# Modul Pelatnas IOAI Indonesia

*Modul 2: Pengenalan Machine Learning*

Tim Pembina IOAI Indonesia
sc.ioai.id@gmail.com

Juli 2025


# Modul Pelatnas IOAI Indonesia

*Modul 2: Pengenalan Machine Learning*

**Penyusun**

Tim Pembina IOAI Indonesia

sc.ioai.id@gmail.com

# Daftar Isi

# Pengantar

Puji syukur kehadirat Tuhan YME atas berkat limpahan rahmat dan karnuia-Nya, Buku Modul Pelatnas IOAI ini telah berhasil kami selesaikan. Buku Modul ini kami susun sebagai salah satu referensi rangkaian pembinaan/pelatihan nasional bagi siswa peserta didik yang mengikuti Pelatnas dalam rangka membentuk tim yang akan mewakili Indonesia pada ajang International Olympiad in Artificial Intelligence (IOAI).

Terima kasih yang sebesar-besarnya kami sampaikan kepada para Pembina, Asisten Pembina, dan para Alumni ajang OSN bidang Informatika dan IOI (TOKI), serta semua pihak yang telah berkontribusi sehingga Buku Modul ini dapat terwujud. Kami menyadari masih banyak kekurangan dalam penulisan Buku Modul ini. Untuk itu kami mohon maaf dan kami sangat mengharapkan masukan untuk perbaikan dan penyempurnaan selanjutnya, sehingga keberadaan Buku Modul ini dapat memberikan manfaat yang sebesar-besarnya bagi semua pihak.

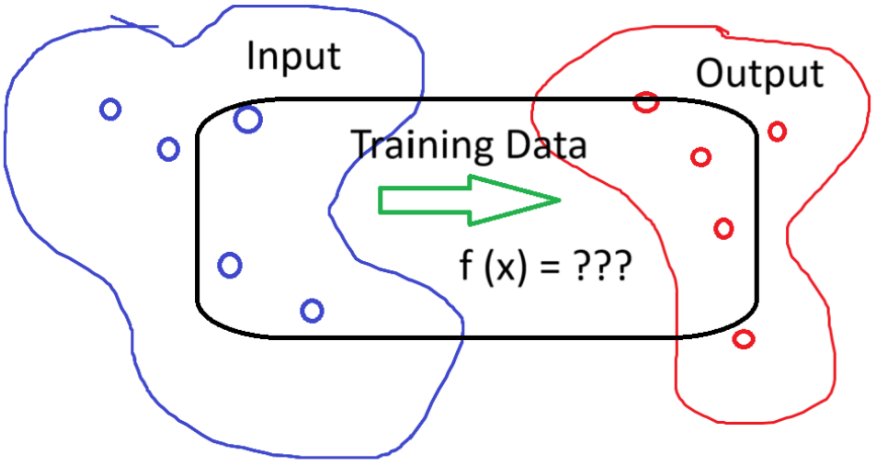
Semoga Buku Modul Pelatnas IOAI ini dapat digunakan sebaik-baiknya untuk menunjang kegiatan Pelatnas IOAI dan mampu membantu menghasilkan calon-calon talenta Indonesia di bidang AI yang mampu memberikan prestasi yang membanggakan di tingkat internasional.

BAB 1

# Pengenalan Data Science dan Machine Learning

## 1.1 Jenis-jenis Machine Learning

Machine Learning (ML) atau pembelajaran mesin adalah cabang dari kecerdasan buatan yang memungkinkan komputer belajar dari data dan membuat keputusan atau prediksi berdasarkan pola yang ditemukan. Ada beberapa paradigma pembelajaran utama dalam ML, yang diklasifikasikan berdasarkan jenis data yang digunakan dan tujuan dari proses pembelajaran. Dalam subbab ini, kita akan membahas tiga pendekatan utama: *supervised learning*, *unsupervised learning*, dan *reinforcement learning*.




**Gambar 1.1.** Ilustrasi Machine Learning

### 1.1.1 Supervised Learning

Supervised learning adalah pendekatan di mana mesin belajar dari data yang telah diberi label. Artinya, setiap contoh data dilengkapi dengan jawaban atau keluaran yang benar. Proses pembelajaran bertujuan untuk menemukan hubungan antara input (fitur) dan output (label) agar bisa digunakan untuk memprediksi hasil pada data baru yang belum pernah dilihat sebelumnya.

**Karakteristik:**

- ◇ Data pelatihan memiliki pasangan input-output.
- ◇ Tujuan model adalah mempelajari fungsi pemetaan dari input ke output.
- ◇ Digunakan ketika kita memiliki pengetahuan eksplisit tentang apa yang ingin diprediksi.

**Contoh tugas dalam supervised learning:**

- ◇ Memprediksi harga rumah berdasarkan ukuran dan lokasi.
- ◇ Memprediksi apakah email termasuk spam atau bukan.
- ◇ Memprediksi kemungkinan seorang pasien memiliki suatu penyakit berdasarkan hasil tes medis.

Supervised learning banyak digunakan dalam situasi dunia nyata di mana data historis tersedia dan dapat digunakan sebagai contoh pembelajaran.

Ada dua jenis pendekatan utama dalam supervised learning, yaitu Classification dan Regression. Classification (klasifikasi) adalah model machine learning yang bertujuan memprediksi kelas (kelompok) yang bersifat *diskret*, misalnya: sehat vs sakit, orang vs mobil vs motor, dan seterusnya. Sedangkan regresi bertujuan untuk memprediksi sebuah nilai yang bersifat *numerik*, misalnya: prediksi curah hujan, suhu, berat/tinggi badan, dan seterusnya.

### Perbandingan Klasifikasi dan Regresi

Dalam machine learning supervised, tugas dibedakan menjadi dua kelompok utama: **klasifikasi** dan **regresi**. Keduanya mempelajari pola dari data yang memiliki label, namun jenis label dan tujuannya berbeda.

| **Aspek** | **Klasifikasi** | **Regresi** |
|---|---|---|
| **Jenis output** | Label kategori diskret | Nilai kontinu (bilangan riil) |
| **Contoh target** | `"spam"` / `"not spam"`, `"A"` / `"B"` / `"C"` | Harga rumah, suhu udara, berat badan |
| **Model umum** | Logistic Regression, KNN, SVM, Decision Tree | Linear Regression, Random Forest Regressor, LASSO |
| **Contoh visualisasi** | Pemisahan area berdasarkan kelas (boundary) | Garis atau kurva yang mendekati titik data |
| **Metrik evaluasi** | Akurasi, Precision, Recall, F1-Score | Mean Squared Error (MSE), Mean Absolute Error (MAE), $R^2$ Score |
| **Kasus umum** | Deteksi email spam, klasifikasi citra, diagnosis penyakit | Prediksi harga, estimasi nilai ujian, peramalan suhu |

Pemahaman tentang perbedaan ini penting karena akan menentukan:

- ◇ Jenis model yang digunakan

- Proses evaluasi model
- Interpretasi hasil

Contohnya, jika kita ingin memprediksi apakah pasien akan mengalami diabetes (`ya/tidak`), maka itu adalah masalah klasifikasi. Namun jika ingin memprediksi kadar gula darah secara numerik, maka itu adalah masalah regresi.

### 1.1.2 Unsupervised Learning

Berbeda dengan supervised learning, pendekatan unsupervised learning digunakan ketika data tidak memiliki label. Mesin hanya diberikan data mentah dan harus menemukan struktur atau pola tersembunyi secara mandiri tanpa arahan eksplisit. Tujuannya bisa berupa pengelompokan, pengurangan dimensi, atau eksplorasi hubungan antar data.

**Karakteristik:**

- Data pelatihan tidak memiliki label.
- Fokus pada eksplorasi dan penemuan pola internal dalam data.
- Tidak ada "jawaban benar" yang diketahui sejak awal.

**Contoh tugas dalam unsupervised learning:**

- Mengelompokkan pelanggan berdasarkan kebiasaan belanja mereka.
- Mengidentifikasi topik-topik utama dari kumpulan dokumen.
- Mengurangi jumlah fitur dari data untuk mempermudah visualisasi atau analisis lanjutan.

Unsupervised learning sangat berguna dalam tahap eksplorasi data, ketika kita belum tahu secara pasti apa struktur atau informasi yang tersembunyi di balik data.

### 1.1.3 Reinforcement Learning

Reinforcement learning merupakan pendekatan pembelajaran berdasarkan pengalaman, di mana agen (komputer atau sistem) belajar melalui interaksi langsung dengan lingkungan. Agen diberikan imbalan (reward) atau hukuman (penalti) berdasarkan tindakan yang diambil, dan bertujuan untuk memaksimalkan total imbalan dalam jangka panjang.

**Karakteristik:**

- Agen belajar dari trial-and-error.
- Tidak ada data pelatihan eksplisit, melainkan umpan balik dari lingkungan.
- Pembelajaran berlangsung secara bertahap dan berorientasi pada keputusan jangka panjang.

**Contoh tugas dalam reinforcement learning:**

- Melatih agen untuk bermain gim papan seperti catur atau Go.
- Mengontrol robot agar bisa berjalan atau menyeimbangkan diri.
- Mengatur strategi penawaran harga dalam sistem lelang otomatis.

Reinforcement learning sering digunakan pada situasi dinamis dan interaktif, di mana sistem harus mengambil keputusan secara adaptif dan belajar dari konsekuensi keputusannya sendiri.

## 1.2 Tugas-Tugas dalam Machine Learning

Setelah memahami jenis-jenis pendekatan pembelajaran dalam machine learning, penting juga untuk mengenal bentuk-bentuk tugas atau permasalahan yang biasa dihadapi. Tugas-tugas ini mencerminkan jenis output yang ingin dipelajari atau diprediksi oleh model berdasarkan data yang tersedia.

### 1.2.1 Klasifikasi (*Classification*)

Klasifikasi adalah tugas di mana model bertujuan untuk mengelompokkan data ke dalam kategori atau kelas yang telah ditentukan. Setiap contoh data memiliki satu label dari sekumpulan label yang terbatas.

**Contoh aplikasi:**

- Mendeteksi apakah email adalah spam atau bukan.
- Memprediksi jenis tanaman berdasarkan ciri-ciri morfologinya.
- Menentukan diagnosis penyakit berdasarkan gejala yang dilaporkan.

### 1.2.2 Regresi (*Regression*)

Berbeda dengan klasifikasi, pada tugas regresi model mencoba memprediksi nilai kontinu atau numerik. Hasil prediksi bisa berupa angka yang tidak terbatas jumlahnya.

**Contoh aplikasi:**

- Memprediksi harga rumah berdasarkan ukuran, lokasi, dan kondisi fisik.
- Memperkirakan suhu udara di hari esok.
- Menghitung estimasi waktu pengiriman barang.

### 1.2.3 Clustering

Clustering merupakan tugas dalam unsupervised learning yang bertujuan mengelompokkan data ke dalam grup-grup berdasarkan kemiripan atau pola yang serupa, tanpa adanya label.

**Contoh aplikasi:**

- Mengelompokkan pelanggan berdasarkan perilaku belanja.

- Menyusun artikel berita berdasarkan topik-topik yang tersembunyi.
- Mengelompokkan wilayah berdasarkan pola sebaran penyakit.

### 1.2.4 Deteksi Anomali (*Anomaly Detection*)

Tugas ini berfokus pada identifikasi data yang tidak biasa atau menyimpang dari pola umum. Sangat penting untuk deteksi kejadian langka atau berbahaya.

