MACHINE LEARNING BERBASIS POHON KEPUTUSAN:

Implementasi Decision Tree dan Random Forest di Google Colab

Sartika Lina Mulani Sitio, S.Kom.,M.Kom

MEDIA PUSTAKA INDO

2026

MACHINE LEARNING BERBASIS POHON KEPUTUSAN:
Implementasi Decision Tree dan Random Forest di Google Colab

Penulis :
Sartika Lina Mulani Sitio, S.Kom., M.Kom

ISBN :
978-634-247-201-9

Diterbitkan Oleh :
PT MEDIA PUSTAKA INDO
Jl. Merdeka RT4/RW2 Binangun, Kab. Cilacap, Jawa Tengah
Website: www.mediapustakaindo.com
E-mail: mediapustakaindo@gmail.com
Anggota IKAPI: 263/JTE/2023

Cetakan Pertama : 2026

KATA PENGANTAR

Puji dan syukur ke hadirat Tuhan Yang Maha Esa atas segala rahmat dan karunia-Nya sehingga buku *Machine Learning Berbasis Pohon Keputusan: Implementasi Decision Tree dan Random Forest di Google Colab* ini dapat disusun dan diterbitkan. Buku ini hadir sebagai upaya untuk menjembatani pemahaman konseptual dan keterampilan praktis dalam penerapan algoritma machine learning, khususnya yang berbasis pohon keputusan.

Perkembangan pesat teknologi data dan kecerdasan buatan menuntut penguasaan metode analisis yang tidak hanya akurat, tetapi juga mudah dipahami dan diimplementasikan. Decision Tree dan Random Forest merupakan algoritma yang populer karena kemampuannya dalam menangani data kompleks serta interpretabilitas model yang relatif baik. Oleh karena itu, buku ini dirancang untuk memberikan pemahaman menyeluruh, mulai dari konsep dasar, prinsip kerja algoritma, hingga tahapan implementasi secara praktis menggunakan Google Colab sebagai platform pembelajaran berbasis komputasi awan.

Penyajian materi dalam buku ini disusun secara sistematis dan bertahap, dilengkapi dengan contoh kasus, penjelasan kode program, serta interpretasi hasil analisis. Dengan demikian, pembaca tidak hanya memahami teori, tetapi juga mampu mengaplikasikannya secara langsung dalam berbagai konteks, baik akademik maupun praktis. Buku ini diharapkan dapat menjadi referensi bagi mahasiswa, dosen, peneliti, serta praktisi yang ingin mempelajari dan mengembangkan kemampuan di bidang machine learning.

Akhir kata, semoga buku ini dapat memberikan kontribusi nyata dalam pengembangan literasi data dan kecerdasan buatan, serta menjadi referensi yang bermanfaat dalam pembelajaran dan penelitian di bidang machine learning.

DAFTAR ISI

BAB 1
Machine Learning Berbasis Pohon

1.1 Paradigma Pembelajaran Mesin dan Posisi Pohon Keputusan

Paradigma pembelajaran mesin *(*machine learning*)* merupakan cabang kecerdasan buatan yang berfokus pada pengembangan algoritma yang mampu belajar dari data dan pengalaman. Berbeda dengan pemrograman konvensional yang mengandalkan aturan eksplisit buatan manusia, pembelajaran mesin memungkinkan sistem membangun model secara otomatis berdasarkan pola yang ditemukan dalam data. Paradigma ini menjadi sangat relevan di era data besar *(*big data*)*, di mana volume, variasi, dan kecepatan data tidak lagi memungkinkan pendekatan manual dalam pengambilan keputusan. Secara konseptual, pembelajaran mesin bertujuan untuk membangun fungsi pemetaan antara data masukan dan keluaran yang optimal. Proses ini dilakukan melalui tahapan utama seperti pengumpulan data, pra-pemrosesan, pelatihan model, evaluasi, dan penerapan model [1]. Keberhasilan suatu model tidak hanya diukur dari kemampuannya menyesuaikan diri terhadap data latih, tetapi juga dari kemampuannya melakukan generalisasi pada data baru yang belum pernah ditemui sebelumnya. Paradigma pembelajaran mesin secara umum diklasifikasikan ke dalam tiga kategori utama, yaitu supervised learning, unsupervised learning, dan reinforcement learning. Supervised learning menggunakan data berlabel untuk melatih model sehingga dapat memprediksi target tertentu, sedangkan unsupervised learning berfokus pada pencarian struktur tersembunyi dalam data tanpa label. Adapun reinforcement learning mengandalkan mekanisme umpan balik berbasis

penghargaan dan hukuman untuk mengoptimalkan keputusan secara bertahap.

Dalam konteks aplikasi praktis, supervised learning merupakan paradigma yang paling luas digunakan karena relevansinya dengan permasalahan nyata seperti klasifikasi, regresi, dan prediksi risiko. Model dalam paradigma ini belajar dari pasangan data input dan output sehingga mampu menghasilkan prediksi yang terukur. Salah satu algoritma yang paling representatif dan banyak digunakan dalam paradigma supervised learning adalah pohon keputusan atau decision tree. Pohon keputusan menempati posisi strategis dalam pembelajaran mesin karena mampu merepresentasikan proses pengambilan keputusan secara struktural dan logis. Model ini bekerja dengan membagi ruang data ke dalam beberapa bagian berdasarkan nilai atribut tertentu, membentuk struktur hierarkis menyerupai pohon. Setiap keputusan yang diambil oleh model dapat ditelusuri kembali melalui jalur dari akar hingga ke daun, menjadikannya sangat transparan.

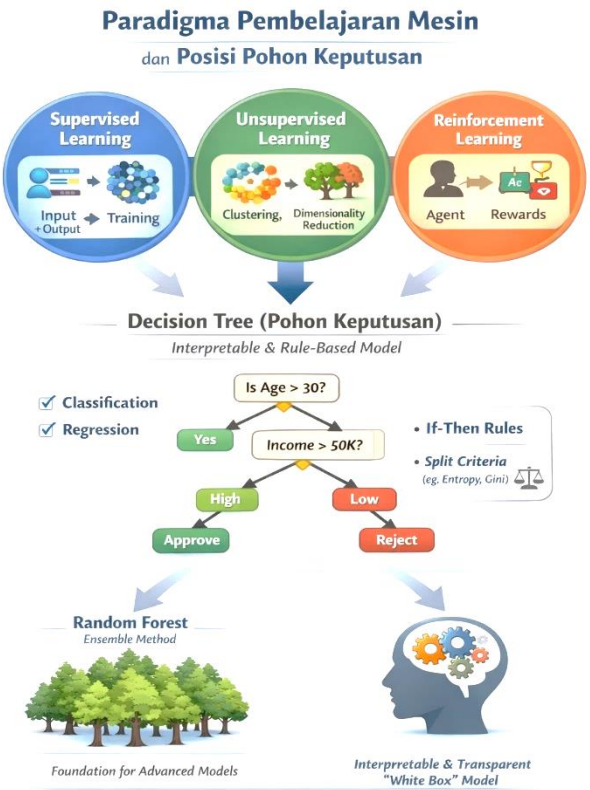
Keunggulan utama pohon keputusan terletak pada interpretabilitasnya yang tinggi. Berbeda dengan banyak algoritma pembelajaran mesin lain yang bersifat black box, pohon keputusan memungkinkan pengguna memahami alasan di balik setiap prediksi. Hal ini menjadikan pohon keputusan sangat cocok digunakan pada domain yang membutuhkan transparansi dan akuntabilitas, seperti sistem pendukung keputusan, analisis kebijakan, dan bidang kesehatan. Dari sisi matematis, pohon keputusan dibangun melalui proses pemilihan atribut terbaik pada setiap node berdasarkan kriteria tertentu. Kriteria yang umum digunakan meliputi information gain, entropy, dan gini impurity, yang bertujuan untuk memaksimalkan homogenitas data pada setiap cabang. Dengan pendekatan ini, model secara iteratif membagi data hingga mencapai kondisi penghentian tertentu. Namun demikian, pohon

keputusan juga memiliki keterbatasan yang perlu diperhatikan dalam paradigma pembelajaran mesin. Salah satu permasalahan utama adalah kecenderungan terhadap overfitting, terutama ketika pohon tumbuh terlalu dalam dan menyesuaikan diri secara berlebihan terhadap data latih. Model yang terlalu kompleks dapat kehilangan kemampuan generalisasi, sehingga performanya menurun pada data uji. Untuk mengatasi permasalahan tersebut, berbagai teknik seperti pruning dikembangkan guna membatasi kompleksitas pohon. Pre pruning menghentikan pertumbuhan pohon lebih awal, sedangkan post pruning memangkas cabang yang tidak memberikan kontribusi signifikan terhadap akurasi model. Teknik-teknik ini menunjukkan bahwa meskipun pohon keputusan bersifat sederhana secara konsep, implementasinya tetap memerlukan pertimbangan metodologis yang matang. Selain sebagai model tunggal, pohon keputusan memiliki peran penting dalam pengembangan paradigma ensemble learning. Dalam pendekatan ini, beberapa model digabungkan untuk menghasilkan prediksi yang lebih akurat dan stabil. Pohon keputusan sering dipilih sebagai base learner karena sifatnya yang fleksibel dan mampu menangkap hubungan non linear dalam data.

Salah satu contoh paling populer dari penerapan pohon keputusan dalam ensemble learning adalah Random Forest. Metode ini membangun banyak pohon keputusan secara independen menggunakan sampel data dan subset fitur yang berbeda, kemudian menggabungkan hasil prediksi dari seluruh pohon [2]. Pendekatan ini secara signifikan mengurangi variansi model dan meningkatkan ketahanan terhadap overfitting. Dengan demikian, posisi pohon keputusan dalam paradigma pembelajaran mesin tidak hanya terbatas sebagai algoritma dasar, tetapi juga sebagai fondasi bagi metode yang lebih kompleks dan canggih. Pohon keputusan menjadi jembatan konseptual antara model sederhana

yang mudah dipahami dan model ensemble yang memiliki performa tinggi. Hal ini menjadikannya alat pembelajaran yang ideal bagi pemula sekaligus solusi praktis bagi praktisi. Dalam konteks pendidikan dan riset, pohon keputusan sering digunakan sebagai titik awal untuk memahami prinsip dasar pembelajaran mesin. Konsep seperti pemilihan fitur, pembagian data, dan evaluasi model dapat dipelajari secara intuitif melalui struktur pohon. Pemahaman ini kemudian dapat diperluas ke algoritma lain yang lebih kompleks dengan landasan yang kuat.

Secara praktis, pohon keputusan juga unggul dalam menangani data campuran yang terdiri dari atribut numerik dan kategorikal. Fleksibilitas ini membuatnya cocok digunakan pada berbagai jenis dataset tanpa memerlukan transformasi data yang kompleks. Selain itu, pohon keputusan relatif efisien secara komputasi dan mudah diimplementasikan pada berbagai platform analisis data. Dalam ekosistem pembelajaran mesin modern, posisi pohon keputusan tetap relevan meskipun muncul berbagai model berbasis pembelajaran mendalam (deep learning). Ketika interpretabilitas, kecepatan, dan kemudahan implementasi menjadi prioritas, pohon keputusan sering kali menjadi pilihan utama. Oleh karena itu, memahami paradigma pembelajaran mesin melalui perspektif pohon keputusan memberikan dasar konseptual yang kuat sekaligus aplikatif bagi pengembangan model yang lebih lanjut.



Gambar 1. Paradigma Pembelajaran Mesin dan Pohon Keputusan

1.2 Karakteristik Model Berbasis Aturan (Rule-Based Learning)

Model berbasis aturan (rule based learning) memiliki karakteristik utama berupa representasi pengetahuan dalam bentuk aturan logis eksplisit, yang umumnya disusun menggunakan pola if th*en*. Setiap aturan menggambarkan hubungan sebab–akibat antara kondisi tertentu pada data dengan keputusan atau kesimpulan yang dihasilkan. Representasi ini membuat proses pengambilan keputusan menjadi terstruktur dan sistematis, karena setiap prediksi bergantung pada pemenuhan kondisi yang telah ditentukan.

Karakteristik yang paling menonjol dari model berbasis aturan adalah interpretabilitas dan transparansi yang tinggi. Setiap keputusan dapat dijelaskan secara langsung melalui aturan yang aktif, sehingga pengguna dapat memahami alasan di balik hasil prediksi tanpa memerlukan analisis matematis yang kompleks. Transparansi ini sangat penting dalam sistem yang menuntut akuntabilitas, seperti sistem pendukung keputusan, sektor kesehatan, hukum, dan keuangan. Model berbasis aturan juga bersifat deterministik, artinya keputusan yang dihasilkan akan selalu konsisten selama aturan dan data input yang digunakan sama. Tidak terdapat unsur stokastik atau ketidakpastian dalam proses inferensi, sehingga perilaku model dapat diprediksi dan diuji dengan mudah [3]. Karakteristik ini memberikan keunggulan dalam proses validasi, audit sistem, dan penelusuran kesalahan (debugging). Selain itu, model berbasis aturan memiliki kemampuan untuk merepresentasikan hubungan non-linear dan kondisi kompleks antar atribut. Dengan mengombinasikan beberapa kondisi dalam satu aturan, model mampu menangkap interaksi antar fitur tanpa memerlukan transformasi matematis tingkat lanjut. Hal ini membuat pendekatan berbasis aturan cukup fleksibel dalam menangani permasalahan klasifikasi dan pengambilan keputusan yang kompleks.

Karakteristik lain yang penting adalah kemudahan integrasi dengan pengetahuan pakar. Aturan yang digunakan dalam model dapat berasal dari hasil pembelajaran data maupun dari pengalaman dan keahlian manusia. Kombinasi ini memungkinkan pengembangan sistem hibrida yang tidak hanya mengandalkan data historis, tetapi juga memanfaatkan intuisi dan kebijakan domain tertentu, sehingga meningkatkan keandalan keputusan. Namun, model berbasis aturan memiliki karakteristik skalabilitas yang terbatas ketika jumlah atribut dan kompleksitas data meningkat. Semakin besar dan beragam data, jumlah

aturan yang dihasilkan cenderung bertambah secara signifikan. Kondisi ini dapat menyebabkan aturan menjadi sulit dikelola, saling tumpang tindih, dan berpotensi mengurangi keterbacaan model secara keseluruhan.

Model berbasis aturan juga sensitif terhadap noise dan data yang tidak bersih. Aturan yang dibentuk dari data bising dapat menangkap pola yang tidak relevan atau bersifat kebetulan, sehingga menurunkan kemampuan generalisasi model pada data baru. Oleh karena itu, pra-pemrosesan data dan penyederhanaan aturan menjadi langkah penting dalam penerapan pendekatan ini. Dari sisi komputasi, model berbasis aturan memiliki efisiensi inferensi yang tinggi, karena proses pengambilan keputusan hanya melibatkan evaluasi kondisi logis sederhana. Hal ini menjadikannya sangat cocok untuk aplikasi real-time dan sistem dengan keterbatasan sumber daya, seperti sistem tertanam (embedded systems) atau aplikasi berbasis layanan cepat. Secara keseluruhan, karakteristik model berbasis aturan mencerminkan keseimbangan antara keterbacaan, konsistensi, dan kemudahan implementasi**,** meskipun dengan keterbatasan dalam skalabilitas dan fleksibilitas pada data berskala besar. Dalam konteks pembelajaran mesin modern, pendekatan ini tetap relevan karena mampu menjembatani kebutuhan interpretabilitas manusia dengan proses pembelajaran otomatis berbasis data, khususnya pada algoritma seperti pohon keputusan dan metode ensemble berbasis aturan.

Tabel 1. Perbandingan Karakteristik Model Berbasis Aturan (Rule-Based Learning)

Karakteristik	Penjelasan	Implikasi Praktis
Representasi Aturan	Pengetahuan direpresentasikan dalam bentuk aturan logis *if then*.	Mudah dipahami dan ditelusuri oleh manusia.
Interpretabilitas	Setiap keputusan dapat dijelaskan berdasarkan aturan yang aktif.	Cocok untuk domain kritis (kesehatan, keuangan, kebijakan).
Transparansi	Proses pengambilan keputusan bersifat terbuka dan tidak tersembunyi.	Mendukung audit dan akuntabilitas sistem.
Deterministik	Input yang sama selalu menghasilkan output yang sama.	Perilaku model konsisten dan mudah divalidasi.
Hubungan Non Linear	Mampu menangkap interaksi kompleks antar atribut.	Tidak memerlukan transformasi matematis rumit.
Integrasi Pengetahuan Pakar	Aturan dapat berasal dari data maupun keahlian manusia.	Memungkinkan sistem hibrida data driven dan expert driven.
Skalabilitas	Jumlah aturan meningkat seiring kompleksitas data.	Berpotensi menurunkan keterbacaan pada data besar.
Sensitivitas terhadap Noise	Mudah terpengaruh data bising atau outlier.	Membutuhkan pra-pemrosesan data yang baik.
Efisiensi Inferensi	Evaluasi aturan bersifat sederhana dan cepat.	Cocok untuk sistem real-time dan resource terbatas.
Generalisasi	Cenderung menurun jika aturan terlalu spesifik.	Perlu pruning atau penyederhanaan aturan.

Contoh Aturan (If–Then) dari Dataset Nyata adalah sebagai berikut:

1. Contoh dari Dataset Iris (Klasifikasi Bunga)

   Dataset *Iris* digunakan untuk mengklasifikasikan jenis bunga berdasarkan panjang dan lebar sepal serta petal. Aturan hasil pembelajaran Decision Tree:

   - **IF** petal_length ≤ 2.45 **THEN** class = *Iris-setosa*
   - **IF** petal_length > 2.45 **AND** petal_width ≤ 1.75 **THEN** class = *Iris-versicolor*
   - **IF** petal_length > 2.45 **AND** petal_width > 1.75 **THEN** class = *Iris-virginica*

   Aturan ini menunjukkan bagaimana model berbasis aturan mampu memisahkan kelas secara jelas dengan kondisi sederhana.

2. Contoh dari Dataset Kredit (Credit Approval / Loan Approval)

   Dataset ini sering digunakan untuk memprediksi kelayakan kredit berdasarkan atribut demografis dan finansial. Contoh aturan:

   - **IF** income > 50.000 **AND** credit_history = "good" **THEN** loan_status = "approved"
   - **IF** income ≤ 50.000 **AND** debt_ratio > 0.4 **THEN** loan_status = "rejected"

   Aturan ini mencerminkan logika pengambilan keputusan yang realistis dan mudah dijelaskan kepada pemangku kepentingan.

3. Contoh dari Dataset Kesehatan (Prediksi Penyakit Jantung)

   Digunakan untuk memprediksi risiko penyakit jantung berdasarkan data klinis. Contoh aturan:

   - **IF** age > 50 **AND** cholesterol > 240 **AND** blood_pressure = "high" **THEN** heart_disease = "high risk"
   - **IF** age ≤ 50 **AND** cholesterol ≤ 200 **THEN** heart_disease = "low risk"

   Aturan ini menunjukkan keunggulan interpretabilitas dalam domain medis.

1.3 Kelebihan dan Keterbatasan Decision Tree dan Random Forest

1. Kelebihan dan Kelemahan Decision Tree [4]

   Kelebihan decision tree:

   a. Interpretabilitas yang sangat tinggi

      Decision Tree memiliki struktur yang menyerupai alur pengambilan keputusan manusia, sehingga setiap hasil prediksi dapat dijelaskan secara logis melalui aturan if then. Pengguna dapat menelusuri jalur keputusan dari akar hingga daun untuk memahami mengapa suatu keputusan diambil. Kelebihan ini menjadikan Decision Tree sangat ideal untuk sistem yang menuntut transparansi dan akuntabilitas.

   b. Mudah dipahami dan diimplementasikan

      Algoritma Decision Tree relatif mudah dipahami bahkan oleh pengguna non-teknis. Proses implementasinya juga sederhana karena tidak memerlukan formulasi matematis yang kompleks. Hal ini membuat Decision Tree sering

digunakan sebagai model awal (baseline model) dalam eksperimen machine learning.

c. Mampu menangani data numerik dan kategorikal

   Decision Tree dapat langsung memproses data dengan berbagai tipe atribut tanpa memerlukan normalisasi atau standardisasi. Kemampuan ini mengurangi beban pra-pemrosesan data dan mempercepat tahap eksperimen.

d. Mampu menangkap hubungan non linear

   Dengan membagi data secara rekursif, Decision Tree dapat memodelkan hubungan non-linear dan interaksi antar fitur yang sulit ditangkap oleh model linier. Hal ini menjadikannya fleksibel untuk berbagai permasalahan klasifikasi dan regresi.

e. Efisiensi komputasi pada dataset kecil hingga menengah

   Untuk dataset dengan ukuran terbatas, Decision Tree memiliki waktu pelatihan yang relatif cepat dan tidak membutuhkan sumber daya komputasi besar.

Kelemahan Decision Tree:

a. Rentan terhadap overfitting

   Decision Tree cenderung membentuk pohon yang sangat kompleks jika tidak dibatasi, sehingga model menyesuaikan diri secara berlebihan terhadap data latih. Akibatnya, performa pada data uji dapat menurun secara signifikan.

b. Tidak stabil terhadap perubahan data

   Perubahan kecil pada data latih dapat menghasilkan struktur pohon yang sangat berbeda. Ketidakstabilan ini

membuat Decision Tree kurang andal pada data yang bervariasi atau mengandung noise tinggi.

c. Akurasi terbatas pada data kompleks

Pada dataset berdimensi tinggi atau memiliki pola yang sangat kompleks, Decision Tree tunggal sering kalah akurat dibandingkan metode ensemble atau model yang lebih canggih.

d. Sensitif terhadap noise dan outlier

Keputusan pemisahan atribut dapat dipengaruhi oleh data yang bising, sehingga menghasilkan aturan yang tidak relevan dan menurunkan kemampuan generalisasi model.

2. Kelebihan dan Kelemahan Random Forest [5]

Kelebihan Random Forest:

a. Akurasi prediksi yang tinggi

Random Forest memiliki tingkat akurasi prediksi yang tinggi karena menggabungkan hasil dari banyak pohon keputusan dalam proses pengambilan keputusan akhir. Setiap pohon dibangun dari subset data yang berbeda, sehingga mampu menangkap variasi pola yang beragam dalam dataset. Dengan melakukan agregasi prediksi melalui mekanisme voting (untuk klasifikasi) atau rata-rata (untuk regresi), Random Forest dapat mengurangi kesalahan prediksi individual yang sering muncul pada Decision Tree tunggal. Pendekatan kolektif ini membuat model lebih representatif terhadap keseluruhan data, sehingga performa prediksi menjadi lebih stabil dan akurat.

b. Tahan terhadap overfitting

Random Forest dirancang untuk mengatasi permasalahan overfitting yang umum terjadi pada Decision Tree. Melalui teknik bagging (bootstrap aggregating), setiap pohon dilatih menggunakan sampel data yang berbeda, sementara pemilihan fitur secara acak pada setiap pemisahan node mencegah model terlalu bergantung pada atribut tertentu. Kombinasi kedua mekanisme ini mengurangi variansi model secara signifikan, sehingga Random Forest tidak mudah menyesuaikan diri secara berlebihan terhadap data latih dan mampu melakukan generalisasi yang lebih baik pada data baru.

c. Robust terhadap noise dan data tidak seimbang

Keunggulan lain Random Forest adalah ketahanannya terhadap noise dan ketidaksempurnaan data. Pada dataset nyata, sering ditemukan data yang bising, tidak lengkap, atau memiliki distribusi kelas yang tidak seimbang. Dalam kondisi ini, kesalahan prediksi dari satu pohon keputusan dapat dikompensasi oleh pohon-pohon lain yang mempelajari pola berbeda. Mekanisme agregasi ini membuat Random Forest lebih toleran terhadap anomali dan kesalahan lokal, sehingga tetap mampu menghasilkan prediksi yang andal meskipun kualitas data tidak ideal.

d. Mampu menangani dataset besar dan berdimensi tinggi

Random Forest sangat efektif digunakan pada dataset berskala besar dan berdimensi tinggi karena setiap pohon hanya menggunakan sebagian fitur dan sebagian data. Pendekatan ini mengurangi kompleksitas komputasi per pohon sekaligus memungkinkan model menangkap

informasi penting dari ruang fitur yang luas. Selain itu, Random Forest dapat diparalelkan dengan mudah karena setiap pohon dilatih secara independen, menjadikannya cocok untuk aplikasi industri dan riset yang melibatkan data dalam jumlah besar dan kompleks.

e. Menyediakan informasi pentingnya fitur (feature importance)

   Random Forest mampu mengukur tingkat kepentingan setiap fitur berdasarkan kontribusinya terhadap peningkatan kualitas pemisahan data di seluruh pohon. Informasi ini memberikan wawasan yang berharga mengenai atribut mana yang paling berpengaruh dalam proses pengambilan keputusan model. Feature importance sangat berguna dalam analisis data eksploratif, seleksi fitur, serta interpretasi hasil penelitian, karena membantu peneliti memahami hubungan antara variabel input dan target secara lebih mendalam meskipun model bersifat ensemble.

Kelemahan Random Forest:

a. Interpretabilitas rendah

   Random Forest memiliki keterbatasan utama pada aspek interpretabilitas karena model ini merupakan gabungan dari banyak pohon keputusan yang bekerja secara kolektif. Setiap pohon menghasilkan keputusan berdasarkan aturan yang berbeda-beda, dan prediksi akhir diperoleh melalui mekanisme agregasi seperti voting atau rata-rata. Akibatnya, sangat sulit untuk menelusuri secara rinci jalur keputusan yang menyebabkan suatu prediksi tertentu.

Kondisi ini menjadikan Random Forest cenderung bersifat black box dibandingkan Decision Tree tunggal, sehingga kurang sesuai untuk aplikasi yang menuntut penjelasan keputusan secara eksplisit, transparansi logika, dan akuntabilitas tinggi.

b. Kebutuhan komputasi dan memori lebih besar

Keterbatasan lain dari Random Forest adalah kebutuhan sumber daya komputasi dan memori yang relatif besar. Model ini harus menyimpan dan memproses banyak pohon keputusan sekaligus, di mana setiap pohon memiliki struktur node dan cabang tersendiri. Seiring bertambahnya jumlah pohon dan fitur, konsumsi memori meningkat secara signifikan, begitu pula beban komputasi selama proses pelatihan dan inferensi. Oleh karena itu, Random Forest kurang efisien ketika diterapkan pada sistem dengan keterbatasan perangkat keras atau lingkungan komputasi yang terbatas.

c. Waktu pelatihan relatif lebih lama

Proses pelatihan Random Forest memerlukan waktu yang lebih lama dibandingkan Decision Tree tunggal karena model ini harus melatih banyak pohon keputusan secara terpisah. Setiap pohon dibangun dari sampel data dan subset fitur yang berbeda, sehingga total waktu pelatihan bertambah seiring dengan peningkatan jumlah pohon (n_estimators) dan kedalaman pohon. Meskipun proses ini dapat dipercepat melalui komputasi paralel, Random Forest tetap kurang ideal untuk aplikasi yang membutuhkan pelatihan model secara cepat atau pembaruan model secara real time.

d. Kurang efisien untuk aplikasi sangat sederhana

Random Forest juga dinilai kurang efisien untuk permasalahan dengan pola data yang sederhana dan mudah dipisahkan. Pada kasus seperti ini, penggunaan banyak pohon keputusan sering kali tidak memberikan peningkatan performa yang signifikan dibandingkan Decision Tree tunggal. Sebaliknya, kompleksitas model justru meningkat tanpa manfaat yang sepadan, baik dari sisi waktu pelatihan maupun penggunaan sumber daya. Oleh karena itu, Random Forest dapat dianggap berlebihan (overkill) untuk aplikasi sederhana, di mana model yang lebih ringan dan mudah dijelaskan sudah mampu memberikan hasil yang memadai.

Eksperimen Perbandingan Decision Tree dan Random Forest di Google Colab:

1. Persiapan Lingkungan : Google Colab telah menyediakan sebagian besar pustaka machine learning. Tahap ini memastikan semua dependensi tersedia.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

2. Memuat dan Memahami Dataset : Dataset Iris terdiri dari 150 sampel dengan 4 fitur numerik dan 3 kelas.

```
iris = load_iris()
X = iris.data
y = iris.target

feature_names = iris.feature_names
target_names = iris.target_names

df = pd.DataFrame(X, columns=feature_names)
df['class'] = y
df.head()
```

3. Pembagian Data Latih dan Data Uji

Rasio 70:30 digunakan agar evaluasi lebih seimbang.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

4. Implementasi Decision Tree

Decision Tree dilatih tanpa tuning awal untuk melihat performa dasar.

```
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

y_pred_dt = dt_model.predict(X_test)
```

Evaluasi Decision Tree

```
print("Accuracy Decision Tree:", accuracy_score(y_test, y_pred_dt))
print(classification_report(y_test, y_pred_dt, target_names=target_names))
```

Visualisasi Struktur Pohon

```
plt.figure(figsize=(16,10))
plot_tree(dt_model, feature_names=feature_names,
        class_names=target_names, filled=True)
plt.show()
```

5. Implementasi Random Forest

   Random Forest dibangun dengan 100 pohon keputusan.

```
rf_model = RandomForestClassifier(
   n_estimators=100,
   random_state=42
)
rf_model.fit(X_train, y_train)

y_pred_rf = rf_model.predict(X_test)
```

Evaluasi Random Forest

```
print("Accuracy Random Forest:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf, target_names=target_names))
```

6. Perbandingan Akurasi Model

```
comparison = pd.DataFrame({
   "Model": ["Decision Tree", "Random Forest"],
   "Accuracy": [
      accuracy_score(y_test, y_pred_dt),
      accuracy_score(y_test, y_pred_rf)
   ]
})
comparison
```

7. Analisis Feature Importance (Random Forest)

```
importance = rf_model.feature_importances_
feature_importance = pd.DataFrame({
   "Feature": feature_names,
   "Importance": importance
}).sort_values(by="Importance", ascending=False)

feature_importance
```

Visualisasi

```
plt.figure(figsize=(8,5))
plt.barh(feature_importance["Feature"],
feature_importance["Importance"])
plt.gca().invert_yaxis()
plt.title("Feature Importance - Random Forest")
plt.show()
```

1.4 Aplikasi Pohon Keputusan dalam Berbagai Domain

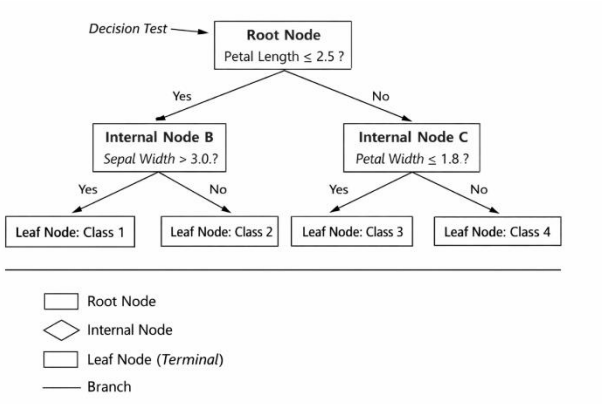
Pohon keputusan (decision tree) merupakan salah satu algoritma pembelajaran mesin yang banyak digunakan karena mampu

menggabungkan kemampuan prediksi yang baik dengan tingkat interpretabilitas yang tinggi. Algoritma ini bekerja dengan cara mempelajari pola dari data dan menyusunnya dalam bentuk struktur keputusan yang menyerupai cara berpikir manusia. Setiap keputusan yang dihasilkan oleh model tidak muncul secara acak, melainkan melalui tahapan evaluasi kondisi yang sistematis dan terstruktur. Secara konseptual, pohon keputusan merepresentasikan proses pengambilan keputusan dalam bentuk struktur hierarkis yang terdiri dari node akar, node internal, cabang, dan node daun. Node akar berfungsi sebagai titik awal pengambilan keputusan, sedangkan node internal merepresentasikan pengujian terhadap suatu atribut [6]. Cabang menunjukkan hasil dari pengujian tersebut, dan node daun menyatakan keputusan atau prediksi akhir. Struktur ini membuat alur logika model mudah ditelusuri dari awal hingga akhir. Keunggulan utama pohon keputusan terletak pada kemampuannya menghasilkan model yang mudah dipahami dan dijelaskan. Setiap jalur dari akar hingga daun dapat diterjemahkan menjadi aturan logis berbentuk if then, sehingga pengguna dapat memahami alasan di balik setiap prediksi. Dengan demikian, pohon keputusan tidak hanya berfungsi sebagai alat prediksi, tetapi juga sebagai sarana penjelasan dan eksplorasi pengetahuan dari data.

