KOMPUTA : Jurnal Ilmiah Komputer dan Informatika
Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

RANGKUMAN MATERI VIDEO MACHINE LEARNING KELOMPOK 16

Muhammad Akmal Syarif[1], Rafly Ariel Hidayat[2]

[1,2] Program Studi Sistem Informasi, STMIK Tazkia Bogor
E-mail : 241572010004.akmal@student.stmik.tazkia.ac.id
241572010019.rafly@student.stmik.tazkia.ac.id

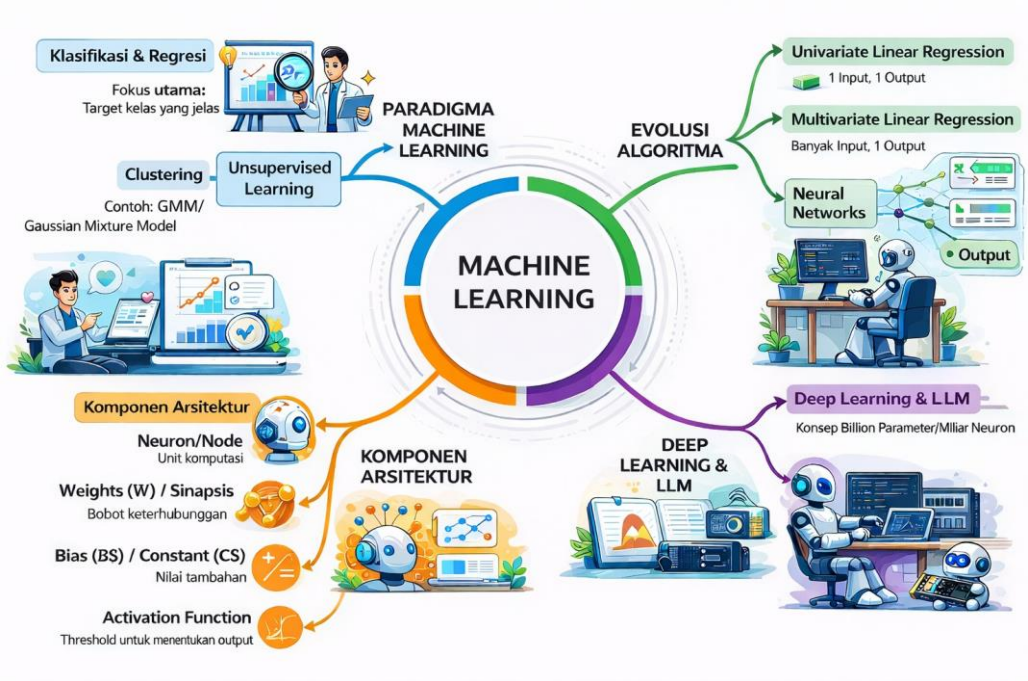
BAB I
PENGENALAN DAN KONSEP FUNDAMENTAL MACHINE LEARNING

1. Deskripsi Singkat Bahasan

Pertemuan ini membahas dasar-dasar Machine Learning (ML) sebagai bagian dari ekosistem Artificial Intelligence (AI). Fokus utama materi adalah memahami definisi formal pembelajaran mesin melalui kerangka kerja Tom Mitchell, hubungan antara data dan pengetahuan, serta prospek karier di bidang AI. Mahasiswa ditekankan untuk menguasai konsep fundamental sebelum masuk ke implementasi praktis, mengingat tren teknologi masa depan akan sangat bergantung pada integrasi AI dengan Cybersecurity dan Blockchain.

2. Mindmap / Taksonomi

Berikut adalah pemetaan konsep dasar yang dibahas:



3. Penjelasan Detail

Definisi operasional *Machine Learning* yang disepakati merujuk pada buku *Introduction to Machine Learning* karya **Tom Mitchell**. Sebuah program komputer dikatakan belajar dari pengalaman jika performanya dalam menjalankan tugas tertentu meningkat seiring dengan bertambahnya pengalaman.
Secara matematis, hubungan ini dapat didefinisikan sebagai berikut:

Definisi Pembelajaran:

Naskah dikirim: dd-mm-yyyy; direvisi dd-mm-yyyy; diterima: dd-mm-yyyy

Sebuah program komputer dikatakan belajar dari pengalaman *E* terhadap sekumpulan tugas *T* dan pengukuran performa *P*, jika performanya pada tugas-tugas di *T*, sebagaimana diukur oleh *P*, meningkat dengan pengalaman *E*.

Dalam konteks teknis, komponen-komponen tersebut dijabarkan sebagai berikut:

- **Tugas (Task – *T*):** Merupakan unit pekerjaan yang harus diselesaikan oleh sistem, misalnya melakukan klasifikasi nilai ujian atau prediksi harga pasar.
- **Performa (Performance - *P*):** Variabel pengukuran keberhasilan sistem. Jika sistem mendapatkan skor rendah, maka diperlukan perbaikan melalui umpan balik data.
- **Pengalaman (Experience - *E*):** Dalam dunia komputasi, pengalaman direpresentasikan sebagai **Data**.

Hubungan Data, Fakta, dan Pengetahuan:

Data didefinisikan sebagai Fakta yang telah terjadi. Akumulasi data yang konsisten akan membentuk sebuah Hipotesis atau Model. Pengetahuan yang dihasilkan harus memiliki sifat konsisten terhadap data asal. Jika terdapat kesalahan dalam model, hal tersebut sering kali disebabkan oleh adanya noise (gangguan) dalam data yang bukan merupakan fakta sebenarnya.

Secara formal, peningkatan kemampuan mesin melalui proses iteratif dapat dimodelkan sebagai fungsi optimasi performa:

$$P_{t+1} > P_t \Leftrightarrow E_{t+1} \text{ memberikan informasi baru yang valid bagi } T$$

$$Di\ mana\ P\ adalah\ nilai\ performa\ pada\ waktu\ t. Jika\ tidak\ ada\ peningkatan\ performa$$

$$(Pt + 1 \leq Pt), makamesintersebutbelumdikatakan \text{belajar.}$$

4. Sample Code

Meskipun pertemuan pertama berfokus pada teori, berikut adalah contoh kode sederhana menggunakan pustaka scikit-learn untuk mendemonstrasikan bagaimana sebuah mesin "belajar" memprediksi nilai (Task *T*) berdasarkan data historis (Experience *E*) dan diukur akurasinya (Performance *P*).

Python

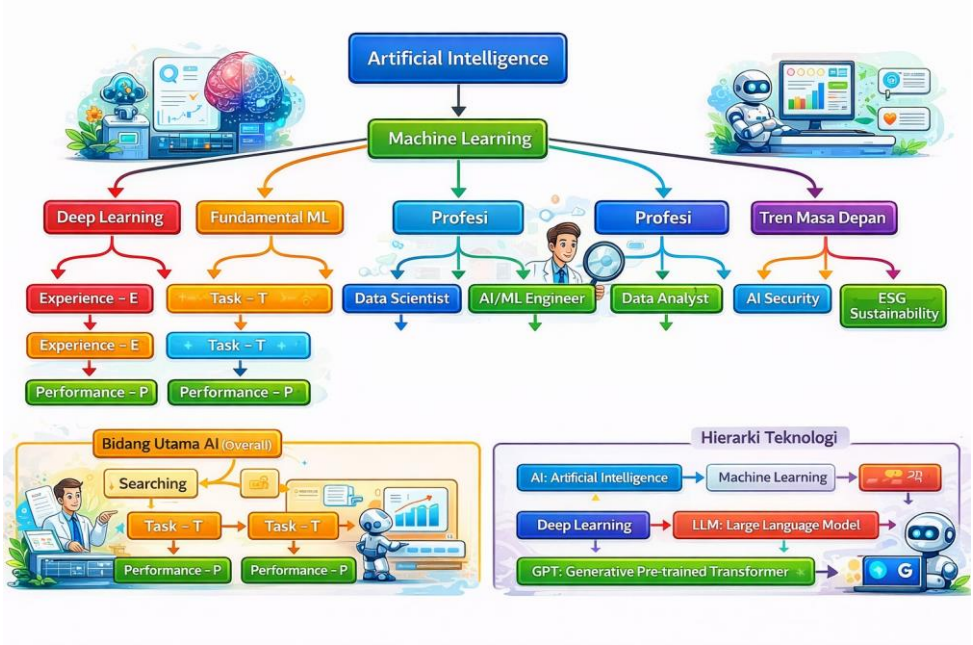
```
from sklearn.linear_model import LinearRegression
import numpy as np
# 1. Experience (E): Data berupa jam belajar (X) dan Nilai Ujian (y)
# Data adalah fakta yang sudah terjadi
X_jam_belajar = np.array([[1], [2], [3], [4], [5]])
y_nilai_ujian = np.array([30, 45, 60, 80, 95])
# 2. Task (T): Membuat model untuk memprediksi nilai berdasarkan jam
belajar
model = LinearRegression()
# Proses "Learning" - Mencari pola yang konsisten terhadap data
model.fit(X_jam_belajar, y_nilai_ujian)
# 3. Pengujian Tugas
jam_baru = np.array([[6]])
prediksi = model.predict(jam_baru)
# 4. Performance (P): Melihat seberapa akurat model (R-squared score)
performa = model.score(X_jam_belajar, y_nilai_ujian)
print(f"Hasil Prediksi untuk 6 jam belajar: {prediksi[0]:.2f}")
print(f"Skor Performa Model (P): {performa:.4f}")
```

BAB II
TAKSONOMI ARTIFICIAL INTELLIGENCE DAN IMPLEMENTASI PADA SUSTAINABILITY

1. Deskripsi Singkat Bahasan

Bab ini mendalami struktur hierarki *Artificial Intelligence* (AI) mulai dari konsep payung besar hingga produk aplikatif seperti Chat GPT. Selain teori, bab ini mengeksplorasi peran vital AI dalam industri *Environmental, Social, and Governance* (ESG), khususnya pada proyek karbon dan reforestasi. Mahasiswa diajak memahami bahwa AI bukan sekadar kode, melainkan alat validasi data citra satelit dan pemenuhan regulasi (*compliance*) global untuk mengatasi krisis iklim.