**Contoh aplikasi:**

- Mendeteksi transaksi penipuan dalam sistem perbankan.
- Menemukan cacat pada produk pabrikasi berdasarkan sensor.
- Mengidentifikasi aktivitas tidak wajar dalam jaringan komputer.

### 1.2.5 Rekomendasi (*Recommendation*)

Sistem rekomendasi bertujuan memprediksi preferensi atau pilihan pengguna berdasarkan data historis.

**Contoh aplikasi:**

- Menyediakan daftar film atau lagu yang disukai pengguna.
- Menyusun produk yang relevan dalam e-commerce.
- Mengarahkan konten berita yang sesuai dengan minat pembaca.

### 1.2.6 Prediksi Urutan (*Sequence Prediction*)

Beberapa tugas ML melibatkan data yang memiliki urutan atau ketergantungan waktu, seperti teks, deret waktu, atau sinyal.

**Contoh aplikasi:**

- Memprediksi harga saham berdasarkan data historis.
- Melanjutkan kalimat dalam sistem penulisan otomatis.
- Menerjemahkan kalimat dari satu bahasa ke bahasa lain.

### 1.2.7 Penguatan (*Reinforcement Learning Tasks*)

Dalam pembelajaran penguatan, tugas yang dihadapi umumnya berupa pengambilan keputusan bertahap dalam lingkungan yang terus berubah. Tugas ini khas karena melibatkan tindakan, keadaan, dan imbalan yang dipelajari dari waktu ke waktu.

**Contoh aplikasi:**

- Mengembangkan agen permainan video yang mampu bermain dan menang.
- Mengontrol pergerakan robot dalam ruangan dinamis.
- Menyusun strategi penjadwalan atau perencanaan jangka panjang.

Tugas-tugas ini menunjukkan keragaman masalah yang dapat dipecahkan dengan machine learning. Setiap jenis tugas memiliki karakteristik dan tantangan tersendiri, dan pemilihan pendekatan yang tepat sangat bergantung pada konteks dan data yang tersedia.

## 1.3 Fondasi Teoritis dalam Pembelajaran Mesin

Machine learning tidak hanya sekadar mengatur data dan menjalankan algoritma — di balik semua itu terdapat fondasi teoritis yang kuat dari berbagai bidang ilmu, khususnya matematika dan statistika. Pemahaman terhadap dasar-dasar ini tidak hanya memperkuat intuisi, tetapi juga sangat penting untuk mengembangkan algoritma baru, memahami perilaku model, serta menganalisis hasil dengan benar.

Beberapa cabang utama yang menjadi dasar teori dalam machine learning antara lain:

- **Aljabar Linear:** Digunakan untuk merepresentasikan data dan operasi-operasi dalam bentuk vektor dan matriks. Sangat penting dalam representasi fitur, transformasi data, dan perhitungan dalam jaringan saraf.
- **Kalkulus (Turunan):** Berperan penting dalam proses pembelajaran model, terutama saat mengoptimalkan fungsi biaya melalui algoritma seperti gradient descent.
- **Statistika dan Teori Peluang:** Menjadi dasar dalam memahami distribusi data, inferensi, evaluasi model, dan estimasi ketidakpastian dalam prediksi.
- **Optimasi:** Digunakan untuk menemukan parameter terbaik dari model, dengan teknik yang memanfaatkan turunan, seperti penurunan paling curam (steepest descent) atau metode lainnya.
- **Matematika Diskret dan Teori Graf:** Berguna dalam representasi struktur data seperti pohon keputusan, jaringan, dan hubungan antar-entitas, serta penting dalam bidang seperti NLP dan algoritma komputasi.

Dengan memahami fondasi-fondasi ini, kita tidak hanya mampu menggunakan alat dan pustaka AI secara efektif, tetapi juga memiliki kemampuan untuk membangun, menganalisis, dan memperbaiki model secara ilmiah. Bab berikutnya akan membahas secara khusus beberapa dasar matematika yang paling relevan dalam pengembangan sistem AI modern.

## 1.4 Penutup

Pada bab ini, kita telah mengenal dasar-dasar penting dalam data science dan machine learning. Kita memahami bahwa data merupakan fondasi utama bagi kecerdasan buatan, dan bahwa machine learning adalah salah satu metode utama untuk memanfaatkan data guna membuat prediksi atau keputusan.

Kita telah mempelajari tiga pendekatan utama dalam machine learning:

- **Supervised Learning**: Belajar dari data berlabel untuk membuat prediksi yang

akurat.

- **Unsupervised Learning**: Mengeksplorasi struktur tersembunyi dalam data tanpa label.
- **Reinforcement Learning**: Belajar dari pengalaman melalui interaksi dengan lingkungan.

Selain itu, kita juga membahas berbagai bentuk tugas pembelajaran yang umum dijumpai, seperti klasifikasi, regresi, clustering, deteksi anomali, sistem rekomendasi, dan prediksi urutan. Masing-masing tugas ini dapat ditemukan dalam berbagai aplikasi nyata, mulai dari pengenalan wajah hingga kendaraan otonom.

#### Referensi dan Bahan Bacaan Lanjutan

Untuk memahami lebih dalam mengenai topik pada bab ini, pembaca dapat merujuk pada:

- Deisenroth, M. P., Faisal, A. A., & Ong, C. S. (2020). *Mathematics for Machine Learning*. Cambridge University Press. [DFO20]
- Google Developers. *Machine Learning Crash Course*. https://developers.google.com/machine-learning/crash-course [Dev23]

# BAB 2
# Praproses Data

## Pendahuluan

Dalam machine learning, kualitas hasil sangat bergantung pada kualitas data yang digunakan. Istilah umum yang sering digunakan adalah: *"garbage in, garbage out"* — jika data yang digunakan buruk, maka model yang dihasilkan juga tidak akan akurat atau dapat diandalkan.

Sebelum sebuah dataset digunakan untuk melatih model, perlu dilakukan proses pembersihan, penyesuaian, dan transformasi data agar sesuai dengan kebutuhan model. Proses ini disebut **praproses data** (data preprocessing), dan mencakup berbagai tahapan seperti deteksi pencilan (outlier), imputasi data hilang, normalisasi skala fitur, dan transformasi bentuk data. Tahapan ini sangat penting untuk meningkatkan performa dan generalisasi model yang dilatih.

## 2.1 Normalisasi Data

#### Mengapa Normalisasi Diperlukan?

Dalam machine learning, banyak algoritma bekerja dengan asumsi bahwa data memiliki skala yang seragam. Jika terdapat fitur dengan skala berbeda (misalnya tinggi badan dalam sentimeter dan penghasilan dalam juta rupiah), maka fitur dengan skala besar bisa mendominasi proses pembelajaran.

**Contoh:** Jika satu fitur memiliki nilai antara 0–1 dan fitur lain 0–10.000, maka fitur kedua akan lebih memengaruhi hasil model, walaupun secara logika belum tentu lebih penting. Untuk menghindari hal ini, diperlukan proses normalisasi atau standarisasi.

#### Jenis Normalisasi

1. **Min-Max Scaling** Menyusun ulang nilai agar berada dalam rentang tertentu, biasanya $[0, 1]$:

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

   Min-max scaling sensitif terhadap outlier karena menggunakan nilai ekstrem.

2. **Z-score Normalization (Standarisasi)** Mengubah data agar memiliki rata-rata 0 dan standar deviasi 1:

$$z = \frac{x - \mu}{\sigma}$$

Cocok untuk data yang mengikuti distribusi normal atau hampir simetris.

#### Contoh Implementasi Normalisasi dengan Python

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# Contoh data
data = np.array([[150], [160], [170], [180], [190]])

# Min-Max Normalisasi ke [0,1]
min_max = MinMaxScaler()
data_minmax = min_max.fit_transform(data)
print("Min-Max:\n", data_minmax)

# Z-score Normalisasi (Standarisasi)
z_scaler = StandardScaler()
data_zscore = z_scaler.fit_transform(data)
print("Z-score:\n", data_zscore)
```

#### Kapan Menggunakan Masing-masing?

- **Gunakan Min-Max** ketika data tidak memiliki outlier ekstrem dan Anda ingin mempertahankan bentuk distribusi asli.
- **Gunakan Z-score** jika data memiliki distribusi normal atau ada potensi outlier yang perlu diseimbangkan pengaruhnya.

#### Kesimpulan

Normalisasi adalah tahap penting agar model machine learning bekerja optimal. Ini memastikan bahwa semua fitur memiliki kontribusi yang seimbang dalam proses pembelajaran, terutama pada algoritma berbasis jarak atau gradien seperti KNN, SVM, dan neural network.

## 2.2 Deteksi dan Penanganan Outlier

#### Apa Itu Outlier?

**Outlier** adalah nilai yang jauh berbeda dari sebagian besar data lainnya. Outlier dapat muncul karena kesalahan pengukuran, pencatatan, atau memang merupakan kejadian

langka yang valid. Keberadaan outlier dapat sangat mempengaruhi rata-rata, standar deviasi, atau performa model, terutama model berbasis regresi.

**Contoh:** Dalam dataset tinggi badan siswa SMA, jika terdapat satu nilai 250 cm, maka nilai tersebut patut dicurigai sebagai outlier.

#### Metode Deteksi Outlier

Beberapa metode umum untuk mendeteksi outlier antara lain:

1. **Z-score (Standard Score)** Digunakan untuk mengukur seberapa jauh suatu data dari rata-ratanya dalam satuan standar deviasi. Formula:

$$z = \frac{x - \mu}{\sigma}$$

   Di mana:

   - $x$ = nilai data
   - $\mu$ = rata-rata
   - $\sigma$ = standar deviasi

   Data dianggap outlier jika $|z| > 3$ (atau sesuai ambang batas yang ditentukan).
2. **IQR (Interquartile Range)** IQR adalah selisih antara kuartil ketiga ($Q_3$) dan kuartil pertama ($Q_1$). Data dianggap outlier jika berada di luar rentang:

$$[Q_1 - 1.5 \times IQR,\ Q_3 + 1.5 \times IQR]$$

   Metode ini lebih tahan terhadap data yang tidak berdistribusi normal.

#### Contoh Deteksi Outlier dengan Python

Berikut contoh implementasi deteksi outlier menggunakan Z-score dan IQR:

```
import numpy as np
from scipy.stats import zscore

# Contoh data
data = np.array([165, 170, 168, 171, 250, 169, 172, 167])

# Metode 1: Z-Score
z_scores = zscore(data)
outlier_z = data[np.abs(z_scores) > 3]
print("Outlier (Z-score):", outlier_z)
```

```
# Metode 2: IQR
Q1 = np.percentile(data, 25)
Q3 = np.percentile(data, 75)
IQR = Q3 - Q1
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

outlier_iqr = data[(data < lower_bound) | (data > upper_bound)]
print("Outlier (IQR):", outlier_iqr)
```

#### Visualisasi Outlier dengan Boxplot

Outlier dapat divisualisasikan secara efektif menggunakan `boxplot`, yang menunjukkan:

- Median dan kuartil data
- Batas atas dan bawah (1.5 IQR dari Q1 dan Q3)
- Titik-titik ekstrem (outlier) sebagai titik terpisah

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.boxplot(x=df["nilai_ujian"])
plt.title("Boxplot Nilai Ujian")
plt.show()
```

Outlier terlihat sebagai titik-titik yang jauh dari box utama. Data ini dapat dianalisis lebih lanjut atau dibersihkan sesuai kebutuhan.

#### Penanganan Outlier

Setelah outlier terdeteksi, ada beberapa pendekatan untuk menanganinya:

- **Menghapus** data outlier (jika dianggap sebagai noise atau kesalahan input).
- **Mengganti** dengan nilai median atau nilai yang dibatasi (clipping).
- **Membiarkan** outlier (jika dianggap penting, misalnya dalam deteksi anomali).