Selain interpretabilitas, pohon keputusan juga memiliki fleksibilitas tinggi dalam menangani berbagai jenis data. Model ini dapat digunakan untuk permasalahan klasifikasi maupun regresi, serta mampu memproses atribut numerik dan kategorikal tanpa memerlukan transformasi data yang kompleks. Fleksibilitas ini menjadikan pohon keputusan mudah diadaptasi pada berbagai konteks permasalahan nyata. Karakteristik pohon keputusan yang transparan dan mudah dipahami menjadikannya sangat sesuai untuk domain yang membutuhkan akuntabilitas dalam pengambilan keputusan. Pada bidang seperti kesehatan, keuangan,

pendidikan, dan pemerintahan, hasil prediksi harus dapat dijelaskan secara rasional dan dapat dipertanggungjawabkan. Pohon keputusan memenuhi kebutuhan tersebut karena setiap keputusan dapat dilacak kembali ke kondisi-kondisi yang menyebabkannya.

Dari sisi praktis, pohon keputusan juga relatif mudah diimplementasikan dan tidak membutuhkan sumber daya komputasi yang besar, terutama untuk dataset berukuran kecil hingga menengah. Hal ini membuatnya sering digunakan sebagai model awal dalam analisis data dan pembelajaran mesin. Selain itu, kemudahan visualisasi struktur pohon turut mendukung proses analisis dan komunikasi hasil kepada pemangku kepentingan non teknis. Dengan kombinasi antara performa prediktif, interpretabilitas, dan fleksibilitas, pohon keputusan menempati posisi penting dalam ekosistem pembelajaran mesin. Algoritma ini tidak hanya relevan sebagai model mandiri, tetapi juga menjadi fondasi bagi pengembangan metode yang lebih kompleks seperti Random Forest. Oleh karena itu, pohon keputusan tetap menjadi salah satu pendekatan yang strategis dan bernilai tinggi dalam berbagai aplikasi pembelajaran mesin modern.



Gambar 2. Aplikasi Pohon Keputusan

a. Aplikasi Pohon Keputusan dalam Bidang Kesehatan

Dalam bidang kesehatan, pohon keputusan banyak digunakan sebagai alat bantu diagnosis dan prediksi risiko penyakit. Model ini mampu menganalisis data medis seperti usia, tekanan darah, kadar kolesterol, hasil laboratorium, dan gejala klinis untuk menentukan kemungkinan suatu penyakit. Keunggulan utama pohon keputusan di domain ini adalah interpretabilitasnya, karena aturan keputusan yang dihasilkan dapat dipahami oleh tenaga medis. Dengan demikian, dokter dapat menggunakan hasil prediksi sebagai pendukung keputusan klinis tanpa menghilangkan pertimbangan profesional dan etika medis.

Contoh: Mengklasifikasikan risiko penyakit jantung (tinggi/rendah)

```
from sklearn.datasets import load_breast_cancer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

model = DecisionTreeClassifier(max_depth=4, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
accuracy_score(y_test, y_pred)
```

b. Aplikasi Pohon Keputusan dalam Sektor Keuangan dan Perbankan

Pada sektor keuangan, pohon keputusan digunakan secara luas dalam penilaian kredit (credit scoring), deteksi penipuan, dan manajemen risiko. Model ini membantu lembaga keuangan mengklasifikasikan nasabah berdasarkan tingkat kelayakan kredit dengan mempertimbangkan pendapatan, riwayat pembayaran, dan rasio utang. Transparansi aturan keputusan memungkinkan bank menjelaskan alasan persetujuan atau penolakan kredit kepada nasabah, sekaligus memenuhi tuntutan regulasi dan audit.

Contoh: Menentukan status kredit (layak/tidak layak)

```
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=1000, n_features=6, random_state=42)
model = DecisionTreeClassifier(criterion="gini", max_depth=5)
model.fit(X, y)
```

c. Aplikasi Pohon Keputusan dalam Bidang Pendidikan

Dalam domain pendidikan, pohon keputusan dimanfaatkan untuk menganalisis performa akademik, memprediksi kelulusan, serta mengidentifikasi siswa atau mahasiswa yang berisiko mengalami kegagalan studi. Dengan mengolah data nilai, kehadiran, latar belakang sosial, dan aktivitas belajar, model ini menghasilkan aturan yang membantu pendidik merancang intervensi pembelajaran yang tepat. Interpretabilitas model memungkinkan pihak sekolah atau universitas memahami faktor dominan yang memengaruhi keberhasilan belajar.

Contoh: Mengidentifikasi mahasiswa berisiko gagal studi

```
X, y = make_classification(n_samples=500, n_features=5, random_state=1)

model = DecisionTreeClassifier(max_depth=3)
model.fit(X, y)
```

d. Aplikasi Pohon Keputusan dalam Bisnis dan Pemasaran

Pada sektor bisnis, pohon keputusan digunakan untuk segmentasi pelanggan, analisis perilaku konsumen, dan prediksi churn. Model ini membantu perusahaan memahami faktor-faktor yang memengaruhi keputusan pembelian pelanggan, seperti usia, preferensi, dan riwayat transaksi. Aturan keputusan yang dihasilkan memudahkan manajemen dalam menyusun strategi pemasaran yang lebih efektif dan tepat sasaran, tanpa memerlukan pemahaman teknis yang mendalam.

Contoh: Menentukan pelanggan yang berpotensi berhenti

```
X, y = make_classification(n_samples=800, n_features=7, random_state=10)
model = DecisionTreeClassifier(max_depth=4)
model.fit(X, y)
```

e. Aplikasi Pohon Keputusan dalam Industri dan Manufaktur

Dalam industri manufaktur, pohon keputusan digunakan untuk pengendalian kualitas, prediksi kegagalan mesin, dan optimasi proses produksi. Model ini menganalisis data sensor, kondisi operasional, dan histori perawatan untuk mendeteksi potensi cacat produk atau kerusakan peralatan. Dengan aturan yang jelas, teknisi

dapat mengetahui kondisi kritis yang memicu kegagalan sehingga tindakan pencegahan dapat dilakukan lebih awal.

Contoh: Mengklasifikasikan produk cacat atau normal

```
X, y = make_classification(n_samples=600, n_features=8, random_state=7)

model = DecisionTreeClassifier(max_depth=6)
model.fit(X, y)
```

f. Aplikasi Pohon Keputusan dalam Pertanian

Pada sektor pertanian, pohon keputusan dimanfaatkan untuk prediksi hasil panen, klasifikasi tanaman, dan rekomendasi pemupukan. Model ini menggabungkan data cuaca, jenis tanah, pola tanam, dan penggunaan pupuk untuk membantu petani mengambil keputusan yang lebih tepat. Interpretabilitas aturan menjadikan rekomendasi mudah dipahami dan diterapkan langsung di lapangan, sehingga meningkatkan produktivitas pertanian.

Contoh: Mengklasifikasikan hasil panen (tinggi/rendah)

```
X, y = make_classification(n_samples=400, n_features=4, random_state=20)

model = DecisionTreeClassifier(max_depth=3)
model.fit(X, y)
```

g. Aplikasi Pohon Keputusan dalam Pemerintahan dan Kebijakan Publik

Dalam pemerintahan, pohon keputusan digunakan untuk analisis data kependudukan, penentuan penerima bantuan sosial, dan evaluasi kebijakan publik. Model ini membantu pemerintah

mengklasifikasikan masyarakat berdasarkan kriteria tertentu secara objektif dan transparan. Dengan aturan yang jelas, keputusan yang diambil dapat dipertanggungjawabkan secara administratif dan sosial.

Contoh: Klasifikasi layak/tidak layak menerima bantuan

```
X, y = make_classification(n_samples=700, n_features=6, random_state=99)

model = DecisionTreeClassifier(max_depth=5)
model.fit(X, y)
```

h. Aplikasi Pohon Keputusan dalam Pengembangan Model Lanjutan

Selain digunakan sebagai model mandiri, pohon keputusan juga berperan penting sebagai fondasi bagi metode ensemble learning seperti Random Forest. Dalam konteks ini, pohon keputusan menjadi komponen dasar yang dikombinasikan untuk meningkatkan akurasi dan stabilitas prediksi. Hal ini memperluas penerapan pohon keputusan ke domain dengan data besar dan kompleks, termasuk analitik industri dan riset skala besar.

1.5 Alur Umum Eksperimen Machine Learning di Google Colab

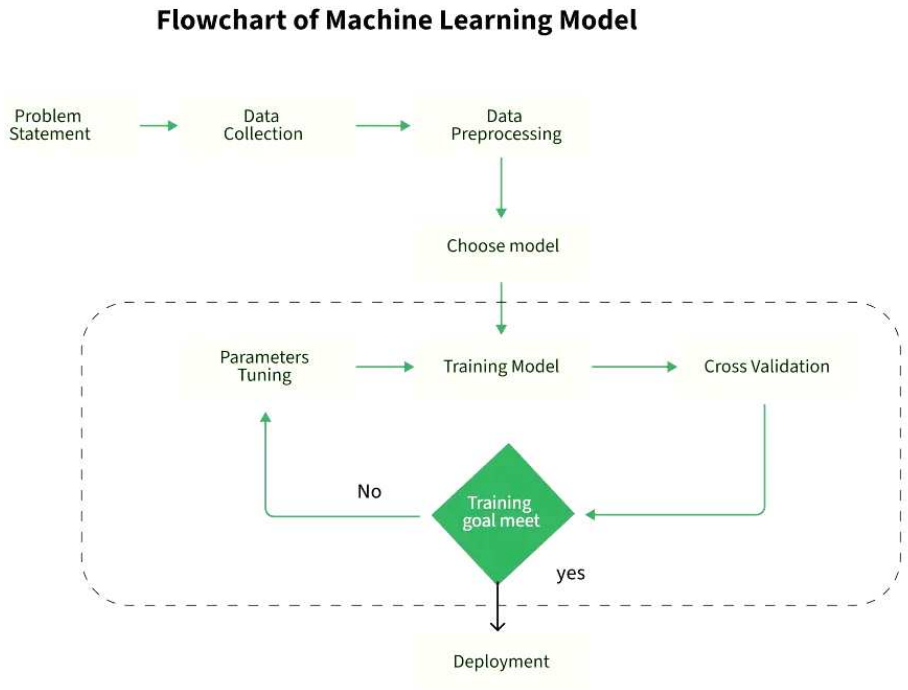
Alur umum eksperimen machine learning di Google Colab merupakan rangkaian tahapan sistematis yang dirancang untuk membangun, mengevaluasi, dan memvalidasi model pembelajaran mesin secara terstruktur dan dapat direproduksi. Setiap tahapan dalam alur ini saling berkaitan dan membentuk sebuah siklus kerja yang logis, mulai dari perumusan masalah hingga evaluasi hasil. Pendekatan yang sistematis ini penting untuk memastikan bahwa model yang dihasilkan tidak hanya mampu mempelajari pola dari data, tetapi juga memiliki kualitas dan

keandalan yang dapat dipertanggungjawabkan secara ilmiah. Google Colab berperan sebagai lingkungan komputasi berbasis cloud yang mendukung seluruh tahapan eksperimen machine learning [7]. Platform ini menyediakan notebook interaktif yang memungkinkan pengguna menuliskan kode, menjalankan eksperimen, serta mendokumentasikan hasil dalam satu tempat. Dengan pendekatan ini, proses eksperimen menjadi lebih terorganisasi dan mudah diikuti, baik untuk tujuan pembelajaran maupun penelitian. Selain itu, Colab mendukung kolaborasi, sehingga eksperimen dapat dibagikan dan dikembangkan bersama oleh banyak pengguna.

Salah satu keunggulan utama Google Colab adalah kemampuannya menyediakan akses langsung ke bahasa pemrograman Python beserta berbagai pustaka machine learning populer. Pustaka seperti NumPy, Pandas, scikit-learn, dan Matplotlib sudah tersedia secara bawaan, sehingga pengguna dapat langsung fokus pada pengembangan model tanpa perlu menghabiskan waktu untuk konfigurasi lingkungan. Hal ini sangat membantu dalam mempercepat proses eksperimen, terutama bagi peneliti dan mahasiswa yang ingin segera melakukan analisis data. Selain pustaka perangkat lunak, Google Colab juga menyediakan dukungan komputasi berupa CPU dan GPU yang dapat digunakan sesuai kebutuhan eksperimen. Dukungan ini memungkinkan pelatihan model dilakukan dengan lebih efisien, terutama untuk dataset berukuran besar atau model yang membutuhkan komputasi intensif. Akses ke sumber daya ini menjadikan Colab sebagai alternatif yang ekonomis dan praktis dibandingkan pengadaan perangkat keras lokal dengan spesifikasi tinggi. Keunggulan lain dari Google Colab adalah tidak diperlukannya instalasi lokal pada perangkat pengguna. Seluruh proses eksperimen dijalankan melalui peramban web, sehingga pengguna dapat mengakses lingkungan kerja dari berbagai perangkat dan sistem operasi. Kondisi ini memberikan

fleksibilitas tinggi dan mengurangi hambatan teknis yang sering muncul pada proses instalasi dan konfigurasi lingkungan machine learning secara mandiri.

Dalam konteks akademik dan riset, alur eksperimen machine learning di Google Colab sangat mendukung prinsip keterulangan (reproducibility). Notebook yang berisi kode, data, dan hasil eksperimen dapat disimpan dan dibagikan dengan mudah, sehingga eksperimen dapat dijalankan ulang atau diverifikasi oleh pihak lain. Hal ini merupakan aspek penting dalam penelitian ilmiah untuk menjamin validitas dan transparansi hasil. Secara keseluruhan, kombinasi antara alur eksperimen machine learning yang sistematis dan dukungan lingkungan Google Colab menjadikan platform ini sangat efektif untuk pembelajaran, eksperimen, dan penelitian [8]. Dengan mengikuti alur yang terstruktur dan memanfaatkan fasilitas yang disediakan, pengguna dapat membangun model pembelajaran mesin secara efisien, terdokumentasi dengan baik, dan siap digunakan untuk pengembangan lebih lanjut maupun penerapan pada permasalahan nyata.



Gambar 3. Flowchart Model Mesin Learning

1. Problem Statement

   Tahap problem statement merupakan langkah awal yang sangat krusial dalam eksperimen machine learning. Pada tahap ini, permasalahan yang akan diselesaikan didefinisikan secara jelas dan terukur, misalnya apakah tujuan eksperimen adalah klasifikasi, regresi, atau prediksi. Perumusan masalah yang tepat akan menentukan arah keseluruhan eksperimen, termasuk jenis data yang dibutuhkan, algoritma yang digunakan, serta metrik evaluasi yang relevan. Tanpa problem statement yang jelas, model yang dibangun berisiko tidak sesuai dengan kebutuhan atau sulit dievaluasi keberhasilannya.

```
# Problem:
# Klasifikasi jenis bunga Iris (Setosa, Versicolor, Virginica)
# berdasarkan panjang dan lebar sepal serta petal
```

2. Data Collection

   Setelah masalah dirumuskan, tahap berikutnya adalah data collection atau pengumpulan data. Data dapat diperoleh dari berbagai sumber, seperti database internal, sensor, survei, dataset publik, atau repositori daring. Kualitas data pada tahap ini sangat menentukan performa model, karena machine learning sangat bergantung pada pola yang terkandung dalam data. Data yang tidak representatif atau mengandung banyak kesalahan dapat menyebabkan model menghasilkan prediksi yang tidak akurat.

```
from sklearn.datasets import load_iris
import pandas as pd

iris = load_iris()
X = iris.data
y = iris.target
df = pd.DataFrame(X, columns=iris.feature_names)
df['class'] = y
df.head()
```

3. Data Preprocessing

   Tahap data preprocessing bertujuan untuk menyiapkan data agar layak digunakan dalam pelatihan model. Proses ini mencakup pembersihan data, penanganan nilai hilang, penghapusan duplikasi, encoding data kategorikal, serta normalisasi atau standardisasi jika

diperlukan. Pra-pemrosesan penting karena data mentah umumnya mengandung noise dan inkonsistensi yang dapat menurunkan kinerja model. Tahap ini juga membantu meningkatkan efisiensi dan stabilitas proses pelatihan.

```
# Cek missing value
df.isnull().sum()

# Cek informasi dataset
df.info()
# Karena dataset Iris bersih,
# tidak diperlukan preprocessing lanjutan
```

4. Choose Model

   Pada tahap choose model, algoritma machine learning dipilih sesuai dengan karakteristik masalah dan data. Pemilihan model dapat berupa Decision Tree, Random Forest, regresi linier, atau algoritma lainnya. Pertimbangan dalam memilih model meliputi kompleksitas data, kebutuhan interpretabilitas, serta ketersediaan sumber daya komputasi. Model yang dipilih akan menjadi kerangka utama dalam proses pembelajaran pola dari data.

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier(
    criterion="gini",
    random_state=42
)
```

5. Training Model

   Tahap training model merupakan inti dari eksperimen machine learning, di mana model dilatih menggunakan data latih. Pada proses ini, algoritma menyesuaikan parameter internalnya agar mampu memetakan hubungan antara fitur input dan target output. Tujuan pelatihan adalah meminimalkan kesalahan prediksi berdasarkan fungsi objektif tertentu. Proses ini dapat dilakukan berulang kali dengan konfigurasi yang berbeda untuk memperoleh performa terbaik.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.3, random_state=42, stratify=y
)
model.fit(X_train, y_train)
```

6. Cross Validation

   Cross validation digunakan untuk mengevaluasi kestabilan dan kemampuan generalisasi model. Data dibagi ke dalam beberapa subset, di mana model dilatih dan diuji secara bergantian. Teknik ini membantu mengurangi bias evaluasi yang dapat terjadi jika hanya menggunakan satu pembagian data latih dan data uji. Dengan cross validation, performa model dapat dinilai secara lebih objektif dan representatif terhadap data nyata.

```
from sklearn.model_selection import cross_val_score

cv_scores = cross_val_score(
   model, X, y, cv=5, scoring='accuracy'
)
```

```
print("Cross Validation Scores:", cv_scores)
print("Average CV Accuracy:", cv_scores.mean())
```

7. Parameters Tuning

   Tahap parameters tuning bertujuan untuk mengoptimalkan hyperparameter model agar performanya meningkat. Hyperparameter seperti kedalaman pohon, jumlah pohon, atau tingkat regularisasi diatur dan diuji dengan berbagai kombinasi. Jika hasil evaluasi belum memenuhi target, proses tuning dilakukan kembali. Tahap ini bersifat iteratif dan sangat berpengaruh terhadap keseimbangan antara bias dan variansi model.

```
from sklearn.model_selection import GridSearchCV

param_grid = {
   'max_depth': [2, 3, 4, 5, None],
   'min_samples_split': [2, 5, 10]
}

grid = GridSearchCV(
   DecisionTreeClassifier(random_state=42),
   param_grid,
   cv=5,
   scoring='accuracy'
)

grid.fit(X_train, y_train)
print("Best Parameters:", grid.best_params_)
model = grid.best_estimator_
```

8. Training Goal Meet (Decision Point)

   Bagian training goal meet merupakan titik keputusan dalam alur eksperimen. Pada tahap ini, performa model dibandingkan dengan kriteria atau target yang telah ditentukan sebelumnya, seperti nilai akurasi minimum atau tingkat kesalahan tertentu. Jika tujuan belum tercapai (No), proses kembali ke tahap tuning atau pelatihan ulang. Jika tujuan telah tercapai (Yes), maka model dianggap layak untuk tahap selanjutnya.

```
from sklearn.metrics import accuracy_score, classification_report
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Test Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
# Decision point
target_accuracy = 0.90
if accuracy >= target_accuracy:
    print("Training goal meet ✓")
else:
    print("Training goal NOT meet ✗ - Need tuning")
```

9. Deployment

   Tahap terakhir adalah deployment, yaitu penerapan model yang telah memenuhi kriteria ke dalam lingkungan nyata. Model dapat diintegrasikan ke dalam sistem informasi, aplikasi web, atau layanan prediksi otomatis. Pada tahap ini, model tidak lagi hanya bersifat eksperimen, tetapi sudah digunakan untuk membantu pengambilan keputusan nyata. Deployment juga menandai bahwa seluruh rangkaian eksperimen machine learning telah berjalan secara lengkap dan sistematis.

```
# Contoh data baru (simulasi)
new_data = [[5.1, 3.5, 1.4, 0.2]]
prediction = model.predict(new_data)
print("Predicted Class:", iris.target_names[prediction][0])
```

BAB 2
Pohon Keputusan

2.1 Representasi Struktur Pohon: Node, Cabang, dan Daun

Representasi struktur pohon merupakan inti dari algoritma Decision Tree dalam pembelajaran mesin karena menjadi dasar bagaimana model memahami dan memproses data. Struktur ini dirancang untuk meniru pola berpikir manusia dalam mengambil keputusan, yaitu dengan mengevaluasi suatu kondisi secara bertahap sebelum sampai pada kesimpulan akhir. Pendekatan bertahap ini memungkinkan permasalahan yang kompleks diuraikan menjadi bagian-bagian yang lebih sederhana dan mudah dianalisis. Dalam proses pengambilan keputusan, struktur pohon membagi suatu masalah besar menjadi sub-masalah yang lebih kecil melalui serangkaian pertanyaan atau pengujian terhadap atribut data. Setiap pengujian berfungsi sebagai langkah seleksi yang menyaring data berdasarkan kriteria tertentu. Dengan cara ini, model secara sistematis mempersempit ruang keputusan hingga diperoleh hasil yang paling sesuai dengan karakteristik data. Secara umum, struktur pohon keputusan tersusun atas tiga komponen utama, yaitu node (simpul), cabang (branch), dan daun (leaf). Node berperan sebagai titik pengambilan keputusan yang mengevaluasi suatu atribut, cabang merepresentasikan hasil dari evaluasi tersebut, sedangkan daun menunjukkan hasil akhir dari proses pengambilan keputusan [9]. Ketiga komponen ini membentuk kerangka kerja yang terorganisasi dan mudah diikuti.

Hubungan antar node, cabang, dan daun bersifat hierarkis, di mana keputusan dimulai dari node paling atas dan mengalir ke bawah melalui

cabang-cabang tertentu hingga mencapai daun. Struktur hierarki ini memungkinkan model memetakan hubungan antara fitur input dan hasil prediksi secara jelas dan terstruktur. Setiap jalur dari node awal hingga daun mencerminkan logika keputusan yang spesifik berdasarkan kondisi data. Dengan representasi struktur seperti ini, Decision Tree tidak hanya berfungsi sebagai alat prediksi, tetapi juga sebagai sarana untuk memahami pola dan hubungan dalam data. Kejelasan struktur pohon memungkinkan pengguna menelusuri proses pengambilan keputusan secara transparan, sehingga model menjadi mudah diinterpretasikan dan relevan untuk berbagai aplikasi yang membutuhkan penjelasan logis atas hasil prediksi.

1. Node (simpul)

   merupakan elemen dasar dalam struktur pohon keputusan yang berfungsi sebagai titik pengambilan keputusan. Setiap node merepresentasikan suatu pengujian terhadap atribut tertentu dalam dataset. Pengujian ini biasanya berbentuk kondisi logis, seperti "apakah nilai fitur lebih besar dari ambang tertentu" atau "apakah atribut termasuk dalam kategori tertentu". Node paling atas disebut root node, yang merupakan titik awal evaluasi seluruh data. Pemilihan atribut pada node dilakukan berdasarkan kriteria tertentu, seperti Information Gain, Gini Impurity, *atau* Variance Reduction, dengan tujuan memaksimalkan pemisahan kelas atau nilai target.

   Contoh Node (Root Node):

   "Apakah penghasilan > Rp5.000.000?"

   Node ini mengevaluasi atribut penghasilan dan membagi data ke dalam dua kondisi. Keputusan pada node ini menentukan jalur selanjutnya yang akan diikuti oleh data. Node dapat berada di posisi awal (root node) maupun di tengah pohon (node internal).

2. Cabang (branch)

   adalah penghubung antar node yang merepresentasikan hasil dari suatu pengujian pada node sebelumnya. Setiap cabang menunjukkan kemungkinan nilai atau kondisi dari atribut yang diuji, misalnya hasil "ya" atau "tidak", atau rentang nilai tertentu pada fitur numerik. Cabang berfungsi sebagai jalur keputusan yang mengarahkan data dari satu node ke node berikutnya. Dalam konteks aturan (rule-based learning), cabang dapat dipandang sebagai bagian dari klausa kondisi dalam aturan if then.

   Contoh Cabang dari Node Penghasilan:

   Cabang 1: Ya (Penghasilan > Rp5.000.000)

   Cabang 2: Tidak (Penghasilan ≤ Rp5.000.000)

   Cabang merepresentasikan kondisi logis dari hasil pengujian node. Setiap cabang membawa data ke langkah keputusan berikutnya, sehingga cabang dapat dipandang sebagai jalur keputusan dalam pohon.

3. Daun (leaf)

   merupakan node terminal yang menandai akhir dari proses pengambilan keputusan. Tidak seperti node internal, daun tidak lagi melakukan pengujian atribut. Sebaliknya, daun menyimpan hasil akhir berupa kelas prediksi (pada klasifikasi) atau nilai numerik (pada regresi). Setiap daun mewakili keputusan final yang dihasilkan oleh pohon berdasarkan jalur cabang yang dilalui dari root node. Dengan demikian, setiap jalur dari root node ke daun dapat diterjemahkan sebagai satu aturan keputusan yang lengkap dan eksplisit.

Contoh Daun:

Layak Kredit

Tidak Layak Kredit

Misalnya, jika data mengikuti cabang "Ya" dari node penghasilan dan memenuhi kondisi tertentu, maka proses berhenti di daun Layak Kredit. Daun inilah yang menjadi output akhir model Decision Tree.

Tabel 2. Perbandingan Komponen Struktur Pohon Keputusan

Komponen	Definisi	Fungsi Utama	Ciri Khas	Contoh
Node (Simpul)	Titik pengambilan keputusan dalam pohon	Menguji nilai suatu atribut	Menggunakan kriteria pemisahan (entropy, gini, dll.)	"Apakah usia > 30?"
Root Node	Node paling atas dalam pohon	Titik awal evaluasi seluruh data	Tidak memiliki parent	"Apakah penghasilan > 5 juta?"
Cabang (Branch)	Penghubung antar node	Menunjukkan hasil pengujian atribut	Mewakili kondisi atau nilai tertentu	"Ya" / "Tidak"
Node Internal	Node selain root dan daun	Melakukan pengujian lanjutan	Memiliki parent dan child	"Apakah riwayat kredit baik?"
Daun (Leaf)	Node terminal	Menyimpan hasil akhir prediksi	Tidak memiliki child	"Layak Kredit"
Jalur (Path)	Urutan node dan cabang	Membentuk aturan keputusan	Dapat ditulis sebagai *if then*	IF usia > 30 AND income tinggi THEN disetujui

2.2 Konsep Splitting dan Recursive Partitioning

Konsep splitting dan recursive partitioning merupakan fondasi utama dalam pembentukan pohon keputusan pada pembelajaran mesin. Kedua konsep ini menjelaskan bagaimana data dibagi secara bertahap ke dalam subset yang lebih homogen dengan tujuan meningkatkan akurasi prediksi. Melalui proses ini, Decision Tree mampu memetakan hubungan kompleks antara fitur input dan target output secara sistematis dan terstruktur. Splitting merujuk pada proses pemilihan atribut dan nilai ambang tertentu untuk membagi dataset pada suatu node menjadi dua atau lebih subset. Setiap pemisahan dilakukan dengan tujuan memaksimalkan keseragaman (homogenitas) data dalam setiap subset hasil pembagian. Dengan kata lain, splitting berupaya mengelompokkan data sedemikian rupa sehingga setiap kelompok mengandung data dengan karakteristik target yang serupa. Proses splitting dilakukan berdasarkan kriteria matematis tertentu yang mengukur kualitas pemisahan. Pada masalah klasifikasi, kriteria yang umum digunakan meliputi entropy, information gain, dan gini impurity. Kriteria-kriteria ini mengevaluasi seberapa baik suatu atribut mampu memisahkan kelas target. Atribut dengan nilai pemisahan terbaik akan dipilih sebagai dasar splitting pada node tersebut.

Pada data numerik, splitting biasanya dilakukan dengan menentukan nilai ambang (threshold). Dataset dibagi menjadi dua bagian, misalnya data dengan nilai fitur kurang dari atau sama dengan ambang tertentu dan data dengan nilai lebih besar dari ambang tersebut. Pemilihan ambang dilakukan dengan menguji berbagai kemungkinan nilai dan memilih yang menghasilkan pemisahan terbaik berdasarkan kriteria yang digunakan. Pada data kategorikal, splitting dilakukan berdasarkan kategori atau kombinasi kategori tertentu. Setiap nilai kategori dapat membentuk cabang tersendiri atau digabungkan dengan kategori lain

tergantung pada algoritma yang digunakan. Pendekatan ini memungkinkan Decision Tree menangani berbagai tipe data tanpa memerlukan transformasi kompleks. Setelah splitting dilakukan pada satu node, proses berlanjut ke tahap recursive partitioning [10]. Recursive partitioning adalah proses pembagian data secara berulang pada setiap subset hasil splitting hingga tercapai kondisi tertentu. Proses ini bersifat rekursif karena setiap subset diperlakukan sebagai dataset baru yang akan dibagi kembali menggunakan prinsip yang sama.

Recursive partitioning memungkinkan Decision Tree membangun struktur hierarkis yang semakin spesifik. Pada setiap level pohon, data menjadi semakin homogen karena hanya subset tertentu yang dipertimbangkan. Dengan pendekatan ini, model mampu menangkap pola non-linear dan interaksi antar fitur yang kompleks secara bertahap. Proses rekursif ini terus berlangsung hingga salah satu kondisi penghentian terpenuhi. Kondisi tersebut dapat berupa seluruh data dalam node berasal dari kelas yang sama, jumlah data dalam node terlalu sedikit, atau kedalaman pohon telah mencapai batas maksimum. Kondisi penghentian penting untuk mencegah pertumbuhan pohon yang berlebihan. Tanpa pembatasan yang tepat, recursive partitioning dapat menghasilkan pohon yang sangat dalam dan kompleks. Kondisi ini sering menyebabkan overfitting, di mana model terlalu menyesuaikan diri terhadap data latih dan kehilangan kemampuan generalisasi. Oleh karena itu, pengendalian proses recursive partitioning merupakan aspek penting dalam desain Decision Tree. Untuk mengatasi masalah tersebut, digunakan teknik seperti pre pruning dan post pruning. Pre pruning menghentikan proses splitting lebih awal berdasarkan kriteria tertentu, sedangkan post-pruning memangkas cabang pohon setelah pohon terbentuk sepenuhnya. Kedua teknik ini bertujuan menjaga keseimbangan antara kompleksitas model dan performa prediksi. Konsep splitting dan

recursive partitioning juga menjelaskan mengapa Decision Tree bersifat tidak stabil. Perubahan kecil pada data latih dapat mengubah urutan pemilihan atribut pada proses splitting awal, yang selanjutnya berdampak pada seluruh struktur pohon. Ketidakstabilan ini menjadi salah satu alasan dikembangkannya metode ensemble seperti Random Forest.

Dalam Random Forest, prinsip splitting dan recursive partitioning tetap digunakan, tetapi diterapkan pada banyak pohon dengan variasi data dan fitur. Pendekatan ini mengurangi variansi model dan menghasilkan prediksi yang lebih stabil. Dengan demikian, splitting dan recursive partitioning tidak hanya relevan pada Decision Tree tunggal, tetapi juga menjadi fondasi metode ensemble. Dari sudut pandang interpretabilitas, splitting dan recursive partitioning memungkinkan setiap jalur dari akar hingga daun diterjemahkan sebagai aturan if then. Setiap aturan mencerminkan hasil dari serangkaian keputusan bertahap yang dihasilkan melalui proses pembagian data secara rekursif. Hal ini menjadikan Decision Tree mudah dipahami dan dijelaskan.

Secara komputasi, recursive partitioning bersifat greedy, artinya keputusan splitting terbaik diambil secara lokal pada setiap node tanpa mempertimbangkan dampak global terhadap struktur pohon. Pendekatan ini membuat proses pelatihan relatif efisien, tetapi juga menjadi sumber keterbatasan karena tidak menjamin solusi global yang optimal. Dalam praktik machine learning modern, pemahaman mendalam tentang splitting dan recursive partitioning sangat penting untuk mengoptimalkan penggunaan Decision Tree. Peneliti dan praktisi perlu memahami bagaimana kriteria splitting, kondisi penghentian, dan teknik pruning saling berinteraksi dalam membentuk model akhir. Secara keseluruhan, konsep splitting dan recursive partitioning menjelaskan mekanisme inti bagaimana Decision Tree belajar dari data. Melalui pembagian data yang bertahap dan rekursif, model mampu membangun

struktur keputusan yang logis, fleksibel, dan interpretatif. Konsep ini menjadi dasar penting tidak hanya bagi Decision Tree, tetapi juga bagi berbagai metode ensemble yang berkembang dalam pembelajaran mesin modern.