2. Mindmap / Taksonomi



3. Penjelasan Detail

A. Pembagian Bidang AI (Pendekatan Stuart Norvig)

Dalam buku *AI: A Modern Approach*, kecerdasan buatan dibagi menjadi empat pilar utama:

1. **Searching:** Algoritma pencarian solusi dalam ruang masalah (contoh: Google Search).
2. **Planning:** Penentuan urutan tindakan untuk mencapai tujuan.
3. **Reasoning:** Pengambilan keputusan berdasarkan logika dan pengetahuan yang ada.
4. **Learning:** Kemampuan sistem untuk memperbaiki diri dari pengalaman.

Pada pilar **Learning**, terdapat perbedaan mendasar antara metode **Induktif** dan **Deduktif**:

- **Pembelajaran Induktif (Machine Learning):** Mesin mempelajari pola dari data spesifik untuk menghasilkan kesimpulan umum.
- **Pembelajaran Deduktif (Expert System):** Mesin bekerja berdasarkan aturan (*rules*) yang telah didefinisikan secara kaku oleh pakar manusia.

Secara formal dalam logika predikat, sistem deduktif mengikuti kaidah:

$$P \rightarrow Q, \quad P \vdash Q$$

Jika aturan $P \rightarrow Q$ (Jika $P\ maka\ Q$) tersedia dan fakta P benar, maka sistem menyimpulkan Q. Namun, Machine Learning (Induktif) bekerja secara probabilistik:

$$P(Q|P) = \frac{P(P|Q)P(Q)}{P(P)}$$

Menggunakan Teorema Bayes untuk menghitung probabilitas kesimpulan Q diberikan fakta P.

B. Studi Kasus: AI untuk Validasi Karbon dan Reforestasi

Implementasi AI saat ini sangat krusial dalam memvalidasi klaim karbon perusahaan. Perusahaan melakukan reforestasi (penghutanan kembali) untuk mendapatkan sertifikat karbon guna menghindari pajak karbon (*carbon tax*).

AI berperan dalam dua aspek teknis utama:

1. **Computer Vision (Image Recognition):** Melakukan validasi jenis pohon melalui foto yang diunggah. AI memastikan bahwa pohon yang ditanam sesuai dengan genusnya (misalnya membedakan pohon durian dengan mangga).
2. **Analisis Citra Satelit:** Memantau tutupan lahan (*land cover*) untuk memastikan area tersebut bukan hutan lindung milik pemerintah atau lahan rawa (*wetland*) yang memiliki regulasi berbeda.

C. Compliance dan Regulasi (Kepatuhan)

Dunia IT tidak hanya berurusan dengan teknis, tetapi juga kepatuhan hukum (*Compliance*). Beberapa standar yang harus dipahami oleh *AI Engineer* meliputi:

- **Nasional:** UU PDP (Perlindungan Data Pribadi), Indeks KAMI, dan SRN (Sistem Registrasi Nasional) untuk klaim karbon.
- **Internasional:** ISO 27001 (Keamanan Informasi), GRI (*Global Reporting Initiative*), dan Verra (Standar Kredit Karbon Global)

4. Sample Code: Klasifikasi Jenis Pohon Sederhana

Contoh kode berikut mensimulasikan logika dasar *Image Classification* menggunakan pendekatan *Deep Learning* (Keras/TensorFlow) untuk membedakan jenis vegetasi berdasarkan fitur citra (disederhanakan).

Python

```python
import tensorflow as tf
from tensorflow.keras import layers, models

# Membangun Arsitektur CNN (Convolutional Neural Network) Sederhana
def build_tree_classifier():
    model = models.Sequential([
        # Layer Konvolusi untuk ekstraksi fitur citra pohon
        layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150,
150, 3)),
        layers.MaxPooling2D((2, 2)),
        layers.Conv2D(64, (3, 3), activation='relu'),
        layers.MaxPooling2D((2, 2)),
        # Flattening untuk masuk ke Fully Connected Layer
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        # Output layer: 2 kelas (Contoh: 0 = Durian, 1 = Mangga)
        layers.Dense(2, activation='softmax')
    ])

    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])
    return model

# Inisialisasi Model
tree_model = build_tree_classifier()

# Penjelasan Performa (P):
# Dalam klasifikasi, performa diukur dengan Akurasi (Accuracy)
# Accuracy = (TP + TN) / (TP + TN + FP + FN)
print("Model AI untuk Validasi Reforestasi siap dikonfigurasi.")
tree_model.summary()
```

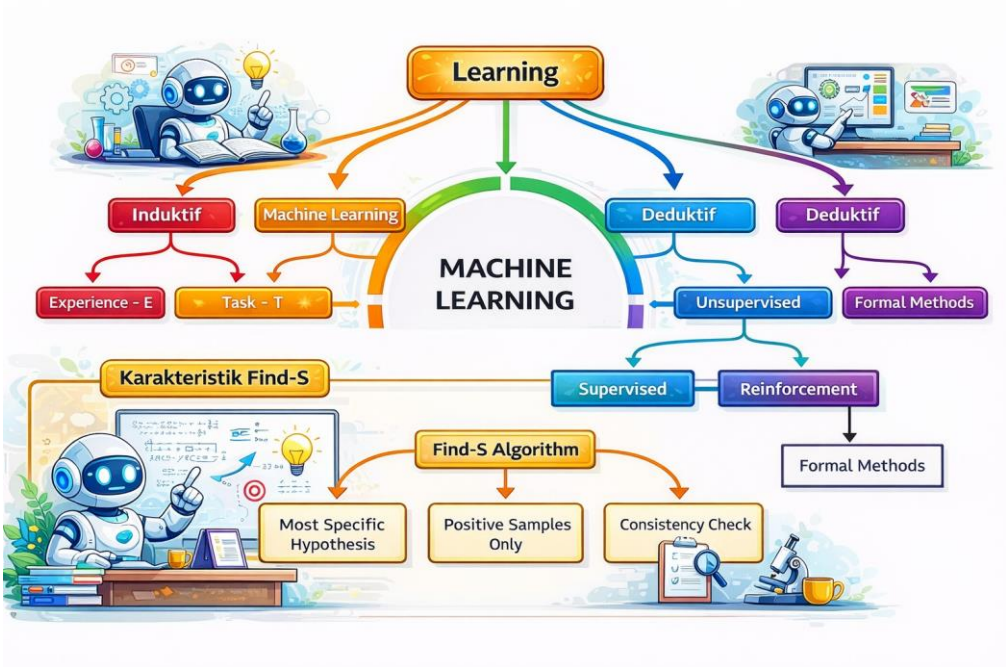
BAB III
PEMBELAJARAN INDUKTIF DAN ALGORITMA FIND-S

1. Deskripsi Singkat Bahasan

Bab ini membahas dikotomi antara pembelajaran induktif dan deduktif, serta bagaimana *Machine Learning* secara spesifik mengadopsi pendekatan induktif untuk membangun pengetahuan dari data spesifik menuju kesimpulan umum. Fokus utama diberikan pada formalisasi fungsi pendekatan $f'(x)$ untuk mengestimasi pengetahuan pakar yang sulit diformalkan, serta pengenalan algoritma **Find-S** sebagai metode dasar dalam mencari hipotesis paling spesifik dalam sebuah ruang pencarian.

1. Mindmap / Taksonomi

Struktur konsep pembelajaran dan kategorisasi algoritma yang dibahas:



2. Penjelasan Detail

A. Pembelajaran Induktif vs. Deduktif

Perbedaan mendasar antara kedua metode ini terletak pada arah aliran logikanya:

- **Pembelajaran Induktif:** Bergerak dari kasus-kasus khusus (data/observasi) menuju aturan umum. *Machine Learning* bersifat induktif karena menggunakan data historis untuk membentuk model prediksi.

- **Pembelajaran Deduktif:** Bergerak dari aturan umum menuju kasus khusus. Contohnya adalah *Expert System* di mana aturan logis sudah ditanamkan sejak awal oleh pakar.

B. Formalisasi Fungsi Pengetahuan

Seringkali, seorang pakar (misal: Dokter) memiliki pengetahuan $f(x)$ namun sulit menyatakannya dalam kode program secara eksplisit. Tugas seorang AI Engineer adalah membuat fungsi pendekatan $f'(x)$ melalui proses *training*.

Ketidakpastian dalam model ini diukur sebagai selisih (error) antara prediksi dan kenyataan:

$$f'(x) \approx f(x)$$

$$Error = \sum_{i=1}^{n} \left(y_i - f'(x_i)\right)^2$$

Pengetahuan yang dihasilkan $(knowledge)$ haruslah bersifat **konsisten terhadap data**. Jika terdapat data baru yang kontradiktif, maka hipotesis H harus diperbarui agar tetap valid.

C. Konsep Algoritma Find-S

Algoritma Find-S bekerja dengan mencari hipotesis yang paling spesifik yang konsisten dengan contoh pelatihan positif.

1. $Inisialisasi\ h\ ke\ hipotesis\ paling\ spesifik\ dalam\ H: h \leftarrow \langle \emptyset, \emptyset, \ldots, \emptyset \rangle$
2. $Untuk\ setiap\ contoh\ pelatihan\ positif\ x$:
    - $Jika\ fitur\ dalam\ x\ tidak\ konsisten\ dengan\ h, perbarui\ h\ menjadi\ lebih\ umum.$
    - $Jikanilaifiturberbeda, gantidengansimboltandatanya(?).$
3. Abaikan contoh negatif.

Simbol dalam ruang hipotesis:

- **∅ :** Tidak menerima nilai apapun (paling spesifik).
- **Value (misal: "Sunny") :** Hanya menerima nilai tersebut.
- **? :** Menerima nilai apapun (paling umum/general).

4. Sample Code: Implementasi Find-S (Dataset Enjoy Sport)

Kode di bawah ini mendemonstrasikan bagaimana algoritma Find-S memproses data langkah demi langkah untuk menghasilkan hipotesis akhir.

