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

Rangkuman Materi Perkuliahan Machine Learning Kelompok 11

Muhamad Naufal Fauzan[1], Siti Tahtia Ainun Zahra[2]

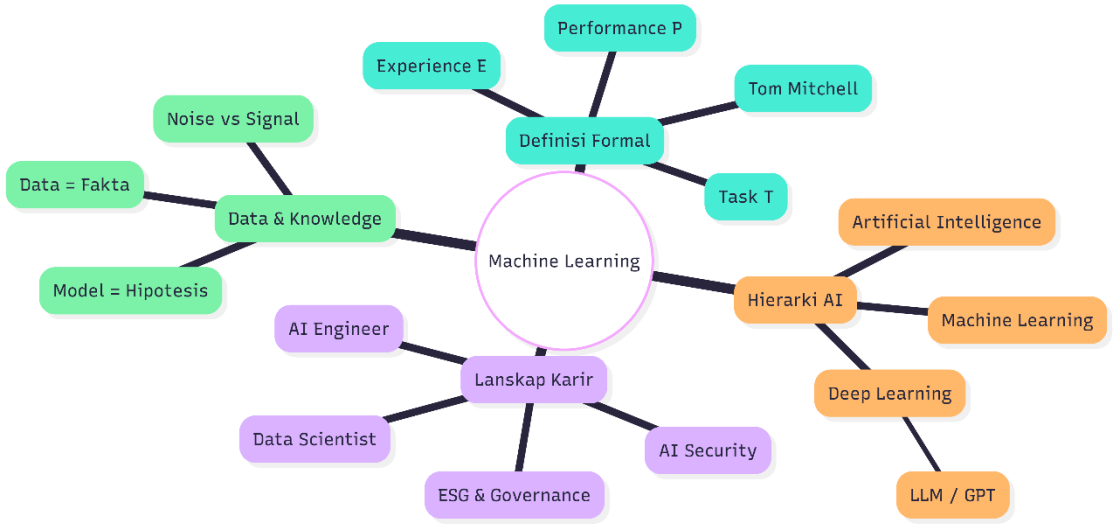
[1,2] Program Studi Sistem Informasi, STMIK Tazkia Bogor
E-mail : 241572010008.naufal@student.stmik.tazkia.ac.id
241572010014.ainun@student.stmik.tazkia.ac.id

BAB I
FUNDAMENTAL MACHINE LEARNING & LANSKAP AI

1. Deskripsi Singkat Bahasan

Bab ini membahas definisi fundamental Machine Learning berdasarkan perspektif Tom Mitchell, perbedaan mendasar antara terminologi AI, Machine Learning, dan Deep Learning, serta tinjauan terhadap lanskap karir masa depan (AI Security, Web3, ESG). Fokus utama adalah memahami bagaimana mesin "belajar" melalui mekanisme Experience, Task, dan Performance Measure, serta pentingnya data sebagai fakta empiris dalam pembentukan model pengetahuan (knowledge/hypothesis).

2. Mindmap / Taksonomi (Visualisasi)



Gambar 1. Mindmap Fundamental Machine Learning.

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

3. Penjelasan Detail

3.1 Definisi Formal Machine Learning

Dalam ranah akademis, definisi yang menjadi standar emas (gold standard) untuk Machine Learning berasal dari Tom Mitchell. Sebuah program komputer dikatakan "belajar" bukan sekadar ketika ia menyimpan data, melainkan ketika ia mampu meningkatkan kinerjanya seiring bertambahnya pengalaman.

Secara matematis, definisi ini dapat diformulasikan sebagai tuple tiga variabel: $< \mathrm{P}, \mathrm{T}, \mathrm{E} >$.

Definisi tersebut berbunyi: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Implementasi konsep ini dapat dianalogikan dengan proses manusia belajar:

- **Task (T):** Tugas spesifik yang harus diselesaikan, misalnya klasifikasi citra atau prediksi harga saham.
- **Performance (P):** Metrik kuantitatif untuk mengukur keberhasilan, seperti akurasi, presisi, atau *mean squared error*.
- **Experience (E):** Data historis atau interaksi lingkungan yang menjadi bahan pembelajaran.

Mesin dianggap "belajar" jika dan hanya jika nilai P meningkat pada tugas T setelah mendapatkan E. Jika performa stagnan meskipun data bertambah, maka proses *learning* gagal terjadi.

3.2 Data Sebagai Fakta dan Pembentukan Hipotesis

Dalam Machine Learning, data didefinisikan sebagai representasi digit dari fakta empiris ("Ground Truth"). Transkrip perkuliahan menekankan bahwa validitas model sangat bergantung pada validitas data. Data yang mengandung *noise* (gangguan) atau bukan fakta akan menghasilkan model yang bias.

Hubungan antara data dan pengetahuan dapat dijelaskan sebagai berikut:

1. **Data:** Fakta mentah yang dikumpulkan (Experience).
2. **Model/Hipotesis:** Pola atau struktur pengetahuan yang diekstraksi dari data. Hipotesis (h) haruslah konsisten terhadap data (D).

Tujuan akhir dari algoritma ML adalah mencari hipotesis h dalam ruang hipotesis H yang paling mendekati fungsi target f yang sebenarnya.

3.3 Tren Masa Depan: AI Security & ESG

Berdasarkan tinjauan industri, fokus pengembangan AI saat ini bergeser dari sekadar pembuatan model (modelling) menuju integrasi sistem yang aman dan berdampak.

- **AI Security:** Mengingat kerentanan model terhadap serangan (seperti *adversarial attacks*) dan isu privasi data (GDPR/UU PDP), integrasi keamanan dalam arsitektur AI menjadi krusial.
- **ESG (Environmental, Social, and Governance):** Pemanfaatan AI untuk menghitung emisi karbon (carbon footprint) dan analisis citra satelit untuk keberlanjutan lingkungan. Ini adalah domain baru di mana *computational power* digunakan untuk audit lingkungan.

4. Sample Code (Ilustrasi Konsep T, P, E)

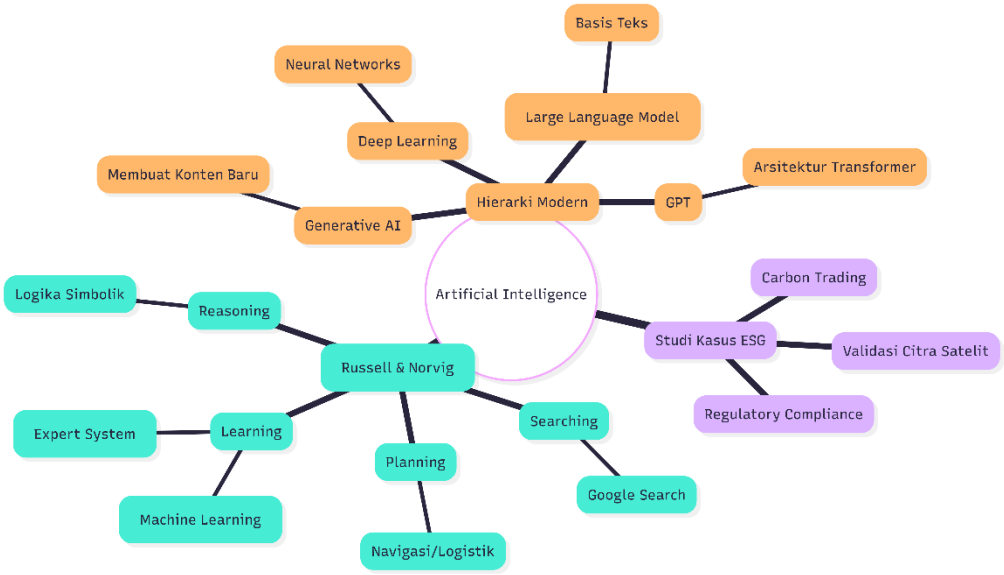
```
import numpy as np

class SimpleLearner:
    """
    Simulasi sederhana konsep T, P, E dari Tom Mitchell.
    Task (T): Memprediksi nilai y = 2x
    Performance (P): Mean Squared Error (MSE) - semakin kecil semakin
baik
    Experience (E): Data training (x, y)
    """
    def __init__(self):
        self.weight = np.random.rand() # Inisialisasi bobot acak
(Model Awal)

    def predict(self, x):
        return self.weight * x  # Hipotesis saat ini

    def learn(self, experience_data, learning_rate=0.01, epochs=5):
        X, y_true = experience_data

        print(f"Initial Weight: {self.weight:.4f}")

        for epoch in range(epochs):
            # 1. Task Execution (Prediksi)
            y_pred = self.predict(X)

            # 2. Performance Measurement (P) -> Hitung Error
            error = y_true - y_pred
            mse = np.mean(error**2)

            # 3. Learning from Experience (E) -> Update Bobot
            # Sederhana: Weight = Weight + (Error * Input * LR)
            gradient = -2 * np.mean(error * X)
            self.weight -= learning_rate * gradient

            print(f"Epoch {epoch+1}: Performance (MSE) = {mse:.4f},
New Weight = {self.weight:.4f}")

# Simulasi
# Experience (E): Fakta bahwa input 2 hasilnya 4, input 3 hasilnya 6
experience = (np.array([1, 2, 3, 4]), np.array([2, 4, 6, 8]))

learner = SimpleLearner()
learner.learn(experience)
```

BAB II
IMPLEMENTASI AI (ESG) & TAKSONOMI LANJUT

1. Deskripsi Singkat Bahasan

Bab ini mengeksplorasi dua domain utama. Pertama, tinjauan implementatif peran Artificial Intelligence dalam sektor Environmental, Social, and Governance (ESG), khususnya dalam validasi perdagangan karbon (Carbon Trading) dan kepatuhan regulasi (Compliance). Kedua, pendalaman taksonomi AI berdasarkan literatur Russell & Norvig, yang membagi AI menjadi empat pilar: Searching, Planning, Reasoning, dan Learning. Bab ini juga membedah hierarki teknologi modern dari AI hingga Generative Pre-trained Transformer (GPT) serta pembedaan fundamental antara pembelajaran Induktif (Machine Learning) dan Deduktif (Expert System).

2. Mindmap / Taksonomi (Visualisasi)



Gambar 2. Mindmap Impementasi AI.

3. Penjelasan Detail

3.1 AI dalam Ekonomi Karbon (ESG Case Study)

Salah satu implementasi *real-world* yang masif saat ini adalah penggunaan AI dalam validasi kredit karbon. Pasar karbon (seperti IDX Carbon atau Verra) membutuhkan verifikasi ketat agar klaim penyerapan karbon (misal: "Hutan ini menyerap 100 ton CO_2") adalah fakta, bukan manipulasi.

AI digunakan dalam dua aspek teknis:

1. **Computer Vision untuk Validasi Spesies:** Menggunakan citra drone/satelit untuk memverifikasi jenis pohon (misal: membedakan Durian vs Jambu) dan menghitung biomassa.
2. **Compliance Automation:** Memastikan proyek reforestasi mematuhi regulasi lokal (seperti SRN - Sistem Registri Nasional) dan internasional.

Secara matematis, total serapan karbon (C_{total}) seringkali diestimasi sebagai fungsi dari luas area (A), kerapatan vegetasi (V), dan koefisien spesies (k), di mana AI memprediksi parameter V dan k dari data sensor:

$$C_{total} = \int_{area} f(V, k)\ dA$$

3.2 Empat Pilar AI (Russell & Norvig)

Berdasarkan buku AI: A Modern Approach, AI tidak hanya soal Machine Learning. Ada 4 pilar utama:

1. Searching: Algoritma pencarian solusi dalam ruang masalah (contoh: Pathfinding, Google Search).
2. Planning: Merencanakan urutan tindakan untuk mencapai tujuan.
3. Reasoning: Penalaran logika (simbolik) untuk menarik kesimpulan.
4. Learning: Meningkatkan performa berdasarkan data.

3.3 Learning: Induktif vs Deduktif

Poin krusial dalam bab ini adalah klasifikasi metode belajar:

- **Inductive Learning (Machine Learning):** Berangkat dari Data spesifik menuju Aturan umum.
  - Input: Data (x, y) → Output: Fungsi $(f(x) = y)$.
- **Deductive Learning (Expert System/Rule-Based):** Berangkat dari Aturan umum menuju Kesimpulan spesifik.
  - Input: Aturan $(If\ A\ then\ B)$ + Fakta (A) → Output: Kesimpulan (B).