Contoh: Implementasi Konsep Splitting dan Recursive Partitioning

1. Import Library

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

2. Membuat Dataset Contoh (Kelayakan Kredit)

```
# Membuat dataset
data = {
   'Usia': [25, 40, 35, 28, 50, 45],
   'Penghasilan': [3, 7, 6, 4, 8, 5],  # dalam juta
   'Riwayat_Kredit': ['Buruk', 'Baik', 'Baik', 'Buruk', 'Baik', 'Buruk'],
   'Kelayakan': ['Tidak', 'Layak', 'Layak', 'Tidak', 'Layak', 'Tidak']
}

df = pd.DataFrame(data)
df
```

	Usia	Penghasilan	Riwayat_Kredit	Kelayakan
0	25	3	Buruk	Tidak
1	40	7	Baik	Layak
2	35	6	Baik	Layak
3	28	4	Buruk	Tidak
4	50	8	Baik	Layak
5	45	5	Buruk	Tidak

3. Preprocessing Data : Decision Tree membutuhkan data numerik, sehingga encoding dilakukan untuk fitur kategorikal.

```
# Encoding fitur kategorikal
df_encoded = df.copy()
df_encoded['Riwayat_Kredit'] =
df_encoded['Riwayat_Kredit'].map({'Buruk': 0, 'Baik': 1})
df_encoded['Kelayakan'] =
df_encoded['Kelayakan'].map({'Tidak': 0, 'Layak': 1})
df_encoded
```

	Usia	Penghasilan	Riwayat_Kredit	Kelayakan
0	25	3	0	0
1	40	7	1	1
2	35	6	1	1
3	28	4	0	0
4	50	8	1	1
5	45	5	0	0

4. Menentukan Fitur (X) dan Target (y)

```
X = df_encoded[['Usia', 'Penghasilan', 'Riwayat_Kredit']]
y = df_encoded['Kelayakan']
```

5. Split Data (Training dan Testing)

```
X_train, X_test, y_train, y_test = train_test_split(
```

```
    X, y, test_size=0.3, random_state=42
)
```

6. Melatih Model Decision Tree : ada tahap ini terjadi splitting dan recursive partitioning secara otomatis oleh algoritma.

```
model = DecisionTreeClassifier(
    criterion='gini',     # kriteria splitting
    max_depth=3,          # membatasi recursive partitioning
    random_state=42
)
model.fit(X_train, y_train)
```

```
DecisionTreeClassifier(max_depth=3, random_state=42)
```

7. Evaluasi Model

```
y_pred = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

8. Confusion Matrix

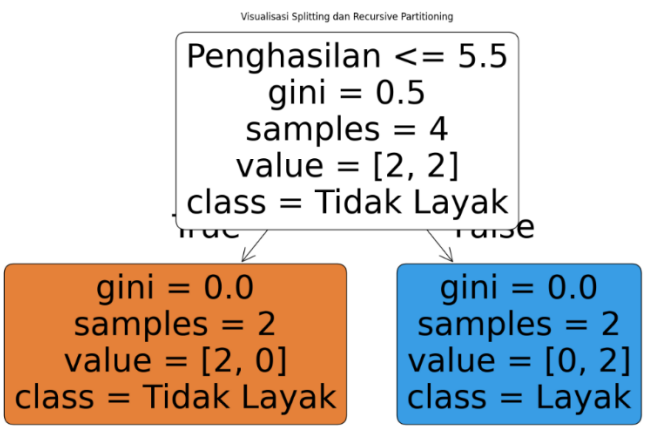
```
Accuracy: 1.0

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
weighted avg       1.00      1.00      1.00         2
```

```
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:\n", cm)
```

9. Visualisasi Pohon Keputusan : Visualisasi ini menunjukkan secara eksplisit proses splitting dan recursive partitioning.

```
Confusion Matrix:
 [[1 0]
 [0 1]]
```

```
plt.figure(figsize=(16,10))
plot_tree(
    model,
    feature_names=['Usia', 'Penghasilan', 'Riwayat_Kredit'],
    class_names=['Tidak Layak', 'Layak'],
    filled=True,
    rounded=True
)
plt.title("Visualisasi Splitting dan Recursive Partitioning")
plt.show()
```



10. Menampilkan Aturan (If–Then Rules)

```
from sklearn.tree import export_text

rules = export_text(
    model,
    feature_names=['Usia', 'Penghasilan', 'Riwayat_Kredit']
)

print("Aturan Keputusan:\n")
print(rules)
```

```
Aturan Keputusan:

|--- Penghasilan <= 5.50
|   |--- class: 0
|--- Penghasilan >  5.50
|   |--- class: 1
```

11. Prediksi Data Baru (Simulasi Deployment)

```
# Data baru
data_baru = pd.DataFrame({
    'Usia': [32],
    'Penghasilan': [6],
    'Riwayat_Kredit': [1]  # Baik
})

prediksi = model.predict(data_baru)
print("Prediksi Kelayakan Kredit:",
      "Layak" if prediksi[0] == 1 else "Tidak Layak")
```

```
Prediksi Kelayakan Kredit: Layak
```

2.3 Kriteria Pemilihan Atribut: Entropy dan Information Gain

Dalam algoritma Decision Tree, pemilihan atribut terbaik pada setiap node merupakan tahapan yang sangat krusial karena secara langsung menentukan kualitas struktur pohon yang dibentuk. Proses ini menjadi fondasi utama dalam mekanisme pemisahan data (splitting), di mana dataset dibagi menjadi beberapa subset yang lebih kecil dan lebih terstruktur. Keputusan pada tahap awal ini akan memengaruhi seluruh proses pembentukan pohon, termasuk kedalaman pohon, jumlah cabang, serta akurasi model secara keseluruhan. Pemilihan atribut dalam Decision Tree tidak dilakukan secara sembarangan atau berdasarkan intuisi semata. Setiap atribut dievaluasi menggunakan kriteria matematis tertentu yang dirancang untuk mengukur efektivitas atribut tersebut dalam memisahkan data. Kriteria ini berfungsi sebagai alat objektif untuk menilai seberapa baik suatu atribut mampu menghasilkan pembagian data yang bermakna dan relevan terhadap variabel target. Tujuan utama dari proses pemilihan atribut adalah menghasilkan subset data yang lebih homogen dibandingkan dataset awal. Homogenitas dalam konteks ini berarti bahwa setiap subset hasil pemisahan memiliki dominasi kelas tertentu atau variasi nilai target yang lebih kecil. Semakin homogen subset yang dihasilkan, semakin rendah tingkat ketidakpastian model dalam melakukan prediksi pada node tersebut.

Untuk mengukur tingkat homogenitas dan ketidakpastian data, digunakan konsep-konsep yang berasal dari teori informasi. Teori informasi menyediakan kerangka matematis yang kuat untuk mengukur seberapa banyak informasi yang terkandung dalam suatu distribusi data. Dalam Decision Tree, kerangka ini dimanfaatkan untuk mengevaluasi kualitas pemisahan data secara kuantitatif dan sistematis. Dua konsep fundamental yang paling banyak digunakan dalam pemilihan atribut adalah Entropy dan Information Gain [5]. Entropy digunakan untuk

mengukur tingkat ketidakpastian atau impuritas dalam suatu kumpulan data. Nilai entropy yang tinggi menunjukkan bahwa data masih bercampur dan sulit diprediksi, sedangkan nilai entropy yang rendah menunjukkan bahwa data cenderung homogen dan lebih mudah diklasifikasikan. Information Gain berperan sebagai ukuran penurunan nilai entropy setelah suatu atribut digunakan untuk melakukan pemisahan data. Dengan kata lain, information gain mengukur seberapa besar pengurangan ketidakpastian yang dihasilkan oleh suatu atribut. Atribut yang menghasilkan penurunan entropy terbesar dianggap paling informatif dan dipilih sebagai dasar pemisahan pada node tersebut. Penggunaan entropy dan information gain memungkinkan Decision Tree membangun struktur pohon secara bertahap dan rasional. Pada setiap node, algoritma memilih atribut yang secara lokal paling efektif dalam memisahkan data. Pendekatan ini bersifat greedy, namun terbukti mampu menghasilkan pohon keputusan yang efisien dan memiliki performa prediksi yang baik dalam banyak kasus praktis.

Secara keseluruhan, pemilihan atribut berdasarkan entropy dan information gain menjadikan Decision Tree sebagai algoritma yang tidak hanya kuat secara prediktif, tetapi juga memiliki dasar teoritis yang jelas. Dengan mengandalkan konsep-konsep dari teori informasi, proses pembentukan pohon keputusan dapat dipahami secara matematis dan logis, sehingga menghasilkan model yang terstruktur, interpretatif, dan relevan untuk berbagai permasalahan pembelajaran mesin.

1. Konsep Entropy

Konsep entropy merupakan salah satu dasar penting dalam algoritma Decision Tree yang digunakan untuk mengukur tingkat ketidakpastian atau ketidakteraturan dalam suatu kumpulan data. Dalam konteks pembelajaran mesin, entropy menggambarkan seberapa sulit model dalam memprediksi kelas target berdasarkan distribusi data yang

ada. Semakin tinggi nilai entropy, semakin besar ketidakpastian yang terkandung dalam data, sedangkan entropy yang rendah menunjukkan bahwa data cenderung homogen dan lebih mudah dipisahkan ke dalam kelas tertentu [11]. Secara konseptual, entropy berasal dari teori informasi yang diperkenalkan oleh Claude Shannon. Dalam teori ini, entropy digunakan untuk mengukur jumlah informasi atau tingkat kejutan yang dihasilkan oleh suatu peristiwa. Ketika diterapkan pada Decision Tree, konsep ini diadaptasi untuk mengukur tingkat pencampuran kelas dalam sebuah node. Dengan demikian, entropy menjadi indikator kuantitatif yang membantu algoritma memahami seberapa informatif suatu node terhadap proses klasifikasi.

Dalam praktiknya, entropy bernilai nol apabila seluruh data dalam suatu node berasal dari satu kelas yang sama. Kondisi ini menunjukkan tidak adanya ketidakpastian karena hasil prediksi dapat ditentukan dengan pasti. Sebaliknya, entropy mencapai nilai maksimum ketika data dalam node terbagi secara merata ke dalam beberapa kelas, karena pada kondisi ini model memiliki tingkat kebingungan yang tinggi dalam menentukan kelas data. Entropy dihitung berdasarkan proporsi masing-masing kelas dalam suatu dataset atau node. Setiap kelas memberikan kontribusi terhadap nilai entropy sesuai dengan tingkat kemunculannya. Pendekatan ini memastikan bahwa entropy tidak hanya mempertimbangkan jumlah kelas, tetapi juga distribusi relatif antar kelas tersebut. Dengan demikian, perubahan kecil pada distribusi data dapat memengaruhi nilai entropy secara signifikan. Peran utama entropy dalam Decision Tree adalah sebagai alat evaluasi sebelum dan sesudah proses pemisahan data. Pada awalnya, entropy dihitung untuk dataset secara keseluruhan guna mengetahui tingkat ketidakpastian awal. Setelah suatu atribut digunakan untuk melakukan pemisahan, entropy dihitung kembali pada setiap subset data. Penurunan nilai entropy setelah

pemisahan menunjukkan bahwa atribut tersebut efektif dalam meningkatkan homogenitas data. Entropy juga berperan penting dalam menentukan struktur akhir pohon keputusan. Dengan meminimalkan entropy pada setiap tahap pemisahan, Decision Tree cenderung menghasilkan node-node daun yang berisi data dengan kelas yang dominan. Hal ini berkontribusi pada peningkatan akurasi model sekaligus menjaga struktur pohon tetap logis dan mudah diinterpretasikan.

Secara keseluruhan, konsep entropy menyediakan dasar matematis yang kuat untuk memahami bagaimana Decision Tree bekerja dalam mengurangi ketidakpastian data. Dengan mengandalkan entropy, algoritma dapat membuat keputusan pemisahan yang objektif dan konsisten. Pemahaman yang baik terhadap konsep ini sangat penting bagi peneliti dan praktisi untuk mengoptimalkan penggunaan Decision Tree serta menilai kualitas hasil pemodelan yang dihasilkan. Secara matematis, entropy dirumuskan sebagai:

$$Entropy(S) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

di mana $p_i$adalah proporsi data dari kelas ke-i dalam himpunan data S. Logaritma basis dua digunakan karena entropy diukur dalam satuan bit. Rumus ini menunjukkan bahwa entropy mempertimbangkan seluruh distribusi kelas dalam suatu node, bukan hanya kelas mayoritas. Dalam praktik, entropy digunakan untuk mengevaluasi kualitas suatu node sebelum dan sesudah dilakukan pemisahan. Tujuan utama algoritma Decision Tree adalah mengurangi entropy secara bertahap, sehingga pada node-node akhir (daun) diperoleh data yang sangat homogen. Dengan demikian, entropy berfungsi sebagai indikator tingkat ketidakpastian yang ingin diminimalkan oleh model.

2. Konsep Information Gain

Konsep Information Gain merupakan ukuran kuantitatif yang digunakan dalam algoritma Decision Tree untuk menentukan seberapa efektif suatu atribut dalam memisahkan data. Information Gain mengukur jumlah penurunan ketidakpastian atau impuritas data setelah dilakukan pemisahan berdasarkan atribut tertentu. Dengan kata lain, konsep ini menunjukkan seberapa besar informasi baru yang diperoleh ketika suatu atribut digunakan sebagai dasar pengambilan keputusan pada sebuah node. Information Gain sangat erat kaitannya dengan konsep entropy. Entropy mengukur tingkat ketidakpastian data sebelum pemisahan, sedangkan Information Gain menghitung selisih antara entropy awal dengan entropy rata-rata setelah data dibagi berdasarkan suatu atribut. Jika suatu atribut mampu menghasilkan subset data yang lebih homogen, maka entropy setelah pemisahan akan lebih rendah, sehingga nilai Information Gain menjadi lebih tinggi. Oleh karena itu, Information Gain dapat dipahami sebagai indikator keberhasilan suatu atribut dalam mengurangi ketidakpastian.

Dalam praktik Decision Tree, setiap atribut yang tersedia dievaluasi nilai Information Gain-nya pada setiap node. Proses ini dilakukan dengan menghitung entropy awal dari dataset, kemudian menghitung entropy dari setiap subset hasil pemisahan, yang selanjutnya dirata-ratakan berdasarkan proporsi jumlah data. Atribut yang menghasilkan nilai Information Gain terbesar akan dipilih sebagai atribut terbaik untuk melakukan pemisahan pada node tersebut, karena dianggap paling informatif terhadap variabel target. Penggunaan Information Gain memungkinkan Decision Tree membangun struktur pohon secara bertahap dan logis. Dengan selalu memilih atribut yang memberikan pengurangan ketidakpastian terbesar, model dapat mencapai tingkat pemisahan kelas yang baik sejak level awal pohon. Pendekatan ini

membantu menghasilkan pohon keputusan yang relatif efisien, dengan jumlah node yang tidak berlebihan dan jalur keputusan yang jelas. Meskipun memiliki keunggulan dalam menghasilkan pemisahan yang informatif, Information Gain juga memiliki keterbatasan. Salah satu kelemahan utamanya adalah kecenderungan untuk memilih atribut dengan banyak nilai unik, karena atribut semacam itu sering kali menghasilkan penurunan entropy yang besar secara matematis. Kondisi ini dapat menyebabkan bias dalam pemilihan atribut, meskipun atribut tersebut belum tentu relevan secara konseptual terhadap permasalahan yang diselesaikan. Secara keseluruhan, Information Gain berperan penting sebagai kriteria pemilihan atribut dalam pembentukan Decision Tree. Konsep ini memberikan dasar matematis yang jelas untuk menentukan atribut paling informatif dalam setiap tahap pemisahan data. Dengan memahami Information Gain secara mendalam, pengguna dapat lebih memahami bagaimana Decision Tree mengambil keputusan dan bagaimana struktur pohon dibentuk secara sistematis berdasarkan prinsip pengurangan ketidakpastian.

Secara matematis, information gain dirumuskan sebagai:

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} \times Entropy(S_v)$$

di mana:

- Sadalah himpunan data awal,
- Aadalah atribut yang dievaluasi,
- $S_v$adalah subset data dengan nilai atribut v,
- $|S_v|/|S|$merupakan bobot proporsi data pada subset tersebut.

Rumus ini menunjukkan bahwa information gain mempertimbangkan baik tingkat impuritas subset maupun proporsi ukuran masing-masing subset. Secara konseptual, information gain menilai efektivitas suatu

atribut dalam mengurangi ketidakpastian. Jika suatu atribut mampu memisahkan data sedemikian rupa sehingga setiap subset hampir murni, maka entropy subset akan kecil dan information gain menjadi besar. Oleh karena itu, information gain menjadi kriteria utama dalam algoritma seperti ID3 dan C4.5.

3. Peran Entropy dan Information Gain dalam Pembentukan Pohon

Peran entropy dan information gain sangat fundamental dalam proses pembentukan pohon keputusan (Decision Tree), karena kedua konsep ini menjadi dasar utama dalam menentukan bagaimana data dipisahkan pada setiap node. Tanpa adanya ukuran kuantitatif seperti entropy dan information gain, pemilihan atribut akan bersifat subjektif dan tidak terarah. Oleh karena itu, kedua konsep ini berfungsi sebagai mekanisme objektif untuk membangun struktur pohon yang logis, efisien, dan berbasis data. Entropy berperan sebagai ukuran awal untuk menilai tingkat ketidakpastian atau ketidakhomogenan data pada suatu node. Pada tahap awal pembentukan pohon, entropy dihitung pada dataset secara keseluruhan yang berada di root node [9]. Nilai entropy ini merepresentasikan kondisi awal data sebelum dilakukan pemisahan, sekaligus menjadi tolok ukur untuk menilai seberapa efektif suatu atribut dalam memperbaiki struktur data melalui proses splitting. Information gain kemudian berperan sebagai alat pembanding antar atribut yang tersedia. Dengan menghitung selisih antara entropy awal dan entropy setelah pemisahan berdasarkan suatu atribut, information gain menunjukkan seberapa besar pengurangan ketidakpastian yang dihasilkan oleh atribut tersebut. Atribut dengan information gain tertinggi dianggap paling informatif dan dipilih untuk menjadi dasar pemisahan pada node tersebut.

Dalam pembentukan pohon keputusan, entropy dan information gain bekerja secara iteratif dan berulang. Setelah pemisahan pertama dilakukan, setiap subset data hasil pemisahan akan menjadi node baru. Pada node-node ini, entropy kembali dihitung, dan information gain kembali digunakan untuk memilih atribut terbaik berikutnya. Proses ini terus berlangsung secara rekursif hingga tercapai kondisi penghentian tertentu, seperti data yang sudah homogen atau batas kedalaman pohon. Peran entropy dan information gain juga sangat penting dalam membentuk struktur pohon yang efisien. Dengan selalu memilih atribut yang paling mampu mengurangi ketidakpastian, pohon keputusan cenderung mencapai pemisahan kelas yang baik pada level-level awal. Hal ini membantu mengurangi jumlah node yang tidak perlu dan menghasilkan pohon yang lebih ringkas serta mudah diinterpretasikan. Selain itu, penggunaan entropy dan information gain berkontribusi terhadap kemampuan interpretabilitas Decision Tree. Setiap keputusan pemisahan dapat dijelaskan secara logis sebagai upaya untuk mengurangi ketidakpastian data. Jalur dari root node hingga leaf node dapat dipahami sebagai rangkaian keputusan yang secara bertahap meningkatkan kepastian terhadap kelas target, sehingga model mudah dipahami oleh manusia.

Namun, peran entropy dan information gain juga memiliki implikasi terhadap kompleksitas pohon. Jika digunakan tanpa pembatasan, proses pemisahan berbasis information gain dapat menghasilkan pohon yang sangat dalam dan kompleks, terutama pada data yang mengandung noise. Oleh karena itu, dalam praktik sering dikombinasikan dengan teknik pembatasan seperti pruning untuk menjaga keseimbangan antara akurasi dan generalisasi. Secara keseluruhan, entropy dan information gain berperan sebagai fondasi matematis dalam pembentukan pohon keputusan. Entropy menyediakan

ukuran ketidakpastian yang ingin dikurangi, sedangkan information gain menjadi kriteria untuk memilih atribut yang paling efektif dalam mengurangi ketidakpastian tersebut. Kombinasi keduanya memungkinkan Decision Tree dibangun secara sistematis, rasional, dan berbasis teori informasi, sehingga menghasilkan model yang akurat, terstruktur, dan mudah diinterpretasikan.

4. Kelebihan dan Keterbatasan Entropy dan Information Gain

   Kelebihan Entropy dan Information Gain:

   a. Dasar Teoritis yang Kuat : Entropy dan Information Gain berasal dari teori informasi yang memiliki landasan matematis yang jelas dan mapan. Penggunaan konsep ini memberikan justifikasi ilmiah yang kuat dalam proses pemilihan atribut pada Decision Tree. Dengan dasar teori yang solid, proses pembentukan pohon keputusan dapat dipahami dan dijelaskan secara rasional.

   b. Mengukur Ketidakpastian Secara Kuantitatif : Entropy mampu mengukur tingkat ketidakpastian atau impuritas data secara kuantitatif. Dengan ukuran numerik ini, algoritma dapat membandingkan berbagai kemungkinan pemisahan data secara objektif. Information Gain kemudian memanfaatkan nilai entropy untuk menilai seberapa efektif suatu atribut dalam mengurangi ketidakpastian tersebut.

   c. Menghasilkan Pemisahan Data yang Informatif : Information Gain secara langsung mengukur pengurangan entropy yang dihasilkan oleh suatu atribut. Hal ini membuat atribut yang dipilih benar-benar memiliki kontribusi signifikan dalam memisahkan kelas target. Dengan demikian, pemisahan data yang dihasilkan

cenderung lebih informatif dan relevan terhadap permasalahan yang diselesaikan.

d. Mendukung Interpretabilitas Model : Karena setiap pemilihan atribut didasarkan pada pengurangan ketidakpastian, struktur pohon yang dihasilkan relatif mudah dijelaskan. Setiap node keputusan dapat diinterpretasikan sebagai langkah yang bertujuan meningkatkan kepastian terhadap hasil prediksi. Hal ini menjadikan Decision Tree berbasis entropy dan information gain mudah dipahami oleh pengguna.

Keterbatasan Entropy dan Information Gain:

a. Bias terhadap Atribut dengan Banyak Nilai : Salah satu keterbatasan utama Information Gain adalah kecenderungannya memilih atribut dengan banyak nilai unik. Atribut semacam ini sering menghasilkan penurunan entropy yang besar secara matematis, meskipun tidak selalu relevan secara konseptual. Bias ini dapat menyebabkan pemilihan atribut yang kurang bermakna dalam konteks masalah.
b. Kompleksitas Komputasi Relatif Tinggi : Perhitungan entropy dan information gain melibatkan operasi logaritma dan evaluasi seluruh distribusi kelas. Pada dataset besar atau dengan banyak atribut, proses ini dapat menjadi lebih mahal secara komputasi dibandingkan kriteria lain seperti Gini Impurity. Hal ini dapat memengaruhi efisiensi pelatihan model.
c. Sensitif terhadap Noise pada Data : Entropy dan Information Gain cukup sensitif terhadap noise dan outlier dalam data. Data yang bising dapat memengaruhi distribusi kelas pada node tertentu, sehingga menghasilkan pemisahan yang kurang optimal. Jika

tidak dikombinasikan dengan teknik pembatasan seperti pruning, pohon yang dihasilkan berisiko mengalami overfitting.

d. Tidak Menjamin Solusi Global Optimal : Pemilihan atribut berdasarkan Information Gain dilakukan secara lokal pada setiap node dengan pendekatan *greedy*. Meskipun efektif dalam banyak kasus, pendekatan ini tidak menjamin bahwa struktur pohon yang dihasilkan merupakan solusi global terbaik. Keputusan awal yang kurang optimal dapat memengaruhi keseluruhan struktur pohon.

Tabel 3. Perbandingan Entropy, Gini, dan Gain Rasio

Kriteria	Entropy	Gini Impurity	Gain Ratio
Konsep Dasar	Mengukur tingkat ketidakpastian atau impuritas data	Mengukur peluang salah klasifikasi	Information Gain yang dinormalisasi
Digunakan pada Algoritma	ID3, C4.5	CART	C4.5
Rentang Nilai	$0 – \log_2(k)$	$0 – (1 − 1/k)$	0 – 1
Nilai Minimum	0 (node murni)	0 (node murni)	0
Nilai Maksimum	Saat kelas seimbang	Saat kelas seimbang	Bergantung split info
Sensitivitas	Sensitif terhadap perubahan distribusi	Lebih cepat dan stabil	Mengoreksi bias entropy

Bias Atribut Banyak Nilai	Tinggi	Sedang	Rendah
Kompleksitas Komputasi	Relatif lebih mahal	Lebih ringan	Paling mahal
Tujuan Optimasi	Memaksimalkan Information Gain	Meminimalkan Gini	Memaksimalkan Gain Ratio
Kelebihan Utama	Dasar teori informasi	Efisien & cepat	Lebih adil pemilihan atribut
Keterbatasan	Bias ke banyak kategori	Kurang informatif	Lebih kompleks

Contoh:

Studi Kasus

Dataset sederhana: Kelayakan Kredit

Target: Layak / Tidak Layak

Step 1 : Import Library

```
import pandas as pd
import numpy as np
from math import log2
```

Step 2 : Membuat Dataset

```
data = {
   'Usia': [25, 40, 35, 28, 50, 45],
   'Penghasilan': [3, 7, 6, 4, 8, 5],
   'Riwayat_Kredit': ['Buruk', 'Baik', 'Baik', 'Buruk', 'Baik', 'Buruk'],
   'Kelayakan': ['Tidak', 'Layak', 'Layak', 'Tidak', 'Layak', 'Tidak']
}
```

```
df = pd.DataFrame(data)
df
```

	Usia	Penghasilan	Riwayat_Kredit	Kelayakan
0	25	3	Buruk	Tidak
1	40	7	Baik	Layak
2	35	6	Baik	Layak
3	28	4	Buruk	Tidak
4	50	8	Baik	Layak
5	45	5	Buruk	Tidak

Step 3 : Fungsi Entropy

```
def entropy(target_col):
    values, counts = np.unique(target_col, return_counts=True)
    entropy_value = 0
    for count in counts:
        p = count / sum(counts)
        entropy_value -= p * log2(p)
    return entropy_value
entropy_total = entropy(df['Kelayakan'])
entropy_total
```

```
np.float64(1.0)
```

Step 4 : Information Gain

```
def information_gain(df, feature, target):
    total_entropy = entropy(df[target])
    values, counts = np.unique(df[feature], return_counts=True)
    weighted_entropy = 0
    for value, count in zip(values, counts):
        subset = df[df[feature] == value]
```

```
        weighted_entropy += (count / sum(counts)) * entropy(subset[target])
    return total_entropy - weighted_entropy
for col in ['Usia', 'Penghasilan', 'Riwayat_Kredit']:
```

```
Usia 1.0
Penghasilan 1.0
Riwayat_Kredit 1.0
```

```
    print(col, information_gain(df, col, 'Kelayakan'))
```

Step 5 : Gini Impurity

```
def gini(target_col):
    values, counts = np.unique(target_col, return_counts=True)
    gini_value = 1
    for count in counts:
        p = count / sum(counts)
        gini_value -= p ** 2
    return gini_value
gini(df['Kelayakan'])
```

```
np.float64(0.5)
```

Step 6 : Gini Setelah Split

```
def gini_split(df, feature, target):
    values, counts = np.unique(df[feature], return_counts=True)
    weighted_gini = 0

    for value, count in zip(values, counts):
        subset = df[df[feature] == value]
        weighted_gini += (count / sum(counts)) * gini(subset[target])
    return weighted_gini
```

for col in ['Usia', 'Penghasilan', 'Riwayat_Kredit']:

```
Usia 0.0
Penghasilan 0.0
Riwayat_Kredit 0.0
```

```
print(col, gini_split(df, col, 'Kelayakan'))
```

Step 7 : Gain Rasio

```
def split_info(df, feature):
    values, counts = np.unique(df[feature], return_counts=True)
    split_info_value = 0
    for count in counts:
        p = count / sum(counts)
        split_info_value -= p * log2(p)
    return split_info_value


def gain_ratio(df, feature, target):
    ig = information_gain(df, feature, target)
    si = split_info(df, feature)
    if si == 0:
        return 0
    return ig / si
```

```
Usia 0.38685280723454163
Penghasilan 0.38685280723454163
Riwayat_Kredit 1.0
```