Contoh penghapusan outlier:

```
# Hapus outlier dengan IQR
cleaned_data = data[(data >= lower_bound) & (data <= upper_bound)]
print("Data bersih:", cleaned_data)
```

#### Kesimpulan

Outlier dapat memengaruhi hasil pelatihan model secara signifikan. Oleh karena itu, mendeteksi dan menangani outlier merupakan salah satu langkah penting dalam praproses data.

## 2.3 Imputasi dan Penanganan Data Hilang

### Apa Itu Data Hilang?

Dalam praktik nyata, dataset sering kali mengandung nilai yang hilang (missing values) karena berbagai alasan, seperti kesalahan pencatatan, sensor gagal membaca, atau responden tidak menjawab. Kehadiran data hilang dapat mengganggu pelatihan model dan harus ditangani sebelum digunakan.

### Tipe Data Hilang

- **MCAR (Missing Completely at Random):** Hilangnya data tidak tergantung pada nilai mana pun.
- **MAR (Missing at Random):** Hilang tergantung pada kolom lain, tapi tidak pada nilai yang hilang itu sendiri.
- **MNAR (Missing Not at Random):** Hilangnya data bergantung pada nilainya sendiri (lebih sulit ditangani).

### Strategi Penanganan Data Hilang

Ada beberapa pendekatan untuk menangani data hilang:

1. **Menghapus baris atau kolom yang mengandung nilai hilang** Cocok jika proporsi data hilang sangat kecil. Namun, berisiko mengurangi informasi penting.
2. **Imputasi: Mengisi nilai hilang dengan nilai pengganti** Beberapa metode umum:
    - **Mean Imputation:** Mengisi dengan rata-rata kolom
    - **Median Imputation:** Mengisi dengan nilai tengah
    - **Mode Imputation:** Untuk data kategorikal, diisi dengan nilai yang paling sering muncul
    - **Model-based Imputation:** Menggunakan algoritma ML untuk memprediksi nilai yang hilang (tidak dibahas di sini)

### Contoh Implementasi dengan Python

```
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer

# Contoh dataset
df = pd.DataFrame({
    'Usia': [15, 16, np.nan, 17, 16],
    'Nilai': [80, np.nan, 78, 90, 85]
})
```

```
print("Data awal:\n", df)

# Imputasi mean untuk data numerik
imputer = SimpleImputer(strategy='mean')
df[['Usia', 'Nilai']] = imputer.fit_transform(df[['Usia', 'Nilai']])

print("\nSetelah imputasi:\n", df)
```

#### Catatan Penting

Imputasi meningkatkan kelengkapan data, namun perlu hati-hati agar tidak menambahkan bias. Misalnya, mengganti semua nilai kosong dengan rata-rata bisa menyamarkan pola sebenarnya jika data awal sangat bervariasi.

Untuk data kategorikal, strategi umum adalah menggunakan `'strategy='most_frequent''` pada `SimpleImputer`.

#### Kesimpulan

Penanganan data hilang merupakan langkah penting dalam proses praproses. Pilihan antara menghapus atau mengisi nilai tergantung pada jumlah data hilang dan konteks penggunaannya. Imputasi sederhana seperti rata-rata atau median sering digunakan sebagai solusi cepat dan cukup efektif dalam banyak kasus.

## 2.4 Transformasi Data

#### Apa Itu Transformasi Data?

Transformasi data adalah proses mengubah data dari satu bentuk ke bentuk lain agar lebih sesuai untuk dianalisis atau digunakan dalam model machine learning. Transformasi dapat membantu:

- Memperbaiki distribusi data (misalnya agar lebih simetris)
- Mengurangi pengaruh outlier
- Menyesuaikan skala atau rentang data
- Meningkatkan performa model

#### Jenis-Jenis Transformasi Umum

1. **Transformasi Logaritma** Cocok untuk data dengan rentang besar atau distribusi miring ke kanan (right-skewed).

$$x' = \log(x + c)$$

   dengan $c$ adalah konstanta untuk menghindari $\log(0)$
2. **Transformasi Akar Kuadrat** Cocok untuk data dengan nilai kecil namun varians

tinggi.

$$x' = \sqrt{x}$$

3. **Transformasi Pangkat (Power Transform)** Termasuk Box-Cox dan Yeo-Johnson, yang secara otomatis mencari transformasi terbaik agar distribusi mendekati normal.
4. **One-Hot Encoding (untuk data kategorikal)** Mengubah kategori menjadi vektor biner (0/1) agar dapat digunakan oleh model ML yang hanya mengenali nilai numerik.

#### Contoh Transformasi Numerik dengan Python

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import PowerTransformer

# Data yang skewed
data = np.array([[1], [2], [5], [20], [50]])

# Transformasi log
log_data = np.log1p(data)  # log(x + 1)
print("Log transform:\n", log_data)

# Power Transform (Box-Cox)
pt = PowerTransformer(method='yeo-johnson')
power_data = pt.fit_transform(data)
print("Power transform:\n", power_data)
```

#### Contoh Transformasi Kategorikal: One-Hot Encoding

```
df = pd.DataFrame({
    'warna': ['merah', 'biru', 'merah', 'kuning']
})

# One-hot encoding
encoded = pd.get_dummies(df, columns=['warna'])
print(encoded)
```

#### Kapan Transformasi Perlu Dilakukan?

Tidak semua model memerlukan transformasi distribusi. Model berbasis pohon (decision tree, random forest) biasanya tidak sensitif terhadap skala atau bentuk distribusi. Namun, model linear, SVM, atau neural network sering kali mendapatkan performa yang lebih baik dengan data yang tertransformasi secara tepat.

#### Kesimpulan

Transformasi data membantu menjadikan data lebih siap untuk dianalisis dan dimodelkan. Pemilihan jenis transformasi sangat tergantung pada jenis data, model yang

digunakan, serta tujuan dari analisis atau prediksi.

## 2.5 Reduksi Dimensi dengan Principal Component Analysis (PCA)

**Principal Component Analysis (PCA)** adalah teknik reduksi dimensi yang bertujuan untuk mengurangi jumlah variabel dalam data tanpa kehilangan terlalu banyak informasi. PCA bekerja dengan mengubah representasi data ke dalam basis baru yang disebut **komponen utama (principal components)**.

#### Mengapa Reduksi Dimensi Diperlukan?

- Data berdimensi tinggi dapat menyebabkan overfitting (curse of dimensionality).
- Visualisasi data berdimensi tinggi menjadi sulit.
- Banyak fitur dalam data sering kali bersifat redundant (berkorelasi).

#### Prinsip Dasar PCA Secara Aljabar Linear

Secara umum, PCA mencari arah (vektor) dalam ruang fitur yang memiliki **variansi terbesar** dan **saling ortogonal**. Proyeksi data ke arah-arah ini disebut **komponen utama**.

#### Prinsip Dasar PCA secara Aljabar Linear

**Principal Component Analysis (PCA)** adalah teknik reduksi dimensi yang mencari arah (vektor) dalam ruang fitur dengan variansi data terbesar dan saling ortogonal. Proyeksi data ke arah-arah tersebut disebut *komponen utama*.

Berikut adalah langkah-langkah utama PCA beserta formulasi matematisnya:

1. **Standarisasi Data** Setiap fitur $x_j$ dari data $X$ (dengan $n$ data dan $d$ fitur) dikurangi rata-rata dan dibagi simpangan baku:

$$z_{ij} = \frac{x_{ij} - \bar{x}_j}{s_j}, \quad \text{untuk } i = 1, \ldots, n$$

   Dengan:

   - $\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij}$: rata-rata fitur ke-$j$
   - $s_j$: simpangan baku fitur ke-$j$

   Hasilnya adalah matriks data terstandarisasi $Z \in \mathbb{R}^{n \times d}$

2. **Hitung Matriks Kovarians** Matriks kovarians $C$ adalah:

$$C = \frac{1}{n-1} Z^\top Z$$

Di mana $C \in \mathbb{R}^{d \times d}$ adalah simetris dan merepresentasikan kovarians antar fitur.

3. **Hitung Eigenvalue dan Eigenvector**

   Temukan pasangan $(\lambda_k, \vec{v}_k)$ dari persamaan:

   $$C\vec{v}_k = \lambda_k \vec{v}_k$$

   Di mana:

   - $\vec{v}_k$: eigenvector ke-$k$ (arah komponen utama)
   - $\lambda_k$: eigenvalue ke-$k$ (variansi yang dijelaskan oleh komponen ke-$k$)

4. **Urutkan Eigenvector**

   Urutkan pasangan $(\lambda_k, \vec{v}_k)$ dari $\lambda$ terbesar ke terkecil:

   $$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d$$

5. **Pilih $k$ Eigenvector Terbesar**

   Ambil $k$ eigenvector pertama dan bentuk matriks proyeksi $W \in \mathbb{R}^{d \times k}$:

   $$W = [\vec{v}_1 \ \vec{v}_2 \ \ldots \ \vec{v}_k]$$

6. **Proyeksikan Data**

   Data baru dalam dimensi lebih rendah diperoleh dengan:

   $$Z_{\text{baru}} = ZW$$

   Di mana $Z_{\text{baru}} \in \mathbb{R}^{n \times k}$ adalah representasi data hasil proyeksi ke ruang berdimensi $k$.

**Catatan:**

- Eigenvector dari matriks kovarians menentukan arah komponen utama (principal directions).
- Eigenvalue menunjukkan besarnya variansi yang ditangkap oleh masing-masing komponen.
- Karena $C$ adalah matriks simetris positif semi-definit, semua eigenvalue $\lambda_k \geq 0$.

#### Makna Principal Component

- Principal component ke-1 adalah arah di ruang data dengan **variansi maksimum**. - Komponen selanjutnya menjelaskan variansi maksimum yang tersisa dan **ortogonal** terhadap yang sebelumnya. - Total variansi yang dijelaskan oleh PCA dapat dihitung sebagai rasio dari jumlah eigenvalue.

#### Menentukan Jumlah Komponen (k)

Umumnya jumlah komponen dipilih berdasarkan:

- **Cumulative explained variance:** Memilih $k$ sehingga komponen tersebut menjelaskan minimal 90–95% dari total variansi.
- Visualisasi grafik **scree plot** atau **elbow method** pada eigenvalue.

#### Contoh Implementasi PCA dengan Python

```
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Muat dan standarisasi data
X, y = load_iris(return_X_y=True)
X_scaled = StandardScaler().fit_transform(X)

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

print("Proporsi variansi:", pca.explained_variance_ratio_)
print("Total variansi yang dijelaskan:",
      sum(pca.explained_variance_ratio_))