Python

```
import numpy as np

# Dataset: Sky, Temp, Humidity, Wind, Water, Forecast, EnjoySport
(Label)
data = np.array([
['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'],
['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'],
['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'],
['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']
])

def train_find_s(concepts, target):
# Inisialisasi hipotesis dengan baris pertama data positif
specific_h = concepts[0].copy()

for i, val in enumerate(concepts):
if target[i] == "Yes":
for x in range(len(specific_h)):
# Jika nilai berbeda, ubah menjadi '?' (Generalize)
if val[x] != specific_h[x]:
specific_h[x] = '?'
# Contoh negatif diabaikan dalam Find-S
return specific_h
```

```
# Memisahkan fitur dan target
features = data[:, :-1]
labels = data[:, -1]

final_hypothesis = train_find_s(features, labels)

print("Hipotesis Akhir (Find-S):")
print(final_hypothesis)
# Output yang diharapkan: ['Sunny' 'Warm' '?' 'Strong' '?' '?']
```

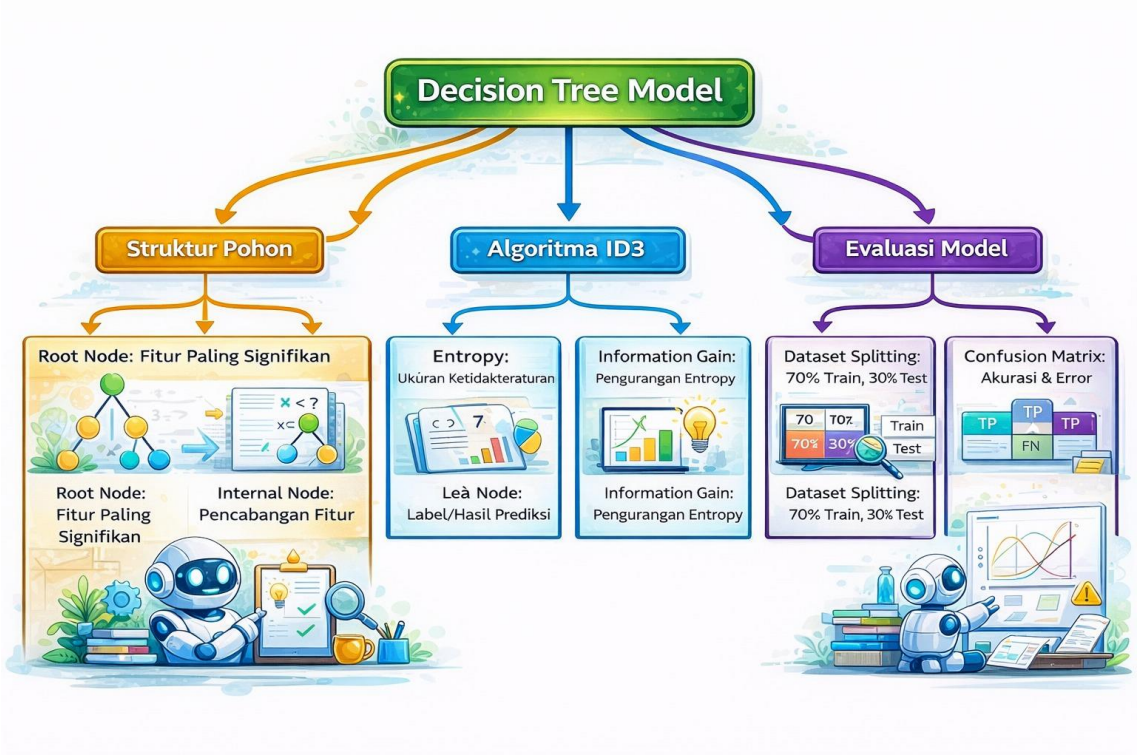
BAB 1V
DECISION TREE DAN ALGORITMA ID3

1.Deskripsi Singkat Bahasan

Bab ini berfokus pada transisi dari algoritma berbasis hipotesis linear (seperti Find-S) menuju struktur model berbasis graf, yaitu **Decision Tree** (Pohon Keputusan). Pembahasan mencakup logika pembentukan pohon menggunakan algoritma **ID3**, peran statistik dalam menentukan signifikansi fitur melalui *Information Gain* dan *Entropy*, serta evaluasi model untuk menghindari kondisi *underfitting* dan *overfitting* dalam skala industri.

2. Mindmap / Taksonomi

Struktur komponen dalam Decision Tree dan alur evaluasi model:



3. Penjelasan Detail

A. Konsep Model vs. Metode

Dalam *Machine Learning*, penting untuk membedakan antara metode dan model:

- **Metode/Algoritma:** Cara atau langkah-langkah untuk membentuk model (contoh: ID3, C45, Backpropagation).
- **Model:** Produk akhir atau output berupa pengetahuan $\left(f'(x)\right)$ yang siap digunakan untuk prediksi (contoh: Struktur pohon, bobot neural network).

B. Formalisasi Entropy dan Information Gain

Algoritma ID3 menentukan posisi *Root* dan *Node* berdasarkan nilai signifikansi fitur. Fitur yang pa.ling mampu memisahkan data ke dalam kelas yang berbeda dipilih berdasarkan nilai **Information Gain** tertinggi

1. **Entropy (S):** Mengukur tingkat ketidakteraturan atau ambiguitas dalam dataset.

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

$$Di\ mana\ pi\ adalah\ proporsi\ sampel\ dalam\ kelas\ i.$$

2. **Information Gain (S, A):** Pengurangan nilai Entropy setelah dataset dibagi berdasarkan atribut A.

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

C. Validasi Model: Confusion Matrix

Untuk memastikan model tidak terlalu longgar (*underfit*) atau terlalu kaku (*overfit*), dilakukan pengujian menggunakan data *testing*. Hasilnya dipetakan ke dalam **Confusion Matrix** untuk menghitung akurasi, sensitivitas, dan spesifisitas model dalam skenario bisnis (misal: deteksi *fraud*).

4. Sample Code: Implementasi ID3 dengan Scikit-Learn

Meskipun ID3 dapat dihitung manual, dalam praktiknya kita menggunakan pustaka seperti scikit-learn yang mengimplementasikan varian pohon keputusan yang efisien (CART).

Python

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, accuracy_score

# 1. Load Dataset (Contoh: Play Tennis)
# Asumsi data sudah di-encode menjadi numerik
data = {
'Outlook': [0, 0, 1, 2, 2, 2, 1, 0, 0, 2, 0, 1, 1, 2],
```

```
'Temp': [0, 0, 0, 1, 2, 2, 2, 1, 2, 1, 1, 1, 0, 1],
'Humidity': [0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0],
'Wind': [0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1],
'Play': [0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0]
}
df = pd.DataFrame(data)

# 2. Split Data (70% Train, 30% Test)
X = df.drop('Play', axis=1)
y = df['Play']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)

# 3. Training Model dengan kriteria 'entropy' (Logika ID3)
clf = DecisionTreeClassifier(criterion='entropy')
clf.fit(X_train, y_train)

# 4. Evaluasi Model
y_pred = clf.predict(X_test)
print(f"Akurasi Model: {accuracy_score(y_test, y_pred) * 100}%")
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Visualisasi Struktur Pohon dalam Teks
tree_rules = export_text(clf, feature_names=list(X.columns))
print("\nStruktur Pohon Keputusan:\n", tree_rules)
```

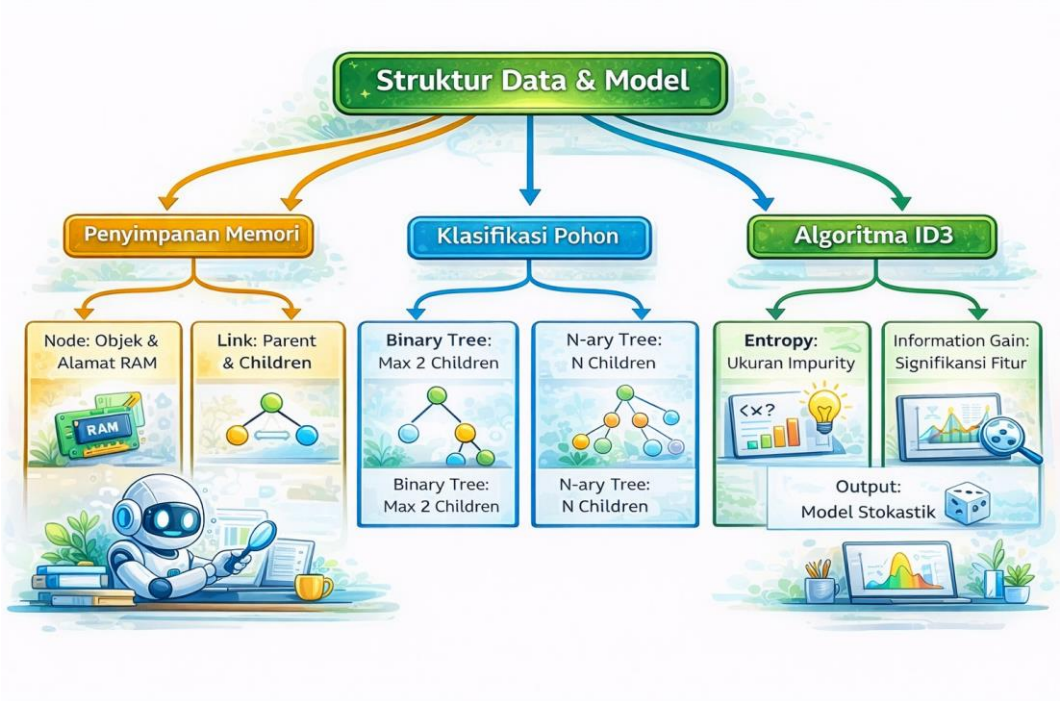
BAB V
IMPLEMENTASI STRUKTUR DATA TREE DAN ALGORITMA ID3

1. Deskripsi Singkat Bahasan

Bab ini mengeksplorasi cara mesin menyimpan dan mengolah model dalam bentuk **Pohon Keputusan (Decision Tree)**. Pembahasan mencakup aspek teknis penyimpanan di memori (RAM) menggunakan konsep *Node* dan *Pointer*, perbedaan antara *Binary Tree* dan *N-ary Tree*, serta mekanika algoritma **ID3** dalam menentukan *Root* melalui perhitungan statistik *Entropy* dan *Information Gain*. Fokus utama adalah bagaimana data diskrit dikonversi menjadi hierarki logika untuk prediksi stokastik.

2. Mindmap / Taxonomi

Struktur teknis dan alur logika yang dipelajari:



3. Penjelasan Detail

A. Penyimpanan Model di Memori

Komputer adalah alat hitung yang memproses data di CPU dan menyimpannya di RAM. Dalam struktur *Tree*, setiap **Node** disimpan dalam "kamar" memori dengan alamat heksadesimal tertentu. Sebuah *Node* minimal harus menyimpan tiga informasi utama:

1. **Data:** Nilai fitur atau label kelas.
2. **Parent Pointer:** Alamat memori induknya (untuk navigasi ke atas).
3. **Children List:** Daftar alamat memori anak-anaknya (untuk navigasi ke bawah).

B. Perhitungan Matematis ID3

Algoritma ID3 menggunakan pendekatan *Top-Down* untuk membangun pohon dengan meminimalkan ketidakteraturan data.

1. Entropy (S): Digunakan untuk mengukur tingkat ketidakmurnian dataset.

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

   - Jika $Entropy = 0$, data sudah murni (semua sampel memiliki label yang sama).
   - Jika $Entropy = 1$, data sangat bervariasi (terbagi rata antara *Yes* dan *No*).

2. Information Gain (S, A): Digunakan untuk memilih atribut A yang memberikan informasi paling signifikan.

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

C. Karakteristik Model

- **Root:** Fitur dengan *Gain* tertinggi (misal: *Outlook*).
- **Leaf:** Hasil akhir atau klasifikasi (misal: *Yes* atau *No*).
- **Stokastik:** Hasil prediksi pohon bersifat probabilitas berdasarkan data historis, bukan kepastian absolut (deterministik).

4. Sample Code: Struktur Node dan Kalkulasi Entropy

Berikut adalah representasi kode Python untuk mendefinisikan struktur *Tree* secara manual dan fungsi penghitung *Entropy*.

Python