```
for col in ['Usia', 'Penghasilan', 'Riwayat_Kredit']:
    print(col, gain_ratio(df, col, 'Kelayakan'))
```

2.4 Gini Impurity dan Variance Reduction

Gini Impurity dan Variance Reduction merupakan dua kriteria penting dalam algoritma Decision Tree yang digunakan untuk menentukan atribut terbaik dalam proses pemisahan data (splitting). Kedua kriteria ini berfungsi sebagai ukuran kualitas pemisahan, namun diterapkan pada jenis permasalahan yang berbeda. Gini Impurity umumnya digunakan pada masalah klasifikasi, sedangkan Variance Reduction digunakan pada masalah regresi. Gini Impurity mengukur tingkat ketidakmurnian atau peluang kesalahan klasifikasi dalam suatu node. Konsep ini merepresentasikan seberapa besar kemungkinan suatu data yang dipilih secara acak dari sebuah node akan diklasifikasikan secara keliru jika labelnya ditentukan secara acak berdasarkan distribusi kelas pada node tersebut. Semakin rendah nilai Gini Impurity, semakin homogen data dalam node tersebut. Secara matematis, Gini Impurity dihitung berdasarkan proporsi masing-masing kelas dalam suatu node. Jika suatu node hanya berisi satu kelas, maka nilai Gini Impurity bernilai nol, yang menunjukkan kondisi paling murni. Sebaliknya, jika data terbagi relatif seimbang ke dalam beberapa kelas, nilai Gini Impurity akan mendekati nilai maksimum, yang menunjukkan tingkat ketidakpastian yang tinggi.

Dalam proses pembentukan Decision Tree, Gini Impurity digunakan untuk mengevaluasi kualitas pemisahan sebelum dan sesudah splitting. Algoritma akan menghitung nilai Gini Impurity pada node awal, kemudian menghitung Gini Impurity rata-rata tertimbang dari subset data hasil pemisahan. Atribut yang menghasilkan nilai Gini Impurity terendah setelah pemisahan akan dipilih sebagai atribut terbaik. Salah satu keunggulan utama Gini Impurity adalah efisiensi komputasinya. Dibandingkan dengan entropy, perhitungan Gini Impurity tidak melibatkan operasi logaritma, sehingga lebih cepat dan

ringan secara komputasi. Oleh karena itu, Gini Impurity banyak digunakan dalam algoritma CART (Classification and Regression Trees) yang dirancang untuk efisiensi dan skalabilitas. Selain efisiensi, Gini Impurity juga cenderung menghasilkan pohon yang relatif seimbang dan stabil. Meskipun secara konseptual mirip dengan entropy, Gini Impurity sering memberikan hasil pemisahan yang hampir sama, namun dengan waktu komputasi yang lebih singkat. Hal ini menjadikannya pilihan populer dalam implementasi praktis Decision Tree dan Random Forest.

Berbeda dengan Gini Impurity yang digunakan untuk klasifikasi, Variance Reduction digunakan sebagai kriteria pemilihan atribut pada Decision Tree untuk masalah regresi. Tujuan utama Variance Reduction adalah meminimalkan variasi atau penyebaran nilai target dalam setiap subset hasil pemisahan. Dengan kata lain, kriteria ini berusaha membuat nilai target dalam setiap node menjadi sedekat mungkin satu sama lain. Variansi merupakan ukuran statistik yang menunjukkan seberapa jauh nilai-nilai data menyebar dari nilai rata-ratanya. Dalam konteks Decision Tree regresi, variansi digunakan untuk mengukur ketidakseragaman nilai target dalam sebuah node. Semakin kecil variansi, semakin homogen nilai target dalam node tersebut, sehingga prediksi model menjadi lebih akurat. Proses Variance Reduction dilakukan dengan membandingkan variansi data sebelum pemisahan dengan variansi rata-rata tertimbang setelah pemisahan. Jika suatu atribut mampu menghasilkan penurunan variansi yang besar, maka atribut tersebut dianggap efektif dalam memisahkan data dan dipilih sebagai dasar splitting pada node tersebut. Secara konseptual, Variance Reduction memiliki tujuan yang serupa dengan entropy dan Gini Impurity, yaitu meningkatkan homogenitas data pada node-node hasil pemisahan. Namun, perbedaannya terletak pada jenis target yang dianalisis.

Variance Reduction berfokus pada nilai numerik kontinu, sedangkan entropy dan Gini Impurity berfokus pada label kelas diskret.

Dalam praktik, penggunaan Variance Reduction memungkinkan Decision Tree regresi membangun model yang mampu menangkap hubungan non-linear antara fitur dan target numerik. Dengan membagi data ke dalam subset dengan variansi rendah, model dapat menghasilkan prediksi yang lebih stabil dan mendekati nilai aktual. Baik Gini Impurity maupun Variance Reduction bekerja secara rekursif dalam pembentukan pohon keputusan. Pada setiap node, kriteria yang sesuai digunakan untuk memilih atribut terbaik, kemudian proses pemisahan diulang pada subset data yang dihasilkan. Proses ini terus berlanjut hingga tercapai kondisi penghentian tertentu, seperti kedalaman maksimum pohon atau jumlah minimum data dalam node. Namun, penggunaan kedua kriteria ini juga memiliki potensi menghasilkan pohon yang terlalu kompleks jika tidak dibatasi. Pada data yang mengandung noise, Gini Impurity dan Variance Reduction dapat menghasilkan pemisahan yang terlalu spesifik terhadap data latih. Oleh karena itu, teknik pembatasan seperti pruning dan pengaturan hyperparameter sangat penting untuk menjaga kemampuan generalisasi model. Dalam konteks Random Forest, baik Gini Impurity maupun Variance Reduction tetap digunakan sebagai kriteria splitting pada masing-masing pohon. Namun, karena Random Forest menggabungkan banyak pohon dengan variasi data dan fitur, dampak keterbatasan kriteria ini dapat diminimalkan. Pendekatan ensemble ini meningkatkan stabilitas dan akurasi model secara keseluruhan.

Tabel 4. Perbandingan Gini Impurity vs Variance Reduction

Aspek Perbandingan	Gini Impurity	Variance Reduction
Jenis Masalah	Klasifikasi	Regresi
Tujuan Utama	Mengukur tingkat ketidakmurnian kelas	Mengurangi variasi nilai target
Tipe Target	Kategorikal (diskret)	Kontinu / Numerik
Konsep Dasar	Probabilitas kesalahan klasifikasi	Penyebaran nilai terhadap rata-rata
Nilai Minimum	0 (node sepenuhnya murni)	0 (nilai target identik)
Nilai Maksimum	Terjadi saat distribusi kelas seimbang	Bergantung pada distribusi data
Kriteria Pemilihan Atribut	Meminimalkan Gini Impurity setelah split	Memaksimalkan penurunan variansi
Cara Evaluasi Split	Gini rata-rata tertimbang	Variansi rata-rata tertimbang
Kompleksitas Komputasi	Lebih ringan (tanpa logaritma)	Relatif ringan (operasi statistik)
Sensitivitas terhadap Noise	Cukup sensitif	Sensitif pada outlier ekstrem
Algoritma yang Menggunakan	CART, Random Forest (klasifikasi)	CART, Random Forest (regresi)
Interpretabilitas	Mudah dipahami secara konseptual	Semakin kecil variansi : prediksi lebih stabil
Kelebihan Utama	Cepat, stabil, efisien	Cocok untuk prediksi numerik

Keterbatasan	Tidak cocok untuk target kontinu	Rentan terhadap outlier
Contoh Aplikasi	Klasifikasi kredit, diagnosis penyakit	Prediksi harga, estimasi permintaan

2.5 Overfitting dan Underfitting pada Pohon Keputusan

A. Overfitting

Overfitting pada pohon keputusan (Decision Tree) merupakan kondisi ketika model belajar terlalu detail dari data latih sehingga tidak mampu melakukan generalisasi dengan baik pada data baru. Dalam situasi ini, pohon keputusan tidak hanya menangkap pola utama yang relevan, tetapi juga mempelajari noise, outlier, dan variasi acak yang seharusnya diabaikan. Akibatnya, meskipun performa pada data latih sangat tinggi, performa pada data uji atau data nyata justru menurun secara signifikan. Salah satu ciri utama overfitting pada pohon keputusan adalah struktur pohon yang sangat dalam dan kompleks. Pohon memiliki banyak node dan cabang, dengan daun yang berisi sangat sedikit sampel, bahkan terkadang hanya satu data. Struktur seperti ini menunjukkan bahwa model telah membuat aturan yang terlalu spesifik terhadap data latih, sehingga setiap variasi kecil diperlakukan sebagai pola penting. Overfitting sering terjadi karena sifat algoritma Decision Tree yang bersifat greedy dan rekursif [12]. Pada setiap node, algoritma memilih atribut yang paling baik secara lokal dalam memisahkan data, tanpa mempertimbangkan dampak jangka panjang terhadap keseluruhan struktur pohon. Jika proses splitting ini dibiarkan tanpa batasan, pohon akan terus tumbuh hingga seluruh data latih terklasifikasi dengan sempurna, meskipun pemisahan tersebut tidak bermakna secara umum.

Faktor lain yang memperparah overfitting adalah keberadaan noise dan outlier dalam data. Pohon keputusan sangat sensitif terhadap data

yang menyimpang karena algoritma akan berusaha menyesuaikan diri dengan setiap pola yang muncul. Noise yang seharusnya diabaikan justru dijadikan dasar pembentukan cabang baru, sehingga aturan yang dihasilkan menjadi tidak stabil dan sulit digeneralisasi. Overfitting pada pohon keputusan juga berkaitan erat dengan jumlah fitur dan kompleksitas dataset. Dataset dengan banyak fitur, terutama yang tidak relevan atau redundan, meningkatkan peluang terbentuknya aturan-aturan yang terlalu spesifik. Pohon keputusan akan mencoba memanfaatkan kombinasi fitur tersebut untuk meningkatkan akurasi data latih, meskipun kombinasi tersebut tidak memiliki makna prediktif yang kuat. Dampak utama overfitting adalah rendahnya kemampuan generalisasi model. Model terlihat sangat baik saat dievaluasi menggunakan data latih, tetapi gagal memberikan prediksi yang akurat pada data baru. Dalam aplikasi nyata, kondisi ini sangat berbahaya karena model menghasilkan keputusan yang tidak konsisten dan sulit dipercaya, meskipun secara teknis terlihat "sempurna" pada tahap pelatihan. Selain menurunkan performa prediksi, overfitting juga menyebabkan penurunan interpretabilitas pohon keputusan. Salah satu keunggulan Decision Tree adalah kemudahan interpretasi melalui aturan if then. Namun, ketika pohon terlalu dalam, aturan yang dihasilkan menjadi sangat panjang dan kompleks, sehingga sulit dipahami oleh manusia dan kehilangan nilai praktis sebagai sistem pendukung keputusan.

Overfitting pada pohon keputusan juga mencerminkan masalah bias variance trade-off. Model yang overfitting memiliki bias yang rendah karena sangat menyesuaikan diri dengan data latih, tetapi memiliki variansi yang tinggi karena sangat sensitif terhadap perubahan kecil pada data. Perubahan kecil pada dataset dapat menghasilkan struktur pohon yang sangat berbeda, menunjukkan ketidakstabilan model. Untuk mengendalikan overfitting, diperlukan pembatasan kompleksitas pohon.

Pendekatan yang umum digunakan adalah pre pruning, yaitu menghentikan pertumbuhan pohon lebih awal dengan membatasi kedalaman pohon, jumlah minimum data pada node, atau penurunan impuritas minimum. Dengan pembatasan ini, pohon dipaksa untuk hanya mempelajari pola yang benar benar penting. Selain itu, pendekatan lain yang efektif untuk mengurangi overfitting adalah penggunaan metode ensemble, seperti Random Forest. Dengan menggabungkan banyak pohon keputusan yang dibangun dari subset data dan fitur yang berbeda, pengaruh overfitting pada satu pohon dapat diminimalkan. Secara keseluruhan, pemahaman mendalam tentang overfitting pada pohon keputusan sangat penting agar model yang dibangun tidak hanya akurat pada data latih, tetapi juga andal dan stabil dalam menghadapi data nyata.

Berikut adalah cara untuk mengatasi Overfitting [13]:

1. Membatasi Kedalaman Pohon (Max Depth) : Salah satu cara paling efektif untuk mengatasi overfitting adalah dengan membatasi kedalaman maksimum pohon. Pohon yang terlalu dalam cenderung mempelajari detail-detail kecil dan noise pada data latih. Dengan membatasi kedalaman, model dipaksa untuk hanya menangkap pola-pola utama yang bersifat umum. Pendekatan ini membantu menjaga keseimbangan antara kompleksitas model dan kemampuan generalisasi.
2. Menentukan Jumlah Minimum Sampel pada Node : Overfitting sering terjadi ketika node daun hanya berisi sedikit data. Dengan menetapkan parameter seperti minimum jumlah sampel untuk melakukan split (min_samples_split) atau minimum sampel pada daun (min_samples_leaf), pertumbuhan pohon dapat dikendalikan. Aturan ini mencegah model membuat keputusan berdasarkan jumlah data yang terlalu kecil dan tidak representatif.

3. Menggunakan Pre Pruning : Pre pruning merupakan teknik penghentian pertumbuhan pohon secara dini. Pada metode ini, proses splitting dihentikan jika pemisahan yang dihasilkan tidak memberikan peningkatan kualitas yang signifikan, misalnya penurunan impuritas yang sangat kecil. Pre-pruning membantu mencegah pohon tumbuh terlalu kompleks sejak awal dan mengurangi risiko overfitting sebelum terjadi.
4. Menggunakan Post Pruning : Berbeda dengan pre-pruning, post pruning dilakukan setelah pohon terbentuk sepenuhnya. Cabang cabang yang tidak memberikan kontribusi signifikan terhadap performa pada data validasi akan dipangkas. Teknik ini memungkinkan model mempelajari pola awal secara lengkap, kemudian menyederhanakan struktur pohon agar lebih general dan stabil.
5. Menggunakan Data Validasi dan Cross Validation : Penggunaan data validasi terpisah atau cross validation sangat penting untuk mendeteksi dan mengatasi overfitting. Dengan mengevaluasi model pada data yang tidak digunakan dalam pelatihan, peneliti dapat mengetahui apakah performa tinggi pada data latih benar-benar mencerminkan kemampuan generalisasi. Parameter model kemudian disesuaikan berdasarkan hasil evaluasi ini.
6. Mengurangi Noise dan Outlier pada Data : Overfitting sering diperparah oleh data yang mengandung noise dan outlier. Melakukan pra pemrosesan data seperti pembersihan data, penghapusan outlier ekstrem, dan seleksi fitur yang relevan dapat membantu model fokus pada pola utama. Data yang lebih bersih menghasilkan aturan keputusan yang lebih stabil dan bermakna.

7. Seleksi dan Reduksi Fitur : Terlalu banyak fitur, terutama yang tidak relevan, meningkatkan risiko overfitting. Dengan melakukan seleksi fitur atau reduksi dimensi, model hanya menggunakan atribut yang benar-benar berkontribusi terhadap prediksi. Pendekatan ini menyederhanakan struktur pohon dan meningkatkan kemampuan generalisasi.
8. Menggunakan Metode Ensemble : Metode ensemble seperti Random Forest merupakan solusi yang sangat efektif untuk mengatasi overfitting pada Decision Tree. Dengan menggabungkan banyak pohon keputusan yang dibangun dari subset data dan fitur yang berbeda, variansi model dapat dikurangi secara signifikan. Kesalahan yang terjadi pada satu pohon akan dikompensasi oleh pohon lainnya, sehingga hasil prediksi menjadi lebih stabil.
9. Mengatur Kriteria Pemisahan dan Hyperparameter : Pemilihan kriteria pemisahan (misalnya Gini atau Entropy) dan pengaturan hyperparameter yang tepat juga berperan dalam mengendalikan overfitting. Hyperparameter tuning secara sistematis memungkinkan peneliti menemukan konfigurasi model yang optimal tanpa membuat pohon terlalu kompleks.

Eksperimen Overfitting pada Decision Tree (Sebelum vs Sesudah)

Studi Kasus:

Dataset: Breast Cancer Wisconsin **(**klasifikasi biner)

Tujuan: Menunjukkan bagaimana pohon terlalu dalam menyebabkan overfitting dan bagaimana pembatasan (regularisasi) memperbaikinya.

1. Persiapan Lingkungan

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
```

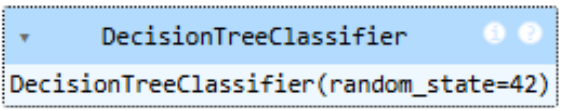
2. Memuat Dataset & Split Data

```
data = load_breast_cancer()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42, stratify=y
)
```

SEBELUM: MODEL OVERFITTING

3. Melatih Decision Tree TANPA Pembatasan

```
dt_overfit = DecisionTreeClassifier(random_state=42)
dt_overfit.fit(X_train, y_train)
```



4. Evaluasi Performa

```
train_acc_overfit = accuracy_score(y_train, dt_overfit.predict(X_train))
test_acc_overfit = accuracy_score(y_test, dt_overfit.predict(X_test))

print("Overfitting Model")
```

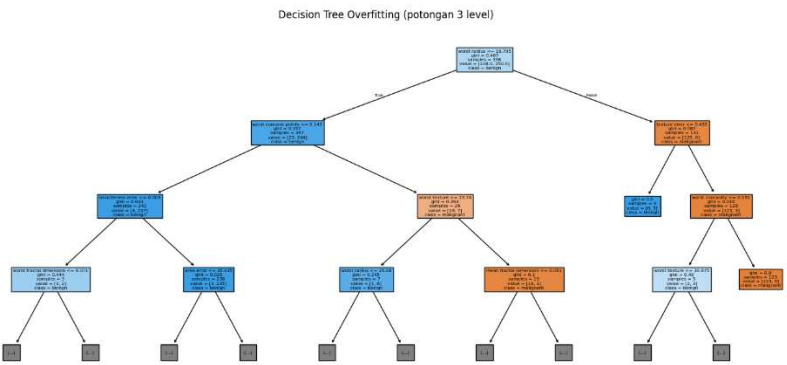
```
print("Train Accuracy:", train_acc_overfit)
print("Test Accuracy :", test_acc_overfit)
```

```
Overfitting Model
Train Accuracy: 1.0
Test Accuracy : 0.9181286549707602
```

5. Visualisasi Pohon (Opsional)

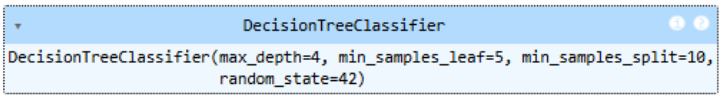
```
plt.figure(figsize=(18,8))
plot_tree(dt_overfit, max_depth=3, filled=True,
          feature_names=data.feature_names,
          class_names=data.target_names)
plt.title("Decision Tree Overfitting (potongan 3 level)")
plt.show()
```

SESUDAH: OVERFITTING DIATASI



6. Melatih Decision Tree DENGAN Pembatasan

```
dt_controlled = DecisionTreeClassifier(
   max_depth=4,
   min_samples_split=10,
   min_samples_leaf=5,
   random_state=42
)
dt_controlled.fit(X_train, y_train)
```



7. Evaluasi Performa

```
train_acc_ctrl = accuracy_score(y_train, dt_controlled.predict(X_train))
test_acc_ctrl = accuracy_score(y_test, dt_controlled.predict(X_test))

print("Controlled Model (Overfitting Reduced)")
print("Train Accuracy:", train_acc_ctrl)
print("Test Accuracy :", test_acc_ctrl)
```

```
Controlled Model (Overfitting Reduced)
Train Accuracy: 0.9723618090452262
Test Accuracy : 0.935672514619883
```

8. Evaluasi Tambahan dengan Cross Validation

```
cv_overfit = cross_val_score(dt_overfit, X, y, cv=5).mean()
cv_ctrl    = cross_val_score(dt_controlled, X, y, cv=5).mean()

print("CV Accuracy Overfitting Model:", cv_overfit)
print("CV Accuracy Controlled Model :", cv_ctrl)
```

9. Perbandingan

```
CV Accuracy Overfitting Model: 0.9173420276354604
CV Accuracy Controlled Model : 0.927930445582984
```

```
print("\nPERBANDINGAN")
print(f"Overfitting -> Train: {train_acc_overfit:.3f}, Test: {test_acc_overfit:.3f}")
print(f"Controlled -> Train: {train_acc_ctrl:.3f}, Test: {test_acc_ctrl:.3f}")
```

```
...
    PERBANDINGAN
    Overfitting  -> Train: 1.000, Test: 0.918
    Controlled   -> Train: 0.972, Test: 0.936
```

10. Interpretasi Akademik
    - Sebelum: Pohon tumbuh bebas : mempelajari noise : overfitting
    - Sesudah: Kompleksitas dibatasi : generalisasi meningkat
    - Penurunan kecil akurasi training adalah hal yang sehat demi performa data baru

B. Underfitting

Underfitting merupakan kondisi dalam pembelajaran mesin ketika sebuah model terlalu sederhana sehingga gagal menangkap pola penting yang terdapat dalam data. Model yang mengalami underfitting tidak mampu merepresentasikan hubungan antara fitur input dan target secara memadai, sehingga menghasilkan performa yang buruk baik pada data latih maupun data uji. Dalam konteks pohon keputusan, underfitting sering terjadi ketika struktur pohon terlalu dangkal atau aturan keputusan yang dibentuk terlalu umum. Salah satu ciri utama underfitting adalah rendahnya akurasi pada data latih. Berbeda dengan overfitting yang menunjukkan performa tinggi pada data latih, model underfitting bahkan tidak mampu menyesuaikan diri dengan data yang digunakan untuk pelatihan. Hal ini menunjukkan bahwa kapasitas model tidak cukup untuk mempelajari kompleksitas permasalahan yang sedang dihadapi. Underfitting pada pohon keputusan biasanya disebabkan oleh pembatasan kompleksitas model yang terlalu ketat. Parameter seperti max_depth yang terlalu kecil, min_samples_split yang terlalu besar, atau min_samples_leaf yang tinggi dapat menghentikan proses pemisahan

data terlalu dini. Akibatnya, pohon keputusan tidak memiliki cukup cabang untuk memodelkan variasi data secara tepat.

Selain pembatasan struktur, underfitting juga dapat disebabkan oleh pemilihan fitur yang kurang tepat. Jika fitur-fitur yang digunakan tidak relevan atau tidak cukup informatif terhadap target, maka model tidak memiliki dasar yang kuat untuk membentuk aturan keputusan yang akurat. Dalam kondisi ini, meskipun struktur pohon diperbesar, model tetap kesulitan menangkap pola yang bermakna. Underfitting juga berkaitan dengan sederhananya asumsi model terhadap data. Model yang terlalu sederhana cenderung mengasumsikan bahwa hubungan dalam data bersifat linier atau mudah dipisahkan, padahal kenyataannya data sering kali memiliki hubungan non-linear dan interaksi antar fitur yang kompleks. Pohon keputusan yang terlalu dangkal tidak mampu merepresentasikan kompleksitas tersebut. Dampak utama underfitting adalah ketidakmampuan model dalam memberikan prediksi yang akurat. Karena model gagal memahami pola dasar dalam data, kesalahan prediksi akan tinggi dan relatif konsisten baik pada data latih maupun data uji. Dalam aplikasi nyata, kondisi ini menyebabkan model menjadi tidak berguna sebagai alat bantu pengambilan keputusan. Underfitting juga berdampak pada rendahnya kepercayaan terhadap model. Ketika model secara konsisten menghasilkan prediksi yang keliru, pengguna akan kehilangan kepercayaan terhadap hasil analisis. Hal ini menjadi masalah serius terutama dalam domain kritis seperti kesehatan, keuangan, dan kebijakan publik, di mana keputusan berbasis model harus dapat diandalkan.

Dalam kerangka bias variance trade off, underfitting dikaitkan dengan bias yang tinggi dan variansi yang rendah. Model dengan bias tinggi terlalu menyederhanakan permasalahan sehingga tidak mampu menyesuaikan diri dengan data. Meskipun model relatif stabil terhadap

perubahan data, stabilitas tersebut dicapai dengan mengorbankan akurasi dan representasi pola yang sebenarnya. Untuk mengatasi underfitting, salah satu pendekatan utama adalah meningkatkan kompleksitas model secara terkontrol. Pada pohon keputusan, hal ini dapat dilakukan dengan menambah kedalaman pohon, mengurangi batas minimum jumlah sampel pada node, atau memperbolehkan proses splitting berjalan lebih jauh. Dengan demikian, model memiliki kapasitas yang cukup untuk mempelajari pola data. Selain itu, penambahan atau perbaikan fitur juga menjadi strategi penting dalam mengatasi underfitting. Fitur yang lebih representatif dan informatif akan membantu model menangkap hubungan yang lebih kompleks. Proses feature engineering sering kali menjadi kunci untuk meningkatkan performa model yang sebelumnya mengalami underfitting. Penggunaan data yang lebih banyak dan beragam juga dapat membantu mengurangi underfitting. Dataset yang terbatas atau tidak representatif dapat membuat model kesulitan mengenali pola yang sesungguhnya. Dengan data yang lebih kaya, model memiliki kesempatan lebih besar untuk belajar dan menyesuaikan diri dengan karakteristik permasalahan.

Berikut adalah cara untuk mengatasi Underfitting [14]:

1. Meningkatkan Kompleksitas Model : Underfitting sering terjadi karena model terlalu sederhana. Cara paling langsung untuk mengatasinya adalah meningkatkan kompleksitas model. Pada pohon keputusan, hal ini dapat dilakukan dengan memperbesar nilai max_depth sehingga pohon dapat tumbuh lebih dalam dan menangkap pola yang lebih kompleks. Dengan kompleksitas yang memadai, model memiliki kapasitas untuk mempelajari hubungan non-linear dan interaksi antar fitur.
2. Mengurangi Pembatasan Struktur Pohon : Pembatasan seperti min_samples_split atau min_samples_leaf yang terlalu besar

dapat menghentikan proses splitting terlalu dini. Untuk mengatasi underfitting, nilai-nilai ini perlu dikurangi secara bertahap agar pohon memiliki fleksibilitas lebih besar dalam membagi data. Dengan demikian, model dapat membentuk aturan keputusan yang lebih spesifik dan representatif.

3. Menambahkan Fitur yang Lebih Informatif : Underfitting juga dapat disebabkan oleh fitur yang kurang relevan atau tidak cukup informatif. Salah satu solusi efektif adalah menambahkan fitur baru melalui proses feature engineering, misalnya dengan mengombinasikan beberapa atribut atau mengekstraksi informasi tambahan dari data mentah. Fitur yang lebih kaya akan membantu model menangkap pola yang sebelumnya tidak terlihat.
4. Memperbaiki Kualitas dan Representativitas Data : Data yang terbatas atau tidak representatif dapat membuat model gagal mempelajari pola yang sebenarnya. Untuk mengatasi hal ini, perlu dilakukan penambahan jumlah data atau perbaikan distribusi data agar lebih mencerminkan kondisi nyata. Data yang lebih beragam memungkinkan model belajar dari berbagai variasi pola yang relevan.
5. Mengurangi Regularisasi yang Terlalu Kuat : Regularisasi bertujuan mencegah overfitting, tetapi jika diterapkan terlalu ketat justru menyebabkan underfitting. Oleh karena itu, pengaturan parameter regularisasi perlu diseimbangkan. Dalam konteks Decision Tree, ini berarti tidak terlalu membatasi kedalaman pohon atau ukuran minimum node, sehingga model tetap memiliki ruang untuk belajar.

6. Menggunakan Algoritma yang Lebih Ekspresif : Jika Decision Tree sederhana masih mengalami underfitting, solusi lain adalah menggunakan algoritma yang lebih ekspresif, seperti Random Forest atau Gradient Boosting. Metode ini mampu menangkap pola yang lebih kompleks dengan mengombinasikan banyak pohon keputusan, sehingga kapasitas model meningkat tanpa harus mengandalkan satu pohon yang terlalu sederhana.
7. Melakukan Evaluasi Bertahap dengan Validasi : Untuk memastikan bahwa peningkatan kompleksitas benar-benar mengatasi underfitting dan tidak berujung pada overfitting, perlu dilakukan evaluasi bertahap menggunakan data validasi atau cross validation. Pendekatan ini membantu menemukan titik keseimbangan yang optimal antara kesederhanaan dan kompleksitas model.

Eksperimen Underfitting pada Decision Tree (Sebelum vs Sesudah)

Studi Kasus:

Dataset: Breast Cancer Wisconsin

Masalah: Klasifikasi (jinak vs ganas)

Tujuan: Menunjukkan dampak pohon terlalu dangkal dan cara mengatasi underfitting

1. Import Library

```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, cross_val_score
```

```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score
```

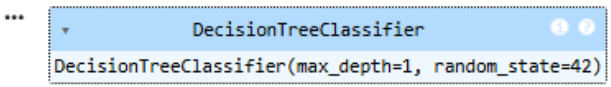
2. Memuat Dataset dan Split Data

```
data = load_breast_cancer()
X, y = data.data, data.target

X_train, X_test, y_train, y_test = train_test_split(
   X, y, test_size=0.3, random_state=42, stratify=y
)
```

SEBELUM: MODEL UNDERFITTING

3. Melatih Decision Tree Terlalu Sederhana

```
dt_underfit = DecisionTreeClassifier(
   max_depth=1,        # terlalu dangkal
   random_state=42
)
dt_underfit.fit(X_train, y_train)
```



4. Evaluasi Model Underfitting

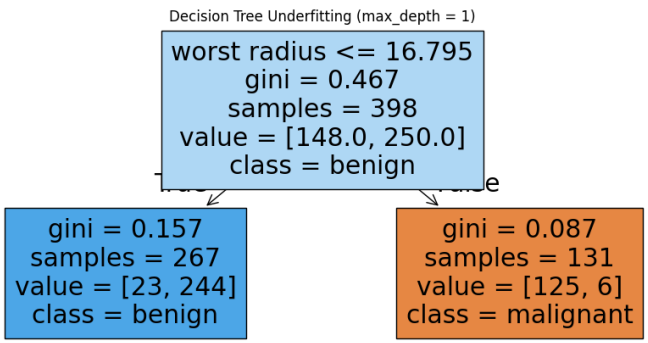
```
train_acc_under = accuracy_score(y_train,
dt_underfit.predict(X_train))
test_acc_under = accuracy_score(y_test,
dt_underfit.predict(X_test))

print("UNDERFITTING MODEL")
print("Train Accuracy:", train_acc_under)
```

```
print("Test Accuracy :", test_acc_under)
```

```
UNDERFITTING MODEL
Train Accuracy: 0.9271356783919598
Test Accuracy : 0.9122807017543859
```

5. Visualisasi Pohon Dangkal

```
plt.figure(figsize=(12,5))
plot_tree(
    dt_underfit,
    feature_names=data.feature_names,
    class_names=data.target_names,
    filled=True
)
plt.title("Decision Tree Underfitting (max_depth = 1)")
plt.show()
```



SESUDAH: UNDERFITTING DIATASI

6. Melatih Decision Tree dengan Kompleksitas Lebih Tinggi

```
dt_fixed = DecisionTreeClassifier(
    max_depth=5,          # kedalaman ditingkatkan
    min_samples_leaf=5,
    random_state=42
```

)

dt_fixed.fit(X_train, y_train)

```
PERBANDINGAN AKHIR
Underfitting -> Train: 0.927, Test: 0.912
Improved     -> Train: 0.972, Test: 0.936
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, min_samples_leaf=5, random_state=42)
```

7. Evaluasi Model Setelah Perbaikan

```
train_acc_fix = accuracy_score(y_train, dt_fixed.predict(X_train))
test_acc_fix  = accuracy_score(y_test,  dt_fixed.predict(X_test))
print("MODEL SETELAH UNDERFITTING DIATASI")
print("Train Accuracy:", train_acc_fix)
print("Test Accuracy :", test_acc_fix)
```

```
MODEL SETELAH UNDERFITTING DIATASI
Train Accuracy: 0.9723618090452262
Test Accuracy : 0.935672514619883
```

8. Evaluasi dengan Cross-Validation

```
cv_under = cross_val_score(dt_underfit, X, y, cv=5).mean()
cv_fix   = cross_val_score(dt_fixed, X, y, cv=5).mean()
print("CV Accuracy Underfitting Model:", cv_under)
print("CV Accuracy Improved Model   :", cv_fix)
```

```
CV Accuracy Underfitting Model: 0.8998447446048751
CV Accuracy Improved Model   : 0.9314702685918336
```

9. Perbandingan Ringkas

```
print("\nPERBANDINGAN AKHIR")
print(f"Underfitting -> Train: {train_acc_under:.3f}, Test: {test_acc_under:.3f}")
print(f"Improved     -> Train: {train_acc_fix:.3f}, Test: {test_acc_fix:.3f}")
```

10. Interpretasi Akademik

Sebelum: Pohon terlalu dangkal → bias tinggi → gagal belajar

Sesudah: Kompleksitas ditambah → pola non-linear tertangkap

Akurasi meningkat pada training dan testing

BAB 3
Algoritma Decision Tree

3.1 Mekanisme Pembentukan Pohon Secara Top Down

Mekanisme pembentukan pohon keputusan secara top down merupakan pendekatan yang paling umum digunakan dalam algoritma Decision Tree. Pendekatan ini membangun pohon dari atas ke bawah, dimulai dari satu node awal yang disebut root node, kemudian secara bertahap membagi data ke dalam node-node yang lebih spesifik. Proses ini meniru cara manusia mengambil keputusan, yaitu dengan memulai dari pertanyaan paling umum lalu bergerak menuju keputusan yang lebih rinci. Pada tahap awal, seluruh dataset ditempatkan pada root node. Node ini merepresentasikan kondisi awal sebelum dilakukan pemisahan apa pun. Tujuan utama pada tahap ini adalah menentukan atribut terbaik yang dapat digunakan untuk memisahkan data sehingga menghasilkan kelompok-kelompok yang lebih homogen terhadap target. Oleh karena itu, seluruh fitur yang tersedia akan dievaluasi untuk menentukan kualitas pemisahan masing-masing. Pemilihan atribut pada root node dilakukan menggunakan kriteria tertentu, seperti Entropy dan Information Gain, Gini Impurity, atau Variance Reduction, tergantung pada jenis permasalahan yang dihadapi. Kriteria ini berfungsi sebagai ukuran objektif untuk menilai seberapa baik suatu atribut mampu memisahkan data. Atribut dengan nilai evaluasi terbaik akan dipilih sebagai dasar pemisahan pertama. Setelah atribut terbaik dipilih, dataset akan dibagi menjadi beberapa subset berdasarkan nilai atau rentang nilai atribut tersebut. Proses ini disebut splitting. Setiap subset hasil splitting akan ditempatkan pada cabang yang berbeda, dan masing-masing cabang

akan mengarah ke node baru. Node-node ini merepresentasikan kondisi data yang sudah lebih spesifik dibandingkan node sebelumnya.

Setiap node baru yang terbentuk kemudian diperlakukan sebagai subproblem yang berdiri sendiri. Mekanisme top-down memastikan bahwa proses pemilihan atribut dan splitting yang sama diterapkan kembali pada setiap node tersebut. Dengan kata lain, proses pembentukan pohon dilakukan secara rekursif, di mana setiap node dianalisis dengan prinsip yang sama seperti pada root node. Pada setiap node internal, algoritma kembali mengevaluasi seluruh atribut yang masih tersedia untuk menentukan atribut terbaik berikutnya. Atribut yang telah digunakan pada tingkat sebelumnya biasanya tidak digunakan kembali pada cabang yang sama, khususnya pada Decision Tree untuk data kategorikal. Hal ini bertujuan menghindari pemisahan yang berulang dan tidak informatif. Proses top-down ini terus berlanjut hingga salah satu kondisi penghentian (stopping criteria) terpenuhi. Kondisi penghentian dapat berupa seluruh data dalam node berasal dari satu kelas yang sama, jumlah data dalam node terlalu sedikit, atau tidak ada lagi atribut yang dapat digunakan untuk pemisahan. Ketika kondisi ini tercapai, node tersebut ditetapkan sebagai leaf node. Leaf node merupakan titik akhir dari mekanisme top down. Node ini tidak lagi melakukan pemisahan, melainkan menyimpan hasil akhir berupa kelas mayoritas pada masalah klasifikasi atau nilai rata-rata pada masalah regresi. Dengan demikian, leaf node merepresentasikan keputusan final yang dihasilkan oleh pohon keputusan.

Salah satu karakteristik penting dari mekanisme top down adalah sifatnya yang greedy. Artinya, pada setiap node algoritma hanya mempertimbangkan keputusan terbaik secara lokal tanpa menjamin bahwa keputusan tersebut optimal secara global [4]. Meskipun demikian, pendekatan greedy terbukti efisien dan efektif dalam banyak kasus

praktis. Keunggulan mekanisme top-down terletak pada kesederhanaan dan efisiensi proses pembentukan pohon. Dengan membagi data secara bertahap dari yang paling umum ke yang paling spesifik, algoritma dapat dengan cepat membangun struktur pohon tanpa memerlukan pencarian solusi yang kompleks. Hal ini menjadikan Decision Tree mudah diimplementasikan dan dipahami. Namun, sifat greedy pada mekanisme top-down juga memiliki konsekuensi. Keputusan pemisahan yang kurang optimal pada tahap awal dapat berdampak besar pada struktur pohon secara keseluruhan. Kesalahan pada root node akan memengaruhi seluruh cabang di bawahnya, sehingga berpotensi menurunkan kualitas model.