# Visualisasi
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y)
plt.xlabel("PC1")
plt.ylabel("PC2")
plt.title("PCA Iris Dataset")
plt.show()
```

**Catatan:** PCA bersifat linear. Untuk pola non-linear, bisa digunakan metode seperti Kernel PCA atau t-SNE.

## Referensi dan Bahan Bacaan Lanjutan

- **Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow** oleh Aurélien Géron — Bab tentang data preprocessing sangat aplikatif dan mudah diikuti. Cocok bagi pemula yang sudah mulai coding Python.
- **Python Data Science Handbook** oleh Jake VanderPlas Buku ini menyediakan penjelasan lengkap dan praktis tentang manipulasi data menggunakan NumPy, Pandas, dan visualisasi dengan Matplotlib. Tersedia juga versi online: https://jakevdp.github.io/PythonDataScienceHandbook/
- **Scikit-learn: Data Preprocessing Guide (resmi)** Dokumentasi resmi Scikit-learn bagian preprocessing, lengkap dengan contoh dan kode Python. https://scikit-learn.org/stable/modules/preprocessing.html
- **Kaggle: Data Cleaning Course (Free)** Kursus interaktif untuk pemula, dengan latihan langsung menggunakan Python dan Pandas. https://www.kaggle.com/learn/data-cleaning
- **Towards Data Science (Medium Blog)** Artikel populer yang menjelaskan konsep seperti missing values, normalization, encoding, dsb. dengan ilustrasi visual. https://towardsdatascience.com/tagged/data-preprocessing

# BAB 3
# Klasifikasi

## 3.1 Pengantar Klasifikasi

Klasifikasi adalah salah satu tugas utama dalam machine learning, di mana tujuan model adalah memetakan suatu input ke salah satu dari sekumpulan kategori atau kelas yang telah ditentukan sebelumnya. Ini termasuk dalam kelompok **supervised learning**, karena pelatihan dilakukan dengan contoh data yang telah diberi label kelas.

**Contoh Klasifikasi:**

- Memprediksi apakah email adalah *spam* atau *bukan spam*.
- Mengklasifikasikan gambar sebagai *kucing*, *anjing*, atau *burung*.
- Menentukan apakah seorang pasien memiliki penyakit tertentu berdasarkan hasil laboratorium.

Model klasifikasi belajar dari data latih yang telah diberi label, dan kemudian digunakan untuk memprediksi kelas dari data baru yang belum diketahui labelnya.

## 3.2 Metrik Evaluasi Klasifikasi

Untuk mengetahui seberapa baik suatu model klasifikasi bekerja, kita memerlukan metrik evaluasi. Beberapa metrik yang umum digunakan antara lain:

### 3.2.1 1. Confusion Matrix

Confusion matrix adalah tabel yang menunjukkan jumlah prediksi yang benar dan salah untuk masing-masing kelas. Ini merupakan dasar dari banyak metrik evaluasi seperti akurasi, precision, recall, dan F1-score.

#### a. Kasus Dua Kelas (Biner)

Untuk kasus klasifikasi dua kelas, confusion matrix berbentuk tabel $2 \times 2$ sebagai berikut:

| | **Pred Positif** | **Pred Negatif** |
|---|---|---|
| **Aktual Positif** | TP (True Positive) | FN (False Negative) |
| **Aktual Negatif** | FP (False Positive) | TN (True Negative) |

**Keterangan:**

- TP: Prediksi positif dan benar
- TN: Prediksi negatif dan benar
- FP: Prediksi positif tapi salah
- FN: Prediksi negatif tapi salah

#### b. Kasus Multi-Kelas

Untuk klasifikasi dengan lebih dari dua kelas (multi-kelas), confusion matrix berbentuk tabel $n \times n$, di mana $n$ adalah jumlah kelas. Setiap baris menunjukkan kelas aktual, dan setiap kolom menunjukkan kelas yang diprediksi.

Contoh untuk tiga kelas (`A`, `B`, `C`):

| | Pred A | Pred B | Pred C |
|---|---|---|---|
| Aktual A | 20 | 2 | 1 |
| Aktual B | 3 | 15 | 4 |
| Aktual C | 0 | 2 | 19 |

Diagonal utama (kiri atas ke kanan bawah) menunjukkan jumlah prediksi yang benar, sedangkan nilai-nilai di luar diagonal menunjukkan kesalahan klasifikasi.

### 3.2.2 2. Akurasi (Accuracy)

Akurasi adalah persentase prediksi yang benar dari keseluruhan prediksi:

$$\text{Akurasi} = \frac{\text{Jumlah prediksi benar}}{\text{Total data}}$$

**Kelemahan:** Akurasi bisa menipu pada data yang tidak seimbang. Misalnya, jika 95% data termasuk kelas A, maka model yang selalu memprediksi A akan mendapat akurasi 95%, walaupun sebenarnya tidak berguna.

### 3.2.3 3. Precision, Recall, dan F1-score (untuk klasifikasi biner)

Untuk kasus dua kelas (positif dan negatif), kita gunakan:

- **Precision:** Proporsi prediksi positif yang benar.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- **Recall:** Proporsi kasus positif yang berhasil ditemukan oleh model.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- **F1-score:** Rata-rata harmonis dari Precision dan Recall.

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

**Catatan:**

- TP = True Positive
- FP = False Positive
- FN = False Negative
- TN = True Negative

#### Contoh Evaluasi Model di Python

```
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score

y_true = [1, 0, 1, 1, 0, 1]
y_pred = [1, 0, 1, 0, 0, 1]

print("Confusion Matrix:\n", confusion_matrix(y_true, y_pred))
print("Akurasi:", accuracy_score(y_true, y_pred))
print("Precision:", precision_score(y_true, y_pred))
print("Recall:", recall_score(y_true, y_pred))
print("F1 Score:", f1_score(y_true, y_pred))
```

## Kesimpulan

Metrik evaluasi sangat penting untuk menilai performa model klasifikasi. Akurasi saja tidak selalu cukup, terutama jika data tidak seimbang. Oleh karena itu, precision, recall, dan F1-score digunakan untuk memberi gambaran yang lebih komprehensif.

## 3.3 Regresi Logistik

#### Motivasi

Regresi logistik adalah model klasifikasi biner yang digunakan untuk memprediksi probabilitas suatu kejadian (kelas 1 atau 0). Meskipun namanya mengandung kata "regresi", model ini termasuk dalam metode klasifikasi karena hasil akhirnya adalah prediksi kelas, bukan nilai kontinu.

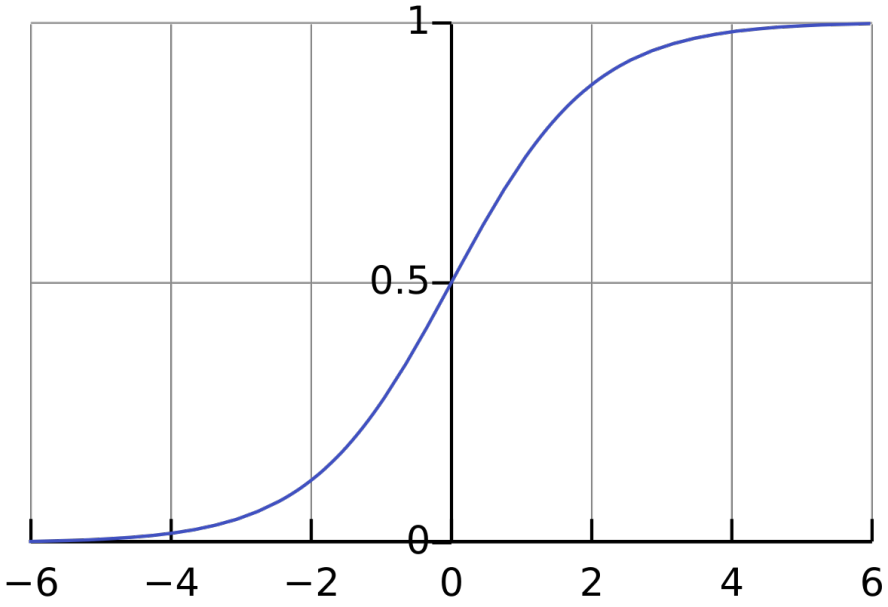
**Contoh:** Menentukan apakah seorang siswa akan lulus ujian (1) atau tidak (0) berdasarkan jumlah jam belajar.

#### Fungsi Aktivasi Sigmoid

Untuk mengubah hasil linear menjadi nilai probabilitas, digunakan fungsi **sigmoid**, yaitu:

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

Gambar 3.1 menunjukkan plot fungsi sigmoid.




**Gambar 3.1.** Fungsi sigmoid

Jika model memiliki parameter $\mathbf{w}$ dan input $\mathbf{x}$, maka:

$$\hat{y} = \sigma\left(\mathbf{w}^\top \mathbf{x} + b\right)$$

Nilai $\hat{y}$ berada di antara 0 dan 1, dan dapat diinterpretasikan sebagai probabilitas bahwa label kelas adalah 1.

#### Fungsi Kerugian (Loss): Binary Cross Entropy

Model dilatih untuk meminimalkan fungsi kerugian (loss) berikut:

$$\mathcal{L}\left(y, \hat{y}\right) = -\left[y \log\left(\hat{y}\right) + (1-y) \log\left(1-\hat{y}\right)\right]$$

Total loss untuk seluruh data (misalnya $n$ data) adalah:

$$J\left(\mathbf{w}, b\right) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(y^{(i)}, \hat{y}^{(i)}\right)$$

#### Gradient Descent untuk Regresi Logistik

Agar model belajar parameter $\mathbf{w}$ dan $b$, kita gunakan metode **gradient descent**. Turunan dari fungsi loss terhadap parameter dihitung sebagai:

$$\frac{\partial J}{\partial w_j} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)}\right) x_j^{(i)}$$

$$\frac{\partial J}{\partial b} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)}\right)$$

Lalu dilakukan update parameter:

$$w_j := w_j - \eta \cdot \frac{\partial J}{\partial w_j} \quad \text{dan} \quad b := b - \eta \cdot \frac{\partial J}{\partial b}$$

di mana $\eta$ adalah **learning rate**.

#### Implementasi Regresi Logistik Manual dengan Python

```
import numpy as np

# Fungsi sigmoid
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Fungsi loss binary cross-entropy
def compute_loss(y, y_hat):
    eps = 1e-15  # Untuk mencegah log(0)
    y_hat = np.clip(y_hat, eps, 1 - eps)
    return -np.mean(y * np.log(y_hat) + (1 - y) * np.log(1 - y_hat))

# Fungsi training regresi logistik
def train_logistic(X, y, lr=0.1, epochs=1000):
    n_samples, n_features = X.shape
    w = np.zeros(n_features)
    b = 0

    for i in range(epochs):
        z = np.dot(X, w) + b
        y_hat = sigmoid(z)

        # Gradien
        dw = (1 / n_samples) * np.dot(X.T, (y_hat - y))
        db = (1 / n_samples) * np.sum(y_hat - y)

        # Update
        w -= lr * dw
        b -= lr * db

        if i % 100 == 0:
            print(f"Epoch {i}, Loss: {compute_loss(y, y_hat):.4f}")

    return w, b

# Prediksi
def predict(X, w, b):
    probs = sigmoid(np.dot(X, w) + b)
    return (probs >= 0.5).astype(int)
```

```
# Contoh data sederhana
X = np.array([[1], [2], [3], [4], [5]])
y = np.array([0, 0, 0, 1, 1])  # threshold di antara 3 dan 4

# Training
w, b = train_logistic(X, y, lr=0.1, epochs=1000)

# Prediksi
print("Prediksi:", predict(X, w, b))
```

#### Interpretasi

Model ini belajar dari data bahwa semakin besar nilai input, semakin besar probabilitas masuk ke kelas 1. Contoh di atas sangat sederhana, namun cukup menunjukkan bagaimana regresi logistik bekerja.

#### Implementasi Menggunakan `scikit-learn`

Selain membuat sendiri model regresi logistik dari awal, kita juga dapat menggunakan pustaka `scikit-learn` yang menyediakan fungsi siap pakai. Hal ini sangat berguna untuk proyek nyata dan skala data yang lebih besar.