3.4 Hierarki Generative AI

Kekeliruan umum di industri diluruskan melalui hierarki berikut:

$$AI \supset ML \supset Deep\ Learning \supset GenAI \supset LLM \supset GPT \supset ChatGPT$$

- **LLM (Large Language Model):** Model bahasa besar.
- **GPT:** Arsitektur spesifik (*Generative Pre-trained Transformer*).
- **ChatGPT:** Produk aplikasi yang menggunakan model GPT.

5. Sample Code (Simulasi Validasi Karbon)

Kode ini mensimulasikan logika "AI Validation" yang dibahas dosen (membedakan spesies pohon dan menghitung karbon) menggunakan Python Class sederhana.

```
class CarbonValidator:
    """
    Simulasi Sistem Validasi Kredit Karbon berbasis AI.
    Menggunakan logika Rule-Based sederhana untuk validasi spesies
    dan kalkulasi serapan CO2.
    """

    def __init__(self):
        # Database Koefisien Serapan Karbon (kg CO2/tahun per pohon)
        self.species_db = {
            "Mangrove": 25.0,
            "Jati": 15.0,
            "Durian": 20.0,
            "Semak": 1.5
        }

    def validate_image(self, image_data):
        """
        Simulasi Computer Vision:
        Mendeteksi spesies pohon dari 'image_data' (mock string).
        """
        # Dalam implementasi nyata, ini menggunakan CNN/ResNet
        detected_species = image_data.split("_")[1] # Misal:
"IMG_Mangrove_001"
        confidence = 0.95 # Mock confidence score
```

```
        if detected_species in self.species_db:
            return detected_species, confidence
        else:
            return "Unknown", 0.0

    def calculate_credit(self, inputs):
        """
        Menghitung potensi kredit karbon.
        """
        total_carbon = 0
        valid_trees = 0

        print("--- Memulai Validasi Lapangan ---")

        for img in inputs:
            species, conf = self.validate_image(img)

            if conf > 0.8: # Threshold validasi AI
                carbon = self.species_db[species]
                total_carbon += carbon
                valid_trees += 1
                print(f"[VALID] {species} terdeteksi. Serapan:
{carbon} kg CO2")
            else:
                print(f"[REJECT] Spesies tidak dikenali atau
confidence rendah.")

        return total_carbon, valid_trees

# Simulasi Data Lapangan (Image Tags)
field_data = ["IMG_Mangrove_01", "IMG_Jati_05", "IMG_Semak_99",
"IMG_Unknown_12"]

validator = CarbonValidator()
total_co2, trees = validator.calculate_credit(field_data)

print(f"\nTotal Valid Trees: {trees}")
print(f"Total Carbon Credit: {total_co2} kg CO2 eq")
```

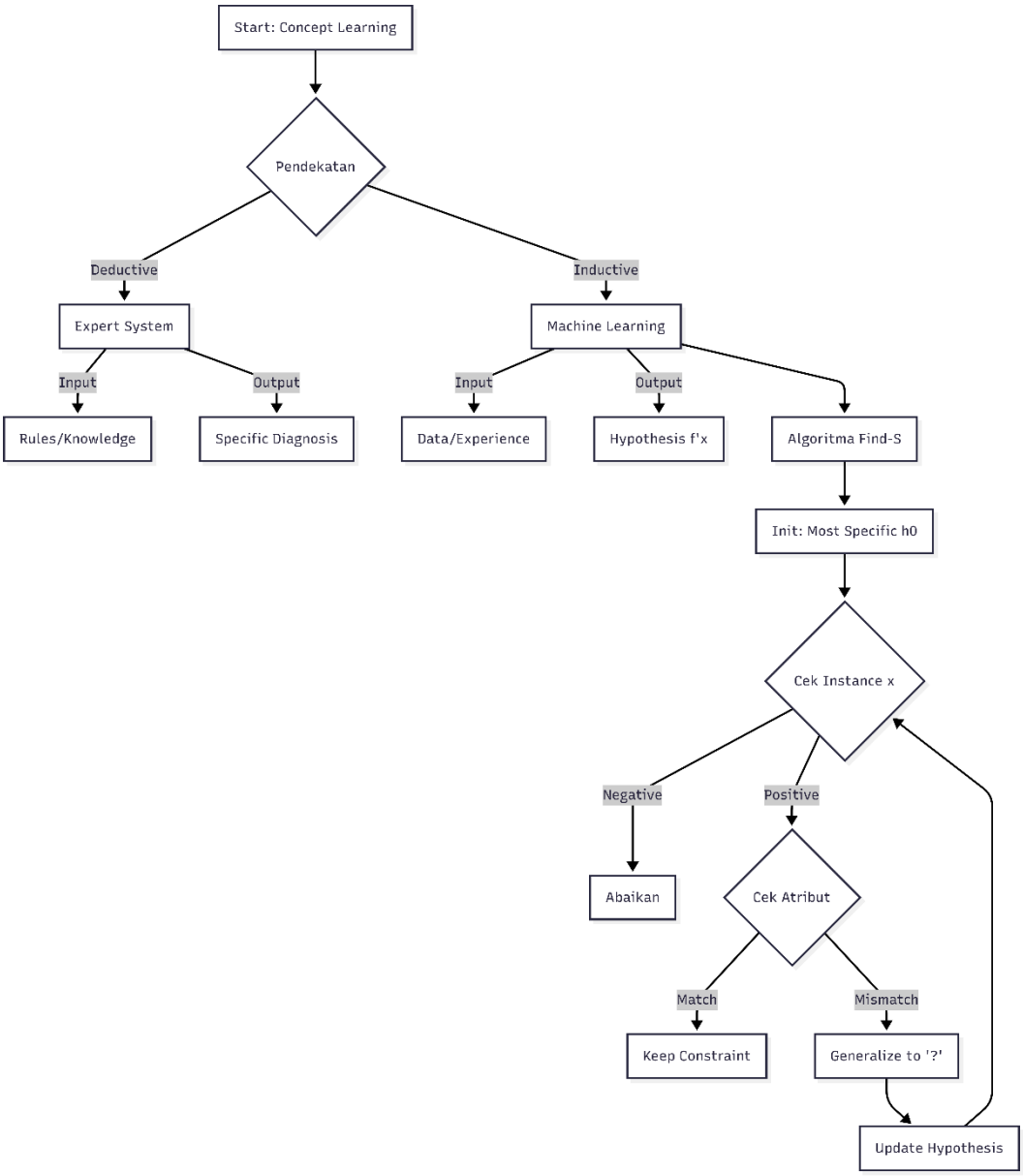
Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

BAB III
INDUCTIVE LEARNING & ALGORITMA FIND-S

1. Deskripsi Singkat Bahasan

Bab ini mendalami paradigma Inductive Learning, yaitu metode pembelajaran mesin yang menyimpulkan aturan umum (hipotesis) dari sekumpulan data spesifik. Diskusi mencakup perbedaan fundamental antara pendekatan Induktif (Machine Learning) dan Deduktif (Expert System). Fokus utama bab ini adalah pembedahan algoritma Find-S (Find-Specific), sebuah algoritma Concept Learning dasar yang bekerja dengan mencari hipotesis paling spesifik yang konsisten dengan semua contoh positif (Positive Instances).

2. Mindmap / Taksonomi (Visualisasi)



Gambar 3. Diagram ini memvisualisasikan alur logika Find-S.

3. Penjelasan Detail

3.1 Inductive vs Deductive Learning

- **Deductive Learning:** Bergerak dari Umum ke Khusus. Kita memiliki pengetahuan/rumus pasti ($f(x)$) lalu diterapan pada kasus. Contoh: Dokter yang sudah hafal ciri-ciri kanker mendiagnosa pasien.
- **Inductive Learning:** Bergerak dari Khusus ke Umum. Kita memiliki banyak data pasien (D), lalu mencari pola untuk membentuk rumus pendekatan ($f'(x)$).

Tujuan Machine Learning adalah mencari fungsi aproksimasi $f'(x)$ yang paling mendekati fungsi target asli $f(x)$:

$$Error = f(x) - f'(x)$$

Semakin kecil error, semakin valid hipotesis yang dibentuk.

3.2 Terminologi Concept Learning

Dalam algoritma Find-S, kita menggunakan dataset klasik "EnjoySport". Beberapa istilah kunci:

- **Instance (X):** Satu baris data tanpa label. Contoh: $\langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$.
- **Target Concept (c):** Label Boolean (Yes/No), dalam kasus ini "EnjoySport".
- **Hypothesis (h):** Vektor batasan yang merepresentasikan pola yang dipelajari.

3.3 Logika Algoritma Find-S

Find-S bekerja dengan prinsip: *"Mulailah dari yang paling spesifik, lalu longgarkan aturan hanya jika dipaksa oleh data positif."*
Notasi Hipotesis:

- $\emptyset$: Menolak semua (Paling Spesifik/Strict).
- ?: Menerima semua (Paling General/Loose).
- $Value$: Nilai spesifik (misal: 'Sunny').

Langkah Kerja:

1. Inisialisasi h ke nilai paling spesifik:

$$h_0 = \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

2. Iterasi setiap *positive instance*. (Abaikan *negative instance*).
3. Untuk setiap atribut dalam *positive instance*, jika nilai atribut berbeda dengan nilai di h, ganti nilai di h dengan yang lebih general.
    - Jika h adalah $\emptyset$, ganti dengan nilai atribut data.
    - Jika h adalah nilai spesifik (misal 'Sunny') dan data baru adalah 'Rainy', ganti dengan ?.

Ilustrasi Perubahan Hipotesis:

- x_1: $\langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$ (+).
    - h_1: $\langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$
- x_2: $\langle Sunny, Warm, High, Strong, Warm, Same \rangle$ (+).
    - Perhatikan atribut ke-3 berubah dari Normal ke High. Maka digeneralisasi menjadi ?.
    - h_2: $\langle Sunny, Warm, ?, Strong, Warm, Same \rangle$

4. Sample Code (Implementasi Find-S)

kode Python ini adalah implementasi teknis dari logika tersebut untuk jurnal.

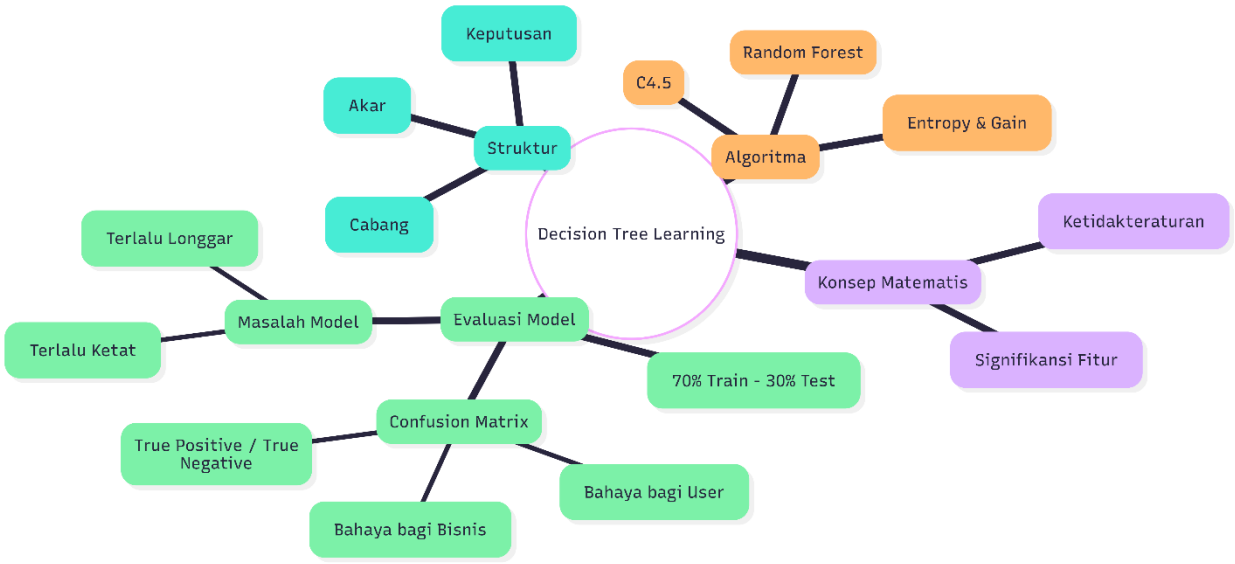
```
import pandas as pd
import numpy as np

def train_find_s(data, target):
    """
    Implementasi Algoritma Find-S
    data: DataFrame fitur
    target: Series label (Yes/No)
    """
    # 1. Inisialisasi hipotesis dengan nilai paling spesifik
    # Mengambil jumlah kolom fitur
    specific_h = ['0'] * len(data.columns)

    print(f"H0 (Init): {specific_h}")

    # 2. Iterasi setiap baris data
    for i, row in data.iterrows():
        # Hanya proses jika labelnya POSITIF (Yes)
        if target[i] == "Yes":
            for x in range(len(specific_h)):
                # Jika hipotesis masih '0' (paling spesifik), ganti
dengan data
                if specific_h[x] == '0':
                    specific_h[x] = row[x]
                # Jika hipotesis tidak sama dengan data, generalisasi
menjadi '?'
                elif specific_h[x] != row[x]:
                    specific_h[x] = '?'

            print(f"H{i+1} (After instance {i+1}): {specific_h}")
        else:
            print(f"H{i+1} (Ignored Negative): {specific_h}")

    return specific_h

# Data 'EnjoySport' sesuai Transkrip
dataset = pd.DataFrame([
    ['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
], columns=['Sky', 'AirTemp', 'Humidity', 'Wind', 'Water',
'Forecast'])

target_label = pd.Series(['Yes', 'Yes', 'No', 'Yes'])

# Eksekusi
final_hypothesis = train_find_s(dataset, target_label)
print(f"\nHipotesis Akhir Find-S: {final_hypothesis}")
```