Mekanisme top-down juga sangat sensitif terhadap noise dalam data. Jika data latih mengandung banyak noise atau outlier, algoritma dapat memilih atribut yang tampak optimal secara lokal tetapi sebenarnya hanya menyesuaikan diri dengan anomali data. Hal ini dapat menyebabkan pohon tumbuh terlalu kompleks dan mengalami overfitting. Untuk mengatasi keterbatasan tersebut, mekanisme top-down sering dikombinasikan dengan teknik pembatasan kompleksitas, seperti pre pruning dan post pruning. Pre-pruning menghentikan proses top-down lebih awal, sedangkan post-pruning menyederhanakan pohon setelah terbentuk. Kedua teknik ini bertujuan menjaga keseimbangan antara akurasi dan kemampuan generalisasi. Dalam konteks pembelajaran mesin modern, mekanisme top-down tetap menjadi fondasi bagi berbagai algoritma berbasis pohon. Algoritma seperti ID3, C4.5, dan CART semuanya menggunakan pendekatan top-down dengan variasi pada kriteria pemilihan atribut dan strategi penghentian. Mekanisme top-down juga digunakan dalam metode ensemble seperti Random Forest. Meskipun Random Forest membangun banyak pohon, setiap pohon individual tetap dibentuk menggunakan prinsip top down. Perbedaannya

terletak pada penggunaan subset data dan fitur yang dipilih secara acak untuk meningkatkan keragaman pohon.

Dari sudut pandang interpretabilitas, mekanisme top-down menghasilkan struktur pohon yang mudah ditelusuri. Setiap jalur dari root node ke leaf node merepresentasikan urutan keputusan yang logis dan hierarkis. Hal ini memungkinkan pengguna memahami bagaimana suatu prediksi dihasilkan langkah demi langkah. Selain itu, mekanisme top down memudahkan penerjemahan pohon keputusan ke dalam bentuk aturan if then. Setiap aturan mencerminkan hasil dari proses pemisahan bertahap yang dimulai dari kondisi paling umum hingga keputusan akhir. Inilah yang menjadikan Decision Tree sangat populer dalam sistem pendukung keputusan. Secara keseluruhan, mekanisme pembentukan pohon secara top-down merupakan pendekatan inti yang menjelaskan bagaimana Decision Tree belajar dari data. Dengan memulai dari kondisi global dan secara bertahap mempersempit ruang keputusan, mekanisme ini memungkinkan pembentukan model yang sistematis, interpretatif, dan efisien. Dengan memahami mekanisme top down secara mendalam, peneliti dan praktisi dapat mengoptimalkan penggunaan Decision Tree serta mengendalikan risiko seperti overfitting dan underfitting. Pemahaman ini juga menjadi dasar penting untuk mempelajari metode ensemble berbasis pohon yang lebih kompleks dalam pembelajaran mesin.

3.2 Algoritma ID3, C4.5, dan CART

Algoritma ID3, C4.5, dan CART merupakan tiga algoritma klasik yang memiliki peran sangat penting dalam perkembangan metode Decision Tree dalam pembelajaran mesin. Ketiganya menjadi fondasi konseptual bagi banyak algoritma berbasis pohon yang digunakan hingga saat ini. Keberadaan ketiga algoritma ini tidak hanya berkontribusi pada aspek praktis pemodelan data, tetapi juga membentuk kerangka teoretis

tentang bagaimana proses pengambilan keputusan dapat direpresentasikan secara hierarkis dan sistematis. Secara umum, ID3, C4.5, dan CART menggunakan pendekatan top down greedy search dalam membangun struktur pohon keputusan . Pendekatan ini dimulai dari sebuah node awal yang merepresentasikan seluruh dataset, kemudian secara bertahap memecah data ke dalam node-node yang lebih spesifik. Pada setiap tahapan, algoritma memilih atribut yang dianggap paling baik dalam memisahkan data berdasarkan kriteria evaluasi tertentu, tanpa mempertimbangkan kembali keputusan sebelumnya. Mekanisme greedy yang digunakan oleh ketiga algoritma tersebut berarti bahwa pemilihan atribut dilakukan berdasarkan optimalitas lokal pada setiap node. Setiap keputusan pemisahan bertujuan untuk menghasilkan pembagian data yang paling homogen pada level tersebut. Meskipun pendekatan ini tidak menjamin solusi global yang optimal, strategi greedy terbukti efisien dan efektif dalam membangun pohon keputusan dengan kompleksitas komputasi yang relatif rendah.

Meskipun ketiganya memiliki tujuan yang sama, yaitu membangun model prediktif yang akurat dan interpretatif, terdapat perbedaan mendasar dalam cara masing-masing algoritma mengevaluasi atribut. ID3 menggunakan Information Gain berbasis entropy, C4.5 mengembangkan pendekatan ini dengan Gain Ratio untuk mengurangi bias, sedangkan CART menggunakan Gini Impurity atau Variance Reduction tergantung pada jenis permasalahan. Perbedaan ini memengaruhi struktur dan karakteristik pohon yang dihasilkan. Selain perbedaan kriteria pemilihan atribut, ketiga algoritma juga berbeda dalam hal dukungan terhadap jenis data. ID3 terutama dirancang untuk data kategorikal, sementara C4.5 dan CART mampu menangani data numerik melalui penentuan nilai ambang tertentu. Fleksibilitas ini membuat C4.5 dan CART lebih sesuai untuk digunakan pada data nyata yang bersifat heterogen dan kompleks.

Perbedaan lainnya terletak pada mekanisme penghentian dan pemangkasan pohon. ID3 tidak menyediakan mekanisme pruning sehingga rentan terhadap overfitting, sedangkan C4.5 dan CART menerapkan teknik pemangkasan untuk mengendalikan kompleksitas pohon. Dengan demikian, meskipun ketiga algoritma memiliki kerangka dasar yang sama, variasi dalam kriteria pemilihan atribut, dukungan data, dan mekanisme pemangkasan menjadikan masing-masing algoritma memiliki karakteristik dan kegunaan yang berbeda dalam penerapan Decision Tree.

1. Algoritma ID3 (Iterative Dichotomiser 3)

Algoritma ID3 (Iterative Dichotomiser 3) merupakan salah satu algoritma paling awal dan berpengaruh dalam sejarah pengembangan metode Decision Tree untuk pembelajaran mesin. Algoritma ini diperkenalkan oleh J. Ross Quinlan pada pertengahan tahun 1980 an dan menjadi tonggak penting dalam penerapan teori informasi pada proses pengambilan keputusan berbasis data. Kehadiran ID3 membuka jalan bagi pendekatan sistematis dalam membangun model klasifikasi yang dapat dijelaskan secara logis. Sebagai algoritma generasi awal, ID3 berperan sebagai fondasi konseptual bagi berbagai pengembangan algoritma pohon keputusan selanjutnya, seperti C4.5 dan C5.0. Meskipun sederhana, ID3 memperkenalkan kerangka berpikir yang kuat mengenai bagaimana data dapat dipecah secara bertahap untuk menghasilkan keputusan yang semakin spesifik. Oleh karena itu, ID3 sering dijadikan rujukan utama dalam pembelajaran dasar Decision Tree. ID3 dirancang khusus untuk menyelesaikan masalah klasifikasi, di mana variabel target bersifat kategorikal. Algoritma ini tidak ditujukan untuk menangani nilai target numerik atau masalah regresi [15]. Fokus ini membuat ID3 sangat sesuai untuk dataset yang kelas targetnya jelas dan terdefinisi dengan baik,

seperti klasifikasi keputusan, diagnosis sederhana, atau sistem rekomendasi berbasis kategori.

Salah satu kontribusi penting ID3 adalah pengenalan pemilihan atribut berbasis teori informasi. Dengan mengadopsi konsep entropy dan information gain, ID3 memberikan dasar matematis yang objektif dalam menentukan atribut terbaik untuk pemisahan data. Pendekatan ini menggantikan pemilihan atribut secara subjektif atau heuristik, sehingga proses pembentukan pohon menjadi lebih terukur dan dapat dipertanggungjawabkan. Prinsip dasar algoritma ID3 adalah membangun pohon keputusan secara top down dengan pendekatan greedy search. Proses ini dimulai dari satu node akar yang memuat seluruh dataset. Pada tahap ini, algoritma mengevaluasi seluruh atribut yang tersedia untuk menentukan atribut mana yang paling efektif dalam memisahkan data ke dalam kelompok yang lebih homogen. Pemilihan atribut pada setiap node dilakukan secara lokal, yaitu berdasarkan kriteria terbaik pada node tersebut tanpa meninjau kembali keputusan sebelumnya. Sifat greedy ini membuat proses pembentukan pohon berlangsung secara efisien dan cepat, karena algoritma tidak melakukan pencarian ulang atau optimasi global. Meskipun demikian, pendekatan ini tetap mampu menghasilkan struktur pohon yang cukup baik dalam banyak kasus. Kriteria utama yang digunakan ID3 dalam memilih atribut adalah Information Gain, yang dihitung berdasarkan nilai Entropy. Entropy berfungsi untuk mengukur tingkat ketidakpastian atau ketidakhomogenan data dalam suatu node, sedangkan Information Gain mengukur seberapa besar pengurangan ketidakpastian yang dihasilkan ketika data dipisahkan menggunakan atribut tertentu. Dengan demikian, ID3 selalu berupaya meminimalkan ketidakpastian pada setiap tahap pemisahan.

Atribut dengan nilai Information Gain tertinggi dianggap sebagai atribut paling informatif dan dipilih sebagai dasar pemisahan pada node

tersebut. Setelah atribut terbaik dipilih, dataset dibagi ke dalam beberapa subset berdasarkan nilai-nilai atribut tersebut. Setiap subset ini kemudian menjadi node anak yang akan diproses kembali menggunakan prinsip yang sama seperti pada node sebelumnya. Proses pemisahan data dalam ID3 berlangsung secara rekursif hingga tercapai kondisi tertentu, yaitu ketika seluruh data dalam suatu node berasal dari kelas yang sama atau ketika tidak ada lagi atribut yang dapat digunakan untuk pemisahan. Pada kondisi tersebut, node akan ditetapkan sebagai node daun, yang menyimpan keputusan akhir berupa kelas target. Dengan mekanisme ini, ID3 membangun pohon keputusan yang lengkap dan terstruktur berdasarkan prinsip teori informasi tanpa mengubah makna dasar dari pendekatan yang digunakan.

Adapun mekanisme kerja Algoritma ID3 (Iterative Dichotomiser 3) adalah sebagai berikut:

a. Inisialisasi Dataset pada Root Node

   Mekanisme kerja algoritma ID3 dimulai dengan menempatkan seluruh dataset pada satu node awal yang disebut root node. Node ini merepresentasikan kondisi awal sebelum dilakukan pemisahan apa pun. Pada tahap ini, semua data dianggap sebagai satu kesatuan, dan tujuan utama algoritma adalah mencari atribut terbaik yang mampu membagi data tersebut ke dalam kelompok yang lebih homogen terhadap kelas target.

b. Menghitung Entropy Dataset

   Setelah dataset berada pada root node, algoritma ID3 menghitung entropy dari dataset tersebut. Entropy digunakan untuk mengukur tingkat ketidakpastian atau ketidakhomogenan kelas dalam data. Nilai entropy ini menjadi acuan awal untuk menilai seberapa baik suatu atribut dapat

mengurangi ketidakpastian jika digunakan sebagai dasar pemisahan.

c. Menghitung Information Gain Setiap Atribut

Pada tahap berikutnya, ID3 mengevaluasi seluruh atribut kandidat dengan menghitung Information Gain masing-masing atribut. Perhitungan ini dilakukan dengan membandingkan entropy awal dataset dengan entropy rata-rata tertimbang dari subset data yang dihasilkan jika atribut tersebut digunakan untuk memisahkan data. Information Gain menunjukkan seberapa besar pengurangan ketidakpastian yang dihasilkan oleh atribut tersebut.

d. Memilih Atribut Terbaik

Setelah nilai Information Gain untuk semua atribut dihitung, algoritma memilih atribut dengan Information Gain tertinggi. Atribut ini dianggap paling informatif karena mampu menghasilkan pemisahan data yang paling signifikan dalam mengurangi ketidakpastian kelas. Atribut terpilih kemudian digunakan sebagai node keputusan pada tingkat tersebut.

e. Melakukan Pemisahan Data (Splitting)

Dataset kemudian dibagi ke dalam beberapa subset berdasarkan nilai-nilai yang dimiliki oleh atribut terpilih. Setiap nilai atribut akan menghasilkan satu cabang, dan setiap cabang akan mengarah ke subset data yang sesuai. Proses ini menghasilkan node-node anak yang merepresentasikan kondisi data yang lebih spesifik dibandingkan node sebelumnya.

f. Pembentukan Node Anak dan Pemrosesan Rekursif

Setiap subset data hasil pemisahan diperlakukan sebagai dataset baru. Algoritma ID3 kemudian diterapkan kembali secara rekursif pada masing-masing node anak dengan

langkah-langkah yang sama, yaitu menghitung entropy, menghitung information gain, dan memilih atribut terbaik berikutnya. Proses ini berlangsung secara top-down hingga kondisi penghentian terpenuhi.

g. Kondisi Penghentian dan Pembentukan Node Daun

Proses rekursif akan berhenti apabila salah satu kondisi penghentian tercapai. Kondisi tersebut antara lain: seluruh data dalam node berasal dari kelas yang sama, tidak ada lagi atribut yang tersedia untuk pemisahan, atau dataset pada node menjadi kosong. Pada kondisi ini, node tersebut ditetapkan sebagai node daun yang menyimpan keputusan akhir berupa kelas target.

h. Pembentukan Aturan Keputusan

Setelah pohon keputusan terbentuk sepenuhnya, setiap jalur dari root node hingga node daun dapat diterjemahkan menjadi sebuah aturan keputusan dalam bentuk if then. Aturan-aturan ini merepresentasikan hasil akhir dari mekanisme kerja ID3 dan menjadi dasar pengambilan keputusan atau prediksi pada data baru.

Kelebihan Algoritma ID3

a. Konsep Sederhana dan Mudah Dipahami

Salah satu kelebihan utama algoritma ID3 adalah kesederhanaan konsepnya. Mekanisme pembentukan pohon yang berbasis entropy dan information gain membuat alur pengambilan keputusan mudah diikuti. Setiap pemisahan data dapat dijelaskan secara logis, sehingga ID3 sangat cocok digunakan sebagai algoritma pengantar untuk memahami prinsip dasar Decision Tree.

b. Interpretabilitas Tinggi

ID3 menghasilkan struktur pohon keputusan yang mudah diinterpretasikan. Setiap jalur dari root node hingga node daun dapat diterjemahkan menjadi aturan if then yang jelas dan transparan. Hal ini menjadikan ID3 sesuai untuk aplikasi yang membutuhkan penjelasan keputusan, seperti sistem pendukung keputusan dan pembelajaran akademik.

c. Dasar Teori Informasi yang Kuat

Algoritma ID3 didasarkan pada teori informasi, khususnya konsep entropy dan information gain. Dasar matematis ini memberikan justifikasi ilmiah yang kuat dalam pemilihan atribut, sehingga proses pembentukan pohon tidak bersifat subjektif, melainkan berbasis pengukuran kuantitatif.

d. Efisien untuk Dataset Kecil hingga Menengah

Untuk dataset dengan ukuran kecil hingga menengah dan jumlah atribut yang terbatas, ID3 dapat bekerja secara efisien. Proses pemilihan atribut dilakukan secara greedy, sehingga waktu komputasi relatif singkat dan mudah diimplementasikan tanpa kebutuhan sumber daya komputasi yang besar.

Keterbatasan Algoritma ID3

a. Bias terhadap Atribut dengan Banyak Nilai

Keterbatasan utama ID3 adalah kecenderungannya memilih atribut dengan banyak nilai unik. Atribut semacam ini sering menghasilkan information gain yang tinggi secara matematis, meskipun tidak selalu relevan secara konseptual. Bias ini dapat menyebabkan pemilihan atribut yang kurang bermakna dan menurunkan kualitas model.

b. Tidak Mendukung Data Numerik Secara Langsung

ID3 dirancang untuk data kategorikal dan tidak dapat menangani data numerik secara langsung. Data numerik harus didiskretisasi terlebih dahulu, yang berpotensi menyebabkan hilangnya informasi. Keterbatasan ini membuat ID3 kurang fleksibel untuk digunakan pada dataset nyata yang umumnya mengandung data numerik.

c. Tidak Memiliki Mekanisme Pruning

ID3 tidak menyediakan mekanisme pemangkasan pohon (pruning). Akibatnya, pohon yang dihasilkan dapat tumbuh sangat kompleks dan menyesuaikan diri secara berlebihan terhadap data latih. Tanpa pruning, ID3 sangat rentan mengalami overfitting, terutama pada data yang besar dan mengandung noise.

d. Sensitif terhadap Noise dan Outlier

Algoritma ID3 cukup sensitif terhadap noise dan outlier dalam data. Karena setiap pemisahan dilakukan untuk memaksimalkan information gain, noise kecil dalam data dapat memengaruhi struktur pohon secara signifikan. Hal ini dapat menghasilkan aturan keputusan yang tidak stabil dan sulit digeneralisasi.

e. Pendekatan Greedy Tidak Menjamin Solusi Global Optimal

ID3 menggunakan pendekatan greedy dalam pemilihan atribut, yaitu memilih keputusan terbaik secara lokal pada setiap node. Meskipun efisien, pendekatan ini tidak menjamin bahwa struktur pohon yang dihasilkan merupakan solusi global terbaik. Kesalahan pemilihan atribut pada tahap awal dapat berdampak besar pada keseluruhan pohon.

Contoh:

Dataset contoh: Kelayakan Kredit (sederhana)

- Dataset

```
import pandas as pd
import numpy as np

data = {
   'Usia': [25, 40, 35, 28, 50, 45],
   'Penghasilan': [3, 7, 6, 4, 8, 5],
   'Riwayat_Kredit': ['Buruk', 'Baik', 'Baik', 'Buruk', 'Baik', 'Buruk'],
   'Kelayakan': ['Tidak', 'Layak', 'Layak', 'Tidak', 'Layak', 'Tidak']
}

df = pd.DataFrame(data)
df
```

	Usia	Penghasilan	Riwayat_Kredit	Kelayakan
0	25	3	Buruk	Tidak
1	40	7	Baik	Layak
2	35	6	Baik	Layak
3	28	4	Buruk	Tidak
4	50	8	Baik	Layak
5	45	5	Buruk	Tidak

- Implementasi ID3 (Manual – Entropy & Information Gain)

```
from math import log2

def entropy(col):
   values, counts = np.unique(col, return_counts=True)
```

```
    return -sum((c/sum(counts)) * log2(c/sum(counts)) for c in
counts)

def information_gain(df, feature, target):
    total_entropy = entropy(df[target])
    values, counts = np.unique(df[feature], return_counts=True)

    weighted_entropy = sum(
        (counts[i]/sum(counts))                                    *
entropy(df[df[feature]==values[i]][target])
        for i in range(len(values))
    )
    return total_entropy - weighted_entropy

for f in ['Usia', 'Penghasilan', 'Riwayat_Kredit']:
    print(f, information_gain(df, f, 'Kelayakan'))
```

```
Usia 1.0
Penghasilan 1.0
Riwayat_Kredit 1.0
```

2. Algoritma C4.5

Algoritma C4.5 merupakan pengembangan lanjutan dari algoritma ID3 yang juga diperkenalkan oleh J. Ross Quinlan sebagai upaya untuk menyempurnakan metode pembentukan pohon keputusan. C4.5 dirancang dengan mempertimbangkan berbagai keterbatasan yang dimiliki oleh ID3, sehingga mampu menghasilkan model yang lebih fleksibel, stabil, dan siap digunakan pada permasalahan nyata. Kehadiran C4.5 menandai tahap penting dalam evolusi algoritma Decision Tree menuju pendekatan yang lebih matang dan praktis. Salah satu tujuan

utama pengembangan C4.5 adalah mengatasi bias ID3 terhadap atribut dengan banyak nilai unik. Dengan memperbaiki mekanisme pemilihan atribut, C4.5 mampu menghasilkan struktur pohon yang lebih adil dan representatif terhadap data. Perbaikan ini membuat C4.5 lebih andal dalam memilih atribut yang benar-benar informatif, bukan sekadar atribut yang secara matematis menghasilkan pemisahan ekstrem. C4.5 juga memperluas kemampuan ID3 dengan mendukung data numerik selain data kategorikal. Algoritma ini mampu menentukan nilai ambang (threshold) yang optimal untuk memisahkan data numerik ke dalam dua kelompok. Dengan dukungan ini, C4.5 dapat diterapkan pada dataset dunia nyata yang umumnya mengandung kombinasi atribut numerik dan kategorikal. Selain dukungan terhadap berbagai tipe data, C4.5 menyediakan mekanisme pemangkasan pohon (pruning) untuk mengendalikan kompleksitas model. Pemangkasan dilakukan setelah pohon dibangun secara penuh dengan tujuan menghilangkan cabang-cabang yang tidak memberikan kontribusi signifikan terhadap performa model. Mekanisme ini sangat penting untuk meningkatkan kemampuan generalisasi dan mengurangi risiko overfitting. Berkat berbagai penyempurnaan tersebut, C4.5 menjadi salah satu algoritma Decision Tree yang paling populer dan banyak digunakan dalam praktik. Kombinasi antara fleksibilitas data, mekanisme pruning, dan struktur pohon yang interpretatif menjadikan C4.5 cocok untuk berbagai aplikasi klasifikasi. Dengan demikian, C4.5 dapat dipandang sebagai jembatan antara kesederhanaan ID3 dan kebutuhan akan model yang lebih robust dalam pembelajaran mesin.

Adapun mekanisme kerja Algoritma C4.5 adalah sebagai berikut:

a. Inisialisasi Dataset pada Root Node : Mekanisme kerja algoritma C4.5 dimulai dengan menempatkan seluruh dataset pada satu node awal yang disebut root node. Node ini

merepresentasikan kondisi awal sebelum dilakukan pemisahan apa pun. Tujuan utama pada tahap ini adalah menentukan atribut yang paling tepat untuk membagi data sehingga menghasilkan subset yang lebih homogen terhadap kelas target.

b. Penanganan Data dan Missing Value : Sebelum pemilihan atribut dilakukan, C4.5 menangani kemungkinan adanya missing value dalam dataset. Algoritma ini tidak langsung membuang data yang tidak lengkap, melainkan memperhitungkannya secara proporsional dalam perhitungan kriteria pemisahan. Pendekatan ini memungkinkan C4.5 tetap memanfaatkan data secara maksimal tanpa kehilangan informasi penting.

c. Perhitungan Entropy Dataset : C4.5 menghitung entropy dari dataset pada node saat ini untuk mengukur tingkat ketidakpastian atau ketidakhomogenan kelas. Nilai entropy ini menjadi dasar untuk mengevaluasi seberapa baik suatu atribut dapat mengurangi ketidakpastian jika digunakan sebagai dasar pemisahan data. Semakin tinggi entropy, semakin tidak homogen data dalam node tersebut.

d. Perhitungan Information Gain Setiap Atribut : Selanjutnya, C4.5 menghitung Information Gain untuk setiap atribut kandidat. Information Gain menunjukkan seberapa besar penurunan entropy yang dihasilkan oleh atribut tersebut. Perhitungan ini dilakukan dengan membandingkan entropy awal dataset dengan entropy rata-rata tertimbang dari subset data hasil pemisahan berdasarkan atribut tersebut.

e. Normalisasi dengan Gain Rasio : Berbeda dengan ID3, C4.5 tidak langsung menggunakan Information Gain untuk memilih atribut. Algoritma ini melakukan normalisasi menggunakan Gain Ratio, yaitu membagi Information Gain dengan nilai Split Information. Gain Ratio digunakan untuk mengurangi bias terhadap atribut yang memiliki banyak nilai unik, sehingga pemilihan atribut menjadi lebih adil dan representatif.

f. Pemilihan Atribut Terbaik : Atribut dengan nilai Gain Ratio tertinggi dipilih sebagai atribut terbaik pada node tersebut. Atribut ini kemudian digunakan sebagai node keputusan. Pemilihan ini memastikan bahwa atribut yang dipilih tidak hanya mengurangi ketidakpastian, tetapi juga membagi data secara seimbang dan bermakna.

g. Penanganan Atribut Numerik : Jika atribut yang dipilih bersifat numerik, C4.5 menentukan nilai ambang (threshold) optimal untuk melakukan pemisahan data. Dataset dibagi menjadi dua subset berdasarkan kondisi nilai atribut kurang dari atau sama dengan threshold dan lebih besar dari threshold. Proses ini memungkinkan C4.5 menangani data numerik secara langsung tanpa perlu diskretisasi manual.

h. Proses Splitting dan Pembentukan Node Anak : Dataset kemudian dibagi ke dalam subset-subset sesuai dengan atribut dan nilai pemisahan yang dipilih. Setiap subset menjadi node anak yang mewakili kondisi data yang lebih spesifik. Node-node ini kemudian diproses kembali menggunakan mekanisme yang sama secara rekursif.

i. Kondisi Penghentian Pembentukan Pohon : Proses rekursif akan berhenti apabila salah satu kondisi penghentian terpenuhi, seperti seluruh data dalam node berasal dari kelas yang sama, jumlah data terlalu sedikit, atau tidak ada lagi atribut yang dapat digunakan. Pada kondisi ini, node akan ditetapkan sebagai node daun yang menyimpan keputusan akhir berupa kelas target.

j. Pemangkasan Pohon (Post-Pruning) : Setelah pohon keputusan terbentuk sepenuhnya, C4.5 menerapkan post pruning untuk menyederhanakan struktur pohon. Cabang-cabang yang tidak memberikan peningkatan performa signifikan pada data validasi akan dipangkas. Mekanisme ini bertujuan mengurangi kompleksitas pohon dan meningkatkan kemampuan generalisasi model.

Kelebihan Algortima C4.5:

a. Mengurangi Bias Pemilihan Atribut

Salah satu kelebihan utama C4.5 adalah penggunaan Gain Ratio sebagai kriteria pemilihan atribut. Gain Ratio merupakan normalisasi dari Information Gain yang bertujuan mengurangi bias terhadap atribut dengan banyak nilai unik. Dengan mekanisme ini, C4.5 mampu memilih atribut yang benar-benar informatif dan relevan terhadap target, bukan sekadar atribut yang menghasilkan pemisahan ekstrem secara matematis.

b. Mendukung Data Numerik dan Kategorikal

Berbeda dengan ID3 yang hanya mendukung data kategorikal, C4.5 mampu menangani data numerik secara langsung. Algoritma ini secara otomatis menentukan nilai ambang (threshold) optimal untuk memisahkan data numerik. Dukungan

terhadap berbagai tipe data menjadikan C4.5 lebih fleksibel dan siap digunakan pada dataset dunia nyata yang bersifat heterogen.

c. Menangani Missing Value

C4.5 memiliki mekanisme untuk menangani nilai yang hilang (missing value) tanpa harus menghapus data tersebut. Algoritma ini memperhitungkan missing value secara proporsional dalam proses perhitungan kriteria pemisahan. Pendekatan ini memungkinkan pemanfaatan data secara lebih optimal dan mengurangi kehilangan informasi penting.

d. Memiliki Mekanisme Pruning

C4.5 menerapkan post pruning untuk menyederhanakan struktur pohon setelah pohon terbentuk secara penuh. Pemangkasan ini bertujuan menghilangkan cabang-cabang yang tidak memberikan kontribusi signifikan terhadap performa model. Dengan adanya pruning, C4.5 memiliki kemampuan generalisasi yang lebih baik dan lebih tahan terhadap overfitting.

e. Interpretabilitas yang Baik

Meskipun lebih kompleks dibandingkan ID3, pohon keputusan yang dihasilkan C4.5 tetap mudah diinterpretasikan. Struktur pohon dapat diterjemahkan ke dalam aturan if then yang jelas dan logis. Hal ini menjadikan C4.5 cocok untuk aplikasi yang membutuhkan transparansi dalam pengambilan keputusan.

Keterbatasan Algoritma C4.5:

a. Kompleksitas Komputasi Lebih Tinggi

Salah satu keterbatasan C4.5 adalah kompleksitas komputasi yang lebih tinggi dibandingkan ID3. Perhitungan Gain Ratio, penanganan data numerik, dan proses pruning membutuhkan

waktu dan sumber daya komputasi yang lebih besar, terutama pada dataset berukuran besar dengan banyak atribut.

b. Sensitif terhadap Noise

   Meskipun lebih baik dari ID3, C4.5 masih relatif sensitif terhadap noise dalam data. Data yang mengandung banyak kesalahan atau outlier dapat memengaruhi proses pemilihan atribut dan menghasilkan struktur pohon yang kurang optimal, meskipun mekanisme pruning telah diterapkan.

c. Struktur Pohon Bisa Tetap Kompleks

   Pada dataset yang sangat kompleks, pohon keputusan yang dihasilkan C4.5 masih berpotensi menjadi besar dan sulit diinterpretasikan. Meskipun pruning membantu mengendalikan ukuran pohon, interpretabilitas dapat menurun jika jumlah node dan cabang tetap terlalu banyak.

d. Tidak Menjamin Solusi Global Optimal

   Seperti algoritma Decision Tree lainnya, C4.5 menggunakan pendekatan **greedy** dalam pemilihan atribut. Pendekatan ini tidak menjamin bahwa struktur pohon yang dihasilkan merupakan solusi global terbaik. Keputusan pemisahan yang kurang optimal pada tahap awal dapat berdampak pada kualitas pohon secara keseluruhan.

e. Implementasi Tidak Tersedia Langsung di Banyak Pustaka Modern

   Dalam praktik modern, C4.5 tidak tersedia secara langsung dalam beberapa pustaka machine learning populer seperti scikit learn. Hal ini membuat pengguna harus mengimplementasikannya secara manual atau menggunakan pustaka khusus, sehingga kurang praktis dibandingkan algoritma berbasis CART yang lebih umum tersedia.

Contoh:

Implementasi C4.5 (Gain Ratio – Manual)

```
def split_info(df, feature):
    values, counts = np.unique(df[feature], return_counts=True)
    return -sum((c/sum(counts)) * log2(c/sum(counts)) for c in counts)

def gain_ratio(df, feature, target):
    ig = information_gain(df, feature, target)
    si = split_info(df, feature)
    return ig / si if si != 0 else 0

for f in ['Usia', 'Penghasilan', 'Riwayat_Kredit']:
    print(f, gain_ratio(df, f, 'Kelayakan'))
```

3. Algoritma CART (Classification and Regression Trees)

```
Usia 0.38685280723454163
Penghasilan 0.38685280723454163
Riwayat_Kredit 1.0
```

Algoritma CART (Classification and Regression Trees) dikembangkan oleh Breiman et al. sebagai pendekatan Decision Tree yang dirancang untuk bersifat lebih umum dan fleksibel. CART tidak hanya ditujukan untuk satu jenis permasalahan tertentu, tetapi dirancang agar dapat diterapkan pada berbagai konteks analisis data. Pengembangan CART bertujuan untuk menyediakan kerangka pohon keputusan yang konsisten dan mudah diadaptasi pada berbagai jenis dataset. Berbeda dengan algoritma ID3 dan C4.5 yang secara khusus difokuskan pada masalah klasifikasi, CART mampu menangani dua jenis permasalahan sekaligus, yaitu klasifikasi dan regresi. Pada masalah klasifikasi, CART memprediksi kelas diskret, sedangkan pada masalah regresi, CART

memprediksi nilai numerik kontinu. Kemampuan ganda ini menjadikan CART lebih serbaguna dan banyak digunakan dalam praktik pembelajaran mesin modern. Salah satu karakteristik utama CART adalah struktur pohonnya yang selalu berbentuk biner, di mana setiap node hanya memiliki dua cabang. Pemisahan data dilakukan berdasarkan kondisi biner, seperti benar–salah atau kurang dari lebih besar dari suatu nilai ambang. Struktur biner ini membuat proses pemisahan lebih sederhana, konsisten, dan efisien, serta memudahkan implementasi CART dalam berbagai algoritma lanjutan seperti Random Forest dan Gradient Boosting.