```
import math

# 1. Definisi Struktur Node
class Node:
    def __init__(self, data=None):
        self.data = data
        self.parent = None
        self.children = []

    def add_child(self, child_node):
        child_node.parent = self
        self.children.append(child_node)

# 2. Fungsi Manual Perhitungan Entropy
def calculate_entropy(yes_count, no_count):
    total = yes_count + no_count
    if total == 0 or yes_count == 0 or no_count == 0:
        return 0

    p_yes = yes_count / total
    p_no = no_count / total

    entropy = -(p_yes * math.log2(p_yes)) - (p_no * math.log2(p_no))
    return entropy

# Contoh Penggunaan:
# Dataset 'Play Tennis' total: 9 Yes, 5 No
s_total_entropy = calculate_entropy(9, 5)
print(f"Entropy Total (S): {s_total_entropy:.4f}")

# Membangun Root Sederhana
root = Node("Outlook")
child_yes = Node("Yes")
root.add_child(child_yes)
```

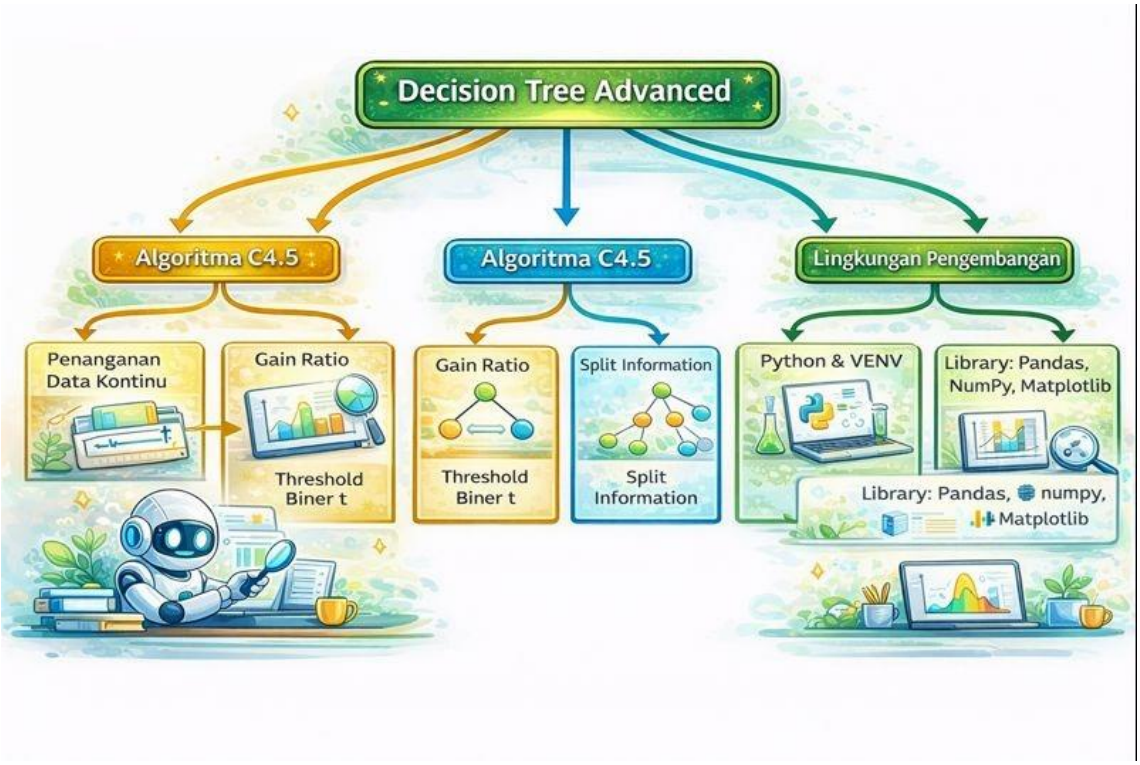
BAB VI
ALGORITMA C4.5 DAN LINGKUNGAN IMPLEMENTASI PYTHON

1. Deskripsi Singkat Bahasan

Bab ini membahas pengembangan dari algoritma ID3, yaitu **C4.5**, yang memiliki kemampuan untuk menangani fitur dengan data kontinu atau numerik. Selain itu, bab ini menjelaskan persiapan infrastruktur teknis yang diperlukan untuk pengembangan model *Machine Learning*, termasuk penggunaan *Virtual Environment* (VENV) untuk isolasi dependensi dan penggunaan *Jupyter Notebook* sebagai media eksperimen data menggunakan pustaka Pandas dan NumPy.

2. Mindmap / Taksonomi

Struktur pengembangan algoritma dan ekosistem implementasi:



3. Penjelasan Detail

A. Algoritma C4.5 dan Data Kontinu

Berbeda dengan ID3 yang hanya menangani data diskrit, algoritma C4.5 mampu memproses data numerik (kontinu) seperti pada *Iris Dataset*. Strategi yang digunakan adalah dengan menentukan nilai ambang biner (t) untuk membagi data menjadi dua kategori: $\leq t \; dan > \; t$.

B. Gain Ratio dan Split Information

C4.5 menggunakan **Gain Ratio** untuk menormalisasi *Information Gain* guna mencegah bias terhadap atribut yang memiliki banyak variasi nilai. Hal ini dicapai dengan menghitung **Split Information**.
Rumus matematis yang digunakan adalah:

1. Split Information (*S, A*):

$$SplitInfo(S,A) = -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \quad \frac{|S_i|}{|S|}$$

2. Gain Ratio (*S, A*):

$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitInfo(S,A)}$$

C. Isolasi Lingkungan Teknis (Virtual Environment)

Dalam implementasi *Engineering*, sangat disarankan untuk menggunakan *Virtual Environment* untuk mengisolasi pustaka proyek agar tidak terjadi konflik dengan sistem utama. Proses ini memastikan bahwa model dapat direproduksi dengan versi pustaka yang konsisten.

4. Sample Code: Eksperimen Data dengan Pandas

Berikut adalah contoh penggunaan pustaka Pandas dalam lingkungan *Jupyter Notebook* untuk memuat dataset dan melakukan komputasi awal pada fitur kontinu.

Python

```
import pandas as pd
import numpy as np

# 1. Memuat Dataset (Contoh: Iris Dataset)
# Asumsi file iris.csv tersedia di direktori lokal
try:
    df = pd.read_csv('iris.csv')
    print("Dataset Berhasil Dimuat:")
    print(df.head())
except FileNotFoundError:
    print("File tidak ditemukan. Pastikan dataset tersedia.")

# 2. Simulasi Perhitungan Perbandingan Fitur Kontinu
# Menghitung nilai tengah (threshold sederhana) untuk fitur
SepalLength
threshold = df['SepalLength'].mean()
print(f"\nNilai Ambang (Threshold) t: {threshold:.2f}")

# 3. Splitting Data berdasarkan Threshold t
df_low = df[df['SepalLength'] <= threshold]
df_high = df[df['SepalLength'] > threshold]

