mahasiswa Menggunakan *Support Vector Machine*	Jumlah SKS, Jenis Kelamin, IPK, Prodi.	*Support Vector Machine* (88,2%)	Model mendeteksi keterlambatan studi mahasiswa berdasarkan kinerja akademik dan faktor prodi dengan akurasi tinggi.

Noviant o dkk. (2024)	PERBANDING AN METODE KLASIFIKASI *RANDOM FOREST* DAN SUPPORT VECTOR MACHINE DALAM MEMPREDIKS I CAPAIAN STUDI MAHASISWA	Jenis Kelamin, Asal Sekolah, Jalur Sekolah, Asal Daerah, IPS 1-4, SKS 1-4, Total SKS, IPK, Status.	*Support Vector Machine* (97,67%), *Random Forest* (97,67%).	Penelitian ini membandin gkan *Random Forest* dan Support Vector Machine (SVM) dalam mempredik si capaian studi mahasiswa. *Random Forest* mencapai akurasi 97,67%, sementara SVM meningkat dari 91,47% menjadi 97,67%

				setelah seleksi fitur, dengan total SKS sebagai atribut utama.
Ricodia dkk. (2024)	Perbandingan Akurasi Algoritma Data Mining dalam Memprediksi Kelulusan Tepat Waktu	tahun_masuk, waktu_kuliah, jenis_kelamin, tipe_sekolah, jurusan, IPS 1-10, SKS 1-10 dan status.	DT (89.12%), NB (79.65%), LR (89.47%), KNN (87.72%), RF (90.88%).	Penelitian ini membandingkan lima algoritma dalam memprediksi kelulusan tepat waktu mahasiswa menggunakan 26 atribut dari data angkatan 2015-2019. *Random Forest*

				mencapai akurasi tertinggi sebesar 90,88% dengan AUC 97,2% dan F1-Score 89,9%, mengunggu li *Decision Tree* (89,12%) dan *Logistic Regression* (89,47%).

2.2 Landasan Teori

2.2.1 Prediksi Kelulusan Mahasiswa

Virtualisasi Prediksi kelulusan mahasiswa adalah cara untuk memperkirakan apakah seorang mahasiswa bisa lulus tepat waktu. Perkiraan ini dilakukan dengan melihat berbagai hal yang memengaruhi perjalanan studi mereka. Hal ini penting bagi kampus agar bisa membantu mahasiswa berhasil dalam belajar dan juga menjaga efisiensi operasional kampus. Prediksi ini berguna untuk menemukan mahasiswa yang mungkin kesulitan lulus tepat waktu sehingga bisa diberikan bantuan lebih awal (Putra & Erwin Harahap, 2024).

Terdapat beberapa faktor yang dapat memengaruhi kelulusan mahasiswa, baik dari aspek akademik, sosial, psikologis, ekonomi, maupun institusi. Menurut (Putri et al., 2024), berikut adalah beberapa faktor utama yang memengaruhi kelulusan mahasiswa:

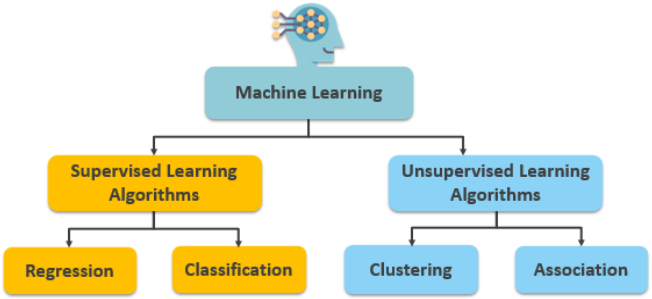
1) Indeks Prestasi Kumulatif (IPK)
   IPK adalah salah satu indikator utama dalam menilai keberhasilan akademik mahasiswa. Semakin tinggi IPK seorang mahasiswa, semakin besar peluangnya untuk lulus tepat waktu. Mahasiswa dengan IPK rendah cenderung mengalami kesulitan dalam menyelesaikan studi tepat waktu.
2) Kehadiran dan Keterlibatan Akademik
   Tingkat kehadiran dalam perkuliahan dan partisipasi aktif dalam kegiatan akademik, seperti seminar atau penelitian, berkontribusi besar terhadap kelulusan mahasiswa. Mahasiswa yang terlibat aktif biasanya lebih mampu menyelesaikan studi sesuai jadwal.
3) Motivasi Belajar
   Mahasiswa yang punya motivasi kuat, seperti keinginan untuk mencapai target tertentu. Mahasiswa dengan motivasi yang tinggi cenderung lebih disiplin dalam belajar dan menyelesaikan tugas akademik.
4) Kondisi Finansial
   Faktor ekonomi memiliki pengaruh signifikan terhadap kelulusan mahasiswa. Mahasiswa dengan dukungan keuangan yang baik lebih mudah berkonsentrasi pada studi dibandingkan mereka yang harus bekerja untuk mencukupi kebutuhan.
5) Dukungan Keluarga dan Sosial