BAB IV
DECISION TREE (ID3) & EVALUASI MODEL

1. Deskripsi Singkat Bahasan

Bab ini meninggalkan algoritma sederhana (seperti Find-S) dan beralih ke Decision Tree, salah satu algoritma Machine Learning paling populer dan interpretatif. Fokus utamanya adalah algoritma ID3 (Iterative Dichotomiser 3) yang membangun pohon keputusan berdasarkan perhitungan matematis Entropy dan Information Gain. Selain itu, bab ini membahas metrik evaluasi model krusial bagi bisnis, yaitu Confusion Matrix (untuk mendeteksi risiko Fraud), serta konsep validasi model (Split Validation) untuk menghindari masalah Overfitting dan Underfitting.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 4. Diagram ini mencakup alur algoritma ID3 dan metrik evaluasi yang dibahas.

3. Penjelasan Detail

3.1 Representasi Tree dalam Memori (Computational View)

Secara visual, Decision Tree adalah diagram alir. Namun secara komputasi, ia adalah struktur data yang tersimpan di RAM. Setiap komponen pohon disebut Node.

- **Node Structure:** Sebuah Node dalam memori setidaknya menyimpan tiga informasi:
    1. **Data:** Label atribut (misal: "Outlook") atau keputusan (misal: "Yes").
    2. **Pointer to Children:** Alamat memori (*memory address*, misal 0x1A) yang menunjuk ke node anak.
    3. **Pointer to Parent:** Alamat memori node induk (untuk *backtracking*).

Dalam algoritma ID3, pohon yang dibentuk adalah **N-ary Tree** (bukan *Binary Tree*), artinya satu parent bisa memiliki N cabang (contoh: *Outlook* punya 3 cabang: Sunny, Overcast, Rain).

3.2 Eksekusi Manual ID3: Iterasi 1 (Root Selection)

Diberikan dataset PlayTennis dengan 14 sampel ($9+$ dan $5-$).

Langkah 1: Hitung Global Entropy (S)

$$Entropy(S) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \approx 0.940$$

Langkah 2: Hitung Information Gain untuk Setiap Atribut

Kita ambil contoh atribut Outlook. Atribut ini memecah data menjadi 3 partisi:

- **Sunny (5 data):** $2+, 3- \rightarrow Entropy = 0.971$
- **Overcast (4 data):** $4+, 0- \rightarrow Entropy = 0.0$ (Murni)
- **Rain (5 data):** $3+, 2- \rightarrow Entropy = 0.971$

Rumus Gain untuk Outlook:

$$Gain(S, Outlook) = Entropy(S) - \sum \frac{|S_v|}{|S|} Entropy(S_v)$$

$$Gain(S, Outlook) = 0.940 - \left[\frac{5}{14}(0.971) + \frac{4}{14}(0) + \frac{5}{14}(0.971)\right] \approx 0.246$$

Setelah dibandingkan dengan atribut lain (*Temperature* ≈ 0.029, *Humidity* ≈ 0.151, *Wind* ≈ 0.048), **Outlook** memiliki Gain tertinggi. Maka, **Outlook menjadi Root Node**.

3.3 Eksekusi Manual ID3: Iterasi 2 (Rekursi Cabang Sunny)

Karena cabang Overcast sudah murni (Entropy 0), ia langsung menjadi Leaf Node (Yes). Namun, cabang Sunny masih kotor (Entropy 0.971). Kita lakukan rekursi hanya pada data Sunny.

Atribut tersisa: Humidity, Temperature, Wind.

Setelah dihitung ulang menggunakan data yang difilter (Sunny only), atribut Humidity memberikan Gain tertinggi.

- Jika *Sunny* AND *Humidity* = *High* → No.
- Jika *Sunny* AND *Humidity* = *Normal* → Yes.

Maka, *Humidity* menjadi anak dari *Outlook (Sunny branch)*.

4. Sample Code (Implementasi ID3 dengan Pandas)

Berikut adalah implementasi perhitungan Entropy dan Gain secara manual menggunakan Python.

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

# Sample Data (Play Tennis - Sesuai Transkrip)
data = {
    'Outlook': ['Sunny', 'Sunny', 'Overcast', 'Rain', 'Rain', 'Rain',
'Overcast', 'Sunny', 'Sunny', 'Rain'],
    'Temperature': ['Hot', 'Hot', 'Hot', 'Mild', 'Cool', 'Cool',
'Cool', 'Mild', 'Cool', 'Mild'],
    'Humidity': ['High', 'High', 'High', 'High', 'Normal', 'Normal',
'Normal', 'High', 'Normal', 'Normal'],
    'Wind': ['Weak', 'Strong', 'Weak', 'Weak', 'Weak', 'Strong',
'Strong', 'Weak', 'Weak', 'Weak'],
    'PlayTennis': ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
'Yes', 'Yes']
}
```

```
df = pd.read_DataFrame(data)

def calculate_entropy(series):
    """Menghitung Entropy dari satu kolom target"""
    counts = series.value_counts()
    entropy = 0
    total = len(series)

    for count in counts:
        p = count / total
        entropy -= p * math.log2(p)

    return entropy

def calculate_information_gain(df, attribute, target_name):
    """Menghitung Information Gain untuk atribut tertentu"""
    # 1. Hitung Entropy Total (S)
    total_entropy = calculate_entropy(df[target_name])

    # 2. Hitung Entropy per Value dari Atribut
    values = df[attribute].unique()
    weighted_entropy = 0
    total_rows = len(df)

    for value in values:
        subset = df[df[attribute] == value]
        prob = len(subset) / total_rows
        weighted_entropy += prob *
calculate_entropy(subset[target_name])

    # 3. Hitung Gain
    gain = total_entropy - weighted_entropy
    return gain

# --- EKSEKUSI ---
print("--- Perhitungan Information Gain (Menentukan Root) ---")
target = 'PlayTennis'
features = ['Outlook', 'Temperature', 'Humidity', 'Wind']

best_gain = 0
best_feature = ''

for feature in features:
    gain = calculate_information_gain(df, feature, target)
    print(f"Gain({feature}): {gain:.4f}")

    if gain > best_gain:
        best_gain = gain
        best_feature = feature

print(f"\nKESIMPULAN: Root Node terbaik adalah '{best_feature}' dengan
Gain {best_gain:.4f}")
```

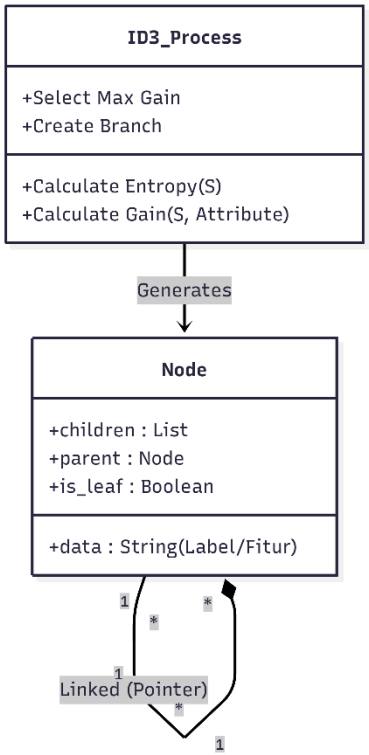
Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

BAB V
IMPLEMENTASI STRUKTUR DATA TREE & KALKULASI ID3

1. Deskripsi Singkat Bahasan

Bab ini berfokus pada dua aspek fundamental dari Decision Tree. Pertama, aspek Computational Structure, yaitu bagaimana pohon keputusan direpresentasikan dalam memori komputer (RAM) menggunakan konsep Node dan Pointer (alamat memori). Diskusi mencakup perbedaan Binary Tree dengan N-ary Tree yang digunakan pada ID3. Kedua, aspek Mathematical Execution, yaitu simulasi manual langkah demi langkah algoritma ID3 untuk menyelesaikan dataset PlayTennis, mulai dari perhitungan Global Entropy hingga terbentuknya rule keputusan final.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 5. Diagram ini menggambarkan struktur memori Node dan alur rekursif ID3.

3. Penjelasan Detail

3.1 Struktur Decision Tree & Algoritma ID3

Berbeda dengan *Find-S* yang linier, *Decision Tree* memecah data berdasarkan struktur hierarki:

- Root Node: Atribut pertama yang paling signifikan memisahkan data.
- Leaf Node: Hasil akhir keputusan (misal: Yes/No).

Algoritma ID3 menentukan atribut mana yang menjadi *Root* dengan menghitung Information Gain. Atribut dengan *Gain* tertinggi dipilih sebagai pemecah data.

3.2 Perhitungan Matematis (Entropy & Gain)

Dasar dari ID3 adalah konsep Entropi (ukuran ketidakteraturan data).

1. Entropy (H):

Mengukur seberapa tidak murninya (impure) suatu kumpulan data S. Jika semua datanya positif, Entropy = 0. Jika 50:50, Entropy = 1.

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

Dimana p_i adalah proporsi sampel yang termasuk dalam kelas i.

2. Information Gain ($Gain$):

   Mengukur pengurangan Entropi setelah data dipisahkan oleh atribut A.

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

   Semakin tinggi nilai $Gain$, semakin bagus atribut tersebut dijadikan Root.

3.3 Evaluasi Model: Confusion Matrix

Dalam kasus bisnis (seperti Fraud Detection), akurasi saja tidak cukup. Kita menggunakan Confusion Matrix:

- **True Positive (TP):** Prediksi Penipu, Faktanya Penipu.
- **False Positive (FP):** Prediksi Penipu, Faktanya Bukan (Customer marah, pindah toko).
- **False Negative (FN):** Prediksi Bukan Penipu, Faktanya Penipu (Bisnis rugi uang).

3.4 Overfitting vs Underfitting

Dosen menggunakan analogi ukuran baju:

- **Overfitting:** Model terlalu kompleks/hafal data training. Ibarat badan ukuran XL dipaksa pakai baju ukuran S (terlalu ketat). Akurasi training tinggi, tapi jelek saat tes.
- **Underfitting:** Model terlalu simpel. Ibarat badan ukuran M pakai baju XXL (terlalu longgar). Pola tidak tertangkap sama sekali.

4. Sample Code (Struktur Data Node)

Berikut adalah implementasi Class Node dalam Python yang merepresentasikan bagaimana struktur pohon disimpan dalam memori, sesuai penjelasan dosen tentang "alamat/pointer".