print(f"Jumlah sampel <= t: {len(df_low)}")
print(f"Jumlah sampel > t: {len(df_high)}")
```

Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

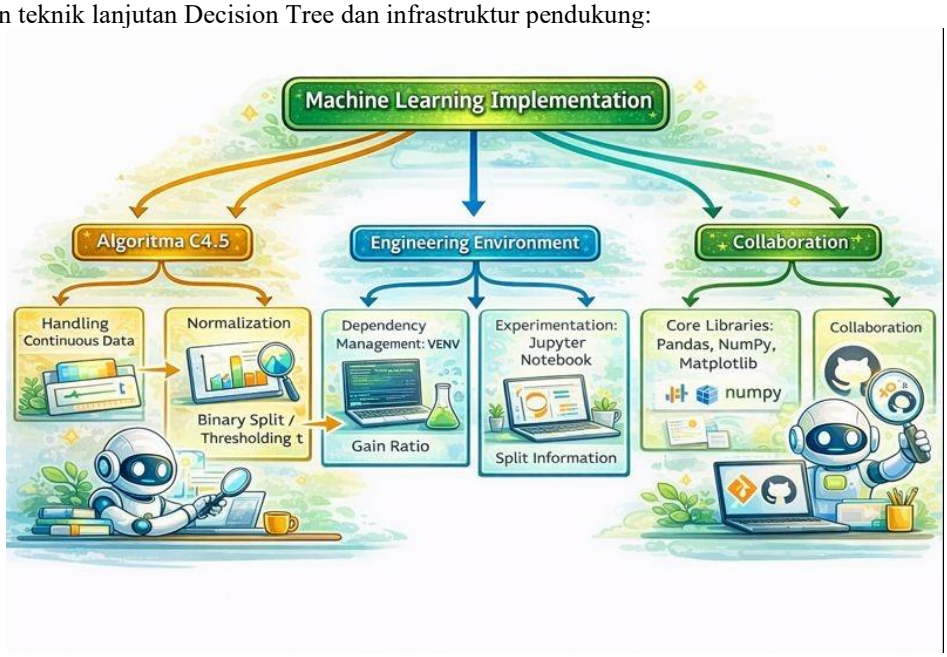
BAB VII
EKSTENSI ALGORITMA C4.5 DAN STANDARISASI LINGKUNGAN PENGEMBANGAN

1. Deskripsi Singkat Bahasan

Bab ini membahas keterbatasan algoritma ID3 dalam menangani data numerik dan solusi yang ditawarkan oleh algoritma **C4.5**. Pembahasan meliputi teknik penentuan ambang batas (*thresholding*) untuk data kontinu serta penggunaan *Gain Ratio* untuk meminimalisir bias pada fitur. Selain aspek algoritma, bab ini juga menekankan pada standarisasi teknis pengembangan perangkat lunak ML, seperti penggunaan *Virtual Environment* (VENV) untuk isolasi dependensi dan integrasi *Jupyter Notebook* sebagai media eksperimen.

2. Mindmap / Taxonomi

Pemetaan teknik lanjutan Decision Tree dan infrastruktur pendukung:



3. Penjelasan Detail

A. Penanganan Data Kontinu pada C4.5

Pada dataset seperti *Iris*, fitur bersifat numerik sehingga tidak bisa langsung dipetakan menjadi cabang. C4.5 melakukan diskritisasi dinamis dengan mencari nilai ambang t. Data kemudian dibagi menjadi dua partisi:

1. $S_1 = \{x \in S \mid \text{fitur}(x) \leq t\}$
2. $S_2 = \{x \in S \mid \text{fitur}(x) > t\}$

B. Gain Ratio dan Split Information

ID3 cenderung memilih fitur yang memiliki banyak nilai unik (seperti ID atau tanggal), yang sebenarnya tidak informatif. C4.5 memperbaiki ini dengan **Gain Ratio**, yang menormalisasi *Information Gain* menggunakan **Split Information**.

Secara matematis, rumusnya adalah:

- Split Information *(S, A)*:

$$SplitInfo(S,A) = -\sum_{i=1}^{n} \frac{|S_i|}{|S|} \log_2 \quad \frac{|S_i|}{|S|}$$

- Gain Ratio (*S*, *A*):

$$GainRatio(S,A) = \frac{Gain(S,A)}{SplitInfo(S,A)}$$

C. Best Practice: Virtual Environment (VENV)

Seorang *AI Engineer* wajib mengisolasi lingkungan Python miliknya. Tanpa VENV, instalasi pustaka akan bersifat global dan berisiko merusak sistem operasi atau menyebabkan konflik antar proyek (*dependency hell*). Isolasi ini menjamin bahwa model ML yang dikembangkan bersifat *reproducible* (dapat dijalankan ulang dengan hasil yang sama di mesin berbeda).

4. Sample Code: Implementasi Manual Entropy & Split (Python)

Berikut adalah simulasi fungsi penghitung Entropy menggunakan Pandas untuk memahami logika di balik pustaka otomatis.

Python

```python
import pandas as pd
import numpy as np

# 1. Menghitung Entropy secara manual menggunakan Pandas
def calculate_entropy(df, target_column):
    # Mendapatkan proporsi setiap label (Yes/No)
    counts = df[target_column].value_counts()
    probabilities = counts / len(df)

    # Rumus Entropy: -sum(pi * log2(pi))
    entropy = -np.sum(probabilities * np.log2(probabilities))
    return entropy

# 2. Contoh Penggunaan dengan data Dummy 'Play Tennis'
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rainy'],
    'Play': ['No', 'No', 'Yes', 'Yes']
}
df_tennis = pd.DataFrame(data)

# Hitung Entropy Total
s_entropy = calculate_entropy(df_tennis, 'Play')
print(f"Entropy Total Dataset: {s_entropy:.4f}")

# 3. Simulasi Split Data Kontinu (Logika C4.5)
# Misal membagi data berdasarkan ambang batas t
# df_filtered = df[df['PetalLength'] <= 2.45]
```

Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

BAB VIII
REGRESI LINEAR DAN OPTIMASI STOCHASTIC GRADIENT DESCENT (SGD)

1. Deskripsi Singkat Bahasan

Bab ini membahas konsep dasar **Regresi Linear** sebagai metode untuk merekam pengetahuan dalam bentuk pola hubungan antar variabel kontinu. Materi mencakup dua pendekatan utama dalam mencari solusi regresi: pendekatan **Analitik** (menggunakan rumus statistik formal *Least Squares*) dan pendekatan **Numerik** (menggunakan algoritma *Stochastic Gradient Descent*). Selain itu, dibahas pula teknis komputasi matriks menggunakan pustaka **NumPy** serta perbandingan paradigma pemrograman antara Java dan Python dalam konteks *AI Engineering*.

2. Mindmap / Taxonomi

Pemetaan metode dan komponen dalam Regresi Linear:

3. Penjelasan Detail

A. Persamaan Dasar Garis Lurus

Model regresi linear berusaha mencari garis terbaik (best fit line) yang meminimalkan error antar titik data. Persamaan dasarnya adalah:

$$y = mx + c$$

Dalam terminologi Machine Learning, sering dituliskan sebagai:

$$f(x) = w_1 x_1 + w_0$$

Di mana w_1 adalah bobot (weight) yang menentukan kemiringan, dan w_0 adalah bias.

B. Metode Analitik (Ordinary Least Squares)

Untuk data dengan fitur tunggal (single variate), nilai m dan c dapat dicari secara pasti menggunakan rumus:

$$m = \frac{n(\sum xy) - (\sum x)(\sum y)}{n(\sum x^2) - (\sum x)^2}$$

$$c = \frac{\sum y - m(\sum x)}{n}$$

Catatan: Terdapat perbedaan penting antara $\sum x^2$ (kuadrat nilai lalu dijumlahkan) dengan $(\sum x)^2$ (jumlah total lalu dikuadratkan).

C. Metode Numerik: Stochastic Gradient Descent (SGD)

Pada data skala besar, pendekatan analitik menjadi berat secara komputasi. SGD menawarkan solusi iteratif dengan memperbarui bobot sedikit demi sedikit berdasarkan *error* prediksi:

1. **Hitung Prediksi:** $\hat{y} = mx + c$
2. **Hitung Error:** $E = \hat{y} - y$
3. **Update Rule:**

$$m_{baru} = m_{lama} - (\alpha \cdot E \cdot x)$$
$$c_{baru} = c_{lama} - (\alpha \cdot E)$$

4. Sample Code: Implementasi SGD Sederhana (NumPy)

Kode berikut mendemonstrasikan bagaimana model "belajar" mencari garis terbaik melalui proses iterasi (*Epoch*).

Python

```
import numpy as np

# 1. Dataset: Luas Rumah (x) dan Harga Rumah (y)
x = np.array([60, 75, 80, 100, 120])
y = np.array([115, 130, 155, 180, 220])

# 2. Inisialisasi Parameter
m = 0.0
c = 0.0
learning_rate = 0.00001
epochs = 1000

# 3. Proses Training (SGD)
for epoch in range(epochs):
    for i in range(len(x)):
        # Prediksi y'
        y_pred = m * x[i] + c

        # Hitung Error
        error = y_pred - y[i]

        # Update Bobot m dan c
        m = m - (learning_rate * error * x[i])
        c = c - (learning_rate * error)

print(f"Hasil Training - Slope (m): {m:.4f}, Intercept (c): {c:.4f}")

# 4. Prediksi Data Baru
luas_baru = 150
prediksi_harga = m * luas_baru + c
print(f"Prediksi harga untuk luas {luas_baru}m2: {prediksi_harga:.2f}
Juta")
```

BAB IX
EVOLUSI MODEL LINEAR MENUJU ARSITEKTUR NEURAL NETWORK

1 Deskripsi Singkat Bahasan

Bab ini membahas perluasan konsep Regresi Linear dari variabel tunggal (*Univariate*) menjadi banyak variabel (*Multivariate*). Fokus utama materi adalah memvisualisasikan bagaimana hubungan linear antara *Input*, *Weights*, dan *Bias* membentuk struktur **Neuron** dalam **Neural Network**. Selain itu, bab ini memberikan tinjauan kritis terhadap persiapan tugas besar, pemilihan dataset yang relevan, serta pengenalan parameter dalam model AI skala besar (seperti DeepSeek R1) yang memiliki miliaran parameter.

2. Mindmap / Taxonomi

Struktur transisi model dan komponen Jaringan Syaraf Tiruan:

3. Penjelasan Detail

A. Formalisasi Multivariate Linear Regression

Dalam kasus nyata, prediksi tidak hanya bergantung pada satu faktor. Jika dalam Bab 6 kita menggunakan $y = mx + c$, maka dalam skala multivariate dengan banyak fitur (x), rumusnya diformalkan menjadi:

$$Y' = w_0x_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$$

Atau dalam bentuk notasi sigma:

$$Y' = \sum_{i=0}^{n} w_ix_i$$

Di mana:

- x_0 adalah **Bias** yang nilainya selalu 1.
- w_0 adalah bobot untuk bias (setara dengan konstanta c).
- w_i adalah **Weights** (bobot) yang merepresentasikan kekuatan hubungan antar fitur.

B. Anatomi Neural Network

Jaringan syaraf tiruan meniru cara kerja neuron otak manusia. Komponen utamanya meliputi:

1. **Input Layer:** Lapisan yang menerima fitur data (misal: luas rumah, emosi, kedisiplinan).
2. **Hidden Layer:** Lapisan antara yang melakukan ekstraksi pola. Semakin banyak lapisan ini, model disebut semakin "Deep" (**Deep Learning**).
3. **Output Layer:** Hasil akhir prediksi. Jika prediksi memiliki banyak kategori (misal: jenis-jenis narkoba), maka *Output Layer* akan memiliki banyak node sesuai jumlah kelas.

C. Skalabilitas Model AI (Billion Parameters)

Model seperti **DeepSeek R1 7B** memiliki arti terdapat **7 Miliar** parameter atau bobot (w) yang saling terhubung. Proses *Thinking* pada AI modern sebenarnya adalah proses kalkulasi probabilitas melalui miliaran sinapsis digital tersebut secara simultan menggunakan bantuan GPU (*Graphics Processing Unit*).

4. Sample Code: Simulasi Multivariate Regression (Multiple Inputs)

Contoh kode berikut menggunakan NumPy untuk menghitung prediksi berdasarkan banyak fitur sekaligus, mensimulasikan logika *Forward Pass* pada satu neuron.

Python