```
from sklearn.linear_model import LogisticRegression

# Contoh data
X = [[1], [2], [3], [4], [5]]
y = [0, 0, 0, 1, 1]

# Buat model dan latih
model = LogisticRegression()
model.fit(X, y)

# Prediksi
print("Prediksi:", model.predict(X))
print("Probabilitas:", model.predict_proba(X))
```

### Penjelasan Fungsi:

- `fit(X, y)` digunakan untuk melatih model
- `predict(X)` menghasilkan prediksi kelas (0 atau 1)
- `predict_proba(X)` menghasilkan probabilitas untuk masing-masing kelas

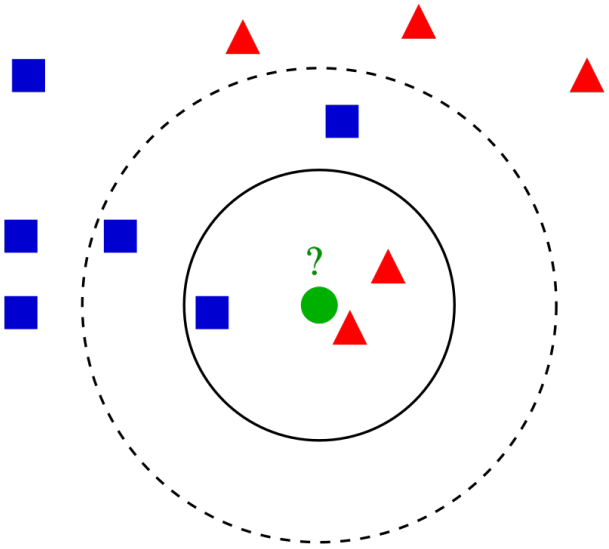
Model dari `scikit-learn` ini secara internal menggunakan optimisasi (biasanya varian dari gradient descent) untuk mencari parameter terbaik. Meskipun penggunaannya lebih singkat, pemahaman konsep di baliknya tetap sangat penting — dan itu telah kita bangun melalui implementasi manual sebelumnya.

## 3.4 K-Nearest Neighbors (KNN)

### Konsep Dasar

K-Nearest Neighbors (KNN) adalah algoritma klasifikasi yang sangat intuitif dan sederhana. Ketika diberikan sebuah data baru yang belum diketahui kelasnya, KNN akan mencari $k$ tetangga terdekat dari data latih (training data) dan melakukan **voting** untuk menentukan kelas mana yang paling umum di antara tetangga tersebut. Gambar 3.2 mengilustrasikan algoritme KNN.

**Prinsip dasar:** "Dekat = mirip".




**Gambar 3.2.** Ilustrasi algoritme KNN

### Langkah-langkah Algoritma KNN

1. Hitung jarak antara data uji dan seluruh data latih.
2. Ambil $k$ data tetangga terdekat (jarak terkecil).
3. Lihat label dari tetangga-tetangga tersebut.
4. Tentukan kelas terbanyak (mayoritas) sebagai prediksi.

### Fungsi Jarak

Biasanya digunakan:

$$\text{Jarak Euclidean} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \dots}$$

Untuk data berdimensi tinggi, bisa juga menggunakan Manhattan distance atau cosine similarity, tergantung konteks.

### Contoh Manual KNN dengan Python

```
import numpy as np
```

```
from collections import Counter

# Fungsi jarak Euclidean
def euclidean(a, b):
    return np.sqrt(np.sum((a - b) ** 2))

# Algoritma KNN manual
def knn_predict(X_train, y_train, x_test, k=3):
    distances = []

    # Hitung jarak dari x_test ke seluruh data latih
    for i in range(len(X_train)):
        dist = euclidean(x_test, X_train[i])
        distances.append((dist, y_train[i]))

    # Urutkan berdasarkan jarak
    distances.sort(key=lambda tup: tup[0])

    # Ambil k tetangga terdekat
    k_neighbors = distances[:k]
    labels = [label for (_, label) in k_neighbors]

    # Voting
    majority = Counter(labels).most_common(1)[0][0]
    return majority

# Contoh data 2D
X_train = np.array([[1, 2], [2, 3], [3, 1], [6, 5], [7, 7]])
y_train = np.array([0, 0, 0, 1, 1])

# Prediksi titik baru
x_test = np.array([5, 5])
pred = knn_predict(X_train, y_train, x_test, k=3)
print("Prediksi kelas:", pred)
```

#### Kelebihan dan Kekurangan

**Kelebihan:**

- Sederhana dan mudah diimplementasikan
- Tidak memerlukan pelatihan model (lazy learner)

**Kekurangan:**

- Waktu prediksi bisa lambat untuk dataset besar
- Sangat bergantung pada skala fitur (fitur harus dinormalisasi)

- Sensitif terhadap nilai $k$ dan outlier

#### Normalisasi Data

Karena KNN menggunakan jarak antar titik, penting untuk menormalisasi fitur ke dalam skala yang sebanding. Jika tidak, fitur dengan skala besar bisa mendominasi perhitungan jarak.

Contoh normalisasi:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
x_test_scaled = scaler.transform([x_test])
```

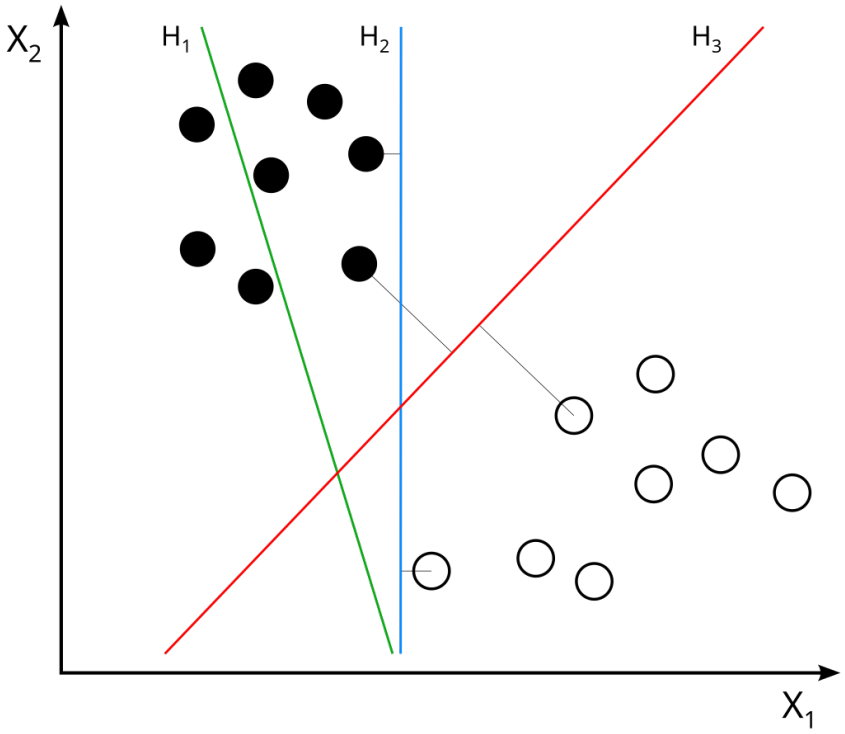
#### Implementasi dengan `scikit-learn`

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train, y_train)

print("Prediksi:", model.predict([x_test]))
```

## 3.5 Support Vector Machine (SVM)

#### Konsep Dasar

Support Vector Machine (SVM) adalah salah satu metode klasifikasi yang sangat kuat dan banyak digunakan. Tujuan utamanya adalah mencari garis atau bidang pemisah (*hyperplane*) yang tidak hanya memisahkan dua kelas, tetapi juga memberikan **margin terbesar** antara titik-titik dari dua kelas tersebut.




Semakin lebar margin antara dua kelas, semakin baik generalisasi model terhadap data baru.

#### Fungsi Keputusan SVM

SVM membentuk model dalam bentuk fungsi keputusan linear:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

Di mana:

- $\mathbf{x}$ adalah vektor fitur (misalnya ukuran bunga)
- $\mathbf{w}$ adalah vektor bobot
- $b$ adalah bias (intersep)

Prediksi kelas dilakukan berdasarkan tanda dari $f(\mathbf{x})$:

$$\hat{y} = \begin{cases} +1 & \text{jika } f(\mathbf{x}) \geq 0 \\ -1 & \text{jika } f(\mathbf{x}) < 0 \end{cases}$$

#### Tujuan Optimasi SVM: Margin Maksimal

SVM berusaha mencari parameter $\mathbf{w}$ dan $b$ yang memaksimalkan margin, yaitu jarak antara hyperplane dan titik terdekat dari masing-masing kelas.

**Formulasi Hard Margin (data tanpa noise):**

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{dengan syarat: } y^{(i)}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b\right) \geq 1$$

**Formulasi Soft Margin (realita: data mengandung noise):**
Untuk mengakomodasi kesalahan klasifikasi, diperkenalkan variabel $\xi_i$ dan penalti $C$:

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{dengan } y^{(i)}\left(\mathbf{w}^\top\mathbf{x}^{(i)} + b\right) \geq 1 - \xi_i$$

- $\xi_i$ adalah toleransi kesalahan pada data ke-$i$
- $C$ adalah parameter yang mengontrol keseimbangan antara margin besar dan jumlah kesalahan.

#### Masalah dengan Data Non-Linier

Jika data tidak bisa dipisahkan dengan garis lurus di ruang asli (misalnya berbentuk melingkar), SVM linear tidak akan cukup. Untuk itu, digunakan pendekatan **kernel**.

#### Kernel Trick

Kernel adalah fungsi yang secara implisit memetakan data ke ruang berdimensi lebih tinggi tanpa perlu menghitung koordinat baru secara eksplisit.

Beberapa jenis kernel:

- **Linear:**
$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top\mathbf{x}'$$
- **Polynomial:**
$$K(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^\top\mathbf{x}' + c\right)^d$$
- **Radial Basis Function (RBF) / Gaussian:**
$$K(\mathbf{x}, \mathbf{x}') = \exp\Big(() - \gamma\|\mathbf{x} - \mathbf{x}'\|^2\Big)$$
- **Sigmoid:**
$$K(\mathbf{x}, \mathbf{x}') = \tanh\left(\alpha\mathbf{x}^\top\mathbf{x}' + c\right)$$

Dengan kernel, SVM bisa memisahkan data yang non-linier dengan efektif.

#### Implementasi SVM pada Dataset Iris (Python)

Kita akan menggunakan dataset `iris` yang tersedia di `scikit-learn`. Kita akan klasifikasikan dua kelas bunga: `versicolor` dan `virginica`, berdasarkan dua fitur: panjang dan lebar kelopak.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

```
# Load data
iris = load_iris()
X = iris.data[50:, 2:4]  # hanya versicolor & virginica, fitur ke-3 dan 4
y = iris.target[50:] - 1  # label: 0 dan 1