```
class Node:
    def __init__(self, name, parent=None):
        self.name = name          # Data (Contoh: "Outlook" atau
"Sunny")
        self.parent = parent      # Pointer ke Parent (Memory Address)
        self.children = {}        # Dictionary Pointer ke Children
        self.decision = None      # Jika Leaf, isinya "Yes/No"

    def add_child(self, branch_value, child_node):
        """
        Menambahkan cabang baru.
        branch_value: Label cabang (misal: "Sunny")
        child_node: Node tujuan (misal: Node "Humidity")
        """
        self.children[branch_value] = child_node
        child_node.parent = self  # Link back ke parent
```

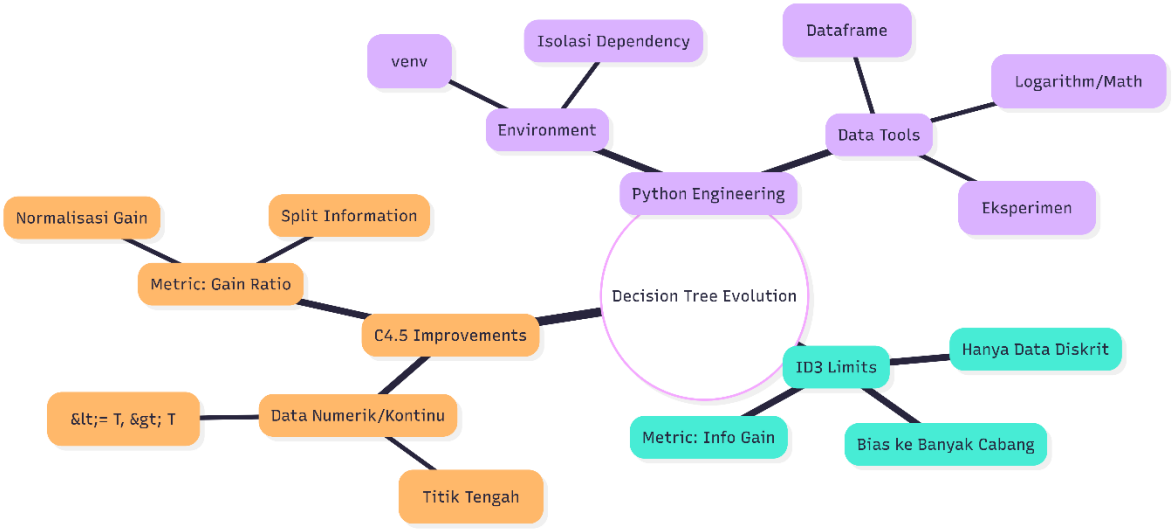
```
    def __repr__(self):
        return f"<Node: {self.name} | MemAddr: {hex(id(self))}>"

# Simulasi Pembentukan Tree Manual (Sesuai kalkulasi Bab 5)
# Level 0: Root
root = Node("Outlook")

# Level 1: Cabang dari Outlook
node_humidity = Node("Humidity", parent=root)
node_wind = Node("Wind", parent=root)
node_leaf_yes = Node("Leaf: Yes", parent=root) # Untuk Overcast

# Linking (Menyimpan Alamat Memori)
root.add_child("Sunny", node_humidity)
root.add_child("Overcast", node_leaf_yes)
root.add_child("Rain", node_wind)

# Level 2: Cabang dari Humidity (Sunny Branch)
leaf_no = Node("Leaf: No")
leaf_yes = Node("Leaf: Yes")

node_humidity.add_child("High", leaf_no)
node_humidity.add_child("Normal", leaf_yes)

# Cetak Struktur Memori
print(f"Root: {root}")
print(f"Children of Root: {root.children}")
print(f"Parent of Humidity Node: {node_humidity.parent}")
```

BAB VI
ALGORITMA C4.5 & PYTHON ENGINEERING

1. Deskripsi Singkat Bahasan

Bab ini membahas evolusi dari algoritma ID3 menuju C4.5, sebuah algoritma Decision Tree yang lebih canggih yang mampu menangani data kontinu (numerik) dan mengatasi bias pada atribut yang memiliki banyak nilai unik. Fokus teoretisnya adalah pengenalan metrik Split Information dan Gain Ratio serta mekanisme thresholding untuk memecah data numerik (seperti pada Iris Dataset). Di sisi praktis, bab ini menekankan standar engineering menggunakan Python: isolasi dependency menggunakan Virtual Environment, manipulasi data dengan Pandas, dan komputasi matematis dengan NumPy.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 6. Diagram ini memetakan perbedaan ID3 vs C4.5 dan stack teknologi yang diperkenalkan.

3. Penjelasan Detail

3.1 Keterbatasan ID3 & Solusi C4.5

Algoritma ID3 murni memiliki kelemahan fatal: ia cenderung memilih atribut dengan banyak cabang (highly branching attributes), seperti "ID Transaksi" atau "Tanggal", yang memiliki Information Gain tinggi namun tidak memiliki nilai prediktif untuk data baru.

C4.5 mengatasi ini dengan memperkenalkan **Gain Ratio**. Gain Ratio menormalisasi *Information Gain* dengan **Split Information**.

3.2 Formulasi Matematis C4.5

1. Split Information ($SplitInfo$):

   Mengukur seberapa luas data terpecah oleh atribut A.

$$SplitInfo_A(S) = -\sum_{j=1}^{v} \frac{|S_j|}{|S|} \log_2 \left(\frac{|S_j|}{|S|} \right)$$

2. Gain Ratio:

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

Atribut dengan $SplitInfo$ yang sangat tinggi (banyak pecahan kecil) akan menurunkan nilai Gain Ratio, sehingga mengurangi bias.

3.3 Handling Continuous Data (Data Numerik)

Salah satu fitur utama C4.5 yang dibahas adalah penanganan data numerik (contoh: Petal Length pada Iris Dataset). ID3 tidak bisa memproses angka $2.45, 2.50, 4.7$ secara langsung sebagai kategori.

Mekanisme C4.5:

1. Urutkan nilai numerik atribut A.
2. Tentukan titik potong kandidat (T) sebagai rata-rata dari dua nilai yang berurutan.

$$T_i = \frac{v_i + v_{i+1}}{2}$$

3. Ubah atribut menjadi biner: $A \leq T \; dan \; A > T$.
4. Hitung *Information Gain* untuk setiap kandidat T dan pilih yang memberikan Gain tertinggi.

Contoh Transkrip: Pada Iris Dataset, nilai 2.45 dipilih sebagai threshold pemisah antara *Setosa* dan spesies lainnya.

3.4 Python Engineering: Virtual Environment

Dalam rekayasa perangkat lunak AI, isolasi adalah kunci. Menginstal library secara global di sistem operasi (misal: /usr/bin/python) adalah praktik buruk karena dapat merusak sistem ("Gendut").

Solusinya adalah **Virtual Environment (venv)**:

- Membuat salinan Python lokal di dalam folder proyek.
- Perintah: python -m venv venv_name.
- Aktivasi: source venv/bin/activate (Linux/Mac) atau venv\Scripts\activate (Windows).

4. Sample Code (Implementasi Pandas & NumPy)

Berikut adalah kode Python yang merepresentasikan tentang penggunaan Pandas untuk menghitung Entropy secara otomatis, menggantikan perhitungan manual spreadsheet.

```
import pandas as pd
import numpy as np

def calculate_entropy(df, target_col):
    """
    Menghitung Entropy S menggunakan Pandas & NumPy.
    Rumus: - sum(p * log2(p))
    """
    # 1. Hitung distribusi kelas (Value Counts)
    counts = df[target_col].value_counts()

    # 2. Hitung probabilitas (p)
    probabilities = counts / len(df)

    # 3. Hitung Entropy (Menggunakan NumPy log2)
    # Menambahkan epsilon 1e-9 untuk menghindari log(0)
    entropy = -np.sum(probabilities * np.log2(probabilities + 1e-9))

    return entropy

def get_split_candidates(df, feature):
```

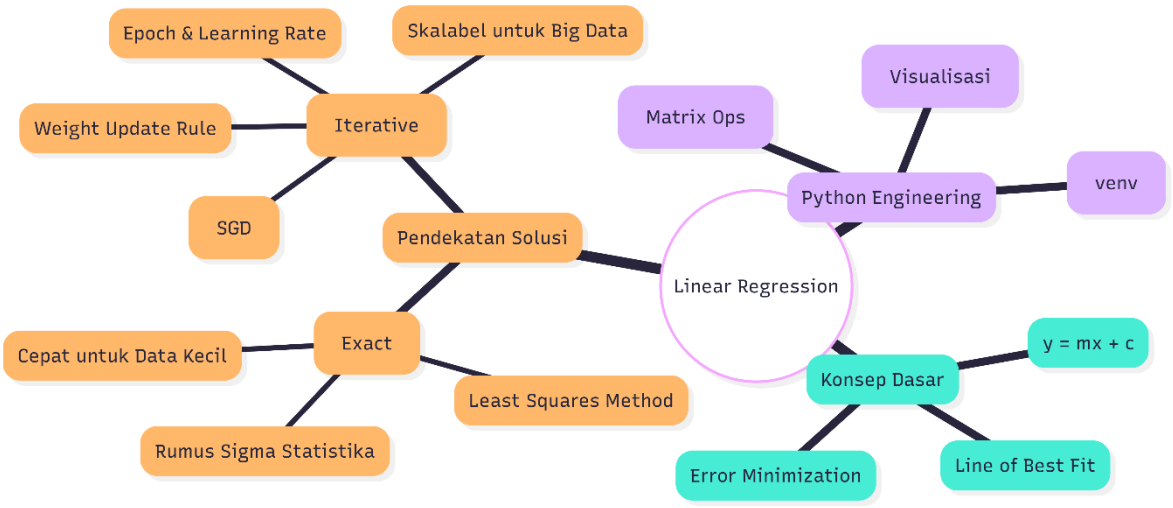
```
    """
    Mencari kandidat threshold (T) untuk data numerik (Logic C4.5)
    """
    unique_values = sorted(df[feature].unique())
    thresholds = []

    for i in range(len(unique_values) - 1):
        # Titik tengah antara dua nilai berurutan
        midpoint = (unique_values[i] + unique_values[i+1]) / 2
        thresholds.append(midpoint)

    return thresholds

# --- Simulasi Penggunaan ---
# Mock Data Iris Sederhana
data = {
    'PetalLength': [1.4, 1.4, 1.3, 4.7, 4.5, 4.9],
    'Species': ['Setosa', 'Setosa', 'Setosa', 'Versicolor',
'Versicolor', 'Versicolor']
}
df_iris = pd.DataFrame(data)

# 1. Hitung Entropy Total
total_entropy = calculate_entropy(df_iris, 'Species')
print(f"Total Entropy: {total_entropy:.4f}")

# 2. Cari Threshold untuk PetalLength
candidates = get_split_candidates(df_iris, 'PetalLength')
print(f"Kandidat Threshold (T): {candidates}")

# Output akan menunjukkan midpoint seperti (1.4+4.5)/2 = 2.95 sebagai
pemisah potensial
```

BAB VII
LINEAR REGRESSION & OPTIMISASI GRADEN

1. Deskripsi Singkat Bahasan

Bab ini membahas Linear Regression Single Variate, metode statistik untuk memodelkan hubungan linear antara variabel independen (X) dan dependen (Y). Fokus utama adalah perbandingan dua pendekatan solusi: Analitik (Metode Least Squares yang pasti/eksak) dan Numerik (Stochastic Gradient Descent yang iteratif/pendekatan). Selain itu, bab ini menekankan praktik Software Engineering dalam Python, seperti penggunaan Virtual Environment untuk isolasi proyek dan penggunaan library NumPy untuk komputasi matriks performa tinggi.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 7. Diagram ini memetakan dua jalur solusi Linear Regression.

3. Penjelasan Detail

3.1 Persamaan Dasar Linear Regression

Tujuan utama algoritma ini adalah mencari garis lurus yang paling representatif terhadap persebaran data. Secara matematis, garis lurus didefinisikan sebagai:

$$y = mx + c$$

Atau dalam notasi Machine Learning modern:

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

Dimana:

- m atau w_1: **Gradient/Slope** (Kemiringan garis). Menunjukkan seberapa kuat X mempengaruhi Y.
- c atau w_0: **Bias/Intercept** (Titik potong sumbu Y). Nilai dasar ketika $X = 0$.

3.2 Pendekatan Analitik (Least Squares)

Pendekatan ini menggunakan rumus statistik tertutup untuk langsung menemukan nilai m dan c yang optimal dalam satu langkah perhitungan.

Rumus untuk mencari gradien (m):

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

Rumus untuk mencari bias (c):

$$c = \bar{y} - m\bar{x}$$

Dimana $\bar{y}$ dan $\bar{x}$ adalah rata-rata (mean) dari data.

- **Kelebihan:** Hasil pasti (deterministik) dan akurat.
- **Kekurangan:** Sangat berat secara komputasi jika jumlah data (n) atau fitur (x) sangat besar (jutaan), karena melibatkan invers matriks atau perhitungan sigma kompleks.

3.3 Pendekatan Numerik (Stochastic Gradient Descent - SGD)

Ini adalah fondasi dari Deep Learning. Alih-alih mencari jawaban pasti, kita "menebak" dan memperbaiki tebakan tersebut berulang kali (Iteratif).

Algoritma SGD:

1. **Inisialisasi:** Tentukan nilai acak untuk m dan c (misal 0).
2. Prediksi ($\hat{y}$): Hitung nilai prediksi untuk satu data poin.
$$\hat{y} = mx_i + c$$
3. Hitung Error: Selisih antara prediksi dan fakta.
$$Error = \hat{y} - y_i$$
4. Update Bobot (Learning): Perbaiki m dan c berdasarkan error dan Learning Rate (α).
$$m_{new} = m_{old} - (\alpha \times Error \times x_i)$$
$$c_{new} = c_{old} - (\alpha \times Error)$$
5. **Iterasi (Epoch):** Ulangi langkah 2-4 untuk seluruh data berkali-kali hingga Error mendekati 0.

3.4 Konsep Epoch & Learning Rate

- **Epoch:** Satu putaran penuh melewati seluruh dataset. Semakin banyak Epoch, semakin garis mendekati titik optimal (namun hati-hati *Overfitting*).
- **Learning Rate (α):** Seberapa besar langkah perbaikan yang diambil. Jika terlalu besar, model tidak konvergen (melompat-lompat). Jika terlalu kecil, proses belajar sangat lama.

4. Sample Code (Implementasi Manual SGD)

Kode ini mensimulasikan tentang bagaimana SGD bekerja "di balik layar" tanpa menggunakan library scikit-learn. Ini menunjukkan logika for-loop untuk update bobot.

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

# Data Sampel (Luas Rumah vs Harga)
X = np.array([60, 75, 100, 140, 180]) # Luas (m2)
y = np.array([115, 130, 155, 190, 230]) # Harga (Juta)

def stochastic_gradient_descent(X, y, epochs=1000,
learning_rate=0.0001):
    """
    Implementasi Manual SGD untuk Linear Regression
```