Lingkungan keluarga dan sosial yang suportif dapat memberikan dorongan emosional dan moral kepada mahasiswa. Dukungan ini membantu mahasiswa untuk tetap fokus dan termotivasi dalam menyelesaikan studinya.

6) Kualitas Pengajaran dan Fasilitas
Institusi yang menyediakan pengajaran berkualitas, pembimbing akademik yang kompeten, dan fasilitas pendukung yang memadai cenderung memiliki tingkat kelulusan yang lebih tinggi.

2.2.2 *Machine Learning*

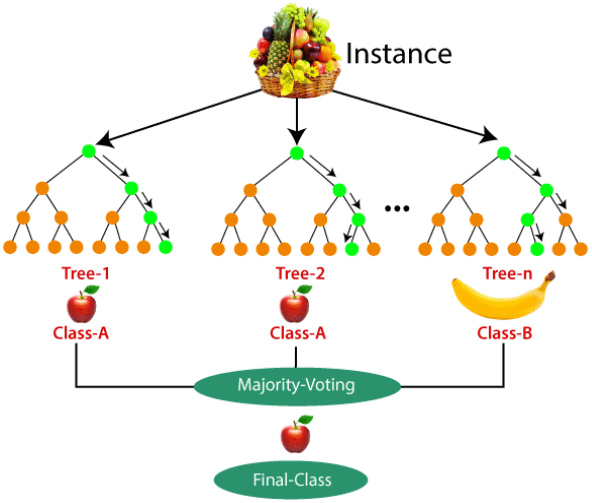
Machine Learning adalah cabang ilmu komputer yang dirancang untuk memungkinkan sistem komputer belajar dari data tanpa harus diprogram secara langsung. Dengan memanfaatkan pengalaman dari data yang dianalisis, model *Machine Learning* dapat meningkatkan performanya secara bertahap (Bi et al., 2019). Secara umum, *Machine Learning* terbagi menjadi dua kategori utama, yaitu *supervised learning* dan *unsupervised learning*.



Gambar 2.1 *Machine Learning*
(Sumber: https://medium.com/@varshini.kasirajan)

2.2.3 *Random Forest*

Menurut Breiman (2001) *Random Forest* merupakan algoritma *ensemble learning* yang membangun beberapa pohon keputusan (*decision trees*) dan menggabungkan prediksi dari setiap pohon untuk meningkatkan akurasi dan mengurangi risiko *overfitting*. Pendekatan ini sangat efektif karena menggabungkan kekuatan dari banyak model sederhana untuk memberikan hasil yang lebih stabil dan *robust*.



Gambar 2.5 Algoritma *Random Forest*
(Sumber: https://www.javatpoint.com/machine-learning-random-forest-algorithm)

Setiap pohon dalam *Random Forest* dibangun menggunakan *subset* data yang diambil secara acak dengan teknik *bootstrap sampling*. Selain itu, pada setiap *node* pohon, hanya *subset* acak dari fitur yang dipertimbangkan untuk menentukan pemisahan terbaik. Teknik ini memperkenalkan variasi tambahan di antara pohon-pohon, sehingga menghasilkan model yang lebih general terhadap data baru.

Proses pemisahan dalam setiap pohon menggunakan kriteria *Gini Index*, yang diukur dengan rumus:

$$Gini = 1 - \sum_{i=1}^{C} p_i^2 \qquad (2.1)$$

- p_i = Proporsi data pada kelas *i* di dalam node tertentu.
- C = Jumlah total kelas, yaitu terlambat dan tidak terlambat.