Adapun mekanisme kerja Algoritma CART (Classification and Regression Trees) adalah sebagai berikut:

a. Inisialisasi Dataset pada Root Node

   Mekanisme kerja algoritma CART dimulai dengan menempatkan seluruh dataset pada satu node awal yang disebut root node. Node ini memuat semua data latih sebelum dilakukan pemisahan apa pun. Tujuan utama pada tahap ini adalah menentukan pemisahan terbaik yang mampu membagi data ke dalam dua kelompok yang lebih homogen terhadap variabel target.

b. Penentuan Jenis Permasalahan

   Sebelum proses pemisahan dilakukan, CART menentukan jenis permasalahan yang sedang dihadapi, yaitu klasifikasi atau regresi. Penentuan ini penting karena CART menggunakan kriteria evaluasi yang berbeda untuk masing-masing jenis permasalahan. Pada klasifikasi digunakan Gini Impurity, sedangkan pada regresi digunakan Variance Reduction atau kesalahan kuadrat.

c. Evaluasi Semua Kandidat Pemisahan

Pada setiap node, CART mengevaluasi seluruh atribut yang tersedia beserta semua kemungkinan pemisahan biner yang dapat dilakukan. Untuk atribut kategorikal, CART menguji berbagai kombinasi pembagian kategori ke dalam dua kelompok. Untuk atribut numerik, CART menguji berbagai nilai ambang (threshold) untuk membagi data menjadi dua subset berdasarkan kondisi kurang dari atau sama dengan dan lebih besar dari threshold tersebut.

d. Perhitungan Kriteria Pemisahan

Setiap kandidat pemisahan dievaluasi menggunakan kriteria yang sesuai. Pada klasifikasi, CART menghitung Gini Impurity untuk mengukur tingkat ketidakmurnian kelas pada node. Pada regresi, CART menghitung Variance Reduction atau penurunan kesalahan kuadrat. Tujuan utama dari perhitungan ini adalah mengukur seberapa baik suatu pemisahan dapat meningkatkan homogenitas data pada node anak.

e. Pemilihan Split Terbaik

Dari seluruh kandidat pemisahan yang diuji, CART memilih pemisahan yang menghasilkan penurunan impuritas atau variansi terbesar. Pemisahan ini dianggap paling optimal secara lokal dan digunakan sebagai dasar pembentukan node keputusan. Keputusan ini bersifat greedy, karena hanya mempertimbangkan hasil terbaik pada node saat ini tanpa menjamin optimalitas global.

f. Pembentukan Cabang Biner

Setelah split terbaik dipilih, CART membagi dataset menjadi dua subset sesuai dengan kondisi pemisahan yang ditentukan.

Setiap subset ditempatkan pada satu cabang node, sehingga setiap node internal dalam CART selalu memiliki dua cabang. Struktur biner ini merupakan ciri khas utama CART dan membedakannya dari algoritma pohon keputusan lainnya.

g. Proses Rekursif pada Node Anak

Setiap subset data hasil pemisahan diperlakukan sebagai node baru dan diproses kembali menggunakan mekanisme yang sama. CART secara rekursif mengevaluasi atribut dan kandidat pemisahan pada setiap node anak. Proses ini berlangsung dari atas ke bawah hingga mencapai kondisi penghentian tertentu.

h. Kondisi Penghentian Pembentukan Pohon

Proses rekursif CART akan berhenti apabila salah satu kondisi penghentian terpenuhi. Kondisi tersebut dapat berupa seluruh data dalam node berasal dari kelas yang sama, jumlah data pada node terlalu sedikit, kedalaman maksimum pohon tercapai, atau penurunan impuritas tidak lagi signifikan. Pada kondisi ini, node tersebut ditetapkan sebagai node daun.

i. Penentuan Nilai pada Node Daun

Pada node daun, CART menyimpan hasil akhir yang digunakan untuk prediksi. Untuk klasifikasi, nilai pada node daun biasanya berupa kelas mayoritas dari data pada node tersebut. Untuk regresi, nilai yang disimpan adalah rata-rata atau nilai representatif lain dari target numerik pada node tersebut.

j. Pemangkasan Pohon (Cost-Complexity Pruning)

Setelah pohon CART dibangun secara penuh, algoritma ini menerapkan cost complexity pruning untuk menyederhanakan struktur pohon. Pemangkasan dilakukan dengan mempertimbangkan trade-off antara kompleksitas pohon dan

kesalahan prediksi. Cabang yang tidak memberikan peningkatan performa signifikan akan dipangkas untuk meningkatkan kemampuan generalisasi.

Kelebihan Algoritma CART (Classification and Regression Trees):

a. Dapat Digunakan untuk Klasifikasi dan Regresi

   Salah satu kelebihan utama CART adalah kemampuannya menangani dua jenis permasalahan, yaitu klasifikasi dan regresi, dalam satu kerangka algoritma yang sama. Pada klasifikasi, CART memprediksi kelas diskret, sedangkan pada regresi CART memprediksi nilai numerik kontinu. Fleksibilitas ini menjadikan CART sangat serbaguna dan banyak digunakan dalam berbagai aplikasi pembelajaran mesin.

b. Struktur Pohon Biner yang Konsisten

   CART selalu menghasilkan pohon biner, di mana setiap node hanya memiliki dua cabang. Struktur ini membuat proses pemisahan data menjadi lebih sederhana, konsisten, dan mudah diimplementasikan. Pohon biner juga memudahkan optimasi dan integrasi CART ke dalam metode ensemble seperti Random Forest dan Gradient Boosting.

c. Mendukung Data Numerik dan Kategorikal

   Algoritma CART mampu menangani data numerik maupun kategorikal secara langsung. Untuk data numerik, CART menentukan nilai ambang optimal, sedangkan untuk data kategorikal CART mengelompokkan kategori ke dalam dua subset. Dukungan terhadap berbagai tipe data membuat CART cocok untuk dataset dunia nyata yang kompleks dan heterogen.

d. Efisiensi Komputasi yang Baik

   Penggunaan kriteria pemisahan seperti Gini Impurity dan Variance Reduction yang relatif sederhana secara matematis membuat CART efisien secara komputasi. Dibandingkan dengan metode berbasis entropy, perhitungan CART lebih cepat dan lebih ringan, terutama pada dataset berukuran besar.

e. Memiliki Mekanisme Pruning yang Sistematis

   CART menerapkan cost complexity pruning untuk mengendalikan kompleksitas pohon. Mekanisme ini memungkinkan pengguna memilih struktur pohon yang optimal dengan mempertimbangkan trade off antara kompleksitas dan kesalahan prediksi. Pruning yang sistematis membantu CART meningkatkan kemampuan generalisasi dan mengurangi overfitting.

f. Menjadi Dasar Metode Ensemble Modern

   CART merupakan fondasi utama bagi berbagai metode ensemble populer seperti Random Forest, Bagging, dan Gradient Boosting. Karakteristik pohon biner yang sederhana dan stabil menjadikan CART sangat cocok digunakan sebagai *base learner* dalam metode ensemble.

Keterbatasan Algoritma CART (Classification and Regression Trees):

a. Rentan terhadap Overfitting

   Seperti algoritma Decision Tree lainnya, CART cenderung menghasilkan pohon yang sangat kompleks jika tidak dibatasi. Tanpa pengaturan parameter dan pruning yang tepat, CART dapat menyesuaikan diri secara berlebihan terhadap data latih, sehingga performa pada data uji menurun.

b. Ketidakstabilan terhadap Perubahan Data

CART bersifat tidak stabil, artinya perubahan kecil pada data latih dapat menghasilkan struktur pohon yang sangat berbeda. Ketidakstabilan ini dapat menurunkan konsistensi model dan menjadi masalah dalam aplikasi yang membutuhkan hasil yang stabil dan dapat direproduksi.

c. Interpretabilitas Menurun pada Pohon Besar

Meskipun secara konsep mudah dipahami, interpretabilitas CART menurun ketika pohon tumbuh terlalu besar dan dalam. Pohon dengan banyak node dan cabang menghasilkan aturan if then yang panjang dan kompleks, sehingga sulit dipahami oleh pengguna non teknis.

d. Bias terhadap Split Tertentu pada Data Tidak Seimbang

CART dapat menunjukkan bias dalam pemilihan split ketika dataset memiliki distribusi kelas yang tidak seimbang. Tanpa penanganan khusus, seperti penyesuaian bobot kelas, CART dapat lebih memihak kelas mayoritas dan menghasilkan performa yang kurang baik pada kelas minoritas.

e. Tidak Menjamin Solusi Global Optimal

CART menggunakan pendekatan greedy dalam pemilihan split, yaitu memilih pemisahan terbaik secara lokal pada setiap node. Pendekatan ini tidak menjamin bahwa struktur pohon yang dihasilkan merupakan solusi global terbaik, sehingga kualitas model sangat bergantung pada keputusan pemisahan di tahap awal.

Contoh:

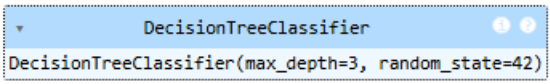
- **Implementasi CART (Scikit-learn – Praktis & Standar)**

  Encoding Data

  ```
  from sklearn.preprocessing import LabelEncoder

  df_enc = df.copy()
  le = LabelEncoder()

  df_enc['Riwayat_Kredit'] =
  le.fit_transform(df_enc['Riwayat_Kredit'])
  df_enc['Kelayakan'] = le.fit_transform(df_enc['Kelayakan'])

  X = df_enc[['Usia', 'Penghasilan', 'Riwayat_Kredit']]
  y = df_enc['Kelayakan']
  ```

  CART – Klasifikasi (Gini Impurity)

  ```
  from sklearn.tree import DecisionTreeClassifier
  ```

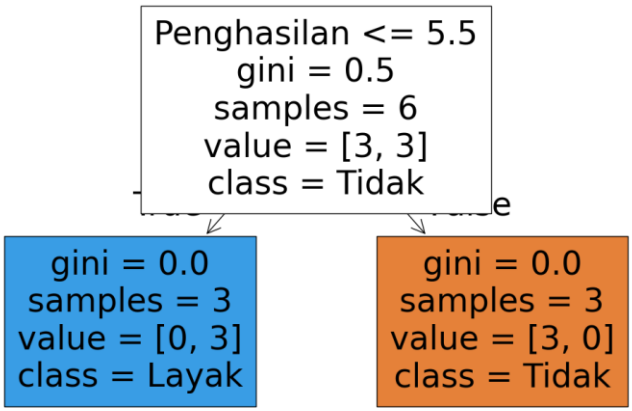
  


  ```
  cart_clf = DecisionTreeClassifier(
     criterion='gini',
     max_depth=3,
     random_state=42
  )
  cart_clf.fit(X, y)
  ```

  Visualisasi CART

  ```
  from sklearn.tree import plot_tree
  ```

```
import matplotlib.pyplot as plt
```




```
plt.figure(figsize=(14,8))
plot_tree(
    cart_clf,
    feature_names=X.columns,
    class_names=['Tidak', 'Layak'],
    filled=True
)
plt.show()
```

3.3 Penanganan Data Numerik dan Kategorikal

Penanganan data numerik dan kategorikal merupakan aspek fundamental dalam penerapan algoritma pembelajaran mesin, khususnya pada metode berbasis pohon keputusan seperti Decision Tree, C4.5, dan CART. Kedua jenis data ini memiliki karakteristik yang berbeda sehingga memerlukan pendekatan pemrosesan yang berbeda pula. Data numerik merepresentasikan nilai kuantitatif yang dapat diukur secara matematis, sedangkan data kategorikal merepresentasikan kelas atau kategori diskret yang tidak memiliki makna numerik langsung. Pemahaman yang tepat mengenai perbedaan ini sangat penting agar model mampu menangkap

pola data secara akurat. Dalam konteks pohon keputusan, data kategorikal umumnya ditangani dengan cara membagi data berdasarkan setiap nilai kategori yang dimiliki oleh atribut tersebut. Pada algoritma seperti ID3 dan C4.5, setiap nilai kategori dapat membentuk satu cabang tersendiri dari sebuah node keputusan. Pendekatan ini memungkinkan model memisahkan data secara eksplisit berdasarkan karakteristik kategori, sehingga mudah diinterpretasikan dan sesuai untuk data dengan jumlah kategori yang terbatas dan bermakna. Namun, penanganan data kategorikal dengan banyak nilai unik dapat menimbulkan masalah, seperti meningkatnya kompleksitas pohon dan risiko overfitting. Oleh karena itu, algoritma yang lebih maju seperti C4.5 dan CART menerapkan strategi tambahan untuk mengatasi kondisi ini. C4.5 menggunakan Gain Ratio untuk mengurangi bias terhadap atribut dengan banyak kategori, sedangkan CART mengelompokkan kategori ke dalam dua subset untuk membentuk pemisahan biner yang lebih sederhana dan stabil.

Berbeda dengan data kategorikal, data numerik tidak dapat langsung dipisahkan berdasarkan nilai uniknya karena jumlah kemungkinan nilai bisa sangat banyak dan kontinu. Oleh karena itu, algoritma pohon keputusan menangani data numerik dengan menentukan nilai ambang (threshold) tertentu yang membagi data ke dalam dua kelompok, misalnya nilai yang lebih kecil atau sama dengan threshold dan nilai yang lebih besar dari threshold tersebut. Pemilihan threshold dilakukan dengan menguji berbagai kemungkinan dan memilih yang menghasilkan pemisahan terbaik berdasarkan kriteria tertentu. Algoritma C4.5 dan CART memiliki kemampuan bawaan untuk menangani data numerik secara langsung. C4.5 menentukan threshold dengan menghitung Gain Ratio untuk setiap kandidat pemisahan, sedangkan CART memilih threshold yang meminimalkan Gini Impurity atau variansi. Pendekatan ini memungkinkan data numerik dimanfaatkan

secara maksimal tanpa perlu diskretisasi manual, sehingga informasi penting dalam data tetap terjaga. Selain itu, penanganan data numerik dan kategorikal juga berkaitan dengan isu missing value dan kualitas data. Beberapa algoritma, seperti C4.5, mampu menangani nilai yang hilang dengan pendekatan probabilistik, sedangkan algoritma lain mungkin memerlukan tahap pra-pemrosesan tambahan. Dengan penanganan yang tepat terhadap kedua jenis data ini, model pohon keputusan dapat dibangun secara lebih fleksibel, akurat, dan mampu melakukan generalisasi dengan baik pada data dunia nyata.

BAB 4
Ensemble Learning dan Random Forest

4.1 Konsep Ensemble Learning dalam Machine Learning

Ensemble Learning merupakan salah satu konsep penting dalam machine learning yang bertujuan untuk meningkatkan kinerja model prediktif dengan cara menggabungkan beberapa model pembelajaran (base learners) menjadi satu model gabungan. Ide utama dari ensemble learning adalah bahwa sekumpulan model yang bekerja bersama dapat menghasilkan prediksi yang lebih akurat, stabil, dan andal dibandingkan dengan satu model tunggal. Pendekatan ini terinspirasi dari prinsip bahwa keputusan kolektif sering kali lebih baik daripada keputusan individu. Konsep dasar ensemble learning berangkat dari kenyataan bahwa setiap model pembelajaran mesin memiliki keterbatasan. Model tunggal dapat mengalami masalah seperti overfitting, underfitting, atau sensitivitas terhadap noise dalam data. Dengan mengombinasikan beberapa model, kelemahan dari satu model dapat dikompensasi oleh kekuatan model lainnya. Dengan demikian, ensemble learning berperan sebagai strategi untuk mengurangi kesalahan prediksi secara keseluruhan.

Dalam ensemble learning, keberagaman (diversity) antar model menjadi faktor yang sangat penting. Jika semua model yang digabungkan menghasilkan prediksi yang sama, maka ensemble tidak memberikan manfaat tambahan. Oleh karena itu, model-model dalam ensemble biasanya dibangun dengan variasi tertentu, seperti menggunakan subset data yang berbeda, subset fitur yang berbeda, atau algoritma pembelajaran yang berbeda. Keberagaman ini memungkinkan ensemble menangkap berbagai perspektif pola dalam data. Salah satu tujuan utama

ensemble learning adalah mengatasi masalah bias variance trade off. Model dengan bias tinggi cenderung underfitting, sedangkan model dengan variansi tinggi cenderung overfitting. Ensemble learning, khususnya metode yang menggabungkan banyak model dengan variansi tinggi, dapat secara efektif menurunkan variansi tanpa meningkatkan bias secara signifikan. Hasilnya adalah model yang lebih seimbang dan mampu melakukan generalisasi dengan lebih baik. Pendekatan ensemble learning umumnya diklasifikasikan ke dalam beberapa metode utama, seperti bagging, boosting, dan stacking. Masing-masing metode memiliki mekanisme penggabungan model yang berbeda dan ditujukan untuk mengatasi jenis kesalahan yang berbeda pula. Pemilihan metode ensemble yang tepat sangat bergantung pada karakteristik data dan permasalahan yang dihadapi.

Bagging (Bootstrap Aggregating) merupakan salah satu metode ensemble learning yang paling sederhana dan populer. Pada bagging, beberapa model dilatih secara independen menggunakan dataset hasil bootstrap sampling, yaitu pengambilan sampel acak dengan pengembalian dari data latih. Setiap model dilatih secara paralel, dan hasil prediksi digabungkan melalui voting (untuk klasifikasi) atau rata-rata (untuk regresi). Bagging efektif dalam mengurangi variansi model. Contoh paling terkenal dari bagging adalah Random Forest, yang menggunakan pohon keputusan sebagai base learner. Random Forest tidak hanya menerapkan bootstrap sampling pada data, tetapi juga melakukan pemilihan fitur secara acak pada setiap node pohon. Kombinasi ini meningkatkan keberagaman antar pohon dan menjadikan Random Forest sangat tahan terhadap overfitting serta mampu bekerja dengan baik pada berbagai jenis data.

Berbeda dengan bagging, boosting merupakan metode ensemble yang membangun model secara berurutan. Setiap model baru difokuskan

untuk memperbaiki kesalahan yang dibuat oleh model sebelumnya. Data yang salah diklasifikasikan akan diberi bobot lebih besar sehingga mendapat perhatian lebih pada tahap berikutnya. Pendekatan ini bertujuan mengurangi bias dan meningkatkan akurasi model secara bertahap. Algoritma boosting yang populer antara lain AdaBoost, Gradient Boosting, dan XGBoost. Metode boosting sering menghasilkan performa yang sangat tinggi, terutama pada dataset kompleks. Namun, boosting juga lebih sensitif terhadap noise dan outlier, karena kesalahan pada data dapat terus diperkuat sepanjang proses pelatihan. Metode ensemble lainnya adalah stacking, yang menggabungkan prediksi dari beberapa model dasar menggunakan model tingkat lanjut (meta learner). Dalam stacking, model-model dasar menghasilkan prediksi awal, kemudian prediksi tersebut digunakan sebagai input untuk model meta yang belajar bagaimana cara mengombinasikan hasil tersebut secara optimal. Pendekatan ini menawarkan fleksibilitas tinggi, tetapi memerlukan desain dan validasi yang lebih kompleks. Ensemble learning juga memberikan keuntungan dari sisi stabilitas dan keandalan model. Karena keputusan akhir dihasilkan dari kombinasi beberapa model, hasil prediksi menjadi kurang sensitif terhadap fluktuasi kecil dalam data. Hal ini sangat penting dalam aplikasi nyata yang melibatkan data dinamis dan tidak sempurna, seperti keuangan, kesehatan, dan sistem rekomendasi.

Meskipun memiliki banyak kelebihan, ensemble learning juga memiliki keterbatasan. Model ensemble umumnya lebih kompleks, membutuhkan sumber daya komputasi yang lebih besar, serta lebih sulit diinterpretasikan dibandingkan model tunggal. Dalam beberapa kasus, kompleksitas ini menjadi tantangan, terutama ketika transparansi dan penjelasan model sangat dibutuhkan. Secara keseluruhan, ensemble learning merupakan strategi yang sangat kuat dalam machine learning untuk meningkatkan performa dan generalisasi model. Dengan

memanfaatkan kombinasi beberapa model yang beragam, ensemble learning mampu mengurangi kesalahan prediksi, meningkatkan stabilitas, dan menghasilkan model yang lebih andal. Oleh karena itu, konsep ensemble learning menjadi fondasi bagi banyak algoritma canggih dan tetap menjadi pendekatan utama dalam pengembangan sistem machine learning modern.

4.2 Bootstrap Aggregating (Bagging)

Bootstrap Aggregating (Bagging) merupakan salah satu metode utama dalam ensemble learning yang bertujuan untuk meningkatkan kinerja dan stabilitas model pembelajaran mesin dengan cara menggabungkan beberapa model yang dilatih secara independen. Konsep bagging diperkenalkan oleh Leo Breiman dan berfokus pada pengurangan variansi model, terutama pada algoritma yang sensitif terhadap perubahan data seperti Decision Tree. Dengan mengurangi variansi, bagging membantu model menghasilkan prediksi yang lebih konsisten dan mampu melakukan generalisasi dengan lebih baik pada data baru. Istilah bootstrap dalam bagging merujuk pada teknik bootstrap sampling, yaitu pengambilan sampel acak dari dataset pelatihan dengan pengembalian (sampling with replacement). Artinya, satu data dapat terpilih lebih dari satu kali dalam satu subset, sementara data lain mungkin tidak terpilih sama sekali. Dari dataset asli, bagging membentuk banyak subset data bootstrap dengan ukuran yang sama, yang masing-masing digunakan untuk melatih satu model dasar (base learner). Setiap model dasar dalam bagging dilatih secara independen dan paralel menggunakan subset data bootstrap yang berbeda. Karena setiap subset memiliki komposisi data yang sedikit berbeda, model-model yang dihasilkan juga akan memiliki perbedaan dalam struktur dan pola yang

dipelajari. Keberagaman antar model inilah yang menjadi kunci utama efektivitas bagging dalam meningkatkan performa ensemble.

Setelah seluruh model dasar dilatih, bagging menggabungkan hasil prediksi dari semua model tersebut untuk menghasilkan prediksi akhir. Pada masalah klasifikasi, penggabungan dilakukan dengan metode majority voting, di mana kelas yang paling banyak diprediksi oleh model-model dasar akan dipilih sebagai hasil akhir. Pada masalah regresi, prediksi akhir diperoleh dengan menghitung rata-rata dari seluruh prediksi model dasar. Keunggulan utama bagging terletak pada kemampuannya dalam mengurangi variansi model tanpa meningkatkan bias secara signifikan. Model seperti Decision Tree cenderung memiliki variansi tinggi karena perubahan kecil pada data latih dapat menghasilkan struktur pohon yang sangat berbeda. Dengan bagging, ketidakstabilan masing-masing model dapat dikompensasi melalui penggabungan prediksi, sehingga hasil akhir menjadi lebih stabil dan andal. Bagging sangat efektif ketika digunakan dengan model dasar yang kompleks dan tidak stabil, seperti pohon keputusan tanpa pruning. Sebaliknya, bagging kurang memberikan manfaat signifikan jika diterapkan pada model yang sudah stabil dan memiliki variansi rendah, seperti regresi linear. Oleh karena itu, pemilihan base learner yang tepat menjadi faktor penting dalam keberhasilan metode bagging. Salah satu contoh paling populer dari penerapan bagging adalah Random Forest. Random Forest merupakan pengembangan dari bagging yang menggunakan Decision Tree sebagai model dasar, dengan tambahan mekanisme pemilihan fitur secara acak pada setiap node. Kombinasi bootstrap sampling dan random feature selection meningkatkan keberagaman antar pohon, sehingga Random Forest menjadi salah satu algoritma paling kuat dan banyak digunakan dalam praktik machine learning.

Selain meningkatkan akurasi, bagging juga memberikan keuntungan dalam hal ketahanan terhadap overfitting. Meskipun masing-masing model dasar mungkin overfitting terhadap subset datanya, penggabungan banyak model tersebut cenderung menyeimbangkan kesalahan individual dan menghasilkan model ensemble yang lebih general. Namun, bagging tidak dirancang untuk mengatasi bias yang tinggi; jika model dasar terlalu sederhana dan mengalami underfitting, bagging tidak akan memberikan perbaikan yang signifikan. Dari sisi komputasi, bagging memiliki kelebihan karena proses pelatihan model dasar dapat dilakukan secara paralel. Namun, konsekuensinya adalah meningkatnya kebutuhan sumber daya komputasi dan memori, terutama ketika jumlah model dasar sangat banyak. Selain itu, interpretabilitas model ensemble hasil bagging cenderung lebih rendah dibandingkan model tunggal, karena keputusan akhir merupakan hasil agregasi banyak model. Secara keseluruhan, Bootstrap Aggregating (Bagging) merupakan teknik ensemble learning yang sederhana namun sangat efektif untuk meningkatkan stabilitas dan akurasi model dengan variansi tinggi. Dengan memanfaatkan bootstrap sampling dan penggabungan prediksi, bagging mampu menghasilkan model yang lebih robust terhadap noise dan fluktuasi data. Oleh karena itu, bagging menjadi fondasi penting bagi berbagai algoritma ensemble modern dan tetap relevan dalam pengembangan sistem machine learning hingga saat ini.

4.3 Mekanisme Kerja Random Forest

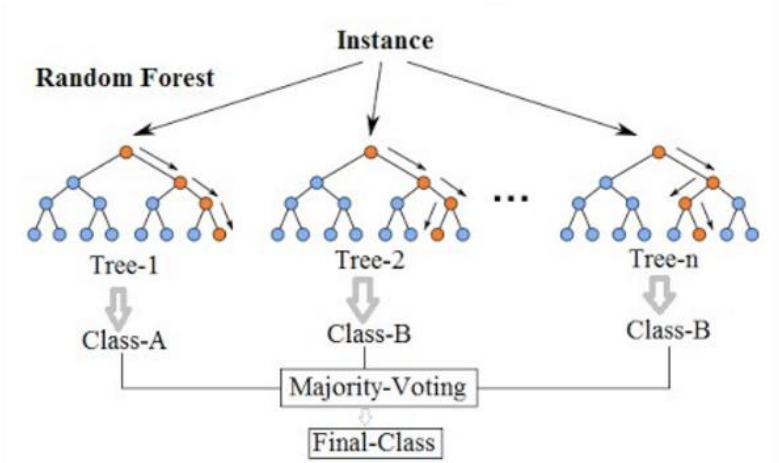
Random Forest merupakan algoritma ensemble learning yang bekerja dengan membangun dan menggabungkan banyak pohon keputusan (Decision Trees) untuk menghasilkan satu prediksi akhir yang lebih akurat dan stabil. Algoritma ini dikembangkan sebagai pengembangan dari metode Bootstrap Aggregating (Bagging) dengan

tambahan mekanisme pemilihan fitur secara acak, sehingga mampu mengurangi overfitting yang sering terjadi pada pohon keputusan tunggal. Mekanisme kerja Random Forest dimulai dengan proses bootstrap sampling. Dari dataset pelatihan asli, algoritma membuat banyak subset data secara acak dengan pengembalian (sampling with replacement). Setiap subset memiliki ukuran yang sama dengan dataset asli, tetapi komposisi datanya berbeda. Proses ini memastikan bahwa setiap pohon keputusan dilatih pada data yang sedikit berbeda, sehingga meningkatkan keberagaman antar pohon. Setelah subset data terbentuk, Random Forest membangun satu pohon keputusan untuk setiap subset. Pohon-pohon ini biasanya dibangun tanpa pruning atau dengan pembatasan minimal, sehingga masing-masing pohon memiliki kecenderungan variansi tinggi. Namun, dalam konteks ensemble, variansi tinggi pada pohon individual justru menguntungkan karena dapat dikompensasi melalui penggabungan banyak pohon. Keunikan utama Random Forest dibandingkan bagging biasa terletak pada pemilihan fitur secara acak di setiap node. Saat sebuah node akan melakukan pemisahan data, algoritma tidak mengevaluasi seluruh fitur yang tersedia, melainkan hanya memilih sebagian kecil fitur secara acak. Dari subset fitur tersebut, dipilih fitur terbaik berdasarkan kriteria tertentu seperti Gini Impurity atau Variance Reduction. Mekanisme ini semakin meningkatkan keberagaman struktur antar pohon.

Proses pembentukan pohon dilakukan secara top down dan rekursif, sama seperti pada algoritma CART. Setiap pohon tumbuh dengan struktur yang berbeda karena perbedaan data bootstrap dan fitur yang dipilih secara acak. Akibatnya, setiap pohon akan menangkap pola data dari sudut pandang yang berbeda-beda. Setelah seluruh pohon selesai dibangun, Random Forest memasuki tahap prediksi. Untuk sebuah data baru, data tersebut akan dimasukkan ke seluruh pohon keputusan

dalam hutan (forest). Setiap pohon akan menghasilkan prediksi masing-masing berdasarkan aturan yang dipelajarinya. Pada masalah klasifikasi, hasil prediksi dari semua pohon digabungkan menggunakan mekanisme majority voting, yaitu kelas yang paling sering diprediksi oleh pohon-pohon akan dipilih sebagai hasil akhir. Pada masalah regresi, prediksi akhir diperoleh dengan menghitung rata-rata dari seluruh prediksi pohon. Proses agregasi inilah yang membuat Random Forest lebih stabil dan akurat dibandingkan satu pohon tunggal.

Keunggulan utama mekanisme Random Forest adalah kemampuannya dalam mengurangi overfitting dan variansi model. Meskipun masing-masing pohon mungkin overfitting terhadap subset datanya, kesalahan individual tersebut cenderung saling meniadakan ketika digabungkan. Selain itu, Random Forest relatif tahan terhadap noise dan outlier karena keputusan akhir tidak bergantung pada satu model saja. Secara keseluruhan, mekanisme kerja Random Forest dapat diringkas sebagai proses membangun banyak pohon keputusan yang beragam melalui bootstrap sampling dan pemilihan fitur acak, kemudian menggabungkan hasil prediksinya. Kombinasi inilah yang menjadikan Random Forest sebagai salah satu algoritma machine learning paling kuat, stabil, dan banyak digunakan dalam praktik, baik untuk klasifikasi maupun regresi.



Gambar 4. Mekanisme Random Forest

4.4 Pemilihan Fitur Acak dan Reduksi Korelasi

Pemilihan fitur acak (***random feature selection***) dan reduksi korelasi merupakan dua konsep kunci yang menjelaskan mengapa algoritma Random Forest mampu mengungguli pohon keputusan tunggal dalam hal stabilitas dan generalisasi. Kedua mekanisme ini bekerja secara sinergis untuk meningkatkan keberagaman (diversity) antar pohon dalam ensemble, yang pada akhirnya menurunkan variansi dan meningkatkan kinerja prediksi. Pada pohon keputusan tunggal, seluruh fitur dievaluasi di setiap node untuk memilih pemisahan terbaik. Pendekatan ini sering menyebabkan fitur yang sangat kuat (misalnya fitur dengan korelasi tinggi terhadap target) dipilih berulang kali di banyak node. Akibatnya, struktur pohon menjadi sangat bergantung pada fitur tertentu, dan perubahan kecil pada data latih dapat menghasilkan pohon yang berbeda secara drastis sebuah gejala variansi tinggi. Pemilihan fitur acak mengatasi masalah tersebut dengan cara membatasi jumlah fitur kandidat yang dievaluasi di setiap node. Alih-alih menilai semua fitur, Random Forest

secara acak memilih subset kecil fitur (misalnya √p untuk klasifikasi atau p/3 untuk regresi, dengan p jumlah fitur total). Dari subset acak ini, fitur terbaik dipilih berdasarkan kriteria seperti Gini Impurity atau Variance Reduction. Pembatasan ini memaksa setiap pohon (bahkan setiap node) untuk "melihat" data dari perspektif fitur yang berbeda. Dampak langsung dari pemilihan fitur acak adalah meningkatnya keberagaman struktur pohon. Pohon-pohon dalam hutan tidak lagi didominasi oleh fitur yang sama di level atas, sehingga jalur keputusan yang terbentuk menjadi lebih beragam. Keberagaman ini penting karena ensemble hanya efektif jika model-model dasarnya tidak terlalu mirip satu sama lain.

Reduksi korelasi antar pohon merupakan konsekuensi lanjutan dari keberagaman tersebut. Jika semua pohon dilatih dengan fitur yang sama dan data yang mirip, kesalahan yang mereka buat cenderung terkorelasi artinya, ketika satu pohon salah, pohon lain juga salah pada contoh yang sama. Dengan pemilihan fitur acak, kesalahan antar pohon menjadi lebih tidak berkorelasi, sehingga saat prediksi digabungkan (voting atau rata-rata), kesalahan individual saling meniadakan. Secara statistik, manfaat reduksi korelasi dapat dipahami melalui efek ensemble: akurasi ensemble meningkat ketika variansi model turun dan korelasi antar kesalahan rendah. Random Forest mencapai tujuan ini tanpa meningkatkan bias secara signifikan, karena setiap pohon tetap cukup kuat (dibangun hingga kedalaman memadai), sementara perbedaan fitur dan data memastikan ketidaksamaan antar pohon. Pemilihan fitur acak juga memberikan keuntungan praktis pada dataset dengan fitur yang saling berkorelasi tinggi. Dalam kondisi seperti ini, pohon keputusan tunggal cenderung memilih salah satu fitur korelatif dan mengabaikan yang lain. Random Forest, melalui pemilihan fitur acak, memberi kesempatan bagi fitur-fitur korelatif tersebut untuk berperan di pohon yang berbeda, sehingga informasi yang tumpang tindih tetap termanfaatkan tanpa mendominasi

seluruh ensemble. Selain meningkatkan kinerja, mekanisme ini juga meningkatkan robustnes terhadap noise. Fitur yang berisik atau kebetulan tampak kuat pada sebagian data tidak akan selalu dipilih di setiap node dan setiap pohon. Dampak noise menjadi terlokalisasi pada beberapa pohon saja dan berkurang saat prediksi digabungkan. Secara keseluruhan, pemilihan fitur acak adalah strategi desain yang cerdas untuk menciptakan pohon-pohon yang beragam, sedangkan reduksi korelasi adalah hasil yang diinginkan dari keberagaman tersebut. Keduanya menjelaskan mengapa Random Forest memiliki performa yang stabil, tahan overfitting, dan unggul pada berbagai jenis data terutama ketika jumlah fitur besar dan terdapat korelasi antar fitur.