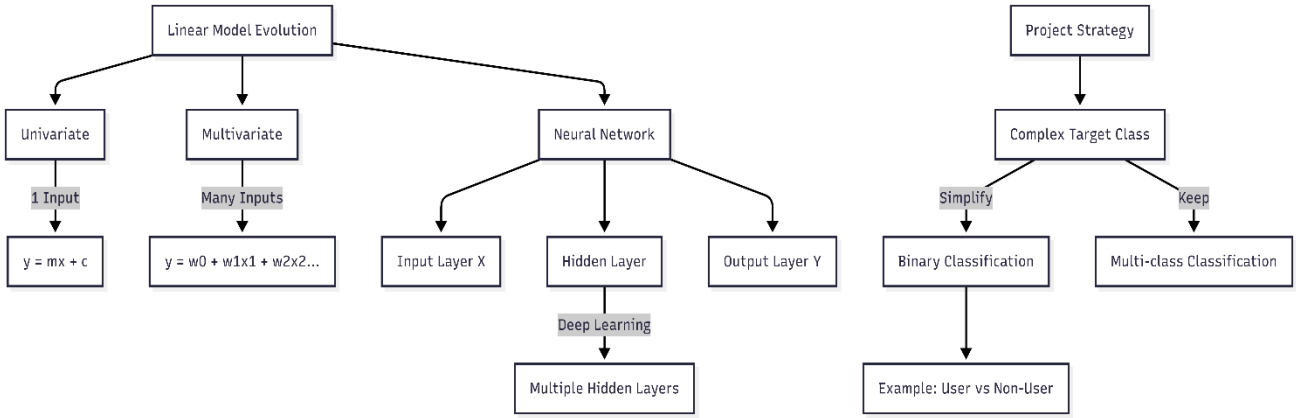
```
    """
    # 1. Inisialisasi Random (Tebakan Awal)
    m = 0.0 # Gradien
    c = 0.0 # Bias
    n = len(X)

    # Riwayat Error untuk visualisasi
    error_history = []

    # 2. Looping Epoch (Iterasi Belajar)
    for epoch in range(epochs):
        epoch_error = 0

        # Iterasi per data point (Stochastic)
        for i in range(n):
            # A. Prediksi
            y_pred = m * X[i] + c

            # B. Hitung Error
            error = y_pred - y[i]
            epoch_error += abs(error) # Akumulasi error absolut

            # C. Update Bobot (Learning Rule)
            # m_new = m_old - (LR * Error * x)
            m = m - (learning_rate * error * X[i])
            # c_new = c_old - (LR * Error)
            c = c - (learning_rate * error)

        error_history.append(epoch_error/n)

        # Print progress setiap 100 epoch
        if epoch % 200 == 0:
            print(f"Epoch {epoch}: m={m:.4f}, c={c:.4f}, Mean
Error={epoch_error/n:.4f}")

    return m, c, error_history

# --- EKSEKUSI ---
print("--- Memulai Training SGD ---")
m_final, c_final, errors = stochastic_gradient_descent(X, y,
epochs=2000, learning_rate=0.00001)

print(f"\nModel Akhir: y = {m_final:.2f}x + {c_final:.2f}")

# Prediksi Harga Rumah 100m2
prediksi_100 = m_final * 100 + c_final
print(f"Prediksi Harga untuk 100m2: {prediksi_100:.2f} Juta")

# Visualisasi
plt.scatter(X, y, color='blue', label='Data Asli')
plt.plot(X, m_final*X + c_final, color='red', label='Garis Regresi
(SGD)')
plt.legend()
plt.show()
```

BAB VIII
TRANSISI KE NEURAL NETWORK & STRATEGI DATASET

1. Deskripsi Singkat Bahasan

Bab ini membahas evolusi model dari Univariate Linear Regression (satu input) menjadi Multivariate Linear Regression (banyak input), yang menjadi prinsip dasar dari satu neuron dalam Neural Network. Fokus utama adalah pemahaman arsitektur jaringan saraf tiruan (Input Layer, Hidden Layer, Output Layer) dan bagaimana kompleksitas model (seperti LLM 7B parameter) terbentuk dari miliaran koneksi bobot (W). Selain itu, bab ini memberikan panduan strategis untuk menyederhanakan Target Class pada dataset yang kompleks (studi kasus: Dataset Konsumsi Obat) agar layak dikerjakan dalam lingkup Tugas Besar.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 8. Diagram ini menggambarkan evolusi dari Regresi ke Neural Network dan strategi penyederhanaan data.

3. Penjelasan Detail

3.1 Dari Univariate ke Multivariate

Pada bab sebelumnya, kita mempelajari Univariate Linear Regression:

$$f(x) = m \cdot x + c$$

Dalam notasi Machine Learning, persamaan ini sering ditulis ulang menggunakan bobot (W) dan bias (W_0):

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

Namun, kasus dunia nyata jarang memiliki satu faktor penentu. Contoh: Harga rumah tidak hanya ditentukan oleh luas (x_1), tapi juga lokasi (x_2), jumlah kamar (x_3), dll. Maka lahirlah Multivariate Linear Regression:

$$f(x) = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + \cdots + w_nx_n$$

Atau dalam notasi Sigma:

$$f(x) = w_0 + \sum_{i=1}^{n} w_i x_i$$

3.2 Arsitektur Neural Network (Jaringan Saraf Tiruan)

Konsep Multivariate di atas adalah dasar dari satu Neuron.

- **Input Layer:** Tempat masuknya fitur (x_1, x_2, x_3).
- **Weights (W):** Garis penghubung antar neuron. Inilah "pengetahuan" yang dipelajari model.
- **Hidden Layer:** Lapisan di tengah yang memproses fitur. Penambahan *Hidden Layer* memungkinkan model menangkap pola non-linear yang kompleks.
- **Output Layer:** Hasil prediksi (Y).

Ketika kita berbicara tentang model LLM (*Large Language Model*) seperti **DeepSeek 7B**, angka "7B" (7 Miliar) merujuk pada jumlah parameter (W) atau koneksi antar neuron yang ada dalam model tersebut.

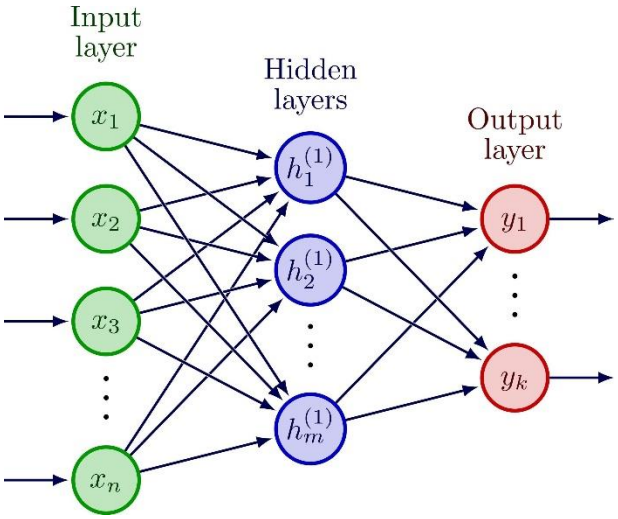
3.3 Strategi Simplifikasi Dataset (Studi Kasus)

Dalam review proposal tugas besar, dibahas kasus dataset dengan Target Class yang sangat kompleks (misal: 18 kategori penggunaan narkoba).

- **Masalah:** Terlalu banyak kelas target (*Multi-class*) dengan fitur yang sedikit dapat menyebabkan akurasi rendah.
- **Solusi Strategis:** Lakukan **Simplifikasi Kelas**.
    - Ubah 18 kelas menjadi 2 kelas saja (*Binary Classification*).
    - Contoh: Menggabungkan "Used in Last Decade", "Used Last Year", "Used Last Month" menjadi satu kelas **"User"**, dan sisanya **"Non-User"**.

4. Sample Code (Simulasi Forward Pass Neural Network)

Kode ini mensimulasikan bagaimana data mengalir dari Input ke Output melalui Hidden Layer menggunakan operasi matriks (Dot Product), merepresentasikan konsep matematika yang dijelaskan.




```
import numpy as np

class SimpleNeuralNetwork:
    """
    Simulasi Forward Pass sederhana dari Neural Network
    (Tanpa Backpropagation/Training) untuk memahami arsitektur.
    """
    def __init__(self, input_size, hidden_size, output_size):
        # Inisialisasi Bobot (Weights) secara acak
        # W1 menghubungkan Input -> Hidden
```

```
        self.W1 = np.random.randn(input_size, hidden_size)
        # W2 menghubungkan Hidden -> Output
        self.W2 = np.random.randn(hidden_size, output_size)

        # Bias
        self.b1 = np.zeros((1, hidden_size))
        self.b2 = np.zeros((1, output_size))

    def forward(self, X):
        """
        X: Data Input (Fitur)
        Rumus: Output = Activation(Input . W + b)
        """
        # 1. Input ke Hidden Layer
        z1 = np.dot(X, self.W1) + self.b1
        a1 = self.sigmoid(z1) # Fungsi Aktivasi

        # 2. Hidden ke Output Layer
        z2 = np.dot(a1, self.W2) + self.b2
        output = self.sigmoid(z2)

        return output

    def sigmoid(self, s):
        # Fungsi aktivasi sederhana untuk mengubah nilai menjadi 0-1
        return 1 / (1 + np.exp(-s))

# --- Simulasi ---
# Kasus: 3 Fitur Input (misal: Luas, Kamar, Lokasi)
X_input = np.array([[0.5, 0.2, 0.9]])

# Arsitektur: 3 Input -> 4 Neuron Hidden -> 1 Output (Harga)
nn = SimpleNeuralNetwork(input_size=3, hidden_size=4, output_size=1)

prediction = nn.forward(X_input)

print(f"Input Features: {X_input}")
print(f"Bobot W1 (Input->Hidden):\n{nn.W1}")
print(f"Hasil Prediksi (Output Layer): {prediction[0][0]:.4f}")
```

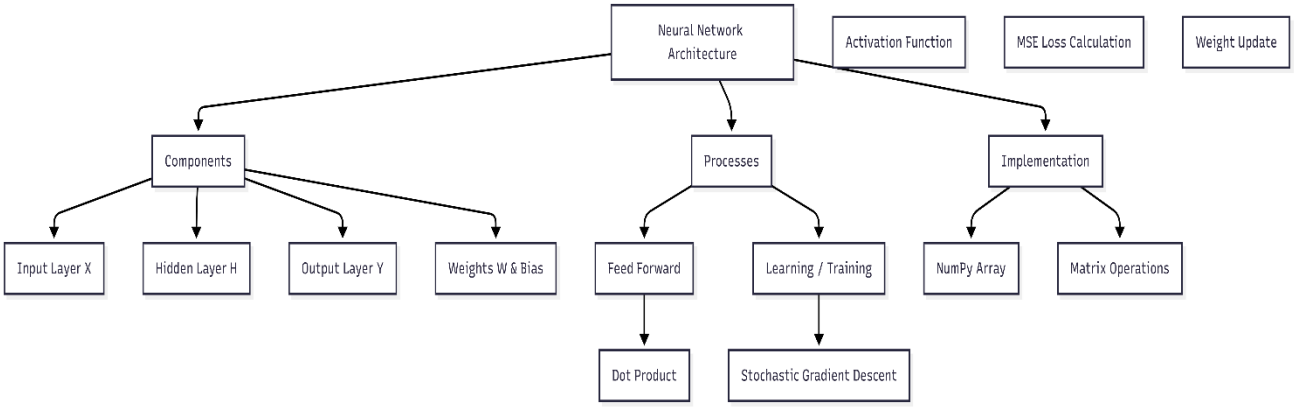
Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

BAB IX
NEURAL NETWORK & FEED FORWARD

1. Deskripsi Singkat Bahasan

Bab ini membahas evolusi konseptual dari Multivariate Linear Regression menuju arsitektur Neural Network (NN). Fokus utamanya adalah memahami bagaimana sebuah neuron bekerja secara matematis (penjumlahan berbobot + bias) dan bagaimana neuron-neuron tersebut disusun menjadi lapisan (Input, Hidden, Output). Bab ini juga mendetailkan algoritma Feed Forward (proses data mengalir dari input ke output) dan Stochastic Gradient Descent (SGD) sebagai metode optimasi untuk mencari nilai bobot (W) terbaik. Diskusi teknis mencakup implementasi struktur data matriks menggunakan NumPy untuk efisiensi komputasi.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 9. Diagram ini menggambarkan struktur Neural Network dan alur algoritma Feed Forward.