```
import numpy as np

# 1. Definisi Fitur (X) dan Bobot (W)
# Fitur: [Bias, Luas Rumah, Jumlah Kamar, Lokasi]
# Bias (X0) selalu bernilai 1
X = np.array([1, 120, 3, 2])

# Bobot (Weights) yang sudah dilatih (contoh)
W = np.array([10.5, 1.5, 20.2, 50.0])

# 2. Fungsi Prediksi Multivariate
def predict_price(features, weights):
    # Melakukan perkalian dot (Sigma Wi * Xi)
    prediction = np.dot(features, weights)
    return prediction

# 3. Eksekusi Prediksi
harga_prediksi = predict_price(X, W)

print(f"Fitur Input (termasuk Bias): {X}")
print(f"Bobot Model (W): {W}")
print(f"Hasil Prediksi Harga: {harga_prediksi:.2f} Juta")

# Penjelasan:
# (1*10.5) + (120*1.5) + (3*20.2) + (2*50.0) = 351.
```

Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

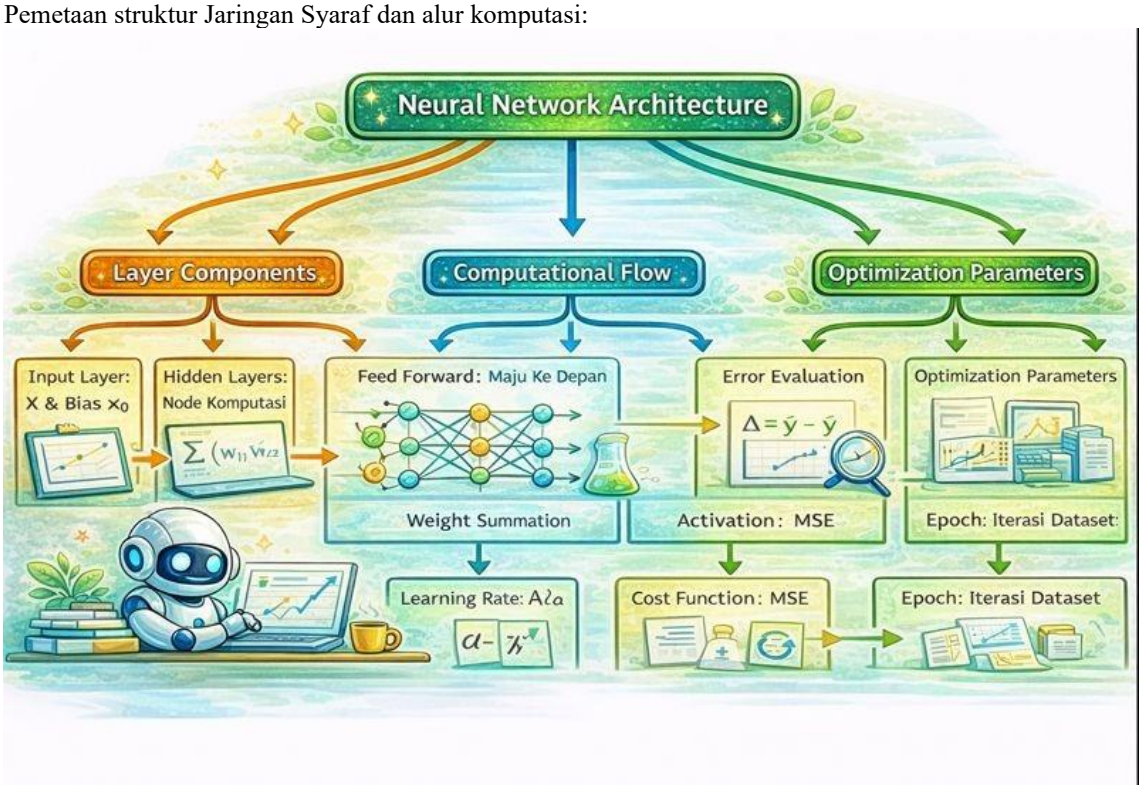
BAB X
ARSITEKTUR NEURAL NETWORK DAN MEKANISME FEED FORWARD

1. Deskripsi Singkat Bahasan

Bab ini membahas struktur mendalam dari *Multilayer Neural Network*, mencakup peran penting *Hidden Layers* dalam menangani data multivariate. Fokus utama materi adalah pada algoritma **Stochastic Gradient Descent (SGD)** dengan pendekatan **Feed Forward** untuk menghitung output model. Selain itu, bab ini menjelaskan cara mengukur kesalahan prediksi menggunakan fungsi biaya (*Cost Function*) **Mean Squared Error (MSE)** dan pengenalan konsep *Momentum* serta *Learning Rate* dalam optimasi bobot.

2. Mindmap / Taxonomi

Pemetaan struktur Jaringan Syaraf dan alur komputasi:



3. Penjelasan Detail

A. Formalisasi Hidden Layer

Dalam arsitektur ini, input tidak langsung menuju output, melainkan diproses terlebih dahulu melalui neuron di *Hidden Layer* (N_j) Setiap neuron menerima sinyal dari seluruh input layer yang dikalikan dengan bobot (*Weight*).
Secara matematis, nilai pada satu neuron hidden layer (N_0) dihitung sebagai:

$$N_0 = (w_{00} \cdot x_0) + (w_{30} \cdot x_1) + (w_{60} \cdot x_2) + (w_{90} \cdot x_3)$$

Atau secara umum untuk setiap neuron j:

$$N_j = \sum_{i=0}^{n} w_{ij} x_i$$

Di mana x_0 adalah Bias yang selalu bernilai 1 sebagai fungsi identitas.

B. Mekanisme Feed Forward

Proses Feed Forward adalah langkah di mana data dimasukkan ke input layer, melewati hidden layer, hingga menghasilkan output (O) Output akhir (O) merupakan hasil akumulasi dari neuron-neuron hidden layer yang dikalikan dengan bobot layer berikutnya:

$$O = \sum_{j=0}^{m} W_{j(out)} N_j$$

C. Pengukuran Kesalahan (Mean Squared Error)

Untuk mengetahui seberapa buruk performa model dalam satu Epoch (satu kali perulangan seluruh dataset), kita menggunakan **MSE**. Rumusnya adalah:

$$MSE = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Di mana:

- n = jumlah data.
- y = nilai target asli.
- $\hat{y}$ atau y' = hasil prediksi model.

Nilai MSE ini akan terus diperbaiki melalui pembaruan bobot (*Weight Update*) di iterasi berikutnya agar garis prediksi semakin mendekati titik data asli (*best fit*).

4. Sample Code: Simulasi Feed Forward Sederhana (Python)

Python

```python
# 1. Definisi Input (Fitur) dan Bobot Awal (Random)
# Fitur: [Bias, Fitur1, Fitur2]
X = [1.0, 0.5, 0.8]
# Bobot menuju 3 Hidden Neurons (masing-masing punya 3 bobot)
W_hidden = [
    [0.2, 0.4, -0.5], # Bobot untuk Hidden 0
    [0.1, -0.3, 0.2], # Bobot untuk Hidden 1
    [0.5, 0.1, 0.4]   # Bobot untuk Hidden 2
]

# 2. Perhitungan Hidden Layer (Feed Forward Step 1)
hidden_outputs = []
for weights in W_hidden:
    # Sigma (Wi * Xi)
    neuron_sum = sum(X[i] * weights[i] for i in range(len(X)))
    hidden_outputs.append(neuron_sum)
```

```
print(f"Nilai Neuron di Hidden Layer: {hidden_outputs}")

# 3. Perhitungan Output Final (Feed Forward Step 2)
W_output = [0.6, -0.2, 0.4] # Bobot dari Hidden ke Output
final_output = sum(hidden_outputs[j] * W_output[j] for j in
range(len(hidden_outputs)))