4.5 Perbandingan Random Forest dan Single Decision Tree

Adapun Perbandingan Random Forest dan Single Decision Tree adalah sebagai berikut [16]:

a. Konsep Dasar Model

   Single Decision Tree merupakan model pembelajaran mesin yang membangun satu struktur pohon keputusan berdasarkan data latih. Model ini mempelajari aturan if then secara langsung dari data dan menghasilkan satu jalur keputusan untuk setiap prediksi.

   Sebaliknya, Random Forest adalah metode ensemble learning yang menggabungkan banyak pohon keputusan. Setiap pohon dilatih menggunakan data dan fitur yang berbeda, kemudian hasil prediksinya digabungkan untuk menghasilkan keputusan akhir.

b. Akurasi dan Generalisasi

   Single Decision Tree cenderung memiliki akurasi yang baik pada data latih, tetapi sering mengalami penurunan performa

pada data uji karena overfitting. Hal ini disebabkan oleh ketergantungan model pada satu struktur pohon saja.

Random Forest umumnya menghasilkan akurasi dan kemampuan generalisasi yang lebih tinggi karena prediksi akhir merupakan hasil agregasi banyak pohon. Kesalahan pada satu pohon dapat dikompensasi oleh pohon lain, sehingga model lebih stabil.

c. Overfitting dan Variansi

Decision Tree tunggal memiliki variansi tinggi, artinya perubahan kecil pada data latih dapat menghasilkan struktur pohon yang sangat berbeda. Kondisi ini membuat model rentan terhadap overfitting.

Random Forest dirancang khusus untuk mengurangi variansi melalui bootstrap sampling dan pemilihan fitur acak. Dengan demikian, Random Forest jauh lebih tahan terhadap overfitting dibandingkan satu pohon keputusan.

d. Kompleksitas Model

Single Decision Tree memiliki struktur yang relatif sederhana dan mudah dipahami, terutama jika kedalaman pohon dibatasi. Kompleksitasnya rendah, baik dari sisi komputasi maupun penyimpanan. Sebaliknya, Random Forest memiliki kompleksitas yang lebih tinggi karena melibatkan banyak pohon keputusan. Proses pelatihan dan prediksi membutuhkan sumber daya komputasi dan memori yang lebih besar.

e. Interpretabilitas

Salah satu keunggulan utama Single Decision Tree adalah interpretabilitas yang tinggi. Setiap keputusan dapat ditelusuri melalui jalur dari root node ke leaf node dan diterjemahkan ke dalam aturan if then yang jelas.

Random Forest memiliki interpretabilitas yang lebih rendah karena keputusan akhir merupakan hasil agregasi banyak pohon. Meskipun demikian, Random Forest masih menyediakan informasi tambahan seperti feature importance untuk membantu interpretasi global.

f. Ketahanan terhadap Noise

Single Decision Tree sangat sensitif terhadap noise dan outlier. Data yang menyimpang dapat memengaruhi struktur pohon secara signifikan dan menghasilkan aturan keputusan yang tidak stabil.

Random Forest lebih robust terhadap noise, karena noise hanya memengaruhi sebagian pohon saja. Ketika prediksi digabungkan, dampak noise tersebut cenderung berkurang.

g. Kecepatan Pelatihan dan Prediksi

Single Decision Tree relatif cepat dilatih dan dieksekusi, sehingga cocok untuk aplikasi yang membutuhkan komputasi ringan dan respons cepat.

Random Forest membutuhkan waktu pelatihan yang lebih lama karena harus membangun banyak pohon, meskipun proses ini dapat dilakukan secara paralel. Waktu prediksi juga sedikit lebih lambat karena melibatkan agregasi hasil dari banyak pohon.

h. Fleksibilitas dan Kinerja Praktis

Dalam praktik, Single Decision Tree sering digunakan untuk analisis awal, eksplorasi data, atau aplikasi yang membutuhkan transparansi tinggi.

Random Forest lebih sering digunakan untuk aplikasi nyata berskala besar karena performanya yang stabil dan akurat pada berbagai jenis data, baik klasifikasi maupun regresi.

BAB 5
Persiapan dan Pra Pemrosesan Data

5.1 Struktur Dataset untuk Model Berbasis Pohon

Struktur dataset merupakan faktor yang sangat menentukan keberhasilan model berbasis pohon, seperti Decision Tree, Random Forest, dan algoritma ensemble sejenis. Model model ini memiliki keunggulan karena mampu bekerja langsung pada data tabular tanpa banyak asumsi statistik, namun tetap memerlukan struktur dataset yang jelas dan konsisten agar proses pembentukan pohon berjalan optimal. Secara umum, dataset untuk model berbasis pohon terdiri atas fitur (atribut) sebagai variabel input dan satu variabel target sebagai output yang akan diprediksi. Komponen utama dalam struktur dataset berbasis pohon adalah fitur (features), yaitu sekumpulan atribut yang merepresentasikan karakteristik setiap objek atau observasi. Fitur-fitur ini biasanya disusun dalam bentuk kolom pada tabel data, sementara setiap baris merepresentasikan satu sampel data [17]. Model berbasis pohon tidak memerlukan normalisasi atau standarisasi fitur karena proses pemisahan data dilakukan berdasarkan perbandingan nilai, bukan jarak numerik seperti pada algoritma berbasis gradien. Selain fitur, dataset juga harus memiliki variabel target (label) yang jelas. Pada masalah klasifikasi, target bersifat kategorikal atau diskret, seperti kelas "ya/tidak" atau "positif/negatif". Pada masalah regresi, target bersifat numerik kontinu, seperti harga, jumlah, atau nilai pengukuran tertentu. Kejelasan tipe target ini penting karena menentukan kriteria pemisahan yang digunakan, misalnya Gini Impurity untuk klasifikasi atau Variance Reduction untuk regresi.

Struktur dataset untuk model berbasis pohon dapat mencakup fitur numerik dan kategorikal secara bersamaan. Fitur numerik diinterpretasikan sebagai nilai kontinu yang dapat dibandingkan menggunakan nilai ambang tertentu, sedangkan fitur kategorikal diperlakukan sebagai himpunan nilai diskret. Model berbasis pohon relatif fleksibel dalam menangani kedua jenis fitur ini, meskipun pada implementasi tertentu fitur kategorikal perlu dikodekan terlebih dahulu agar dapat diproses oleh perangkat lunak. Kualitas struktur dataset juga sangat dipengaruhi oleh kelengkapan data (missing values). Beberapa algoritma pohon, seperti C4.5, mampu menangani nilai yang hilang secara internal, sementara algoritma lain memerlukan tahap pra-pemrosesan. Dataset yang memiliki banyak nilai kosong tanpa penanganan yang tepat dapat menghasilkan struktur pohon yang bias atau tidak stabil. Oleh karena itu, struktur dataset idealnya memiliki data yang lengkap atau telah melalui proses penanganan missing value.

Aspek penting lainnya adalah keseimbangan distribusi kelas pada dataset klasifikasi. Struktur dataset yang sangat tidak seimbang, misalnya satu kelas jauh lebih dominan dibandingkan kelas lainnya, dapat menyebabkan model berbasis pohon lebih memihak kelas mayoritas. Meskipun pohon keputusan cukup adaptif, ketidakseimbangan kelas tetap dapat memengaruhi kualitas split dan performa model, sehingga sering diperlukan penyesuaian tambahan seperti pembobotan kelas atau resampling. Struktur dataset untuk model berbasis pohon juga sebaiknya memperhatikan relevansi dan redundansi fitur. Fitur yang tidak relevan atau sangat berkorelasi dapat meningkatkan kompleksitas pohon tanpa memberikan informasi tambahan yang berarti. Meskipun model berbasis pohon relatif tahan terhadap fitur yang tidak penting, struktur dataset yang lebih ringkas dan informatif akan menghasilkan pohon yang lebih sederhana, interpretatif, dan stabil. Secara keseluruhan, struktur dataset

untuk model berbasis pohon idealnya berbentuk data tabular dengan baris sebagai sampel dan kolom sebagai fitur, memiliki target yang jelas, mendukung fitur numerik dan kategorikal, serta memiliki kualitas data yang baik. Dengan struktur dataset yang tepat, model berbasis pohon dapat membangun aturan keputusan yang akurat, mudah dipahami, dan mampu melakukan generalisasi dengan baik pada data baru.

5.2 Penanganan Missing Value dan Outlier

Penanganan missing value dan outlier merupakan tahap penting dalam pra-pemrosesan data untuk memastikan kualitas dan stabilitas model pembelajaran mesin, termasuk model berbasis pohon seperti Decision Tree dan Random Forest. Kedua permasalahan ini dapat memengaruhi proses pembentukan aturan keputusan, kualitas split, serta kemampuan generalisasi model. Berikut penjelasan lengkap mengenai cara penanganannya.

1. Penanganan Missing Value

Penanganan missing value merupakan tahap krusial dalam proses pra-pemrosesan data karena nilai yang hilang dapat memengaruhi kualitas analisis dan kinerja model pembelajaran mesin. Missing value terjadi ketika suatu atribut tidak memiliki nilai pada sebagian observasi, baik akibat kesalahan pencatatan, kegagalan sistem, maupun ketidaktersediaan informasi. Jika tidak ditangani dengan tepat, missing value dapat menyebabkan bias, menurunkan akurasi model, atau bahkan membuat algoritma gagal dijalankan. Langkah awal dalam penanganan missing value adalah mengidentifikasi pola dan proporsinya. Tidak semua missing value memiliki dampak yang sama; missing value yang jumlahnya sangat kecil dan bersifat acak mungkin tidak terlalu bermasalah, sedangkan missing value yang banyak atau terstruktur dapat memengaruhi distribusi data secara signifikan. Analisis awal ini

membantu menentukan strategi penanganan yang paling sesuai. Salah satu pendekatan paling sederhana adalah menghapus data yang mengandung missing value. Penghapusan dapat dilakukan pada tingkat baris (observasi) atau kolom (fitur). Pendekatan ini hanya disarankan jika jumlah missing value sangat kecil dan penghapusan tersebut tidak menghilangkan informasi penting. Jika digunakan secara berlebihan, metode ini berisiko mengurangi ukuran dataset dan menurunkan kemampuan generalisasi model.

Pendekatan yang lebih umum digunakan adalah imputasi nilai, yaitu menggantikan missing value dengan nilai tertentu. Untuk data numerik, nilai rata-rata, median, atau modus sering digunakan sebagai pengganti. Untuk data kategorikal, nilai kategori yang paling sering muncul biasanya dipilih. Metode ini mudah diterapkan dan cukup efektif ketika proporsi missing value relatif kecil dan distribusi data stabil. Namun, imputasi sederhana memiliki keterbatasan karena tidak mempertimbangkan hubungan antar fitur. Oleh karena itu, imputasi berbasis statistik lanjutan sering digunakan, seperti imputasi menggunakan median pada data yang tidak berdistribusi normal atau imputasi per kelompok berdasarkan kategori tertentu. Pendekatan ini menjaga karakteristik data dengan lebih baik dibandingkan imputasi global. Pendekatan yang lebih canggih adalah imputasi berbasis model, di mana missing value diprediksi menggunakan model pembelajaran mesin lain. Misalnya, regresi atau pohon keputusan dapat digunakan untuk memperkirakan nilai yang hilang berdasarkan fitur lain. Metode ini lebih akurat karena mempertimbangkan korelasi antar fitur, tetapi membutuhkan waktu dan sumber daya komputasi yang lebih besar. Dalam konteks model berbasis pohon tertentu, seperti C4.5, missing value dapat ditangani secara internal tanpa perlu imputasi eksplisit. Algoritma ini mendistribusikan bobot data yang memiliki missing value ke beberapa

cabang berdasarkan probabilitas kemunculan nilai yang ada. Pendekatan ini memungkinkan pemanfaatan data secara maksimal tanpa mengubah nilai asli dataset.

Selain teknik penggantian nilai, penandaan missing value sebagai kategori tersendiri juga dapat digunakan, terutama untuk data kategorikal. Dengan membuat kategori khusus seperti "tidak diketahui", model dapat mempelajari apakah ketiadaan informasi itu sendiri memiliki makna prediktif. Pendekatan ini berguna ketika missing value tidak bersifat acak. Penanganan missing value juga harus mempertimbangkan dampaknya terhadap distribusi data dan bias model. Imputasi yang tidak tepat dapat mengubah distribusi fitur dan memperkenalkan bias baru. Oleh karena itu, evaluasi model setelah imputasi sangat penting untuk memastikan bahwa strategi yang digunakan benar-benar meningkatkan performa dan stabilitas model.

2. Penanganan Outlier

Penanganan outlier merupakan tahap penting dalam pra-pemrosesan data karena keberadaan nilai ekstrem dapat memengaruhi kualitas analisis dan kinerja model pembelajaran mesin. Outlier adalah observasi yang memiliki nilai sangat jauh berbeda dari mayoritas data lainnya, baik karena kesalahan pencatatan, anomali sistem, maupun kejadian langka yang valid secara kontekstual. Tanpa penanganan yang tepat, outlier dapat menyebabkan model belajar pola yang tidak representatif. Langkah awal dalam penanganan outlier adalah identifikasi dan pemahaman karakteristiknya. Tidak semua outlier harus dihilangkan, karena sebagian outlier justru membawa informasi penting, terutama dalam domain seperti deteksi fraud atau analisis anomali. Oleh karena itu, penting untuk memahami apakah outlier merupakan kesalahan data atau fenomena nyata sebelum menentukan strategi penanganannya. Salah satu

metode umum untuk mendeteksi outlier adalah pendekatan statistik, seperti z score. Nilai dengan z-score yang sangat tinggi atau rendah dianggap menyimpang dari distribusi normal. Metode ini sederhana dan efektif untuk data yang mendekati distribusi normal, tetapi kurang cocok untuk data yang memiliki distribusi miring atau tidak simetris.

Metode statistik lain yang sering digunakan adalah Interquartile Range (IQR). Pendekatan ini mengidentifikasi outlier sebagai nilai yang berada di luar rentang Q1 − 1,5×IQR atau Q3 + 1,5×IQR. IQR lebih robust terhadap distribusi yang tidak normal dan sering digunakan pada data dunia nyata yang memiliki variasi tinggi. Selain metode statistik, pendekatan visual seperti boxplot dan scatter plot sangat membantu dalam mendeteksi outlier. Visualisasi memungkinkan analis melihat pola distribusi data secara langsung dan mengidentifikasi nilai ekstrem dengan cepat. Pendekatan ini juga membantu dalam memahami konteks outlier, terutama ketika dikombinasikan dengan pengetahuan domain. Setelah outlier teridentifikasi, salah satu strategi penanganan adalah penghapusan outlier. Penghapusan dapat dilakukan jika outlier terbukti sebagai kesalahan pencatatan atau data yang tidak valid. Namun, penghapusan harus dilakukan dengan hati-hati karena dapat mengurangi ukuran dataset dan berpotensi menghilangkan informasi penting jika outlier sebenarnya valid.

Alternatif dari penghapusan adalah transformasi data, seperti transformasi logaritmik, akar kuadrat, atau Box Cox. Transformasi ini bertujuan mengurangi pengaruh nilai ekstrem dengan membuat distribusi data lebih simetris. Transformasi sering digunakan pada data numerik dengan skala yang sangat lebar atau distribusi yang sangat miring. Strategi lain adalah winsorization, yaitu membatasi nilai ekstrem pada ambang tertentu. Dalam pendekatan ini, nilai outlier tidak dihapus, tetapi diganti dengan nilai batas atas atau bawah yang ditentukan. Metode ini

mempertahankan jumlah data sekaligus mengurangi dampak nilai ekstrem terhadap model. Dalam konteks pembelajaran mesin, pembatasan kompleksitas model juga dapat digunakan sebagai cara tidak langsung untuk menangani outlier. Pada model berbasis pohon, pengaturan parameter seperti min_samples_leaf dan max_depth dapat mencegah model membentuk aturan berdasarkan satu atau dua data ekstrem. Dengan demikian, pengaruh outlier dapat diminimalkan tanpa memodifikasi data secara eksplisit. Penggunaan metode ensemble, seperti Random Forest, merupakan pendekatan efektif lainnya. Karena Random Forest menggabungkan banyak model yang dilatih pada subset data yang berbeda, outlier hanya memengaruhi sebagian pohon saja. Ketika prediksi digabungkan, dampak outlier tersebut cenderung berkurang secara signifikan. Penanganan outlier juga perlu mempertimbangkan konteks dan tujuan analisis. Dalam beberapa kasus, outlier justru menjadi fokus utama, misalnya dalam deteksi penipuan atau kegagalan sistem. Dalam konteks ini, outlier tidak boleh dihilangkan, melainkan dianalisis secara khusus sebagai bagian dari tujuan pemodelan.

Evaluasi model setelah penanganan outlier merupakan langkah penting untuk memastikan bahwa strategi yang diterapkan memberikan dampak positif. Perbandingan performa model sebelum dan sesudah penanganan outlier dapat membantu menentukan apakah pendekatan yang digunakan sudah tepat. Evaluasi ini juga mencegah keputusan pra-pemrosesan yang justru menurunkan kualitas model. Secara keseluruhan, penanganan outlier merupakan proses yang memerlukan keseimbangan antara analisis statistik, pemahaman domain, dan tujuan pemodelan. Dengan strategi yang tepat mulai dari deteksi, penghapusan selektif, transformasi, hingga penggunaan model yang robust outlier dapat dikelola secara efektif. Penanganan outlier yang baik akan meningkatkan

stabilitas, akurasi, dan kemampuan generalisasi model pembelajaran mesin pada data dunia nyata.

5.3 Validasi Model dan Cross Validation

Validasi model dan cross validation merupakan dua konsep fundamental dalam pembelajaran mesin yang bertujuan untuk mengevaluasi kinerja model secara objektif dan memastikan kemampuan generalisasi model terhadap data yang belum pernah dilihat sebelumnya. Tanpa proses validasi yang tepat, model yang dibangun berisiko memberikan hasil yang menyesatkan karena performa yang tampak baik pada data latih belum tentu mencerminkan performa pada data nyata. Validasi model pada dasarnya berfokus pada pengukuran performa model menggunakan data yang terpisah dari data pelatihan. Data ini berfungsi sebagai representasi data baru yang akan dihadapi model saat diterapkan di dunia nyata. Dengan memisahkan data latih dan data evaluasi, peneliti dapat memperoleh gambaran yang lebih realistis mengenai kualitas prediksi model. Salah satu alasan utama dilakukannya validasi model adalah untuk menghindari overfitting. Overfitting terjadi ketika model terlalu menyesuaikan diri dengan data latih, termasuk noise dan pola yang tidak relevan, sehingga kehilangan kemampuan generalisasi. Validasi model membantu mendeteksi kondisi ini dengan membandingkan performa pada data latih dan data uji. Selain overfitting, validasi model juga berperan dalam mengidentifikasi underfitting [18]. Underfitting terjadi ketika model terlalu sederhana dan gagal menangkap pola penting dalam data. Jika performa model rendah baik pada data latih maupun data uji, maka validasi menunjukkan bahwa model membutuhkan peningkatan kompleksitas atau perbaikan fitur.

Pendekatan validasi yang paling sederhana dan paling umum digunakan adalah train test split. Dalam metode ini, dataset dibagi

menjadi dua bagian, yaitu data latih untuk membangun model dan data uji untuk mengevaluasi kinerjanya. Rasio pembagian data biasanya ditentukan secara empiris, misalnya 70% untuk pelatihan dan 30% untuk pengujian. Meskipun sederhana, metode train–test split memiliki keterbatasan karena hasil evaluasi sangat bergantung pada satu kali pembagian data. Jika pembagian tersebut tidak representatif, maka estimasi performa model bisa menjadi bias. Oleh karena itu, metode ini kurang stabil terutama pada dataset berukuran kecil atau dengan distribusi kelas yang tidak seimbang. Untuk mengatasi keterbatasan tersebut, digunakan cross validation sebagai teknik validasi yang lebih komprehensif. Cross-validation mengevaluasi model dengan menggunakan beberapa pembagian data yang berbeda, sehingga hasil evaluasi tidak bergantung pada satu skenario pembagian saja. Pendekatan ini memberikan estimasi performa yang lebih andal dan stabil.

Metode cross-validation yang paling populer adalah k fold cross validation. Pada metode ini, dataset dibagi menjadi k bagian atau fold dengan ukuran yang relatif sama. Model kemudian dilatih menggunakan k−1 fold dan diuji pada satu fold yang tersisa. Proses ini diulang hingga setiap fold pernah digunakan sebagai data uji. Hasil evaluasi dari setiap fold kemudian dirata-ratakan untuk memperoleh nilai performa akhir. Dengan cara ini, setiap data dalam dataset berperan sebagai data latih dan data uji, sehingga evaluasi menjadi lebih menyeluruh. K fold cross validation sangat efektif dalam memberikan estimasi performa yang mendekati kondisi nyata. Pemilihan nilai k dalam k fold cross validation juga memiliki implikasi penting. Nilai k yang kecil menghasilkan evaluasi yang lebih cepat tetapi kurang stabil, sedangkan nilai k yang besar, seperti leave one out cross validation, memberikan evaluasi yang lebih mendalam namun membutuhkan komputasi yang lebih besar. Oleh karena itu, nilai k biasanya dipilih antara 5 hingga 10 sebagai kompromi yang seimbang.

Dalam permasalahan klasifikasi dengan distribusi kelas yang tidak seimbang, digunakan stratified cross-validation. Metode ini memastikan bahwa proporsi kelas pada setiap fold tetap sama dengan proporsi kelas pada dataset asli. Pendekatan ini penting agar setiap fold merepresentasikan kondisi data secara adil dan hasil evaluasi tidak bias terhadap kelas mayoritas.

Cross validation juga memainkan peran penting dalam pemilihan model dan penentuan hyperparameter. Dengan mengevaluasi berbagai konfigurasi model menggunakan cross-validation, peneliti dapat memilih model yang tidak hanya memiliki performa terbaik pada satu subset data, tetapi juga konsisten pada berbagai subset data. Hal ini meningkatkan kepercayaan terhadap model yang dipilih. Dalam konteks pengembangan model berbasis pohon, seperti Decision Tree dan Random Forest, cross validation sangat membantu dalam menentukan parameter seperti kedalaman pohon, jumlah pohon, dan jumlah fitur yang dipilih. Parameter yang dipilih berdasarkan cross-validation cenderung menghasilkan model yang lebih stabil dan tidak mudah overfitting. Meskipun sangat bermanfaat, cross-validation memiliki biaya komputasi yang lebih tinggi dibandingkan validasi sederhana. Model harus dilatih dan diuji berulang kali, sehingga waktu komputasi dan penggunaan sumber daya meningkat. Oleh karena itu, pemilihan teknik validasi harus mempertimbangkan ukuran dataset dan kompleksitas model. Validasi model dan cross-validation juga berperan dalam evaluasi metrik kinerja, seperti akurasi, presisi, recall, F1-score, atau error regresi. Dengan mengevaluasi metrik-metrik ini pada beberapa fold, peneliti dapat memperoleh gambaran yang lebih lengkap mengenai kekuatan dan kelemahan model. Selain itu, hasil cross-validation dapat digunakan untuk menilai stabilitas model. Model yang memiliki performa konsisten pada setiap fold dianggap lebih stabil dan lebih dapat diandalkan.

Sebaliknya, model dengan variasi performa yang besar antar fold menunjukkan sensitivitas tinggi terhadap data dan berpotensi kurang robust.

Dalam praktik, validasi model dan cross-validation sering dikombinasikan dengan teknik lain seperti early stopping atau nested cross validation untuk mencegah kebocoran data (data leakage). Pendekatan ini sangat penting dalam penelitian akademik dan aplikasi industri yang menuntut evaluasi model secara ketat dan transparan. Secara keseluruhan, validasi model dan cross-validation merupakan komponen esensial dalam siklus pengembangan machine learning. Dengan menerapkan teknik validasi yang tepat, peneliti dapat memastikan bahwa model yang dibangun tidak hanya berkinerja baik pada data latih, tetapi juga memiliki kemampuan generalisasi yang kuat, stabil, dan siap digunakan pada data dunia nyata.

BAB 6
Implementasi Decision Tree di Google Colab

6.1 Implementasi Decision Tree untuk Klasifikasi

Implementasi Decision Tree untuk klasifikasi merupakan penerapan algoritma pohon keputusan untuk memprediksi label kelas diskret berdasarkan sekumpulan fitur input. Decision Tree bekerja dengan membangun struktur pohon yang berisi aturan keputusan berbentuk if then, sehingga sangat populer dalam masalah klasifikasi karena mudah dipahami, interpretatif, dan fleksibel terhadap berbagai jenis data. Pada tahap awal implementasi, dataset klasifikasi harus disiapkan dengan jelas, yaitu terdiri atas fitur (atribut prediktor) dan label kelas (target). Label kelas bersifat kategorikal, misalnya ya/tidak, positif/negatif, atau kelas A/B/C. Model Decision Tree tidak memerlukan normalisasi data dan dapat menangani fitur numerik maupun kategorikal, meskipun dalam implementasi perangkat lunak tertentu fitur kategorikal sering perlu dikodekan terlebih dahulu. Proses pelatihan Decision Tree dimulai dengan menempatkan seluruh data latih pada root node. Algoritma kemudian memilih atribut terbaik untuk memisahkan data berdasarkan kriteria tertentu, seperti Gini Impurity atau Entropy/Information Gain. Pada klasifikasi, tujuan utama pemilihan atribut adalah meminimalkan ketidakmurnian kelas pada node-node hasil pemisahan, sehingga setiap subset data menjadi semakin homogen. Setelah atribut terbaik dipilih, data dibagi ke dalam beberapa subset sesuai dengan nilai atribut tersebut. Untuk fitur numerik, pemisahan dilakukan menggunakan nilai ambang (threshold), sedangkan untuk fitur kategorikal, pemisahan dilakukan berdasarkan kategori. Setiap subset hasil pemisahan membentuk node

anak, dan proses ini diulang secara rekursif pada setiap node hingga kondisi penghentian terpenuhi.

Kondisi penghentian dalam implementasi Decision Tree klasifikasi dapat berupa seluruh data dalam node berasal dari kelas yang sama, tidak ada lagi atribut yang dapat digunakan, atau pembatasan tertentu seperti kedalaman maksimum pohon. Ketika proses berhenti, node tersebut menjadi node daun (leaf node) yang menyimpan hasil prediksi berupa kelas mayoritas dari data dalam node tersebut. Untuk mencegah pohon tumbuh terlalu kompleks dan mengalami overfitting, implementasi Decision Tree biasanya melibatkan pengaturan hyperparameter, seperti max_depth, min_samples_split, dan min_samples_leaf. Pengaturan ini berfungsi sebagai bentuk regularisasi yang mengendalikan kompleksitas pohon. Parameter-parameter tersebut sangat penting dalam implementasi praktis agar model memiliki kemampuan generalisasi yang baik.

Setelah model dilatih, Decision Tree digunakan untuk melakukan prediksi pada data baru. Setiap data uji akan ditelusuri dari root node hingga node daun berdasarkan aturan pemisahan yang telah dipelajari. Kelas yang tersimpan pada node daun menjadi hasil prediksi untuk data tersebut. Proses ini sangat efisien karena hanya melibatkan evaluasi kondisi sederhana. Evaluasi kinerja Decision Tree untuk klasifikasi dilakukan menggunakan metrik seperti akurasi, precision, recall, F1-score, dan confusion matrix. Evaluasi ini biasanya dilakukan pada data uji atau melalui cross-validation untuk memastikan bahwa model tidak hanya akurat pada data latih, tetapi juga mampu melakukan generalisasi dengan baik. Salah satu keunggulan utama implementasi Decision Tree untuk klasifikasi adalah interpretabilitasnya yang tinggi. Struktur pohon dapat divisualisasikan dan setiap jalur keputusan dapat diterjemahkan menjadi aturan logis yang mudah dipahami oleh manusia. Hal ini menjadikan Decision Tree sangat cocok untuk aplikasi yang membutuhkan

transparansi, seperti sistem pendukung keputusan, analisis risiko, dan bidang kesehatan. Namun, dalam implementasi nyata, Decision Tree klasifikasi juga memiliki keterbatasan, terutama terkait variansi tinggi dan sensitivitas terhadap data. Oleh karena itu, dalam banyak kasus, Decision Tree digunakan sebagai model dasar yang kemudian dikembangkan menjadi metode ensemble seperti Random Forest untuk meningkatkan stabilitas dan akurasi.