3. Penjelasan Detail

3.1 Dari Regresi ke Neuron

Sebuah neuron dalam Neural Network pada dasarnya adalah fungsi Multivariate Linear Regression. Jika kita memiliki input $x_1, x_2, \dots, x_n$, maka output neuron (y) sebelum fungsi aktivasi adalah:

$$y = w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2 + \cdots + w_n \cdot x_n$$

Dimana x_0 adalah Bias yang nilainya selalu 1.

Dalam notasi Sigma (Σ):

$$y = \sum_{i=0}^{n} w_i x_i$$

3.2 Arsitektur Neural Network (Multilayer Perceptron)

Kekuatan Neural Network terletak pada Hidden Layer. Arsitektur standar terdiri dari:

1. **Input Layer:** Lapisan fitur (misal: luas rumah, lokasi).
2. **Hidden Layer:** Lapisan "Black Magic" yang mengekstraksi fitur kompleks. Jumlah neuron di sini menentukan kapasitas model. Model seperti LLM (DeepSeek/GPT) memiliki miliaran parameter (W) di lapisan ini.
3. **Output Layer:** Hasil prediksi.

Hubungan antar lapisan didefinisikan oleh matriks bobot (W). Misalnya, W_{ij} adalah bobot yang menghubungkan neuron i di layer sebelumnya ke neuron j di layer berikutnya.

3.3 Algoritma Feed Forward (Maju)

Proses menghitung prediksi dari input disebut Feed Forward.

1. **Inisialisasi:** Bobot W diisi nilai random (misal -0.5 s.d 0.5).
2. **Kalkulasi:**
   - Nilai neuron di Hidden Layer (H) dihitung dari Input (X).
   - Nilai Output (O) dihitung dari Hidden Layer (H).

$$H = f\left(\sum W_{input \to hidden} \cdot X\right)$$

$$O = f\left(\sum W_{hidden \to output} \cdot H\right)$$

3.4 Stochastic Gradient Descent (SGD) & MSE

Setelah mendapatkan Output (O), kita hitung kesalahannya dibandingkan Target Asli (Y).

- Mean Squared Error (MSE):

$$MSE = \frac{1}{2n}\sum (Y - O)^2$$

- Learning (Update Bobot):

  Bobot diperbarui sedikit demi sedikit (iteratif) untuk mengurangi error.

$$W_{baru} = W_{lama} - (\alpha \times \text{Gradient})$$

  Dimana α adalah Learning Rate (misal 0.01).

4. Sample Code (Simulasi Feed Forward dengan NumPy)

Kode ini mengimplementasikan perkalian matriks (Dot Product) untuk simulasi Feed Forward secara manual.

```
import numpy as np

def sigmoid(x):
    """Fungsi Aktivasi Sigmoid: Mengubah nilai menjadi range 0-1"""
    return 1 / (1 + np.exp(-x))

# 1. Inisialisasi Data & Bobot
# Input: 3 Fitur (Bias x0=1, x1, x2)
X = np.array([1.0, 0.5, 0.2])

# Bobot Layer 1 (Input -> Hidden)
# Misal: 3 Input Node -> 4 Hidden Node
W1 = np.random.uniform(-0.5, 0.5, (3, 4))

# Bobot Layer 2 (Hidden -> Output)
# Misal: 4 Hidden Node -> 1 Output Node
W2 = np.random.uniform(-0.5, 0.5, (4, 1))

print("--- Initial Weights ---")
print(f"W1 Shape: {W1.shape}")
print(f"W2 Shape: {W2.shape}")
```

```
# 2. Proses Feed Forward
# Step A: Hitung nilai Hidden Layer
# z1 = X dot W1
z1 = np.dot(X, W1)
a1 = sigmoid(z1) # Aktivasi
print(f"\nHidden Layer Output:\n{a1}")

# Step B: Hitung nilai Output Layer
# z2 = a1 dot W2
z2 = np.dot(a1, W2)
output = sigmoid(z2) # Output akhir (y_pred)

print(f"\nFinal Output (Prediction): {output[0]:.4f}")

# 3. Simulasi Error Calc (MSE)
y_target = 1.0 # Misal target asli adalah 1
mse = 0.5 * (y_target - output)**2
print(f"MSE Loss: {mse[0]:.4f}")
```

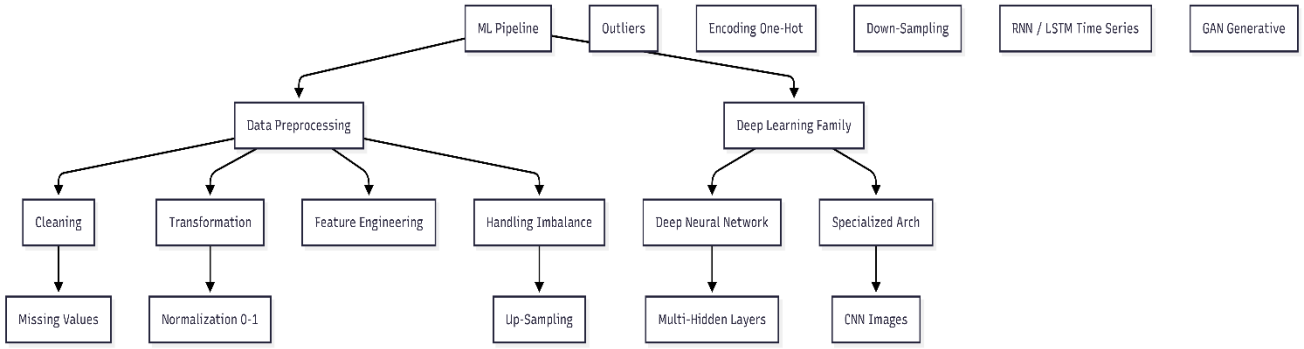
Vol. xx, No. x, mmmm yyyy, ISSN : 2089-9033, e- ISSN : 2715-7849

BAB X
DATA PREPROCESSING & TAKSONOMI DEEP LEARNING

1. Deskripsi Singkat Bahasan

Bab ini membahas ekosistem lengkap Machine Learning Pipeline. Dimulai dari tahap Data Preprocessing, yaitu teknik membersihkan dan mengubah data mentah agar layak dikonsumsi algoritma (termasuk penanganan Missing Values, Normalization, One-Hot Encoding, dan Imbalanced Data). Selanjutnya, bab ini memperluas wawasan ke ranah Deep Learning, membahas evolusi dari Neural Network standar menuju arsitektur yang lebih kompleks seperti Convolutional Neural Network (CNN) untuk citra, Long Short-Term Memory (LSTM) untuk data runtun waktu, dan Generative Adversarial Network (GAN).

2. Mindmap / Taksonomi (Visualisasi)



Gambar 10. Diagram ini memetakan tahapan Preprocessing dan varian Deep Learning.

3. Penjelasan Detail

3.1 Data Preprocessing: Fondasi Model

Data mentah di dunia nyata seringkali "kotor". Tanpa preprocessing, model akan mengalami Garbage In, Garbage Out.

- **Encoding:** Mengubah data kategori (teks) menjadi angka.
  - *Label Encoding:* Fraud=1, Legal=0.
  - *One-Hot Encoding:* Membuat kolom baru biner (misal: is_Jakarta, is_Bandung) untuk menghindari urutan angka yang menyesatkan.
- Normalization (Min-Max Scaling):

  Menyamakan skala data (misal: Gaji 10 Juta vs Umur 25 Tahun) agar gradien turun lebih cepat (konvergen).

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

  Dengan ini, semua data berada dalam rentang $[0,1]$.

3.2 Handling Imbalanced Data

Dalam kasus deteksi penipuan (Fraud), data seringkali tidak seimbang (misal: 10.000 Transaksi Legal vs 100 Penipuan). Jika dibiarkan, model akan bias ke kelas mayoritas.

- **Up-Sampling (Oversampling):** Menduplikasi data minoritas (Fraud) agar jumlahnya setara.
- **Down-Sampling (Undersampling):** Memangkas data mayoritas (Legal).

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

3.3 Taksonomi Deep Learning

Deep Learning adalah Neural Network dengan banyak Hidden Layer.

1. Convolutional Neural Network (CNN):

   Spesialis data Grid (Gambar). Menggunakan teknik Convolution dan Pooling untuk mengekstrak fitur visual secara otomatis.

2. Recurrent Neural Network (RNN) & LSTM:

   Spesialis data Runtun Waktu (Time Series) atau Teks. LSTM (Long Short-Term Memory) mengatasi masalah "lupa" pada data panjang dengan mekanisme Gate.

   Konsep dasarnya mirip Markov Chain: $t_{sekarang}$ dipengaruhi oleh t_{lalu}.

3. Generative Adversarial Network (GAN):

   Dua jaringan yang saling beradu: Generator (pembuat palsu) vs Discriminator (polisi). Hasilnya adalah data sintetis yang sangat realistis.

4. Sample Code (Preprocessing Pipeline)

Kode ini mensimulasikan teknik preprocessing lengkap: Normalisasi manual dan One-Hot Encoding menggunakan Pandas.

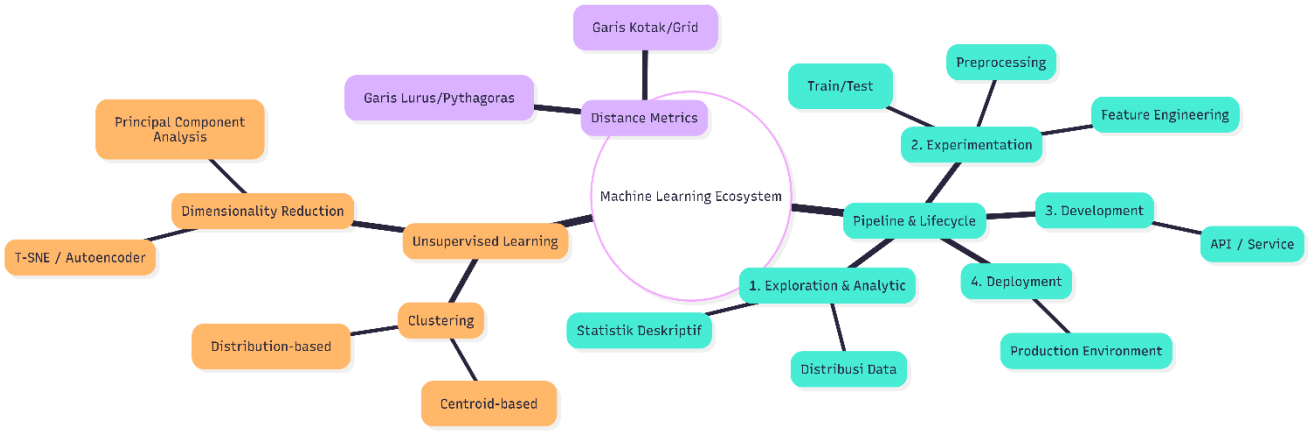
```
import pandas as pd
import numpy as np

# Simulasi Dataset Transaksi (Kotor)
data = {
    'Amount': [1000000, 50000, 2500000, 10000000, 150000],
    'City': ['Jakarta', 'Bandung', 'Jakarta', 'Surabaya', 'Bandung'],
    'Is_Fraud': ['No', 'No', 'No', 'Yes', 'No']
}

df = pd.DataFrame(data)

def normalize_minmax(series):
    """Melakukan Min-Max Scaling ke range 0-1"""
    return (series - series.min()) / (series.max() - series.min())

print("--- Data Asli ---")
print(df)

# 1. Normalisasi Amount
df['Amount_Norm'] = normalize_minmax(df['Amount'])

# 2. One-Hot Encoding untuk City
# Mengubah kolom 'City' menjadi 'City_Jakarta', 'City_Bandung', dll.
df_encoded = pd.get_dummies(df, columns=['City'], prefix='City')

# 3. Label Encoding untuk Target
df_encoded['Is_Fraud'] = df_encoded['Is_Fraud'].map({'Yes': 1, 'No':
0})

print("\n--- Data Setelah Preprocessing ---")
print(df_encoded[['Amount_Norm', 'City_Jakarta', 'City_Bandung',
'City_Surabaya', 'Is_Fraud']])
```

BAB XI
MACHINE LEARNING PIPELINE & UNSUPERVISED LEARNING

1. Deskripsi Singkat Bahasan

Bab ini membahas dua topik besar. Pertama, Machine Learning Pipeline, yaitu alur kerja end-to-end dalam membangun solusi AI, mulai dari Data Exploration, Experimentation (Preprocessing & Modeling), hingga Deployment ke produksi. Kedua, bab ini memperkenalkan Unsupervised Learning, paradigma di mana data tidak memiliki label atau target kelas. Algoritma yang dibahas meliputi K-Means (berbasis Centroid) dan Gaussian Mixture Model (GMM) (berbasis distribusi probabilitas) untuk Clustering, serta Principal Component Analysis (PCA) untuk reduksi dimensi data yang kompleks.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 11. Diagram ini menggambarkan alur Pipeline dan cabang Unsupervised Learning.