Setiap pohon dilatih secara independen, dan hasil prediksi untuk suatu data ditentukan berdasarkan metode *voting* mayoritas, yaitu kelas dengan jumlah prediksi terbanyak dari semua pohon akan menjadi keluaran akhir.

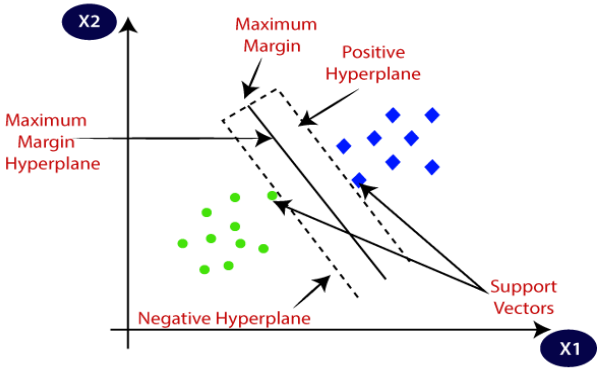
Untuk mengoptimalkan performa model, *Parameter* utama yang diatur meliputi:

1) Jumlah pohon ($n\_estimators$)
   Menentukan jumlah pohon keputusan dalam *Random Forest*. Dalam penelitian ini, nilai awal yang diuji adalah $n\_estimators \in \{50,100,200\}$.
2) Maksimum fitur (max_featuresmax_featuresmax_features)
   Menentukan jumlah maksimum fitur yang dipertimbangkan pada setiap node. Nilai yang diuji meliputi $sqrt(total\ fitur)$ atau $log2(total\ fitur)$.
3) Kedalaman maksimum pohon (max_depth)
   Membatasi kedalaman setiap pohon untuk mencegah *overfitting*. Nilai awal yang diuji adalah $max\_depth \in \{5,10,15\}$.
4) Minimum data per daun ($min\_samples\_leaf$)
   Menentukan jumlah minimum data yang harus ada dalam setiap node daun, dengan nilai awal diuji pada 1 hingga 5.

Menurut (Beliarding Sucahyo et al., 2024), cara kerja *Random Forest* dimulai dengan pemilihan *subset* data secara acak (*bootstrap sampling*). Setiap *subset* data digunakan untuk melatih sebuah *decision tree*. Selama proses pembentukan pohon, algoritma juga memilih *subset* fitur secara acak untuk mencegah *overfitting*. Hasil akhir diperoleh dari kombinasi prediksi setiap pohon melalui metode *voting* (untuk klasifikasi) atau rata-rata (untuk regresi).

2.2.4 *Support Vector Machines (SVM)*

SVM mencari garis pemisah (*hyperplane*) terbaik yang membagi dua kelas target (lulus tepat waktu dan tidak tepat waktu). Algoritma ini memaksimalkan margin antara dua kelas untuk memastikan pemisahan yang paling optimal. Dalam kasus data yang tidak terpisah secara linear, SVM menggunakan teknik *kernel* untuk memetakan data ke ruang fitur yang lebih tinggi agar dapat dipisahkan dengan lebih baik (Cortes & Vapnik, 1995).



Gambar 2.6 Algoritma *Support Vector Machine*
(Sumber: https://medium.com/@fraidoonomarzai99)

Menurut (Cortes & Vapnik, 1995), *Hyperplane* dirumuskan sebagai berikut:

$$w \cdot x + b = 0 \tag{2.2}$$

Di mana:

w = Vektor bobot.
x = Vektor fitur *input*.
b = Bias (intersep).

Hyperplane dipilih untuk memaksimalkan margin, yaitu jarak antara *hyperplane* dengan data terdekat dari kedua kelas (disebut *support vectors*).

SVM memecahkan masalah optimasi untuk memaksimalkan margin sambil meminimalkan kesalahan klasifikasi. Fungsi optimasi untuk kasus linear dinyatakan sebagai:

$$\begin{aligned} Minimize: (1/2)\left|\left|w\right|\right|^2 \\ Subject\ to: y_i(w \cdot x_i + b) \geq 1, \forall i \end{aligned} \tag{2.3}$$

Di mana:

y_i = Label kelas (1 untuk tidak terlambat dan -1 untuk terlambat).
x_i = Vektor data mahasiswa.
$\left|\left|w\right|\right|^2$ = Norm kuadrat dari vektor bobot w.