Contoh:

Studi Kasus Dataset: Breast Cancer Wisconsin

Tujuan: Mengklasifikasikan tumor jinak (benign) atau ganas (malignant)

1. Import Library

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

2. Memuat Dataset

```
data = load_breast_cancer()

X = data.data
y = data.target
```

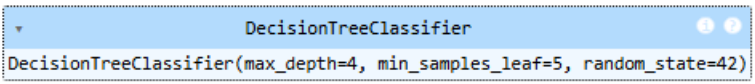
```
feature_names = data.feature_names
target_names = data.target_names
```

3. Split Data (Train & Test)

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42,
    stratify=y
)
```

4. Membangun Model Decision Tree

```
dt_model = DecisionTreeClassifier(
    criterion='gini',     # atau 'entropy'
    max_depth=4,          # mencegah overfitting
    min_samples_leaf=5,
    random_state=42
)

dt_model.fit(X_train, y_train)
```



5. Prediksi Data Uji

```
y_pred = dt_model.predict(X_test)
```

6. Evaluasi Model

Akurasi

```
accuracy = accuracy_score(y_test, y_pred)
```

```
print("Accuracy:", accuracy)
```

```
Accuracy: 0.935672514619883
```

Classification Report

```
print(classification_report(y_test, y_pred,
target_names=target_names))
```

```
              precision    recall  f1-score   support

   malignant       0.92      0.91      0.91        64
      benign       0.94      0.95      0.95       107

    accuracy                           0.94       171
   macro avg       0.93      0.93      0.93       171
weighted avg       0.94      0.94      0.94       171
```

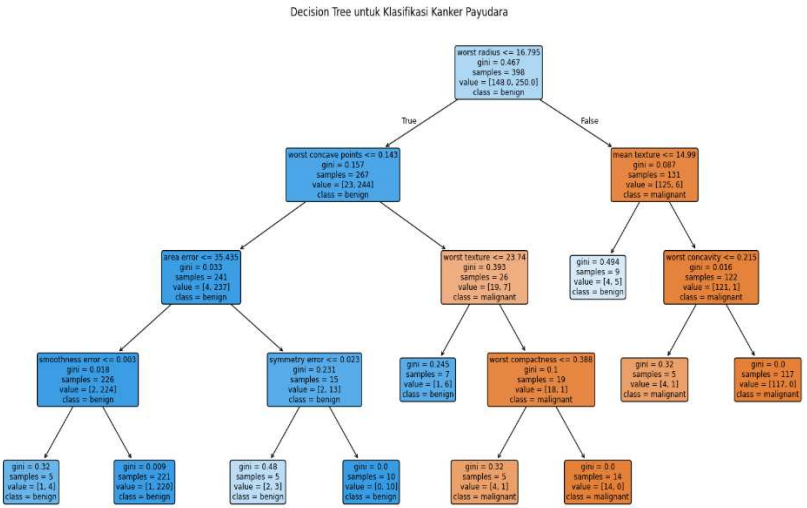
Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

7. Visualisasi Pohon Keputusan

```
[[ 58   6]
 [  5 102]]
```

```
plt.figure(figsize=(20,10))
plot_tree(
   dt_model,
   feature_names=feature_names,
   class_names=target_names,
   filled=True,
   rounded=True
)
plt.title("Decision Tree untuk Klasifikasi Kanker Payudara")
plt.show()
```



6.2 Implementasi Decision Tree untuk Regresi

Decision Tree untuk regresi merupakan penerapan algoritma pohon keputusan yang digunakan untuk memprediksi nilai numerik kontinu, seperti harga rumah, jumlah penjualan, suhu, atau nilai ekonomi lainnya. Berbeda dengan Decision Tree klasifikasi yang menghasilkan label kelas, Decision Tree regresi menghasilkan nilai prediksi berupa angka yang biasanya dihitung sebagai nilai rata-rata dari data pada node daun. Pada Decision Tree regresi, proses pembentukan pohon dilakukan secara top down dan rekursif, dimulai dari root node yang memuat seluruh data latih [19]. Pada setiap node, algoritma akan memilih atribut dan nilai ambang (threshold) terbaik untuk membagi data menjadi dua subset. Tujuan utama pemisahan ini adalah meminimalkan variasi nilai target pada node-node hasil pemisahan. Kriteria pemilihan atribut pada Decision Tree regresi umumnya menggunakan Variance Reduction atau Mean Squared Error (MSE). Variansi mengukur seberapa besar penyebaran nilai target dalam suatu node. Atribut yang dipilih adalah atribut yang mampu

menghasilkan penurunan variansi terbesar setelah pemisahan data, sehingga nilai target dalam setiap subset menjadi lebih homogen.

Setelah atribut terbaik dipilih, data dibagi menjadi dua subset berdasarkan kondisi numerik, misalnya x ≤ threshold dan x > threshold. Setiap subset kemudian diproses kembali dengan mekanisme yang sama. Proses ini berlangsung hingga tercapai kondisi penghentian, seperti kedalaman maksimum pohon, jumlah minimum sampel pada node, atau tidak adanya penurunan variansi yang signifikan. Pada node daun (leaf node), Decision Tree regresi tidak menyimpan kelas, melainkan nilai prediksi, biasanya berupa rata-rata nilai target dari seluruh data yang berada pada node tersebut. Ketika data baru diberikan, model akan menelusuri pohon dari root hingga leaf dan menghasilkan nilai numerik sebagai hasil prediksi. Salah satu keunggulan utama Decision Tree regresi adalah kemampuannya dalam menangkap hubungan non linear tanpa memerlukan asumsi bentuk fungsi tertentu. Selain itu, model ini bersifat interpretatif, karena setiap prediksi dapat ditelusuri melalui aturan keputusan yang jelas. Namun, Decision Tree regresi juga rentan terhadap overfitting jika pohon terlalu dalam dan tidak dibatasi.

Contoh Implementasi Decision Tree untuk Regresi (Google Colab)

Studi Kasus Dataset: California Housing Dataset

Tujuan: Memprediksi harga rumah (median house value)

1. Import Library

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
```

```
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

2. Memuat Dataset (contoh)

```
import pandas as pd
from io import StringIO

data_csv = StringIO("""
Luas_Tanah,Luas_Bangunan,Jumlah_Kamar,Jumlah_Kamar_Mandi,Jarak_Pusat_Kota,Umur_Bangunan,Harga_Rumah
120,90,3,2,5,10,850
150,120,4,3,3,5,1200
90,70,2,1,8,15,600
200,160,5,4,2,2,1800
110,85,3,2,6,12,780
130,100,3,2,4,8,950
170,140,4,3,3,6,1450
80,60,2,1,10,20,500
220,180,6,4,1,1,2200
160,130,4,3,4,7,1350
140,110,3,2,5,9,1050
100,75,2,1,9,18,580
190,150,5,4,2,4,1700
125,95,3,2,6,11,820
210,170,5,4,2,3,1950
""")
```

```
df = pd.read_csv(data_csv)
df.head()
```

	Luas_Tanah	Luas_Bangunan	Jumlah_Kamar	Jumlah_Kamar_Mandi	Jarak_Pusat_Kota	Umur_Bangunan	Harga_Rumah
0	120	90	3	2	5	10	850
1	150	120	4	3	3	5	1200
2	90	70	2	1	8	15	600
3	200	160	5	4	2	2	1800
4	110	85	3	2	6	12	780

3. Pisahkan Fitur dan Target

```
X = df.drop("Harga_Rumah", axis=1)
y = df["Harga_Rumah"]
```

4. Split data train test

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.3,
    random_state=42
)
```

5. Bangun model decision tree regresi

```
from sklearn.tree import DecisionTreeRegressor

dt_reg = DecisionTreeRegressor(
    criterion='squared_error',  # Variance Reduction
    max_depth=4,
    min_samples_leaf=2,
    random_state=42
)
```

```
dt_reg.fit(X_train, y_train)
```

6. Prediksi

```
y_pred = dt_reg.predict(X_test)
```

7. Evaluasi Model

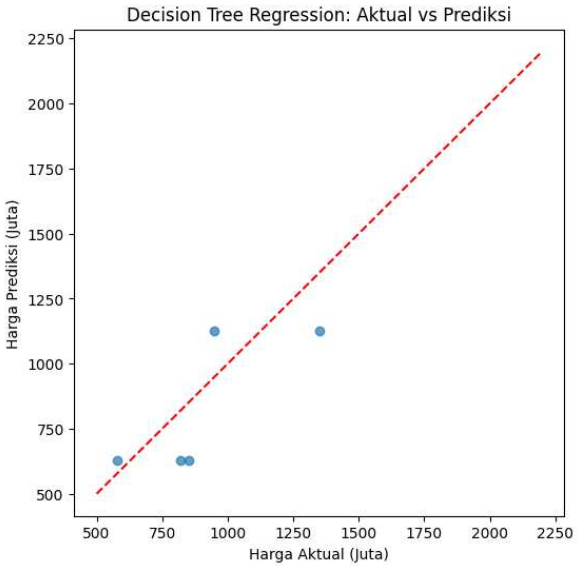
```
from sklearn.metrics import mean_squared_error,
mean_absolute_error, r2_score
import numpy as np
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print("MSE :", mse)
print("RMSE:", rmse)
print("MAE :", mae)
print("R²  :", r2)
```

```
MSE : 34136.66666666667
RMSE: 184.76110701840545
MAE : 172.66666666666669
R²  : 0.45952079375131927
```

8. Visualisasi Aktual vs Prediksi

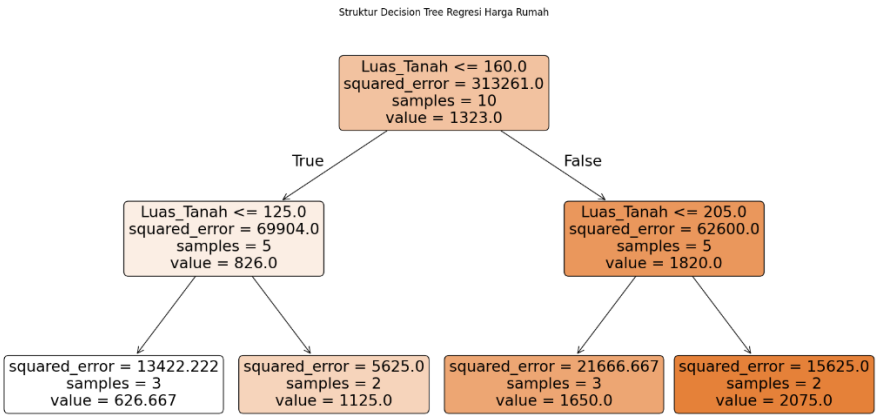
```
import matplotlib.pyplot as plt

plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.xlabel("Harga Aktual (Juta)")
plt.ylabel("Harga Prediksi (Juta)")
```

```
plt.title("Decision Tree Regression: Aktual vs Prediksi")
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--')
plt.show()
```



9. Visualisasi pohon keputusan

```
from sklearn.tree import plot_tree

plt.figure(figsize=(20,10))
plot_tree(
    dt_reg,
    feature_names=X.columns,
    filled=True,
    rounded=True
)
plt.title("Struktur Decision Tree Regresi Harga Rumah")
plt.show()
```



6.3 Analisis Hasil dan Interpretasi Model

Analisis hasil dan interpretasi model merupakan tahap akhir yang sangat penting dalam proses pembelajaran mesin, karena pada tahap inilah kinerja model dievaluasi secara menyeluruh dan makna dari hasil prediksi dijelaskan secara konseptual. Tahap ini tidak hanya bertujuan untuk mengetahui seberapa baik model bekerja secara numerik, tetapi juga untuk memahami bagaimana dan mengapa model menghasilkan keputusan tertentu. Analisis yang baik akan membantu memastikan bahwa model yang dibangun layak digunakan, dapat dipercaya, dan sesuai dengan tujuan penelitian atau aplikasi. Langkah awal dalam analisis hasil adalah mengevaluasi metrik kinerja model. Pada kasus regresi, metrik yang umum digunakan meliputi Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), dan koefisien determinasi (R^2) [20]. Nilai MSE dan RMSE menggambarkan besar kesalahan prediksi secara kuadrat dan akar kuadrat, sehingga sensitif terhadap kesalahan besar. Sementara itu, MAE memberikan gambaran rata-rata kesalahan absolut, dan R^2 menunjukkan seberapa besar variansi data target yang dapat dijelaskan oleh model. Setelah metrik dihitung, analisis dilanjutkan dengan membandingkan hasil prediksi

dengan nilai aktual. Visualisasi seperti grafik sebar (scatter plot) antara nilai aktual dan nilai prediksi sangat membantu dalam melihat pola kesalahan. Jika titik-titik data mendekati garis diagonal, maka model memiliki kemampuan prediksi yang baik. Pola penyimpangan tertentu dapat mengindikasikan bias model, misalnya kecenderungan underestimasi atau overestimasi pada rentang nilai tertentu.

Tahap berikutnya adalah analisis stabilitas dan generalisasi model. Performa model pada data latih dibandingkan dengan data uji untuk mengetahui apakah model mengalami overfitting atau underfitting. Jika kinerja pada data latih jauh lebih baik daripada data uji, maka model kemungkinan besar overfitting. Sebaliknya, jika kinerja rendah pada kedua data tersebut, model cenderung underfitting dan belum mampu menangkap pola utama dalam data. Interpretasi model berbasis pohon, seperti Decision Tree, dapat dilakukan dengan menganalisis struktur pohon keputusan. Setiap node dalam pohon merepresentasikan kondisi pemisahan berdasarkan fitur tertentu, sedangkan setiap cabang menunjukkan hasil keputusan dari kondisi tersebut. Dengan menelusuri jalur dari root hingga leaf node, pengguna dapat memahami aturan keputusan yang digunakan model dalam menghasilkan prediksi. Selain struktur pohon, feature importance merupakan aspek penting dalam interpretasi model. Nilai feature importance menunjukkan kontribusi relatif setiap fitur terhadap proses prediksi. Fitur dengan nilai importance tinggi memiliki pengaruh besar dalam menentukan hasil prediksi. Analisis ini membantu peneliti memahami faktor-faktor dominan yang memengaruhi target, misalnya luas bangunan atau jarak ke pusat kota dalam prediksi harga rumah. Analisis hasil juga perlu mempertimbangkan kesalahan prediksi (error analysis) secara lebih mendalam. Dengan mengamati contoh data yang memiliki kesalahan prediksi besar, peneliti dapat mengidentifikasi kemungkinan penyebab

kesalahan, seperti outlier, data yang tidak representatif, atau fitur penting yang belum dimasukkan ke dalam model. Pendekatan ini membantu meningkatkan kualitas model pada iterasi berikutnya.

Dalam konteks penelitian atau aplikasi nyata, interpretasi model juga harus dikaitkan dengan pengetahuan domain. Hasil model perlu diperiksa apakah masuk akal secara logis dan sesuai dengan realitas. Misalnya, jika model menunjukkan bahwa jarak ke pusat kota sangat memengaruhi harga rumah, hasil tersebut harus dibandingkan dengan teori atau pengalaman di bidang properti untuk memastikan konsistensinya. Analisis hasil dan interpretasi model juga mencakup evaluasi keterbatasan model. Setiap model memiliki asumsi dan batasan tertentu, seperti sensitivitas terhadap perubahan data atau keterbatasan dalam menangkap hubungan yang sangat kompleks. Menyadari keterbatasan ini penting agar hasil model tidak disalahartikan atau digunakan di luar konteks yang tepat. Pada tahap akhir, hasil analisis dan interpretasi perlu dirangkum dalam bentuk kesimpulan yang jelas dan terstruktur. Kesimpulan ini menjelaskan apakah model telah memenuhi tujuan yang ditetapkan, seberapa baik kinerjanya, dan bagaimana model dapat dimanfaatkan dalam pengambilan keputusan. Dalam konteks akademik, kesimpulan juga sering disertai dengan saran pengembangan lebih lanjut.

6.4 Feature Importance dan Analisis Kontribusi Fitur

Feature importance dan analisis kontribusi fitur merupakan bagian penting dalam interpretasi model pembelajaran mesin, khususnya pada model berbasis pohon seperti Decision Tree dan Random Forest. Tujuan utamanya adalah untuk memahami sejauh mana setiap fitur berperan dalam menghasilkan prediksi, sehingga model tidak hanya dinilai dari akurasi, tetapi juga dari sisi keterjelasan dan makna keputusan yang

diambil. Feature importance mengacu pada ukuran kuantitatif yang menunjukkan tingkat kepentingan relatif suatu fitur dalam proses pembentukan model [8]. Pada model berbasis pohon, nilai ini biasanya dihitung berdasarkan kontribusi fitur terhadap penurunan impuritas (misalnya Gini Impurity atau Variance Reduction) di seluruh node tempat fitur tersebut digunakan. Semakin sering dan semakin besar penurunan impuritas yang dihasilkan oleh suatu fitur, semakin tinggi nilai importance-nya. Dalam Decision Tree, feature importance dapat diinterpretasikan secara langsung melalui struktur pohon. Fitur yang muncul pada level atas pohon (dekat root) umumnya memiliki pengaruh besar karena digunakan untuk memisahkan data dalam jumlah besar sejak awal. Sebaliknya, fitur yang hanya muncul di cabang bawah biasanya memiliki kontribusi yang lebih kecil dan bersifat lokal pada subset data tertentu.

Pada Random Forest, feature importance dihitung sebagai rata-rata kontribusi fitur di seluruh pohon dalam ensemble. Pendekatan ini menghasilkan estimasi yang lebih stabil dibandingkan satu pohon tunggal, karena menggabungkan banyak perspektif pemisahan data. Nilai importance pada Random Forest sering digunakan untuk mengidentifikasi fitur dominan dalam dataset berdimensi tinggi. Analisis kontribusi fitur tidak berhenti pada nilai numerik feature importance saja, tetapi juga mencakup pemahaman arah dan konteks pengaruh fitur. Misalnya, meskipun suatu fitur memiliki importance tinggi, perlu dianalisis bagaimana fitur tersebut memengaruhi prediksi apakah melalui threshold tertentu, kombinasi dengan fitur lain, atau hanya pada kondisi data tertentu. Analisis ini biasanya dilakukan dengan menelusuri aturan keputusan atau node-node penting dalam pohon. Selain kontribusi global, penting juga memahami kontribusi fitur secara lokal, yaitu bagaimana fitur memengaruhi prediksi untuk satu observasi tertentu. Pada model

berbasis pohon, hal ini dapat dilihat dengan menelusuri jalur keputusan dari root hingga leaf node untuk satu data. Setiap split menunjukkan fitur apa yang berperan dalam menentukan prediksi akhir untuk observasi tersebut.

Feature importance juga berguna dalam seleksi fitur (feature selection). Fitur dengan importance sangat rendah dapat dipertimbangkan untuk dihapus guna menyederhanakan model, mengurangi noise, dan meningkatkan efisiensi komputasi. Namun, penghapusan fitur harus dilakukan dengan hati-hati, karena fitur dengan kontribusi kecil secara global bisa saja penting pada subset data tertentu. Dalam analisis lanjutan, feature importance perlu dikaitkan dengan pengetahuan domain. Fitur yang dinilai penting oleh model sebaiknya masuk akal secara konseptual. Jika model menilai fitur yang secara logis tidak relevan sebagai sangat penting, hal ini dapat menjadi indikasi adanya bias data, korelasi semu, atau masalah kualitas data. Penting juga disadari bahwa feature importance pada model berbasis pohon bersifat relatif, bukan absolut. Nilai importance menunjukkan perbandingan antar fitur dalam satu model tertentu dan dapat berubah jika dataset, parameter model, atau metode pelatihan berubah. Oleh karena itu, hasil analisis sebaiknya tidak ditafsirkan secara terpisah dari konteks pemodelan. Secara keseluruhan, feature importance dan analisis kontribusi fitur berfungsi sebagai alat interpretasi yang membantu menjembatani kinerja numerik model dengan pemahaman konseptual. Dengan analisis ini, pengguna dapat mengetahui faktor-faktor utama yang memengaruhi prediksi, meningkatkan kepercayaan terhadap model, serta mendukung pengambilan keputusan yang lebih transparan dan berbasis data.

Contoh:

1. Pastikan Dataset & Model Sudah Dimuat (dataset contoh)

```
import pandas as pd
from io import StringIO

data_csv = StringIO("""
Luas_Tanah,Luas_Bangunan,Jumlah_Kamar,Jumlah_Kamar_Mandi,J
arak_Pusat_Kota,Umur_Bangunan,Harga_Rumah
120,90,3,2,5,10,850
150,120,4,3,3,5,1200
90,70,2,1,8,15,600
200,160,5,4,2,2,1800
110,85,3,2,6,12,780
130,100,3,2,4,8,950
170,140,4,3,3,6,1450
80,60,2,1,10,20,500
220,180,6,4,1,1,2200
160,130,4,3,4,7,1350
140,110,3,2,5,9,1050
100,75,2,1,9,18,580
190,150,5,4,2,4,1700
125,95,3,2,6,11,820
210,170,5,4,2,3,1950
""")

df = pd.read_csv(data_csv)

X = df.drop("Harga_Rumah", axis=1)
y = df["Harga_Rumah"]
```

2. Train Model Decision Tree Regresi

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

dt_model = DecisionTreeRegressor(
    max_depth=4,
    min_samples_leaf=2,
    random_state=42
)

dt_model.fit(X_train, y_train)
```

FEATURE IMPORTANCE (GLOBAL)

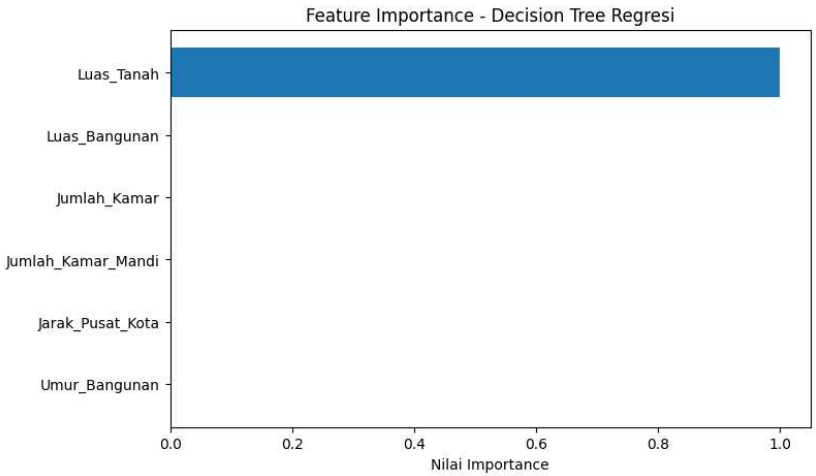
3. Mengambil Nilai Feature Importance

```
import numpy as np

importance = dt_model.feature_importances_

feature_importance_df = pd.DataFrame({
    "Fitur": X.columns,
    "Importance": importance
}).sort_values(by="Importance", ascending=False)
feature_importance_df
```

	Fitur	Importance
0	Luas_Tanah	1.0
1	Luas_Bangunan	0.0
2	Jumlah_Kamar	0.0
3	Jumlah_Kamar_Mandi	0.0
4	Jarak_Pusat_Kota	0.0
5	Umur_Bangunan	0.0

4. Visualisasi Feature Importance

```
import matplotlib.pyplot as plt

plt.figure(figsize=(8,5))
plt.barh(
    feature_importance_df["Fitur"],
    feature_importance_df["Importance"]
)
plt.xlabel("Nilai Importance")
plt.title("Feature Importance - Decision Tree Regresi")
plt.gca().invert_yaxis()
plt.show()
```



ANALISIS KONTRIBUSI FITUR (LOKAL)

5. Menelusuri Jalur Keputusan untuk 1 Data

```
from sklearn.tree import export_text


tree_rules = export_text(
   dt_model,
   feature_names=list(X.columns)
)
print(tree_rules)
```

```
|--- Luas_Tanah <= 160.00
|   |--- Luas_Tanah <= 125.00
|   |   |--- value: [626.67]
|   |--- Luas_Tanah >  125.00
|   |   |--- value: [1125.00]
|--- Luas_Tanah >  160.00
|   |--- Luas_Tanah <= 205.00
|   |   |--- value: [1650.00]
|   |--- Luas_Tanah >  205.00
|   |   |--- value: [2075.00]
```

6. Analisis Kontribusi Fitur untuk Satu Sampel

```
sample = X_test.iloc[[0]]
```

```
prediction = dt_model.predict(sample)

print("Data Uji:")
print(sample)
print("\nPrediksi Harga Rumah:", prediction[0])
```

```
Data Uji:
   Luas_Tanah  Luas_Bangunan  Jumlah_Kamar  Jumlah_Kamar_Mandi  \
9         160            130             4                   3

   Jarak_Pusat_Kota  Umur_Bangunan
9                 4              7

Prediksi Harga Rumah: 1125.0
```

FEATURE SELECTION SEDERHANA

7. Seleksi Fitur Berdasarkan Importance

```
selected_features = feature_importance_df[
    feature_importance_df["Importance"] > 0.05
]["Fitur"].tolist()

selected_features
```

8. Latih Ulang Model dengan Fitur Terpilih

```
X_selected = X[selected_features]

X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(
    X_selected, y, test_size=0.3, random_state=42
)

dt_selected = DecisionTreeRegressor(
    max_depth=4,
    min_samples_leaf=2,
    random_state=42
```

```
)

dt_selected.fit(X_train_s, y_train_s)

print("Akurasi model dengan fitur terpilih:")
print("R²:", dt_selected.score(X_test_s, y_test_s))
```

```
Akurasi model dengan fitur terpilih:
R²: 0.45952079375131927
```

6.5 Perbandingan Performa Random Forest dengan Decision Tree

Decision Tree dan Random Forest merupakan dua algoritma pembelajaran mesin yang sama-sama berbasis struktur pohon, namun memiliki karakteristik performa yang berbeda secara signifikan. Decision Tree bekerja sebagai model tunggal yang membangun satu struktur pohon keputusan, sedangkan Random Forest merupakan metode ensemble yang menggabungkan banyak pohon keputusan untuk menghasilkan satu prediksi akhir. Dari sisi akurasi, Random Forest umumnya menunjukkan performa yang lebih tinggi dibandingkan Decision Tree. Hal ini disebabkan oleh mekanisme agregasi prediksi dari banyak pohon yang mampu mengurangi kesalahan individual. Decision Tree tunggal sering kali sangat akurat pada data latih, tetapi performanya dapat menurun drastis pada data uji. Dari sisi akurasi, Random Forest umumnya menunjukkan performa yang lebih tinggi dibandingkan Decision Tree. Hal ini disebabkan oleh mekanisme agregasi prediksi dari banyak pohon yang mampu mengurangi kesalahan individual. Decision Tree tunggal sering kali sangat akurat pada data latih, tetapi performanya dapat menurun drastis pada data uji. Dalam konteks bias variance trade off, Decision Tree biasanya memiliki bias rendah tetapi variansi tinggi. Sebaliknya, Random Forest dirancang untuk menurunkan variansi tanpa

meningkatkan bias secara signifikan. Penurunan variansi inilah yang membuat Random Forest memiliki kemampuan generalisasi yang lebih baik.

Performa Random Forest juga unggul dalam menghadapi noise dan outlier. Pada Decision Tree, satu atau dua data ekstrem dapat memengaruhi struktur pohon secara signifikan. Pada Random Forest, pengaruh noise cenderung terlokalisasi pada beberapa pohon saja dan berkurang saat prediksi digabungkan. Dari sisi stabilitas model, Decision Tree sangat sensitif terhadap perubahan kecil pada data latih. Perubahan sedikit saja dapat menghasilkan struktur pohon yang sangat berbeda. Random Forest jauh lebih stabil karena struktur prediksi tidak bergantung pada satu pohon saja, melainkan pada koleksi pohon. Dalam hal kompleksitas komputasi, Decision Tree lebih ringan dan cepat untuk dilatih maupun digunakan untuk prediksi. Random Forest membutuhkan waktu dan sumber daya komputasi yang lebih besar karena harus membangun dan mengevaluasi banyak pohon, meskipun proses ini dapat dilakukan secara paralel. Perbandingan performa juga terlihat pada skalabilitas. Decision Tree dapat bekerja dengan baik pada dataset kecil hingga menengah, tetapi sering mengalami degradasi performa pada data yang sangat kompleks. Random Forest lebih mampu menangani dataset besar dan berdimensi tinggi dengan performa yang konsisten.

Dari sudut pandang interpretabilitas, Decision Tree unggul karena struktur pohonnya dapat divisualisasikan dan diterjemahkan langsung ke dalam aturan if–then. Random Forest memiliki interpretabilitas yang lebih rendah karena keputusan akhir merupakan hasil agregasi banyak pohon, meskipun masih menyediakan informasi seperti feature importance. Dalam praktik evaluasi performa, Random Forest hampir selalu menghasilkan nilai metrik evaluasi yang lebih baik, seperti akurasi, F1-score, R^2, atau error yang lebih rendah dibandingkan Decision Tree

tunggal. Hal ini menjadikan Random Forest pilihan utama dalam banyak aplikasi dunia nyata. Namun, keunggulan performa Random Forest tidak selalu berarti Decision Tree menjadi tidak relevan. Decision Tree sering digunakan sebagai baseline model atau sebagai alat eksplorasi awal untuk memahami struktur data dan hubungan antar variabel sebelum menggunakan model yang lebih kompleks. Perbedaan performa juga terlihat pada kemampuan menangani fitur yang berkorelasi. Decision Tree cenderung memilih satu fitur dominan dan mengabaikan fitur korelatif lainnya. Random Forest, melalui pemilihan fitur acak, mampu memanfaatkan fitur-fitur korelatif secara lebih seimbang.

Dalam konteks ketahanan terhadap data tidak seimbang, Random Forest umumnya lebih unggul karena agregasi banyak pohon dapat mengurangi bias terhadap kelas mayoritas. Decision Tree lebih rentan memihak kelas dominan jika tidak dilakukan penyesuaian tambahan. Dari sisi general purpose modeling, Random Forest lebih fleksibel karena dapat digunakan untuk klasifikasi dan regresi dengan performa yang stabil pada berbagai domain. Decision Tree lebih sering digunakan ketika interpretasi dan kesederhanaan menjadi prioritas utama. Random Forest juga lebih konsisten dalam cross validation, di mana variasi performa antar fold cenderung lebih kecil dibandingkan Decision Tree. Hal ini menunjukkan bahwa Random Forest memiliki kestabilan performa yang lebih baik terhadap variasi data. Dalam pengembangan sistem skala industri, Random Forest lebih sering dipilih karena robust, akurat, dan minim tuning ekstrem. Decision Tree jarang digunakan sebagai model akhir pada sistem produksi tanpa pengaturan dan pengawasan yang ketat. Namun demikian, Random Forest memiliki keterbatasan berupa biaya komputasi dan interpretasi. Dalam situasi dengan keterbatasan sumber daya atau kebutuhan transparansi tinggi, Decision Tree masih menjadi pilihan yang lebih tepat meskipun performanya lebih rendah. Secara metodologis,

Decision Tree sering berperan sebagai building block bagi Random Forest. Dengan kata lain, Random Forest tidak menggantikan Decision Tree, tetapi mengembangkannya ke tingkat yang lebih tinggi melalui pendekatan ensemble.

DAFTAR PUSTAKA

[1] N. Rofiq and S. L. M. Sitio, *Pengenalan Dasar Analisis Data dengan Python di Google Colab*. Eureka Media Aksara, 2024.

[2] S. C. Matz, C. S. Bukow, H. Peters, C. Deacons, A. Dinu, and C. Stachl, "Using machine learning to predict student retention from socio-demographic characteristics and app-based engagement metrics," *Sci. Rep.*, vol. 13, no. 1, p. 5705, 2023, doi: 10.1038/s41598-023-32484-w.

[3] A. Alagic *et al.*, "Machine Learning for an Enhanced Credit Risk Analysis: A Comparative Study of Loan Approval Prediction Models Integrating Mental Health Data," *Mach. Learn. Knowl. Extr.*, vol. 6, no. 1, 2024, doi: 10.3390/make6010004.

[4] V. G. Costa and C. E. Pedreira, "Recent advances in decision trees: an updated survey," *Artif. Intell. Rev.*, vol. 56, no. 5, pp. 4765–4800, 2023, doi: 10.1007/s10462-022-10275-5.

[5] H. Dong, R. Liu, and A. W. Tham, "Accuracy Comparison between Five Machine Learning Algorithms for Financial Risk Evaluation," *J. Risk Financ. Manag.*, vol. 17, no. 2, 2024, doi: 10.3390/jrfm17020050.

[6] A. Ahmed *et al.*, "Students ' performance prediction employing Decision Tree," vol. 16, pp. 42–51, 2024, doi: 10.22144/ctujoisd.2024.321.

[7] S. L. M. Sitio *et al.*, "Comparison of the Ensemble Xgboost and Transformer Models With Machine Learning for Classification of Indonesian Music Moods of the 70'S and 80'S Era," *J. Theor. Appl. Inf. Technol.*, vol. 102, no. 24, pp. 9157–9165, 2024.

[8] S. Lina, M. Sitio, and N. Rofiq, "Classification of Creditworthy Customer Using Support Vector Machine Algorithm," vol. 10, no.

2, pp. 339–345, 2025, doi: 10.31572/inotera.Vol10.Iss2.2025.ID502.

[9] D. H. Depari *et al.*, "Perbandingan Model Decision Tree , Naive Bayes dan Random Forest untuk Prediksi Klasifikasi Penyakit Jantung," vol. 4221, pp. 239–248, 2022.

[10] R. F. Ramadhan and W. M. Ashari, "Performance Comparison of Random Forest and Decision Tree Algorithms for Anomaly Detection in Networks," vol. 8, no. 2, pp. 367–375, 2024.

[11] P. A. Firnanda, "Analisis Perbandingan Decision Tree dan Random Forest dalam Klasifikasi Penjualan Produk pada Supermarket," vol. 3, no. 1, pp. 445–461, 2025.

[12] M. Ikermane, M. Mohy-eddine, and Y. Rachidi, "Credit Card Fraud Detection: Comparing Random Forest and XGBoost Models with Explainable AI Interpretations BT - Innovative Technologies on Electrical Power Systems for Smart Cities Infrastructure," I. Aboudrar, F. Ilahi Bakhsh, A. Nayyar, and I. Ouachtouk, Eds., Cham: Springer Nature Switzerland, 2025, pp. 126–135.

[13] H. Sun, Y. Jiang, and X. Tang, "Artificial Intelligence-Driven Academic Performance Early Warning System for Engineering Universities: Application of XGBoost-SHAP Algorithm," *Int. J. High Speed Electron. Syst.*, vol. 0, no. 0, p. 2540536, doi: 10.1142/S0129156425405364.

[14] X. Wu, L. Xiao, Y. Sun, J. Zhang, T. Ma, and L. He, "A survey of human-in-the-loop for machine learning," *Futur. Gener. Comput. Syst.*, vol. 135, pp. 364–381, 2022, doi: https://doi.org/10.1016/j.future.2022.05.014.

[15] H. Putra, K. Nasution, E. Rilvani, U. P. Bangsa, and K. Bekasi, "PREDIKSI KELULUSAN MAHASISWA TEPAT WAKTU MENGGUNAKAN DECISION TREE : STUDI PERBANDINGAN PREDIKSI KELULUSAN MAHASISWA TEPAT WAKTU

MENGGUNAKAN DECISION TREE : STUDI PERBANDINGAN," vol. 3, no. 7, 2025.

[16] S. Sharma and P. Chaudhary, "Machine learning and deep learning," *Quantum Comput. Artif. Intell. Train. Mach. Deep Learn. Algorithms Quantum Comput.*, pp. 71–84, 2023, doi: 10.1515/9783110791402-004.

[17] K. Di *et al.*, "Penerapan Algoritma Random Forest Untuk Analisis Sentimen," *J. Nas. Komputasi dan Teknol. Inf.*, vol. 5, no. 1, 2022.

[18] S. Fatima, A. Hussain, S. Bin Amir, S. H. Ahmed, and S. M. H. Aslam, "XGBoost and Random Forest Algorithms: An in Depth Analysis," *Pakistan J. Sci. Res.*, vol. 3, no. 1, 2023, doi: 10.57041/pjosr.v3i1.946.

[19] Y. Lou and K. F. Colvin, "Performance prediction using educational data mining techniques: a comparative study," *Discov. Educ.*, vol. 4, no. 1, p. 112, 2025, doi: 10.1007/s44217-025-00502-w.

[20] F. Alshareef, H. Alhakami, T. Alsubait, and A. Baz, "Educational data mining applications and techniques," *Int. J. Adv. Comput. Sci. Appl.*, vol. 11, no. 4, 2020, doi: 10.14569/IJACSA.2020.0110494.

TENTANG PENULIS

Sartika Lina Mulani Sitio, S.Kom.,M.Kom, Lahir di Sipolha, 24 Mei 1987. Saya menempuh Magister Komputer konsentrasi Software Engineering di STMIK Eresha tahun 2016. Sejak Lulus tahun 2016 saya menjadi dosen tetap di Prodi Teknik Informatika Universitas Pamulang. Penelitian yang sudah saya buat sampai saat ini berfokus pada sistem informasi, software engineering dan data science