# Bagi data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

# Model SVM dengan kernel RBF
model = svm.SVC(kernel='rbf', C=1.0, gamma=0.5)
model.fit(X_train, y_train)

# Prediksi
print("Akurasi:", model.score(X_test, y_test))

# Visualisasi
h = .02
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm, edgecolors='k')
plt.xlabel('Panjang Kelopak')
plt.ylabel('Lebar Kelopak')
plt.title('SVM dengan Kernel RBF')
plt.show()
```

#### Kesimpulan

SVM adalah algoritma klasifikasi yang sangat kuat, terutama saat data memiliki batas yang jelas atau bentuk non-linier. Pemahaman konsep margin, penalti kesalahan, dan penggunaan kernel sangat penting untuk menerapkan SVM dengan benar.

## 3.6 Decision Tree dan Random Forest

#### Pohon Keputusan (Decision Tree)

Decision Tree adalah metode klasifikasi yang menyerupai proses pengambilan keputusan manusia. Data dibagi berdasarkan fitur yang paling "informatif", membentuk struktur seperti pohon, dengan:

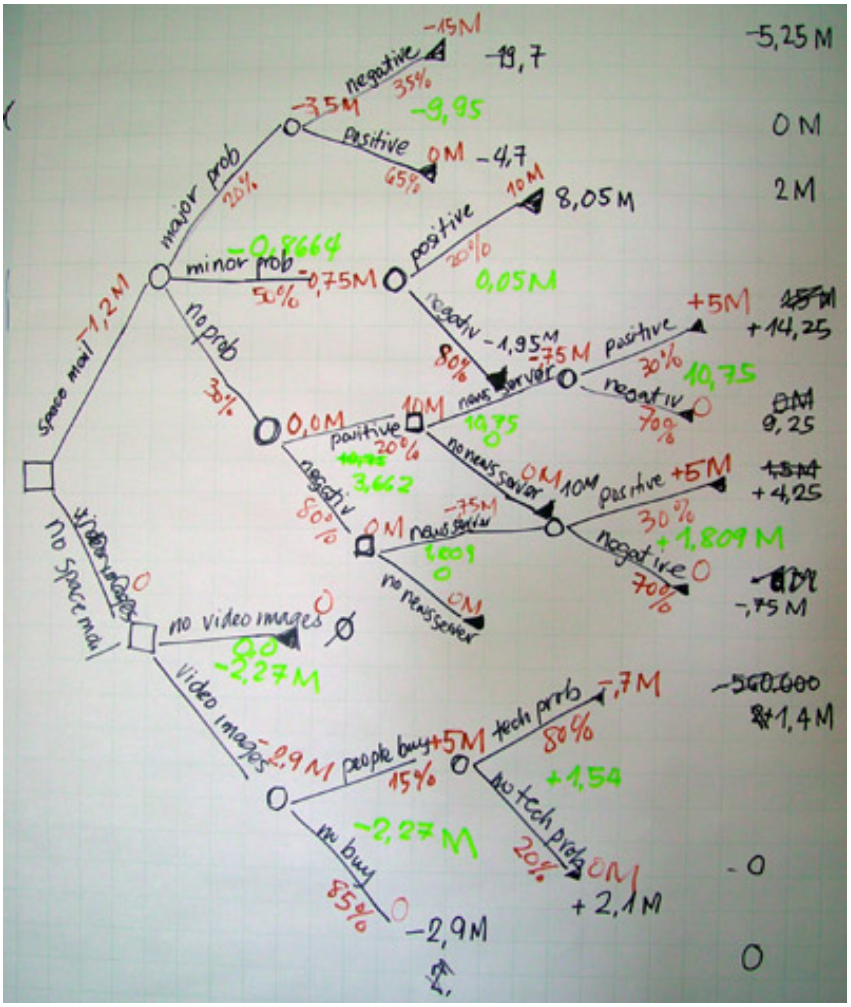
- **Node internal**: fitur yang digunakan untuk memisahkan data
- **Cabang**: hasil dari pengujian kondisi
- **Leaf node (daun)**: prediksi kelas akhir

### Contoh Sederhana

Misalnya, kita ingin memprediksi apakah seseorang membeli es krim atau coklat berdasarkan suhu:

- Jika suhu $> 30°$ C, maka beli es krim
- Jika tidak, maka beli coklat

Sebelum teori *machine learning* berkembang, pohon keputusan biasanya dibuat secara manual dengan menggunakan berbagai pertimbangan dan pengetahuan dari *domain* permasalahan. Contoh pohon keputusan yang dibuat secara manual dapat dilihat di bawah ini.



### Kriteria Pemisahan (Splitting)

Pohon memutuskan pemisahan data berdasarkan seberapa "murni" (homogen) suatu subset. Ukuran umum:

- **Entropy**:

$$H(S) = -\sum_{i=1}^{k} p_i \log_2 p_i$$

- **Gini Index**:

$$G(S) = 1 - \sum_{i=1}^{k} p_i^2$$

Semakin rendah nilai entropy atau Gini, semakin baik pemisahan data.

#### Overfitting dan Pruning

Pohon yang terlalu dalam bisa overfit — terlalu cocok dengan data latih. Solusi:

- **Pruning**: memotong cabang pohon yang terlalu spesifik
- **Maksimum depth**: membatasi kedalaman pohon
- **Minimum samples per leaf**: membatasi jumlah data minimum untuk membuat cabang baru

#### Random Forest (RF)

Random Forest adalah algoritma **ensemble learning** yang menggabungkan banyak Decision Tree untuk membuat model yang lebih kuat dan stabil. Alih-alih hanya membuat satu pohon, RF membentuk **ratusan bahkan ribuan pohon** kecil, kemudian menggabungkan hasilnya.

### Intuisi: Kebun Keputusan (Forest of Trees)

Jika satu pohon bisa salah (karena terlalu dalam atau overfit), maka membuat banyak pohon dari potongan data berbeda dan mengambil keputusan mayoritas akan menghasilkan prediksi yang lebih stabil.

### Bagging: Bootstrap Aggregation

**Bagging** adalah teknik utama yang digunakan dalam Random Forest. Prosesnya:

- Dari dataset asli (misalnya 150 data), dibuat **n** dataset baru dengan pengambilan acak **dengan pengembalian** (bootstrap sampling).
- Untuk setiap dataset bootstrap, dibuat satu Decision Tree.
- Ketika memprediksi data baru, semua pohon memberi suara (voting), dan hasil akhirnya adalah mayoritas.

### Pengacakan Fitur (Feature Randomness)

Untuk mencegah semua pohon menjadi serupa, Random Forest juga melakukan pengacakan fitur saat membentuk pohon:

- Pada setiap node, hanya subset kecil dari fitur yang dipertimbangkan untuk pemisahan terbaik.
- Ini menghasilkan variasi antarpohon, sehingga meningkatkan keragaman model dan mengurangi korelasi antar pohon.

### Prediksi dan Voting

- Untuk **klasifikasi**: hasil akhir adalah **voting mayoritas** dari semua pohon.
- Untuk **regresi**: hasil akhir adalah **rata-rata** dari semua prediksi pohon.

#### Mengapa Random Forest Lebih Baik?

**Masalah pada Decision Tree:**

- Mudah overfitting
- Terlalu sensitif terhadap perubahan kecil pada data

**Keunggulan Random Forest:**

- Lebih akurat dan stabil
- Mengurangi overfitting
- Dapat menangani data besar dan fitur yang banyak
- Tidak perlu normalisasi data
- Bisa digunakan untuk klasifikasi dan regresi

#### Kelebihan dan Kekurangan Random Forest

**Kelebihan:**

- Robust terhadap overfitting
- Tidak terlalu sensitif terhadap outlier atau noise
- Dapat menangani missing values (secara terbatas)
- Secara default memiliki fitur `feature importance` untuk interpretasi

**Kekurangan:**

- Lebih lambat dibanding pohon tunggal
- Lebih sulit diinterpretasi (karena banyak pohon)
- Membutuhkan memori lebih besar

#### Feature Importance

Random Forest dapat digunakan untuk mengetahui fitur mana yang paling penting dalam prediksi.

#### Implementasi di Python (Dataset Iris)

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

# Load data
iris = load_iris()
X, y = iris.data, iris.target

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Decision Tree
tree_model = DecisionTreeClassifier(max_depth=3)
tree_model.fit(X_train, y_train)
print("Akurasi Decision Tree:", tree_model.score(X_test, y_test))

# Random Forest
rf_model = RandomForestClassifier(n_estimators=100)
rf_model.fit(X_train, y_train)
print("Akurasi Random Forest:", rf_model.score(X_test, y_test))
```

#### Visualisasi Decision Tree

```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

plt.figure(figsize=(10,6))
plot_tree(tree_model, filled=True, feature_names=iris.feature_names,
          class_names=iris.target_names)
plt.title("Visualisasi Pohon Keputusan")
plt.show()

import pandas as pd
import matplotlib.pyplot as plt

# Ambil pentingnya fitur
importances = rf_model.feature_importances_
feature_names = iris.feature_names

# Plot
```

```
pd.Series(importances, index=feature_names).sort_values().plot(kind='barh')
plt.title("Feature Importance dari Random Forest")
plt.show()
```

#### Catatan Tambahan

- Decision Tree mudah diinterpretasi dan cocok untuk eksplorasi awal.
- Random Forest lebih kuat untuk akurasi, tetapi lebih sulit dipahami.
- Keduanya bisa digunakan untuk klasifikasi maupun regresi.

### Catatan Penting: Ketidakseimbangan Data (Data Imbalance)

Dalam tugas klasifikasi, terkadang data memiliki distribusi kelas yang sangat tidak seimbang. Misalnya, dalam dataset deteksi kanker, 95% data bisa merupakan kasus `tidak kanker`, dan hanya 5% sisanya adalah `kanker`. Ini disebut sebagai **data imbalance**.

- Jika kita melatih model pada data seperti ini dan hanya mengejar akurasi, maka model bisa saja hanya menebak kelas mayoritas (`tidak kanker`) dan tetap mendapatkan akurasi 95%.
- Namun, model tersebut sama sekali tidak berguna dalam menemukan kasus `kanker` yang sebenarnya sangat penting.

**Contoh:**

| Kelas Sebenarnya | Prediksi | Hasil |
|---|---|---|
| tidak kanker | tidak kanker | benar |
| tidak kanker | tidak kanker | benar |
| tidak kanker | tidak kanker | benar |
| kanker | tidak kanker | **salah fatal!** |

**Solusi untuk Menghadapi Data Imbalance:**

- Gunakan metrik yang lebih tepat seperti **Precision**, **Recall**, dan **F1-Score** daripada hanya akurasi.
- Gunakan `class_weight='balanced'` saat melatih model (misalnya pada Logistic Regression atau SVM).
- Lakukan **oversampling** kelas minoritas atau **undersampling** kelas mayoritas.

Data imbalance sangat penting diperhatikan, terutama dalam aplikasi dunia nyata dan dalam soal-soal kompetisi seperti IOAI. Mengabaikan hal ini dapat menyebabkan model gagal pada kasus yang justru paling penting.

## Referensi dan Bahan Bacaan Lanjutan

- Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 3rd Edition, 2022. [Bab 3 dan 4 membahas klasifikasi

dan metrik evaluasi].

- Google Developers Machine Learning Crash Course: https://developers.google.com/machine-learning/crash-course/classification/video-lecture
- Scikit-learn documentation: *Classification Overview* https://scikit-learn.org/stable/supervised_learning.html#supervised-learning
- Hastie, Tibshirani, Friedman. *The Elements of Statistical Learning*, 2nd edition, Springer, 2009. https://web.stanford.edu/~hastie/ElemStatLearn/

# BAB 4

# Regresi

## 4.1 Pengantar Regresi

Dalam machine learning, **regresi** adalah metode untuk memprediksi nilai kontinu dari suatu variabel target berdasarkan satu atau lebih fitur (input). Ini berbeda dengan **klasifikasi**, yang bertujuan memetakan input ke dalam kelas-kelas tertentu.

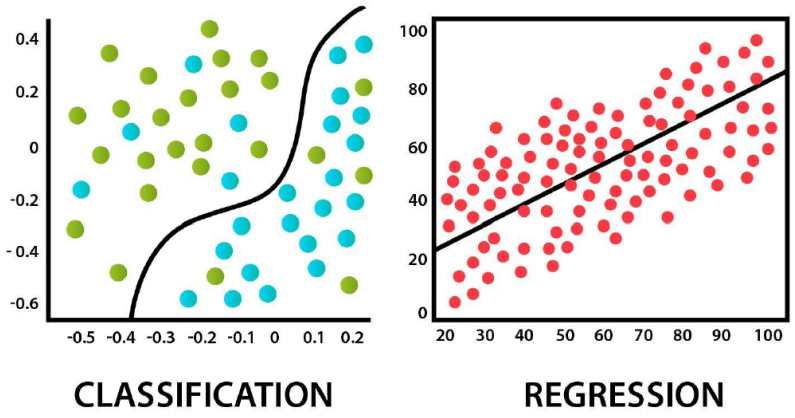
**Contoh perbedaan:**

- **Klasifikasi:** Apakah email ini spam atau bukan?
- **Regresi:** Berapa harga rumah berdasarkan ukurannya?