Untuk data yang tidak sepenuhnya dapat dipisahkan secara linear, fungsi optimasi dimodifikasi dengan menambahkan *Parameter* ξ_i (*slack variables*) untuk mengakomodasi kesalahan klasifikasi:

$$Minimize: (1/2)||w||^2 + C\Sigma\xi_i$$
$$Subject\ to: y_i(w \cdot x_i + b) \geq 1 - \xi_i, \forall i \quad (2.4)$$

Di mana:

ξ_i = Ukuran kesalahan untuk data ke-i

C = *Parameter* regularisasi yang mengontrol *trade-off* antara margin maksimum dan jumlah kesalahan.

Untuk kasus dimana data tidak dapat dipisahkan secara linear, SVM menggunakan fungsi *kernel* untuk memetakan data ke dimensi yang lebih tinggi. *Kernel* yang digunakan dalam penelitian ini adalah *Radial Basis Function* (RBF), yang dirumuskan sebagai:

$$K(x_i, x_j) = exp\left(-\gamma \left|\left|x_i - x_j\right|\right|^2\right) \quad (2.5)$$

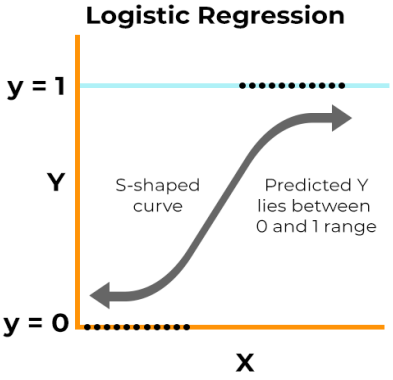
Di mana:

$K(x_i, x_j)$ = *Kernel* RBF untuk pasangan data x_i dan x_j.

Γ = *Parameter* yang mengontrol pengaruh dari setiap titik data terhadap *hyperplane*.

2.2.5 *Logistic Regression*

Sistem *Logistic Regression* adalah algoritma yang digunakan untuk klasifikasi, terutama untuk memprediksi probabilitas suatu kejadian. Algoritma ini bekerja dengan mengubah *output* model linier menjadi nilai probabilitas melalui fungsi sigmoid. *Output* dari *Logistic Regression* adalah nilai antara 0 dan 1, yang dapat digunakan untuk memprediksi kelas atau kategori dalam data (James dkk., 2013)..



Gambar 2.7 Algoritma *Logistic Regression*
(Sumber: https://velog.io/@chiroya/12-Logistic-Regression)

Menurut (Adriani Tampil dkk., n.d.), *Logistic Regression* berfungsi dengan menghitung nilai logit, yang merupakan logaritma dari rasio peluang (*odds*) dari probabilitas suatu kejadian. Model ini mengasumsikan hubungan linier antara variabel independen dengan logaritma peluang kejadian. Kelebihan *Logistic Regression* adalah kesederhanaannya dan kemampuannya untuk memberikan interpretasi probabilistik terhadap data, tetapi model ini kurang efektif untuk *dataset* dengan hubungan non-linier yang kompleks.

Menurut (Alfayulanda & Murni, 2024), Model *Logistic Regression* memanfaatkan fungsi *sigmoid* untuk memetakan kombinasi linier dari fitur-fitur independen menjadi probabilitas kelas target $\big(P(y = 1)\big)$ Fungsi *sigmoid* dirumuskan sebagai berikut:

$$\mathrm{P(y = 1|X)} = \frac{1}{\Big(1 + exp\big(-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ \ldots \ + \beta_n X_n)\big)\Big)} \quad (2.6)$$

Dimana:

$P(y = 1 \mid X)$	= Probabilitas mahasiswa mengalami keterlambatan
e	= Bilangan eksponensial.
β_0	= Bias (intersep).
$\beta_1, \beta_2, \ldots, \beta_n$	= Koefisien regresi dari setiap fitur.
$X_1, X_2, \ldots, X_n$	= Nilai dari fitur-fitur *input*.