3. Penjelasan Detail

3.1 Machine Learning Pipeline

Membangun AI bukan hanya soal coding model, melainkan sebuah siklus rekayasa (Engineering).

1. **Data Exploration & Analytic:** Memahami karakteristik statistik data.
2. **Experimentation:** Fase "laboratorium" di mana *Data Scientist* mencoba berbagai teknik *Preprocessing* dan algoritma.
3. **Development:** Membungkus model menjadi layanan (misal: REST API).
4. **Deployment:** Menerapkan model ke lingkungan nyata (*Production*).

3.2 Konsep Unsupervised Learning

Berbeda dengan Supervised Learning yang memiliki target Y (misal: Harga Rumah, Fraud/Legal), Unsupervised Learning hanya memiliki data input X.

- **Tujuan:** Menemukan struktur tersembunyi atau pola pengelompokan (*Clustering*) dalam data.
- **Analogi:** Kita melihat sekelompok pembeli di mall. Kita tidak tahu siapa mereka, tapi kita bisa mengelompokkan mereka berdasarkan perilaku belanja (misal: "Kelompok Belanja Besar" vs "Kelompok Hemat").

3.3 Algoritma Clustering

1. K-Means:

Algoritma ini menentukan k titik pusat (Centroid) secara acak, lalu mengelompokkan data berdasarkan jarak terdekat ke centroid tersebut.

- Jarak biasanya diukur menggunakan Euclidean Distance:

$$d(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

- Atau Manhattan Distance (jarak blok kota):

$$d(p,q) = \sum_{i=1}^{n}|p_i - q_i|$$

2. Gaussian Mixture Model (GMM):

   Mengasumsikan bahwa data berasal dari beberapa distribusi normal (Gaussian). Setiap "gunung" atau puncak distribusi merepresentasikan satu klaster.

$$P(x) = \sum_{i=1}^{k} w_i \mathcal{N}(x|\mu_i, \Sigma_i)$$

3.4 Dimensionality Reduction (PCA)

Manusia hanya bisa memvisualisasikan maksimal 3 dimensi (X, Y, Z). Jika data memiliki 100 fitur (100 dimensi), kita mengalami Curse of Dimensionality.

PCA (Principal Component Analysis) mereduksi dimensi dengan memproyeksikan data ke sumbu baru (Principal Components) yang mempertahankan variansi (informasi) terbesar, membuang fitur yang kurang relevan.

4. Sample Code (K-Means & PCA dengan Scikit-Learn)

Kode ini mensimulasikan pengelompokan data customer menggunakan K-Means dan mereduksi visualisasinya menggunakan PCA.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

# 1. Simulasi Data (4 Fitur: Usia, Gaji, Skor Belanja, Jam Berkunjung)
# Kita punya 4 dimensi, sulit divisualisasikan langsung.
data = np.array([
    [25, 5000, 80, 2], [28, 5500, 85, 3], [30, 6000, 90, 2], # Kelompok A
    [50, 15000, 20, 10], [55, 16000, 15, 11], [60, 17000, 10, 12], # Kelompok B
    [20, 2000, 40, 18], [22, 2500, 45, 19], [19, 1800, 35, 17]  # Kelompok C
])

# 2. Implementasi K-Means (Clustering)
kmeans = KMeans(n_clusters=3, random_state=42)
labels = kmeans.fit_predict(data)
centroids = kmeans.cluster_centers_

print("Label Klaster untuk setiap data:", labels)
```

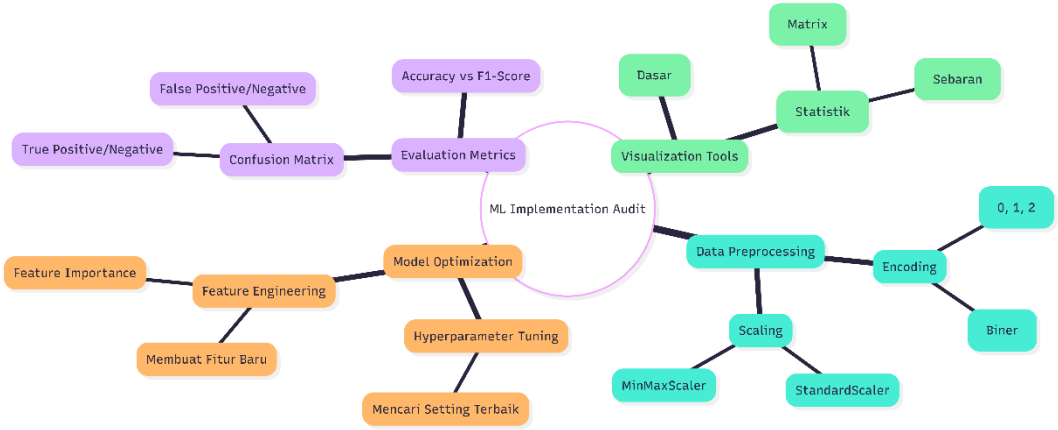
```
print("Koordinat Centroid (4 Dimensi):\n", centroids)

# 3. Implementasi PCA (Reduksi Dimensi untuk Visualisasi)
# Mengubah 4 dimensi menjadi 2 dimensi
pca = PCA(n_components=2)
data_2d = pca.fit_transform(data)
centroids_2d = pca.transform(centroids)

# 4. Visualisasi Hasil
plt.scatter(data_2d[:, 0], data_2d[:, 1], c=labels, cmap='viridis',
label='Data Points')
plt.scatter(centroids_2d[:, 0], centroids_2d[:, 1], s=200, c='red',
marker='X', label='Centroids')
plt.title('Visualisasi Clustering K-Means dengan PCA (4D -> 2D)')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.legend()
plt.show()
```

BAB XII
REVIEW IMPLEMENTASI & EVALUASI MODEL

1. Deskripsi Singkat Bahasan

Bab ini merupakan evaluasi teknis terhadap implementasi proyek Machine Learning. Fokus utama adalah mengidentifikasi dan memperbaiki kesalahan umum dalam pipeline ML, seperti teknik Encoding (transformasi data kategorikal ke numerik), pemahaman Confusion Matrix untuk evaluasi performa klasifikasi, serta peran vital Visualisasi Data (menggunakan Library Seaborn/Matplotlib) dalam menjelaskan hasil prediksi. Bab ini juga membahas konsep Feature Importance untuk mengetahui atribut mana yang paling berpengaruh terhadap keputusan model.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 12. Diagram ini memetakan komponen evaluasi proyek ML.

3. Penjelasan Detail

3.1 Teknik Encoding: Kapan Pakai 0/1?

Dalam review, ditemukan kasus di mana mahasiswa mengubah label teks (misal: "Bisnis" vs "Kecimen") menjadi angka 0 dan 1. Ini adalah Label Encoding.

- Label Encoding: Cocok untuk data ordinal (bertingkat) atau target kelas biner.

$$Teks \rightarrow [0,1,2,\dots]$$

- **One-Hot Encoding:** Wajib digunakan jika data nominal tidak memiliki urutan (misal: Warna Merah, Hijau, Biru) agar model tidak salah menginterpretasikan urutan angka.

3.2 Interpretasi Confusion Matrix

Akurasi 87% terlihat bagus, tapi di mana letak kesalahannya? Confusion Matrix menjawab ini.

Matriks C berukuran $N \times N$ (dimana N adalah jumlah kelas):

$$C_{ij} = \text{Jumlah observasi yang diketahui masuk grup } i \text{ tapi diprediksi masuk grup } j$$

- **Diagonal Utama (C_{ii}):** Prediksi Benar (True Positive/True Negative).
- **Off-Diagonal ($C_{ij}, i \neq j$):** Error (False Positive/False Negative).
    - *Kasus Transkrip:* Model salah memprediksi 19 data "Kecimen" sebagai kelas lain. Ini insight bisnis yang lebih berharga daripada sekadar "Akurasi 87%".

3.3 Feature Importance & Visualisasi

Mengapa model memilih "Yes"? Feature Importance (pada algoritma Tree-based seperti Random Forest) memberikan skor kontribusi setiap fitur.

$$Importance_j = \sum_{t \in Trees} \text{Improvement}_j(t)$$

Visualisasi menggunakan Bar Chart atau Heatmap (dari library Seaborn) sangat krusial untuk menjelaskan "Black Box" model kepada stakeholder non-teknis.

4. Sample Code (Evaluasi & Visualisasi)

Kode ini mensimulasikan pembuatan Confusion Matrix dan Plot Feature Importance yang dibahas dalam review.

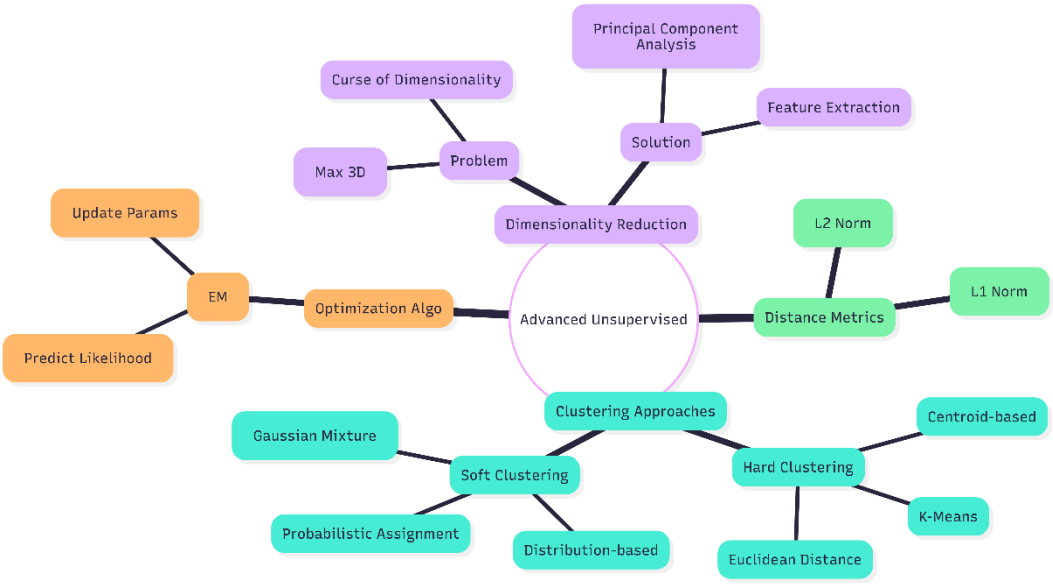
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier

# 1. Simulasi Data & Model
# Fitur: [Umur, Gaji, Skor Kredit]
X = np.random.rand(100, 3)
y = np.random.randint(0, 2, 100) # Target Biner (0/1)
features = ['Umur', 'Gaji', 'Skor Kredit']

model = RandomForestClassifier()
model.fit(X, y)
y_pred = model.predict(X)

# 2. Visualisasi Confusion Matrix (Evaluasi)
cm = confusion_matrix(y, y_pred)

plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Prediksi: 0', 'Prediksi: 1'],
            yticklabels=['Asli: 0', 'Asli: 1'])
plt.title('Confusion Matrix: Analisis Error')
plt.show()

# 3. Visualisasi Feature Importance (Interpretasi)
importances = model.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(8, 4))
plt.title('Feature Importances: Fitur Mana yang Dominan?')
plt.barh(range(len(indices)), importances[indices], color='b',
align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```

BAB XIII
ALGORITMA CLUSTERING LANJUT & REDUKSI DIMENSI

1. Deskripsi Singkat Bahasan

Bab ini mengeksplorasi dua algoritma clustering utama dengan pendekatan yang berbeda: K-Means (berbasis jarak/sentroid) dan Gaussian Mixture Model (GMM) (berbasis distribusi probabilitas). Diskusi mencakup algoritma Expectation-Maximization (EM) yang menjadi mesin pembelajaran bagi GMM. Selain itu, bab ini membahas tantangan Curse of Dimensionality ketidakmampuan manusia memvisualisasikan data di atas 3 dimensi dan solusinya menggunakan Principal Component Analysis (PCA), serta metrik pengukuran jarak (Distance Metrics) seperti Euclidean dan Manhattan.

2. Mindmap / Taksonomi (Visualisasi)



Gambar 13. Diagram ini memetakan perbedaan fundamental antara K-Means dan GMM serta konsep pendukungnya.

3. Penjelasan Detail

3.1 K-Means vs Gaussian Mixture Model (GMM)

Perbedaan mendasar kedua algoritma ini terletak pada cara pandang terhadap data:

- K-Means (Hard Clustering):

  Mengasumsikan klaster berbentuk bola (spherical). Setiap titik data secara tegas dimasukkan ke satu klaster terdekat berdasarkan jarak ke sentroid.

- GMM (Soft Clustering):

  Mengasumsikan data dihasilkan dari kombinasi beberapa distribusi Gaussian (Normal). Data tidak "dimiliki" oleh satu klaster, melainkan memiliki probabilitas keanggotaan.

  Model GMM dideskripsikan oleh dua parameter utama untuk setiap klaster k:

  1. **Mu (μ_k):** Titik tengah (puncak "gunung" distribusi).
  2. **Sigma (Σ_k):** KovaSSrians (lebar/penyebaran "gunung").

Fungsi densitas probabilitas Gaussian:

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^d|\Sigma|}}\exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

3.2 Algoritma Expectation-Maximization (EM)

Jika Linear Regression menggunakan SGD untuk meminimalkan Error, GMM menggunakan EM untuk memaksimalkan Likelihood (kemiripan).

1. **E-Step (Expectation):** Menghitung probabilitas setiap data x_i berasal dari klaster k berdasarkan parameter saat ini.
2. M-Step (Maximization): Memperbarui parameter μ_k (geser puncak gunung) dan Σ_k (ubah lebar gunung) agar lebih pas dengan data, berdasarkan bobot probabilitas dari E-Step.

$$\mu_k^{baru} = \frac{\sum w_{ik}x_i}{\sum w_{ik}}$$

3.3 Dimensionality Reduction (PCA)

Manusia hanya mampu memvisualisasikan maksimal 3 dimensi (X, Y, Z). Data dengan ribuan fitur sulit dianalisis.

PCA (Principal Component Analysis) bekerja dengan memutar sumbu data untuk menemukan arah (Principal Component) yang memiliki variansi (informasi) terbesar, lalu membuang dimensi yang variansinya kecil.

3.4 Distance Metrics (Metrik Jarak)

Dalam K-Means dan KNN, "kedekatan" diukur secara matematis:

- Euclidean Distance: Jarak garis lurus (Pythagoras).

$$d(x,y) = \sqrt{\sum(x_i - y_i)^2}$$

- Manhattan Distance: Jarak blok kota (Grid).

$$d(x,y) = \sum|x\_i - y\_i|$$

4. Sample Code (GMM & PCA Visualization)

Kode ini mensimulasikan pembuatan data 4 dimensi, mengelompokkannya dengan GMM, lalu menggunakan PCA untuk memvisualisasikannya dalam 2 dimensi.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.decomposition import PCA
from sklearn.datasets import make_blobs

# 1. Generate Data Dummy (4 Dimensi, 3 Klaster)
# Kita tidak bisa melihat 4D, jadi kita butuh PCA nanti.
X, y_true = make_blobs(n_samples=300, n_features=4, centers=3,
cluster_std=1.0, random_state=42)

# 2. Modeling dengan GMM (Expectation Maximization)
gmm = GaussianMixture(n_components=3, random_state=42)
gmm.fit(X)
labels = gmm.predict(X)
probs = gmm.predict_proba(X) # Soft clustering probabilities
```