print(f"Prediksi Output Final (Y'): {final_output:.4f}")
```

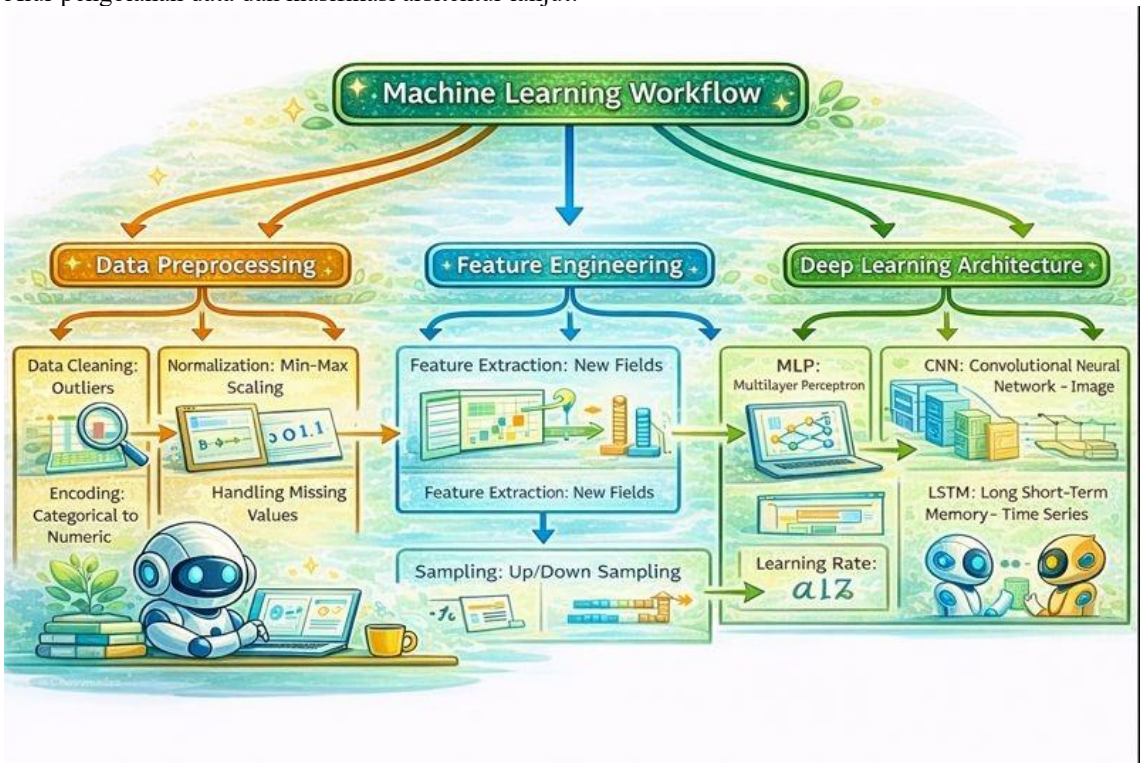
BAB X1
DATA PREPROCESSING, FEATURE ENGINEERING, DAN PENGENALAN DEEP LEARNING

1. Deskripsi Singkat Bahasan

Bab ini membahas tahapan vital sebelum melakukan *training* model, yaitu **Data Preprocessing**. Fokus utama materi adalah menangani data dunia nyata yang tidak ideal (kotor) melalui teknik normalisasi, penanganan nilai kosong (*missing values*), dan penyelarasan dataset yang tidak seimbang (*imbalanced dataset*). Selain itu, bab ini memperkenalkan konsep **Deep Learning** sebagai evolusi dari *Neural Network* dengan arsitektur yang lebih kompleks seperti CNN untuk gambar dan LSTM untuk data urutan waktu (*time-series*).

2. Mindmap / Taxonomi

Alur pengolahan data dan klasifikasi arsitektur lanjut:



3. Penjelasan Detail

A. Fungsi Aktivasi Sigmoid

Dalam Neural Network, hasil penjumlahan bobot seringkali dilewatkan ke fungsi aktivasi untuk memetakan nilai ke rentang tertentu (0 hingga 1).

$$f(z) = \frac{1}{1 + e^{-z}}$$

B. Teknik Normalisasi (Min-Max Scaling)

Normalisasi diperlukan agar fitur dengan rentang nilai besar tidak mendominasi proses learning, sehingga model lebih cepat mencapai konvergensi (error minimum).

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

C. Penanganan Dataset Tidak Seimbang (Imbalanced Dataset)

Jika distribusi label sangat timpang (misal: 99% transaksi normal, 1% *fraud*), model akan cenderung mengabaikan kelas minoritas. Solusinya adalah:

1. **Upsampling:** Menduplikasi data kelas minoritas.
2. **Downsampling:** Mengurangi data kelas mayoritas untuk menyamai jumlah kelas minoritas.

D. Arsitektur Deep Learning

Deep Learning dibedakan dari *Neural Network* biasa melalui jumlah *Hidden Layer* yang sangat banyak (dalam skala puluhan hingga ratusan) dan adanya fase **Pre-training**.

- **CNN (Convolutional Neural Network):** Menggunakan operasi *convolution* untuk mendeteksi kontur dan pola pada pikel gambar.
- **LSTM (Long Short-Term Memory):** Memanfaatkan konsep *Markov Chain* di mana input saat ini dipengaruhi oleh informasi dari masa lalu.

4. Sample Code: Preprocessing & Normalisasi (Python)

Berikut adalah implementasi sederhana teknik normalisasi dan encoding kategori menggunakan pustaka Pandas dan NumPy.

Python

```
import pandas as pd
import numpy as np
```

```
# 1. Dataset kotor sederhana
data = {
    'Amount': [15000000, 1000000, 50000000, np.nan], # Ada missing
value (NaN)
    'Category': ['Fraud', 'Non-Fraud', 'Fraud', 'Non-Fraud']
}
df = pd.DataFrame(data)

# 2. Handling Missing Value (Mengisi dengan rata-rata)
df['Amount'] = df['Amount'].fillna(df['Amount'].mean())

# 3. Encoding Kategori ke Angka (Manual Label Encoding)
# Fraud = 1, Non-Fraud = 0
df['Category_Code'] = df['Category'].map({'Fraud': 1, 'Non-Fraud': 0})

# 4. Normalisasi Min-Max Scaling (Amount)
min_val = df['Amount'].min()
max_val = df['Amount'].max()
df['Amount_Normalized'] = (df['Amount'] - min_val) / (max_val -
min_val)

print("Data Setelah Preprocessing:")
print(df[['Amount_Normalized', 'Category_Code']])
```

Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

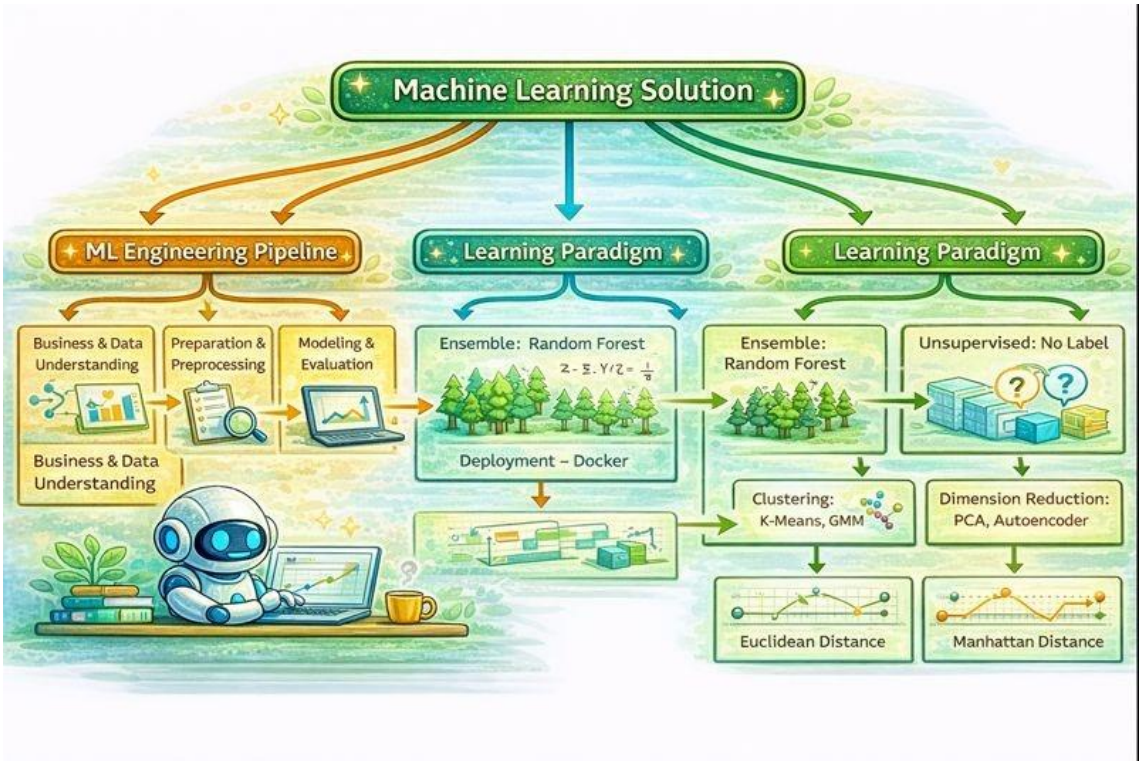
BAB X11
PIPELINE ML ENGINEERING DAN METODE UNSUPERVISED LEARNING

1. Deskripsi Singkat Bahasan

Bab ini menjelaskan dikotomi antara *Machine Learning Science* yang berfokus pada menjawab fenomena data, dengan *Machine Learning Engineering* yang berfokus pada pembangunan solusi teknis. Materi mencakup standarisasi alur kerja industri menggunakan framework CRISP-DM, perbedaan tipe data *Flat Table* dan *Time Series*, serta eksplorasi mendalam pada **Unsupervised Learning** melalui algoritma *K-Means* dan *Gaussian Mixture Model* (GMM) untuk pengolahan data tanpa label.

2. Mindmap / Taxonomi

Struktur ekosistem ML Engineering dan klasifikasi algoritma:



3. Penjelasan Detail

A. Pipeline ML Engineering (CRISP-DM)

Dalam membangun solusi nyata (misal: proyek karbon atau ESG), *Engineer* harus mengikuti tahapan

CRISP-DM:

1. **Business Understanding:** Memahami domain masalah (seperti Paris Agreement atau Net Zero).
2. **Data Preparation:** Melakukan pembersihan dan transformasi.
3. **Modeling:** Memilih algoritma yang sesuai (*Stochastic* vs *Deterministic*).
4. **Deployment:** Menerapkan model ke dalam *server* menggunakan kontainerisasi (Docker).

B. Ensemble Learning (Random Forest)

Ensemble adalah metode penggabungan beberapa model untuk meningkatkan akurasi. Pada Random Forest, sistem menggenerasi banyak pohon keputusan secara acak, lalu hasil akhirnya ditentukan melalui sistem voting:

$$\text{Output Final} = \text{mode}\{T_1(x), T_2(x), \dots, T_n(x)\}$$

C. Unsupervised Learning: Clustering

Berbeda dengan *Supervised*, metode ini tidak memiliki target kelas (Y).Algoritma mencari pola alami dari distribusi fitur.

1. **K-Means (Jarak):** Mengelompokkan data berdasarkan kedekatan dengan titik pusat (*Centroid*).
    - Euclidean Distance:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

2. GMM - Gaussian Mixture Model (Probabilitas): Melihat data sebagai kumpulan distribusi normal atau "gunung" densitas. Jika K-Means menggunakan jarak, GMM menggunakan Likelihood (kecocokan) terhadap fungsi distribusi Gaussian:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

D. Dimension Reduction

Digunakan untuk menyederhanakan data berdimensi tinggi (ratusan fitur) menjadi dimensi rendah (2D atau 3D) tanpa menghilangkan informasi esensial. Contoh utama adalah **PCA** (*Principal Component Analysis*).

Python

```
from sklearn.cluster import KMeans
import numpy as np
import pandas as pd

# 1. Dataset tanpa label (Contoh: Amount dan Jumlah Transaksi)
data = {
    'Amount': [15, 12, 100, 110, 5, 8],
    'Frequency': [2, 3, 10, 12, 1, 2]
}
X = pd.DataFrame(data)

# 2. Inisialisasi K-Means dengan K=2 (dua kelompok)
kmeans = KMeans(n_clusters=2, random_state=42)
```

```
# 3. Proses Clustering
kmeans.fit(X)