Dalam perancangan *Logistic Regression*, beberapa langkah teknis dilakukan untuk membangun model. Langkah-langkah tersebut adalah sebagai berikut:

1) Definisi *Parameter* Model
   Logistic Regression memiliki beberapa *Parameter* utama yang akan disesuaikan dalam proses perancangan. *Parameter-Parameter* tersebut meliputi:
   a. *Solver*: Algoritma optimasi yang digunakan. Penelitian ini menggunakan *solver 'lbfgs'*, yang cocok untuk *dataset* dengan jumlah data menengah hingga besar.
   b. Penalty: Tipe regularisasi yang digunakan untuk mencegah *overfitting*. Regularisasi L2 dipilih karena cocok untuk model *Logistic Regression* standar.

c. C: *Parameter* pengaturan kekuatan regularisasi. Nilai *default* C=1.0 digunakan untuk memberikan keseimbangan antara bias dan variansi.
d. *Max Iterations*: Batas iterasi maksimum untuk mencapai konvergensi. Nilai ini disesuaikan menjadi 200 agar model memiliki waktu yang cukup untuk menemukan solusi optimal.

2) Proses Pembentukan Model
Model *Logistic Regression* dirancang dengan mengkombinasikan fitur-fitur yang telah dipilih dari *dataset*. Kombinasi linier dari fitur-fitur ini dikonversi menjadi probabilitas menggunakan fungsi *sigmoid*. Nilai probabilitas ini kemudian dibandingkan dengan ambang batas tertentu (*default*: 0,5) untuk menentukan kelas akhir mahasiswa, yaitu terlambat atau tidak terlambat.

3) Persamaan Model
Persamaan *Logistic Regression* yang digunakan dapat dinyatakan dalam:

$$z = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n$$
$$P(y = 1|X) = \frac{1}{\left(1 + exp(-z)\right)} \quad (2.7)$$

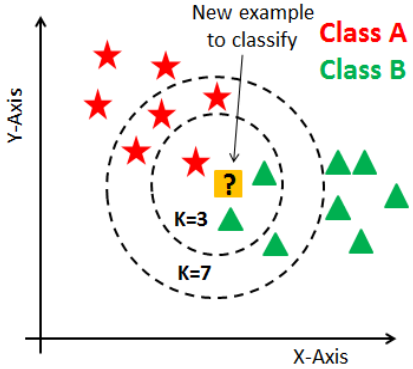
Pada persamaan ini, nilai βi menunjukkan besarnya pengaruh masing-masing fitur *Xi* terhadap probabilitas mahasiswa mengalami keterlambatan. Koefisien ini akan dianalisis lebih lanjut setelah model selesai dibangun.

4) Interpretasi Koefisien Model
Koefisien $(\beta 1, \beta 2, \ldots, \beta n)$ yang dihasilkan oleh model *Logistic Regression* akan digunakan untuk mengidentifikasi fitur-fitur mana yang memiliki dampak paling besar terhadap peluang keterlambatan mahasiswa.
a. Koefisien dengan nilai positif menunjukkan bahwa peningkatan nilai fitur tersebut akan meningkatkan probabilitas keterlambatan.
b. Sebaliknya, koefisien dengan nilai negatif menunjukkan bahwa peningkatan nilai fitur tersebut akan menurunkan probabilitas keterlambatan.

2.2.6 *K-Nearest Neighbors* (KNN)

K-Nearest Neighbors (KNN) adalah algoritma *Machine Learning* yang berbasis pada teknik pembelajaran *instance* (*instance-based learning*). KNN bekerja dengan cara mencari titik data yang paling dekat dengan titik data yang sedang dianalisis, kemudian mengklasifikasikan titik tersebut berdasarkan mayoritas kelas dari tetangga terdekatnya (Talamantes & Chavez, 2021).

KNN memiliki cara kerja yang sederhana namun efektif. Setiap data baru yang ingin diklasifikasikan akan dianalisis dengan menghitung jarak antara data tersebut dan data lainnya di dalam *dataset*, menggunakan metrik seperti *Euclidean* atau *Manhattan*. Kelemahan utama dari KNN adalah bahwa algoritma ini bisa sangat lambat pada *dataset* yang besar dan memerlukan banyak memori, karena harus menyimpan seluruh *dataset* untuk melakukan prediksi (Luqyana Zakiya Almas et al., 2024).