**Aplikasi umum regresi:**

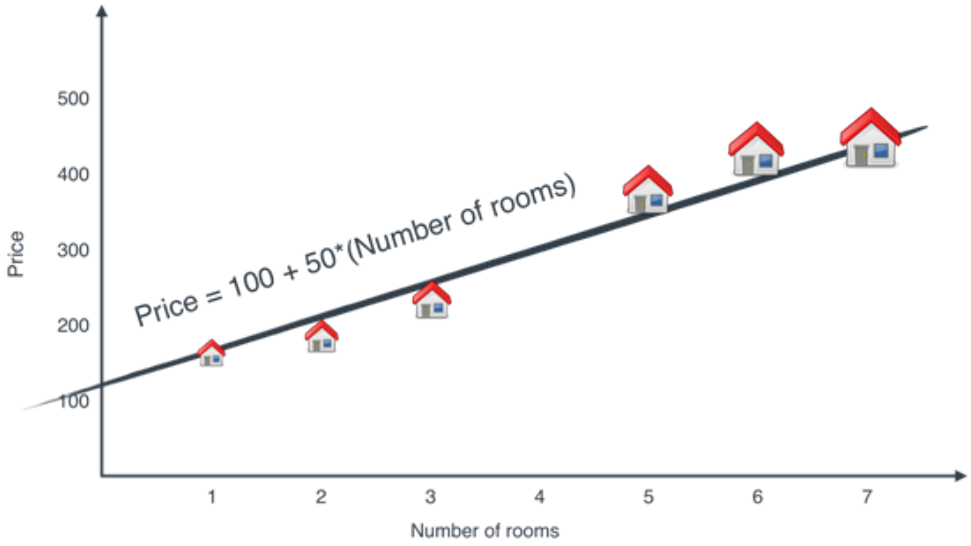
- Memprediksi harga (rumah, mobil, saham)
- Perkiraan suhu atau curah hujan
- Prediksi berat badan berdasarkan tinggi dan umur
- Proyeksi penjualan di masa depan

#### Visualisasi Regresi vs Klasifikasi




Pada regresi, model mempelajari pola hubungan antar variabel input dan output berupa garis atau kurva kontinu, bukan batas kelas seperti pada klasifikasi.

**Contoh:** Jika kita memiliki data tentang banyaknya kamar/ruangan dalam sebuah rumah dan harganya, maka model regresi akan mencoba menemukan garis yang **paling**




**Gambar 4.1.** Contoh regresi linear (sumber: livebook.manning.com)

**cocok** untuk memetakan dari jumlah kamar/ruangan ke harga (Gambar 4.1).

#### Tujuan Model Regresi

Tujuan model regresi adalah menghasilkan sebuah model prediksi yang meminimalkan perbedaan antara nilai prediksi dan nilai sebenarnya (disebut *error* atau *residual*). Kita akan membahas metrik error ini pada subbab berikutnya.

## 4.2 Metrik Evaluasi Regresi

Dalam model regresi, kita perlu mengukur seberapa baik model mampu memprediksi nilai output yang mendekati nilai sebenarnya. Berbeda dengan klasifikasi (yang menggunakan akurasi atau confusion matrix), regresi menggunakan ukuran **error** atau **selisih** antara nilai prediksi dan nilai aktual.

Beberapa metrik umum dalam regresi adalah:

### 4.2.1 Mean Absolute Error (MAE)

MAE mengukur rata-rata dari nilai absolut perbedaan antara nilai sebenarnya ($y_i$) dan nilai prediksi ($\hat{y}_i$).

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

**Ciri:**

- Lebih mudah dipahami
- Tidak terlalu sensitif terhadap outlier

### 4.2.2 Mean Squared Error (MSE)

MSE menggunakan kuadrat dari error:

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

**Ciri:**

- ◇ Memberi penalti besar terhadap error yang besar
- ◇ Cocok saat kita ingin menghindari prediksi buruk (besar)

### 4.2.3 Root Mean Squared Error (RMSE)

RMSE adalah akar kuadrat dari MSE. Nilainya kembali ke satuan asli, sehingga lebih mudah diinterpretasikan.

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

### 4.2.4 Koefisien Determinasi ($R^2$)

Metrik ini mengukur seberapa baik variasi dalam data target dijelaskan oleh model.

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

- ◇ Nilai $R^2$ mendekati 1 artinya prediksi sangat baik.
- ◇ Nilai $R^2 = 0$ artinya model tidak lebih baik dari rata-rata.
- ◇ Bisa juga negatif jika model sangat buruk.

### 4.2.5 Contoh Implementasi (Python)

```
from sklearn.metrics import mean_absolute_error,
    mean_squared_error, r2_score
import numpy as np

# Nilai aktual dan prediksi
y_true = [3, -0.5, 2, 7]
y_pred = [2.5, 0.0, 2, 8]

mae = mean_absolute_error(y_true, y_pred)
mse = mean_squared_error(y_true, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_true, y_pred)

print("MAE:", mae)
print("MSE:", mse)
```

```
print("RMSE:", rmse)
print("R^2:", r2)
```

### 4.2.6 Kapan Menggunakan Metrik Tertentu?

- Gunakan MAE jika Anda ingin menghindari penalti besar akibat outlier.
- Gunakan MSE atau RMSE jika Anda ingin lebih keras terhadap prediksi yang jauh meleset.
- Gunakan $R^2$ untuk melihat seberapa baik model menjelaskan variabilitas data.

## 4.3 Regresi Linier Sederhana

Regresi linier sederhana adalah bentuk paling dasar dari regresi. Tujuannya adalah memodelkan hubungan linier antara satu variabel input ($x$) dan satu variabel output ($y$). Modelnya dinyatakan sebagai:

$$\hat{y} = ax + b$$

- $a$ adalah kemiringan (slope) garis
- $b$ adalah intersep (nilai potong sumbu-$y$)

### 4.3.1 Tujuan Regresi Linier

Model ini mencoba mencari garis lurus yang paling mendekati semua titik data. Garis ini disebut **garis regresi terbaik** atau *least squares line*, karena ia meminimalkan jumlah kuadrat dari error prediksi (residual):

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

Gambar 4.2 menunjukkan sebuah garis regresi linier dan residualnya.

### 4.3.2 Estimasi Parameter $a$ dan $b$

Untuk menghitung $a$ dan $b$ dari data, digunakan rumus berikut:

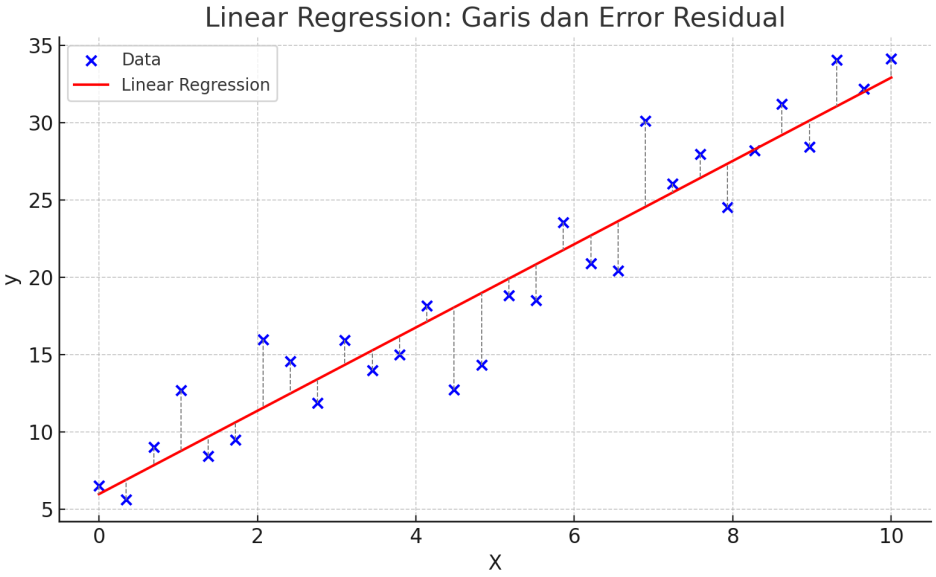
$$a = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \quad \text{dan} \quad b = \bar{y} - a\bar{x}$$

Dengan:

- $\bar{x}$ = rata-rata semua nilai $x$
- $\bar{y}$ = rata-rata semua nilai $y$

### 4.3.3 Implementasi Regresi Linier Manual (Python)

```
# Dataset sederhana
x = [1, 2, 3, 4, 5]
```




**Gambar 4.2.** Regresi Linier dan residual/errornya

```
y = [2, 4, 5, 4, 5]

# Hitung rata-rata
x_mean = sum(x) / len(x)
y_mean = sum(y) / len(y)

# Hitung a dan b
num = sum((x[i] - x_mean)*(y[i] - y_mean) for i in range(len(x)))
den = sum((x[i] - x_mean)**2 for i in range(len(x)))
a = num / den
b = y_mean - a * x_mean

print("a (slope):", a)
print("b (intercept):", b)

# Prediksi dan plotting
y_pred = [a * xi + b for xi in x]

# Visualisasi
import matplotlib.pyplot as plt
plt.scatter(x, y, label='Data')
plt.plot(x, y_pred, color='red', label='Garis Regresi')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Regresi Linier Sederhana')
plt.legend()
plt.grid(True)
```

```
plt.show()
```

#### 4.3.4 Interpretasi

Model di atas mencari garis yang meminimalkan error antara prediksi ($\hat{y}$) dan nilai sebenarnya ($y$). Setelah memahami cara kerja regresi linier sederhana, kita dapat mengembangkan ke:

- Regresi linier multivariat (lebih dari satu fitur)
- Model regularisasi (misalnya LASSO)
- Model regresi non-linier

#### 4.3.5 Catatan Tambahan

Meskipun sederhana, regresi linier banyak digunakan di berbagai bidang seperti ekonomi, kedokteran, teknik, dan sains sosial karena mudah ditafsirkan dan cukup efektif untuk hubungan yang memang linier.