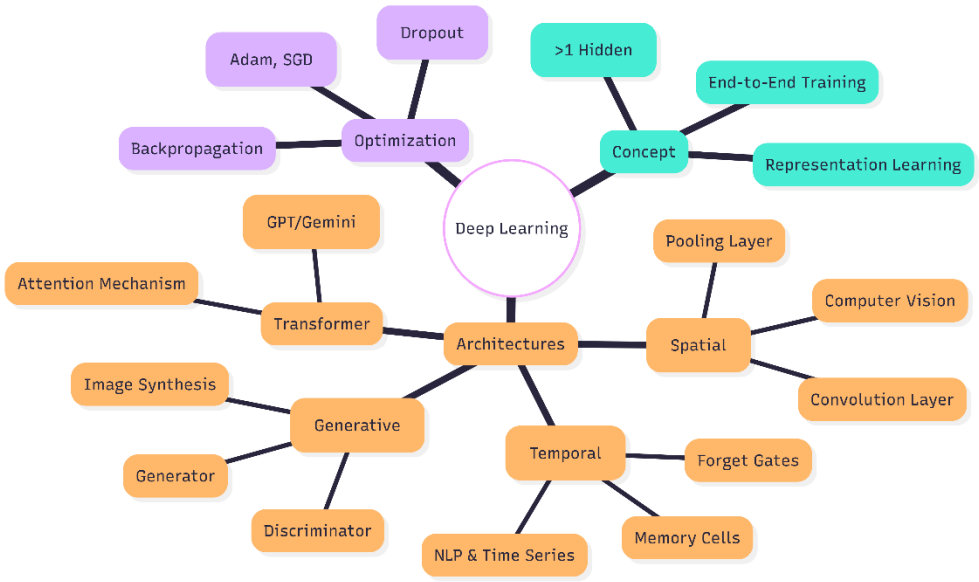
```
print("Probabilitas data pertama masuk ke tiap klaster:")
print(f"Klaster 0: {probs[0][0]:.4f}, Klaster 1: {probs[0][1]:.4f},
Klaster 2: {probs[0][2]:.4f}")

# 3. Dimensionality Reduction dengan PCA (4D -> 2D)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# 4. Visualisasi
plt.figure(figsize=(8, 6))
# Plot titik data, warna berdasarkan prediksi GMM
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=labels, s=40, cmap='viridis',
alpha=0.7)
plt.title("Visualisasi GMM Clustering (4D diproyeksikan ke 2D via
PCA)")
plt.xlabel("Principal Component 1")
plt.ylabel("Principal Component 2")
plt.colorbar(label='Cluster Label')
plt.show()
```

BAB XIV
DEEP LEARNING ARCHITECTURES

1. Deskripsi Singkat Bahasan

Bab penutup ini membahas paradigma *Deep Learning*, yaitu evolusi dari *Neural Network* yang memiliki banyak lapisan tersembunyi (*Deep Hidden Layers*). Topik khusus yang dibedah meliputi arsitektur spesifik untuk berbagai jenis data: Convolutional Neural Network (CNN) untuk data grid/citra, Recurrent Neural Network (RNN) dan Long Short-Term Memory (LSTM) untuk data sekuensial/waktu, serta Generative Adversarial Network (GAN) untuk generasi konten. Bab ini juga menyinggung konsep *Feature Learning* otomatis yang membedakan Deep Learning dengan Machine Learning tradisional..

2. Mindmap / Taksonomi (Visualisasi)



Gambar 14. Diagram ini memetakan keluarga besar arsitektur Deep Learning.

3. Penjelasan Detail

3.1 Definisi Deep Learning

Deep Learning adalah sub-bidang Machine Learning yang menggunakan algoritma yang terinspirasi oleh struktur dan fungsi otak, yang disebut jaringan saraf tiruan (Artificial Neural Networks). Kata "Deep" merujuk pada jumlah lapisan (layers) dalam jaringan.

Jika $f(x)$ adalah fungsi jaringan, maka Deep Learning dapat direpresentasikan sebagai komposisi fungsi yang dalam:

$$f(x) = f_n\left(\dots f_3\left(f_2\left(f_1(x)\right)\right)\right)$$

Dimana setiap f_i adalah lapisan yang mengekstrak fitur pada tingkat abstraksi yang berbeda.

3.2 Convolutional Neural Network (CNN)

CNN dirancang khusus untuk memproses data yang memiliki struktur grid, seperti citra (grid piksel 2D).

- Convolution Layer: Inti dari CNN. Melakukan operasi konvolusi matematis antara input (I) dan filter/kernel (K) untuk menghasilkan Feature Map.

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

- **Pooling Layer:** Mengurangi dimensi spasial (down-sampling) untuk mengurangi beban komputasi dan menghindari *overfitting* (misal: *Max Pooling*).

3.3 Recurrent Neural Network (RNN) & LSTM

RNN dirancang untuk data sekuensial (runtun waktu atau teks) di mana x_t (input saat ini) dipengaruhi oleh x_{t-1} (input sebelumnya).

- **Masalah RNN:** *Vanishing Gradient* (lupa pada sekuens panjang).
- Solusi LSTM (Long Short-Term Memory): Menggunakan mekanisme Gating (Input, Output, Forget Gate) untuk mengatur aliran informasi.

  Status sel (C_t) diperbarui dengan rumus:

  $$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

  Dimana f_t adalah forget gate (seberapa banyak melupakan masa lalu).

3.4 Generative Adversarial Network (GAN)

GAN terdiri dari dua jaringan yang "bermusuhan" dalam permainan zero-sum game:

1. **Generator (G):** Berusaha membuat data palsu yang mirip asli.
2. Discriminator (D): Berusaha membedakan data asli (x) dan palsu ($G(z)$).

   Fungsi tujuan (Objective Function):

   $$\min_G \max_D V(D,G) = E_x[\log D(x)] + E_z\left[\log\left(1 - D\left(G(z)\right)\right)\right]$$

4. Sample Code (Implementasi Konsep Konvolusi Manual)

Untuk menunjukkan pemahaman mendalam, berikut adalah implementasi manual operasi Konvolusi 2D (inti dari CNN) menggunakan NumPy, tanpa framework instan.

```
import numpy as np

def convolution2d(image, kernel):
    """
    Simulasi operasi Konvolusi 2D manual.
    image: Matriks input 2D
    kernel: Matriks filter (misal 3x3)
    """
    m, n = kernel.shape
    y, x = image.shape

    # Hitung ukuran output
    y_out = y - m + 1
    x_out = x - n + 1

    # Matriks kosong untuk output (Feature Map)
    output = np.zeros((y_out, x_out))

    # Operasi Sliding Window (Konvolusi)
    for i in range(y_out):
        for j in range(x_out):
            # Ambil region gambar seukuran kernel
            region = image[i:i+m, j:j+n]
```

```
            # Kalikan elemen-wise dan jumlahkan
            output[i][j] = np.sum(region * kernel)

    return output

# --- Simulasi ---
# Citra Sederhana (5x5) - Angka 10 merepresentasikan garis vertikal
img = np.array([
    [10, 10, 10, 0, 0],
    [10, 10, 10, 0, 0],
    [10, 10, 10, 0, 0],
    [10, 10, 10, 0, 0],
    [10, 10, 10, 0, 0]
])

# Filter Deteksi Tepi Vertikal (Vertical Edge Detector)
kernel_filter = np.array([
    [1, 0, -1],
    [1, 0, -1],
    [1, 0, -1]
])

feature_map = convolution2d(img, kernel_filter)

print("Input Image:\n", img)
print("\nFilter (Kernel):\n", kernel_filter)
print("\nFeature Map (Hasil Konvolusi):\n", feature_map)
# Hasil Feature Map akan menunjukkan nilai tinggi di area transisi
(tepi)
```