Gambar 2.8 Algoritma *K-Nearest Neighbors* (KNN)
(Sumber: https://rifqimulyawan.com/kamus/nearest-neighbor-mapping/)

Menurut (Zardini et al., 2023), Untuk menentukan tetangga terdekat, metrik jarak yang digunakan dalam penelitian ini adalah *Euclidean Distance*, yang dirumuskan sebagai berikut:

$$d(x_i, x_j) = \sqrt{\sum_{m=1}^{n} (x_{im} - x_{jm})^2} \quad (2.8)$$

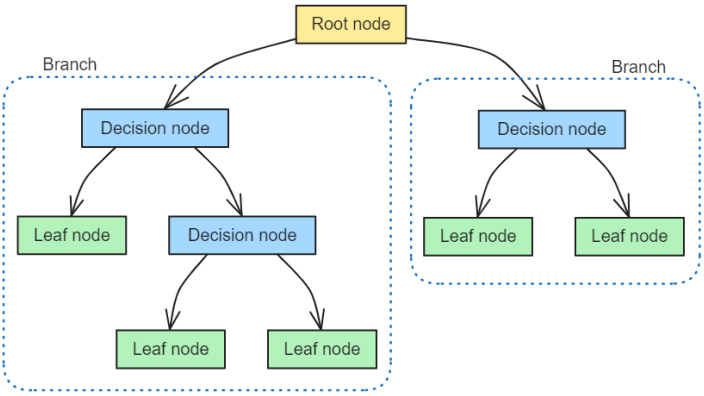
Di mana:
x_i dan x_j = Dua vektor data.

m = Indeks fitur dari data.
n = Jumlah fitur dalam *dataset*.

Nilai k, yang menunjukkan jumlah tetangga terdekat yang dipertimbangkan, merupakan *Parameter* yang harus ditentukan dengan hati-hati. Nilai k yang terlalu kecil dapat menyebabkan model terlalu sensitif terhadap data *outlier* (*overfitting*), sementara nilai k yang terlalu besar dapat menyebabkan model kehilangan kemampuan untuk mengenali pola lokal (*underfitting*). Dalam penelitian ini, nilai k akan ditentukan melalui eksperimen menggunakan teknik *k-fold cross-validation*, dengan mencoba beberapa nilai, seperti $k \in \{3, 5, 7, 9, 11\}$.

2.2.7 *Decision Tree*

Decision Tree adalah algoritma yang membangun model klasifikasi atau regresi dalam bentuk pohon keputusan, yang setiap simpulnya mewakili keputusan berdasarkan fitur tertentu. Algoritma ini membagi *dataset* menjadi *subset* yang lebih kecil berdasarkan aturan tertentu, yang kemudian digunakan untuk menentukan prediksi pada setiap cabang pohon (Ramadhon et al., 2024).



Gambar 2.9 Algoritma *Decision Tree*
(Sumber: https://mlpills.dev/machine-learning/introduction-to-decision-trees/)

Decision Tree membangun model dengan memilih fitur terbaik yang membagi data dengan cara memaksimalkan informasi *gain* atau mengurangi *entropy*. Pohon yang terbentuk dari pembagian ini akan terus bercabang sampai mencapai titik di mana data dalam setiap cabang cukup homogen. Keunggulan dari *Decision Tree* adalah kemampuannya untuk menangani data yang tidak memerlukan pra-pemrosesan yang rumit dan memberikan hasil yang mudah dipahami, meskipun model ini rentan terhadap *overfitting* jika tidak dipangkas dengan baik.