# 4. Mendapatkan Centroid dan Label Hasil Prediksi
centroids = kmeans.cluster_centers_
labels = kmeans.labels_

print("Centroid yang ditemukan:")
print(centroids)
print("\nData masuk ke kluster:")
print(labels) # Output: 0 atau 1
```

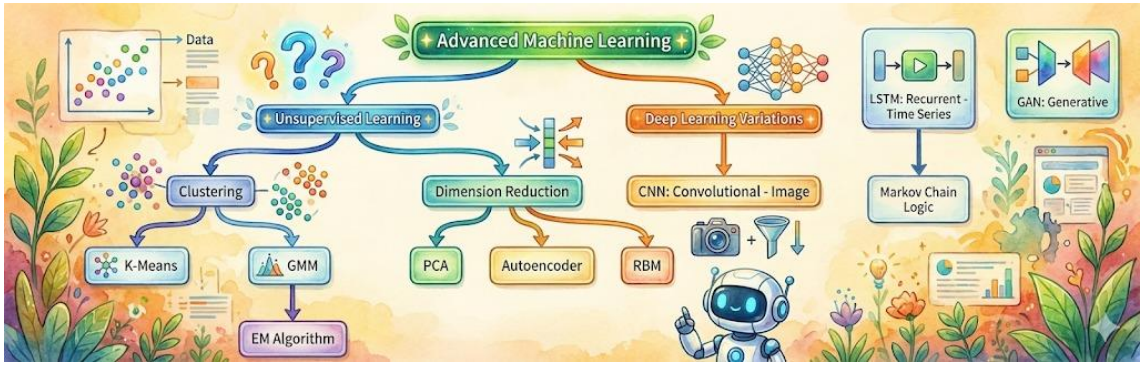
BAB XIII
METODE PROBABILISTIK CLUSTERING DAN ARSITEKTUR DEEP LEARNING

1. Deskripsi Singkat Bahasan

Bab ini membahas pergeseran paradigma dalam *Unsupervised Learning* dari pendekatan berbasis jarak (*Distance-based*) menuju pendekatan berbasis distribusi probabilitas (**Gaussian Mixture Model**). Materi mencakup penggunaan algoritma **Expectation-Maximization (EM)** untuk maksimisasi *likelihood*, teknik reduksi dimensi untuk data kompleks, serta pengenalan arsitektur *Deep Learning* khusus seperti **LSTM** untuk data *time-series* dan **CNN** untuk pemrosesan citra.

2. Mindmap / Taxonomi

Hierarki algoritma lanjut dan pemrosesan data:



3. Penjelasan Detail

A. Gaussian Mixture Model (GMM) dan Algoritma EM

Jika K-Means mengandalkan jarak Euclidean, GMM mengasumsikan bahwa data merupakan gabungan dari beberapa distribusi Gaussian (Normal). Setiap kluster direpresentasikan sebagai sebuah "gunung" distribusi dengan parameter μ (rata-rata) dan σ (simpangan baku).

Untuk mengoptimasi model ini, digunakan algoritma **Expectation-Maximization (EM)**:

- **Stochastic Gradient Descent (SGD):** Digunakan untuk **menekan error** (meminimalkan kerugian).

- **Expectation-Maximization (EM):** Digunakan untuk **memaksimalkan Likelihood** (kecocokan data terhadap suatu kelompok).
- Fungsi distribusi Gaussian multivariate:

$$f(x) = \frac{1}{\sqrt{(2\pi)^k |\Sigma|}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

B. Reduksi Dimensi (Dimension Reduction)

Manusia hanya mampu memvisualisasikan data hingga 3 dimensi $(X, Y,)$ Pada dataset industri yang memiliki ratusan fitur, digunakan **PCA** (*Principal Component Analysis*) untuk mereduksi jumlah fitur tanpa kehilangan informasi esensial (esensi data).

C. Time Series dan LSTM (Long Short-Term Memory)

Pada data yang memiliki urutan waktu (seperti harga saham atau pola belanja), berlaku prinsip Markov Chain di mana kondisi masa depan $(t+)$dipengaruhi oleh kondisi saat ini (t)

$$P(X_{t+1} \mid X_t, X_{t-1}, \dots, X_0) = P(X_{t+1} \mid X_t)$$

LSTM adalah arsitektur Neural Network yang mampu "mengingat" informasi dari urutan waktu yang panjang dengan memasukkan output sebelumnya kembali menjadi input.

4. Sample Code: Implementasi GMM Clustering (Python)

Berikut adalah contoh penggunaan GMM untuk mengelompokkan data berdasarkan distribusi probabilitas menggunakan scikit-learn.

Python

```python
import numpy as np
import pandas as pd
from sklearn.mixture import GaussianMixture
import matplotlib.pyplot as plt

# 1. Dataset dummy (Contoh: Distribusi Suhu dan Tekanan)
data = {
    'Suhu': [20, 22, 21, 80, 85, 82, 50, 52],
    'Tekanan': [1, 1.2, 1.1, 5, 5.5, 5.2, 3, 3.2]
}
X = pd.DataFrame(data)

# 2. Inisialisasi GMM dengan 3 Komponen (3 Gunung Distribusi)
gmm = GaussianMixture(n_components=3, random_state=42)

# 3. Proses EM (Expectation-Maximization)
gmm.fit(X)

# 4. Prediksi Likelihood (Data masuk ke kelompok mana)
labels = gmm.predict(X)
probs = gmm.predict_proba(X) # Probabilitas masuk ke setiap kluster

print("Label Kelompok per Data:")
print(labels)
print("\nSkor Probabilitas (Likelihood):")
print(np.round(probs, 2))
```

KOMPUTA : Jurnal Ilmiah Komputer dan Informatika
Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

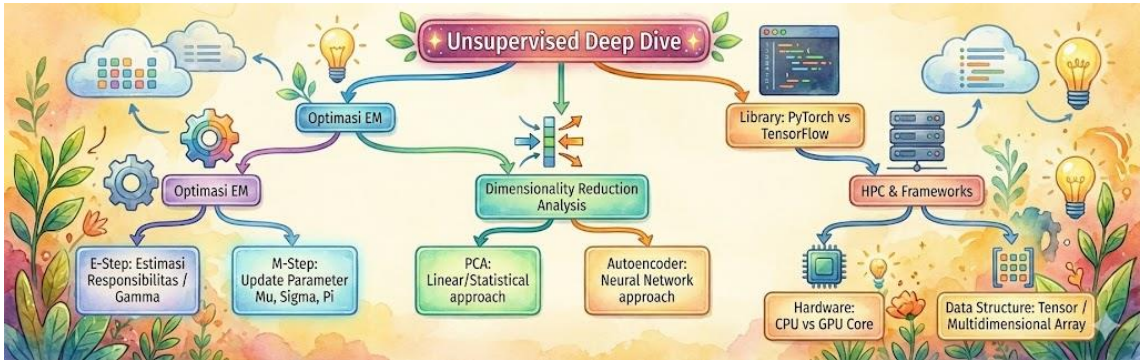
BAB XIV
OPTIMASI EXPECTATION-MAXIMIZATION DAN EKOSISTEM KOMPUTASI TENSOR

1. Deskripsi Singkat Bahasan

Bab ini mengeksplorasi proses iteratif dalam *Unsupervised Learning* untuk mencapai konvergensi model melalui algoritma **Expectation-Maximization (EM)**. Pembahasan meliputi pencarian parameter μ (mean) dan σ (standar deviasi) yang optimal pada GMM, perbandingan efektivitas **PCA** versus **Autoencoder** dalam memvisualisasikan data berdimensi tinggi, serta peran **GPU Acceleration** dan arsitektur **Tensor** (PyTorch/TensorFlow) dalam mempercepat komputasi matriks skala besar.

2. Mindmap / Taxonomi

Struktur optimasi dan infrastruktur AI:



3. Penjelasan Detail

A. Algoritma Expectation-Maximization (EM)

Algoritma ini digunakan untuk mencari estimasi *Maximum Likelihood* ketika model memiliki variabel laten (tersembunyi). Prosesnya terdiri dari dua tahap utama:

1. E-Step (Expectation): Menghitung probabilitas (responsibilitas) sebuah data x_i berasal dari kluster k menggunakan nilai parameter saat ini. Nilai ini disimbolkan dengan Gamma (y)

$$\gamma(z_{ik}) = \frac{\pi_k \mathcal{N}(x_i \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_i \mid \mu_j, \Sigma_j)}$$

M-Step (Maximization): Memperbarui parameter model (μ, Σ, π) untuk memaksimalkan nilai *likelihood* berdasarkan hasil E-step.

Update Mean (μ):

$$\mu_k^{new} = \frac{1}{N_k} \sum_{i=1}^{N} \gamma(z_{ik}) x_i$$

B. Reduksi Dimensi: PCA vs Autoencoder

- **PCA:** Menggunakan pendekatan statistik linear untuk memproyeksikan data ke dimensi yang lebih rendah. Cepat namun terkadang kehilangan detail non-linear.

- **Autoencoder:** Menggunakan arsitektur *Neural Network* (Encoder-Decoder) untuk mengekstrak fitur penting secara non-linear. Hasil visualisasi biasanya lebih terkelompok (*fit*) dibandingkan PCA.

C. Konsep Tensor dan GPU

Model AI modern seperti DeepSeek atau GPT membutuhkan miliaran operasi perkalian matriks. **Tensor** adalah struktur data array multidimensi yang dioptimalkan untuk berjalan di atas **GPU (Graphics Processing Unit)**. Berbeda dengan CPU yang memiliki sedikit *core* pintar, GPU memiliki ribuan *core* sederhana yang bekerja secara paralel (keroyokan) untuk menyelesaikan kalkulasi tensor dengan sangat cepat.

4. Sample Code: Komputasi Tensor dengan PyTorch

Berikut adalah contoh sederhana penggunaan PyTorch untuk mendefinisikan Tensor dan melakukan operasi matriks yang diakselerasi (logika dasar di balik training GMM/Deep Learning).

Python

```
import torch

# 1. Inisialisasi Tensor (mirip NumPy namun mendukung GPU)
# Membuat matriks bobot acak 3x3
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Menggunakan perangkat: {device}")

weights = torch.rand(3, 3).to(device)
inputs = torch.tensor([[1.0, 0.5, 0.2], [0.1, 0.9, 0.4], [0.5, 0.2,
0.8]]).to(device)

# 2. Operasi Perkalian Matriks (Matrix Multiplication)
# Logika dasar Feed Forward / E-Step
output = torch.mm(inputs, weights)

# 3. Menghitung Log-Likelihood sederhana (Contoh stabilisasi)
log_likelihood = torch.log(torch.sum(output))

print("\nMatriks Input:\n", inputs)
print("\nHasil Komputasi Tensor:\n", output)
print(f"\nSkor Likelihood: {log_likelihood.item():.4f}")
```