Menurut (Tangirala, 2020), Struktur pohon terdiri dari *node* akar, *node* cabang, dan *node* daun, di mana setiap *node* cabang memisahkan data berdasarkan aturan tertentu, dan setiap *node* daun merepresentasikan prediksi kelas. Untuk menentukan pemisahan terbaik pada setiap *node*, metrik *Gini Index* digunakan, yang diukur dengan rumus:

$$Gini = 1 - \sum_{i=1}^{C} p_i^2 \tag{2.9}$$

Di mana p_i adalah proporsi data pada kelas *i* di dalam *node* tertentu, dan *C* adalah jumlah total kelas (pada kasus ini *C* = 2 yaitu kelas terlambat dan tidak terlambat). *Gini Index* digunakan untuk menilai impuritas data, di mana pemisahan terbaik adalah pemisahan yang menghasilkan nilai *Gini Index* terkecil. Algoritma pembentukan *Decision Tree* dimulai dengan seluruh *dataset* pada *node* akar, di mana setiap pemisahan dilakukan secara iteratif dengan memilih fitur dan nilai batas (*threshold*) yang meminimalkan impuritas data di *node* yang dihasilkan. Proses ini berlanjut hingga salah satu dari kondisi berhenti terpenuhi, yaitu: semua data dalam *node* termasuk ke dalam satu kelas, tidak ada fitur yang tersisa untuk pemisahan, atau kedalaman maksimum pohon telah tercapai.

2.2.8 *Website*

Website adalah kumpulan halaman *web* yang menyajikan berbagai informasi dalam bentuk teks, gambar, dan suara. Halaman *web* yang saling terhubung disebut *hyperlink*, sedangkan teks yang terhubung dengan teks lain dikenal sebagai *hypertext*. Menurut (Garett et al., 2016), terdapat lima unsur dalam sebuah *website*, yaitu:

1) *Domain*
   Domain adalah alamat dari *website* yang membutuhkan alamat IP (IP *Address*) berupa serangkaian angka. Karena IP sulit diingat, maka dibuatlah penamaan untuk alamat *website*. Dengan nama *domain*, pengguna tidak perlu mengingat angka untuk mengakses *website*.
2) *Hosting*
   Hosting merujuk pada *server* yang menyimpan file *website*, sehingga dapat diakses dan dikelola melalui internet. *Web hosting* dapat mencakup teks, gambar, video, dan lainnya.
3) Konten
   Setiap *website* memiliki konten yang menjelaskan tujuan dari *website* tersebut. Konten bisa berupa informasi, tempat belanja, atau lainnya, tergantung pada tujuan pembuatan *website*.
4) Bahasa Pemrograman
   Website memerlukan bahasa pemrograman untuk pembangunannya. Bahasa yang umum digunakan adalah *HyperText Markup Language* (HTML). Seiring dengan perkembangan teknologi, banyak bahasa pemrograman lain yang dapat digunakan, seperti *Cascading Style Sheets* (CSS) untuk pengaturan tampilan, dan *JavaScript* untuk menambah interaktivitas.
5) Tampilan
   Selain konten, tampilan *website* juga penting. Tampilan harus menarik dan *user-friendly* agar pengguna dapat dengan mudah mengakses dan menggunakan *website* tersebut.

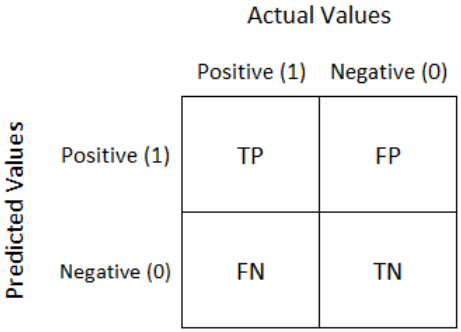
2.2.9 *Python*

Python adalah bahasa pemrograman tingkat tinggi yang bersifat umum, mudah dipelajari, dan memiliki sifat yang dinamis (Assaf & Noyé, 2008). Bahasa *Python* berkembang pesat dan banyak diminati karena mampu meningkatkan produktivitas serta mempermudah pekerjaan. Salah satu keunggulan utama *Python* adalah dukungannya terhadap berbagai paradigma pemrograman, termasuk pemrograman berorientasi objek (*object-oriented*), pemrograman imperatif, fungsional, serta prosedural. Beberapa fitur utama *Python* antara lain:

1) Sederhana
   Sintaks *Python* dirancang agar mudah dibaca dan dipahami, sehingga cocok untuk pemula.
2) Bersifat *open source*
   Python dapat digunakan secara bebas tanpa lisensi, memungkinkan pengembangan kolaboratif di komunitas global.
3) Mendukung berbagai teknologi
   Python kompatibel dengan berbagai teknologi modern seperti kecerdasan buatan, *big data*, dan pengembangan *web*.
4) Mempunyai banyak *library*
   Python menyediakan berbagai *library* bawaan maupun eksternal yang mempermudah pengembangan aplikasi.
5) Berorientasi objek
   Python mendukung konsep *object-oriented programming* untuk mempermudah pengorganisasian dan pengelolaan kode.
6) Bahasa yang ditafsirkan
   Python merupakan bahasa yang ditafsirkan (*interpreted*), sehingga kode dapat dijalankan langsung tanpa perlu dikompilasi terlebih dahulu.

2.2.10 *Confussin Matrix*

Confusion matrix adalah alat analisa dan evaluasi yang menyajikan ringkasan kinerja model dengan menampilkan jumlah prediksi yang benar dan salah. Dengan memanfaatkan metode ini, prediksi model dipetakan ke dalam kategori tertentu untuk menilai akurasi prediksi (Bengio et al., 2003).



Gambar 2.10 *Confusion Matrix*

Pada *confusion matrix*, terdapat empat elemen utama, yaitu:

1) *True Positive* (TP) = Ketika nilai aktual dan prediksi sama-sama benar.
2) *False Positive* (FP) = Ketika nilai aktual salah, tetapi model memprediksi benar.
3) *False Negative* (FN) = Ketika nilai aktual benar, tetapi model memprediksi salah.
4) *True Negative* (TN) = Ketika nilai aktual dan prediksi sama-sama salah.

Prediksi yang benar ditunjukkan pada diagonal utama matriks, sedangkan prediksi yang salah berada di luar diagonal. Dengan menggunakan *confusion matrix*, evaluasi kinerja model dapat dilakukan menggunakan *Parameter* berikut:

Recall = TP / (TP + FN)
Precision = TP / (TP + FP)
Accuracy = (TP + TN) / Total
Error Rate = (FP + FN) / Total

Penjelasan masing-masing *Parameter* evaluasi model dapat dilihat pada tabel 2.2 berikut:

Tabel 2.2 Penjelasan performa evaluasi *confusion matrix*

Performa	Deskripsi
Recall	Kemampuan model dalam mendeteksi hasil positif yang benar.
Precision	Kemampuan model dalam meminimalkan hasil negatif yang salah.
Accuracy	Persentase keseluruhan prediksi yang benar.
Error Rate	Persentase keseluruhan prediksi yang salah.

2.2.11 *Black Box Testing*

Pengujian fungsionalitas dilakukan dengan menggunakan metode *Black Box Testing*. *Black Box Testing* adalah metode pengujian yang berfokus pada fungsi perangkat lunak tanpa memperhatikan struktur internal atau kode program. Tujuannya adalah memastikan bahwa masukan, proses, dan keluaran sistem sesuai dengan spesifikasi yang ditentukan (Dwi Wijaya & Wardah Astuti, 2021).

Metode ini memanfaatkan batas atas dan batas bawah dari data masukan untuk menguji berbagai skenario. Pengujian ini juga berguna untuk memverifikasi apakah sistem dapat menangani masukan yang tidak diharapkan tanpa menyebabkan data yang tersimpan menjadi tidak valid.

2.2.12 *User Acceptance Testing*

User Acceptance Testing (UAT) adalah proses untuk memastikan bahwa solusi yang diimplementasikan pada sistem sudah sesuai dengan kebutuhan pengguna akhir (Leung & Wong, 1997).

Secara teknis, pengujian *black box* sudah cukup untuk mengevaluasi kelayakan perangkat lunak. Namun, UAT memungkinkan identifikasi masalah yang mungkin tidak terdeteksi pada pengujian sebelumnya. Proses ini dilakukan langsung oleh pengguna atau klien untuk memastikan bahwa perangkat lunak bekerja sesuai ekspektasi. UAT tidak difokuskan pada masalah teknis sederhana seperti kesalahan ejaan atau *crash*, tetapi lebih kepada memastikan bahwa solusi perangkat lunak memenuhi kebutuhan bisnis secara keseluruhan.