## 4.4 Regresi Linier Multivariat

#### Apa Itu Regresi Linier Multivariat?

Pada regresi linier sederhana, kita hanya memodelkan hubungan antara **satu** variabel prediktor ($x$) dan satu target ($y$). Dalam banyak kasus nyata, prediksi bergantung pada lebih dari satu faktor. Misalnya:

- Harga rumah dipengaruhi oleh ukuran, lokasi, dan jumlah kamar
- Nilai siswa dipengaruhi oleh jam belajar, kehadiran, dan latar belakang ekonomi

Model regresi linier multivariat memiliki bentuk:

$$\hat{y} = a_1 x_1 + a_2 x_2 + \cdots + a_p x_p + b$$

atau dalam bentuk vektor:

$$\hat{y} = \mathbf{x}^\top \mathbf{a} + b$$

#### Mengapa Metode Sebelumnya Tidak Cukup?

Pada regresi linier sederhana, kita menggunakan rumus eksplisit untuk menghitung slope ($a$) dan intersep ($b$). Namun, untuk lebih dari satu variabel:

- Tidak ada rumus eksplisit sederhana untuk banyak variabel
- Perhitungannya membutuhkan operasi matriks, seperti inverse matriks
- Maka, kita sering menggunakan library atau pendekatan numerik

**Contoh: Regresi Linier Multivariat dengan `scikit-learn`**

Kita akan:

1. Membuat tiga variabel prediktor acak: $x_1$, $x_2$, $x_3$
2. Membuat target $y$ dari hubungan linier: $y = 3x_1 - 2x_2 + 1.5x_3 + \epsilon$
3. Melatih model regresi menggunakan `LinearRegression` dari `scikit-learn`

```
import numpy as np
from sklearn.linear_model import LinearRegression

# Set seed untuk reprodusibilitas
np.random.seed(42)

# Buat 100 data acak untuk 3 prediktor
n = 100
x1 = np.random.rand(n)
x2 = np.random.rand(n)
x3 = np.random.rand(n)

# Gabungkan menjadi matriks X
X = np.column_stack((x1, x2, x3))

# Buat target y secara linier dari ketiga variabel, tambahkan noise kecil
epsilon = np.random.normal(0, 0.1, size=n)
y = 3*x1 - 2*x2 + 1.5*x3 + epsilon

# Buat model dan latih
model = LinearRegression()
model.fit(X, y)

# Tampilkan koefisien dan intersep
print("Koefisien:", model.coef_)   # Harus mendekati [3, -2, 1.5]
print("Intersep:", model.intercept_)

# Prediksi dan evaluasi
y_pred = model.predict(X)

# Hitung R^2
from sklearn.metrics import r2_score
print("R^2:", r2_score(y, y_pred))
```

**Interpretasi**

- `model.coef_` menunjukkan seberapa besar pengaruh masing-masing fitur terhadap target.
- Jika data dihasilkan dengan $y = 3x_1 - 2x_2 + 1.5x_3 + \epsilon$, maka koefisien yang dipelajari

oleh model seharusnya mendekati nilai tersebut.

- Skor $R^2$ yang tinggi (mendekati 1) menandakan bahwa model menangkap pola dengan baik.

#### Visualisasi Hasil (Opsional)

Visualisasi regresi multivariat sulit dilakukan langsung karena kita memiliki lebih dari dua dimensi. Namun, kita bisa memvisualisasi hubungan antar satu fitur dan target.

```
import matplotlib.pyplot as plt

plt.scatter(x1, y, label="x1 vs y", alpha=0.6)
plt.scatter(x1, y_pred, label="x1 vs prediksi", alpha=0.6)
plt.xlabel("x1")
plt.ylabel("y / y_pred")
plt.legend()
plt.title("Hubungan antara x1 dan y")
plt.grid(True)
plt.show()
```

## 4.5 Regresi dengan Regularisasi: Ridge, LASSO, dan Elastic Net

#### Mengapa Perlu Regularisasi?

Pada model regresi biasa, seperti regresi linier sederhana atau multivariat, kita berusaha mencari koefisien yang meminimalkan kesalahan prediksi. Namun dalam praktiknya, sering terjadi dua masalah umum:

1. **Overfitting:** Model terlalu kompleks dan menyesuaikan secara berlebihan dengan data pelatihan, sehingga buruk dalam memprediksi data baru.
2. **Multikolinearitas:** Terjadi jika terdapat korelasi tinggi antar fitur (variabel prediktor), yang menyebabkan ketidakstabilan dan koefisien menjadi sangat besar.

Untuk mengatasi hal ini, digunakan **regularisasi**, yaitu teknik menambahkan penalti terhadap besar koefisien pada saat proses pelatihan model.

**Tujuan Regularisasi:**

- Menyederhanakan model dengan menekan nilai koefisien
- Mencegah overfitting dan meningkatkan kemampuan generalisasi
- Meningkatkan stabilitas model ketika fitur sangat banyak atau saling berkorelasi

#### Prinsip Parsimony dan Occam's Razor

Regularisasi juga sejalan dengan **prinsip parsimony** dalam sains, yaitu bahwa dari dua model yang sama baiknya dalam menjelaskan data, kita sebaiknya memilih model yang

lebih sederhana.

**Occam's Razor** menyatakan bahwa:

> *"Solusi atau penjelasan yang paling sederhana terhadap sebuah fenomena biasanya adalah penjelasan terbaik."*

Dengan kata lain, kita sebaiknya memilih model dengan jumlah parameter yang lebih sedikit atau dengan kompleksitas yang lebih rendah, jika model tersebut tetap dapat memprediksi dengan baik.

### Pengaruh terhadap Generalisasi dan Varians

Regularisasi membantu menurunkan **varians** model terhadap data pelatihan. Model yang overfitting cenderung memiliki varians yang tinggi — performanya berubah drastis terhadap data baru.

Dengan regularisasi:

- Nilai koefisien ditekan mendekati nol
- Kompleksitas model menurun
- **Bias mungkin sedikit meningkat**, tetapi
- **Varians menurun secara signifikan**

Hal ini menciptakan **trade-off antara bias dan varians**, yang sering kali menghasilkan model dengan *generalization error* yang lebih rendah — yakni performa yang lebih baik terhadap data baru (yang tidak dilihat saat pelatihan).

**Tujuan Regularisasi:**

- Mencegah model terlalu kompleks
- Memperkecil nilai koefisien yang tidak penting
- Dalam beberapa kasus, menghilangkan fitur yang tidak relevan sama sekali

### Fungsi Loss dengan Regularisasi

Fungsi loss umum untuk regresi linier adalah **Mean Squared Error (MSE)**:

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

Regularisasi dilakukan dengan menambahkan *penalti* terhadap fungsi loss regresi, yaitu:

$$\text{Loss} = \text{MSE} + \text{penalti terhadap koefisien}$$

Jenis penalti yang digunakan akan membedakan metode regularisasi, seperti Ridge (L2), LASSO (L1), dan Elastic Net (gabungan keduanya).

### LASSO Regression (L1 Regularization)

**LASSO (Least Absolute Shrinkage and Selection Operator)** adalah metode regresi linier dengan regularisasi L1. Tidak seperti Ridge yang hanya mengecilkan koefisien, LASSO memiliki kemampuan untuk menyusutkan beberapa koefisien menjadi nol, sehingga secara otomatis melakukan seleksi fitur.

### Fungsi Loss

Fungsi objektif yang diminimalkan adalah:

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p}|w_j|$$

### Karakteristik LASSO

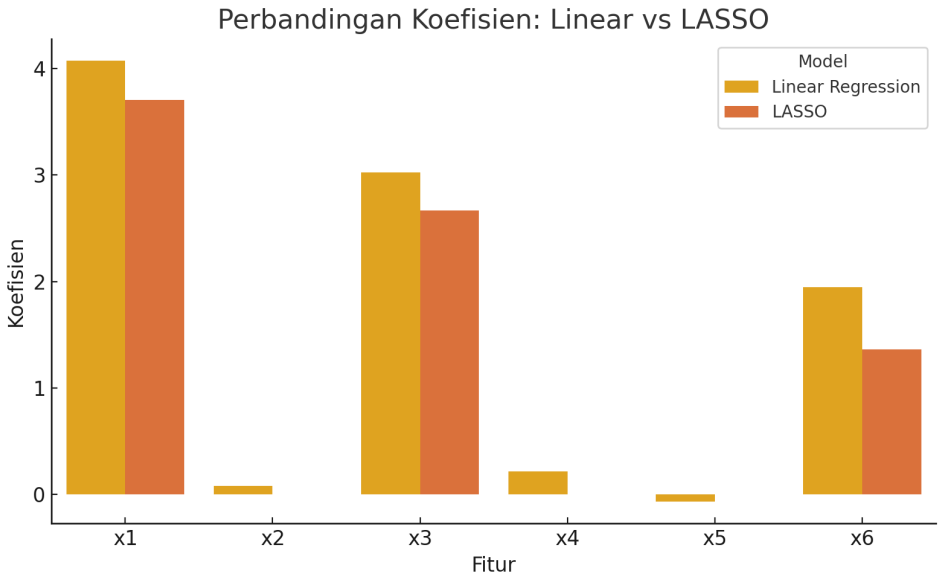
- Cocok ketika hanya sebagian kecil fitur yang benar-benar relevan
- Menghasilkan model yang lebih sederhana dan dapat diinterpretasikan
- Dapat digunakan untuk seleksi fitur

### Contoh Implementasi dengan `scikit-learn`

```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
import numpy as np

# Data sama seperti sebelumnya
np.random.seed(0)
n = 100
X = np.random.randn(n, 3)
true_coef = np.array([3, -2, 1.5])
y = X @ true_coef + np.random.normal(0, 0.5, size=n)

# LASSO regression
model = Lasso(alpha=0.1)
model.fit(X, y)

print("Koefisien:", model.coef_)
print("Intersep:", model.intercept_)
print("MSE:", mean_squared_error(y, model.predict(X)))
```

### Visualisasi: Efek LASSO terhadap Koefisien

Untuk memahami efek LASSO secara visual, Gambar 4.3 menunjukkan grafik yang membandingkan koefisien hasil regresi linier biasa dengan LASSO. Terlihat bahwa beberapa fitur memiliki koefisien nol akibat penalti L1.




**Gambar 4.3.** Perbandingan koefisien: Linear Regression vs LASSO

### Kelebihan dan Keterbatasan

LASSO sangat berguna ketika jumlah fitur sangat besar, tetapi juga memiliki keterbatasan:

- Jika ada fitur-fitur yang sangat berkorelasi, LASSO mungkin memilih salah satunya secara acak.
- Tidak stabil jika jumlah fitur lebih besar dari jumlah sampel.

### Ridge Regression (L2 Regularization)

**Ridge Regression** adalah bentuk regresi linier dengan regularisasi L2, yaitu menambahkan penalti berupa kuadrat dari nilai koefisien ke dalam fungsi loss. Tujuannya adalah untuk mencegah nilai koefisien menjadi terlalu besar, terutama saat terdapat multikolinearitas antar fitur.

### Fungsi Loss

Fungsi objektif yang diminimalkan pada Ridge Regression adalah:

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda \sum_{j=1}^{p} w_j^2$$

di mana:

- $w_j$ adalah koefisien regresi
- $\lambda$ adalah parameter regularisasi (juga dikenal sebagai *alpha* di beberapa library)

**Semakin besar** nilai $\lambda$, maka semakin kecil nilai koefisien (lebih "dipaksa" mendekati nol), dan model menjadi lebih sederhana.

### Karakteristik Ridge Regression

- Tidak menghilangkan fitur (tidak menghasilkan koefisien nol), hanya mengecilkan
- Cocok jika semua fitur dianggap penting
- Lebih stabil ketika fitur saling berkorelasi (multikolinearitas)

Perbedaan utama dari LASSO:

- Penalti berupa jumlah nilai kuadrat dari koefisien (bukan nilai absolut)
- Tidak mendorong koefisien menjadi tepat nol

### Contoh Implementasi dengan `scikit-learn`

```
from sklearn.linear_model import Ridge
from sklearn.metrics import mean_squared_error
import numpy as np

# Buat data sintetik
np.random.seed(0)
n = 100
X = np.random.randn(n, 3)  # 3 fitur
true_coef = np.array([3, -2, 1.5])
y = X @ true_coef + np.random.normal(0, 0.5, size=n)

# Ridge regression
model = Ridge(alpha=1.0)  # alpha = lambda
model.fit(X, y)

# Hasil
print("Koefisien:", model.coef_)
print("Intersep:", model.intercept_)
print("MSE:", mean_squared_error(y, model.predict(X)))
```

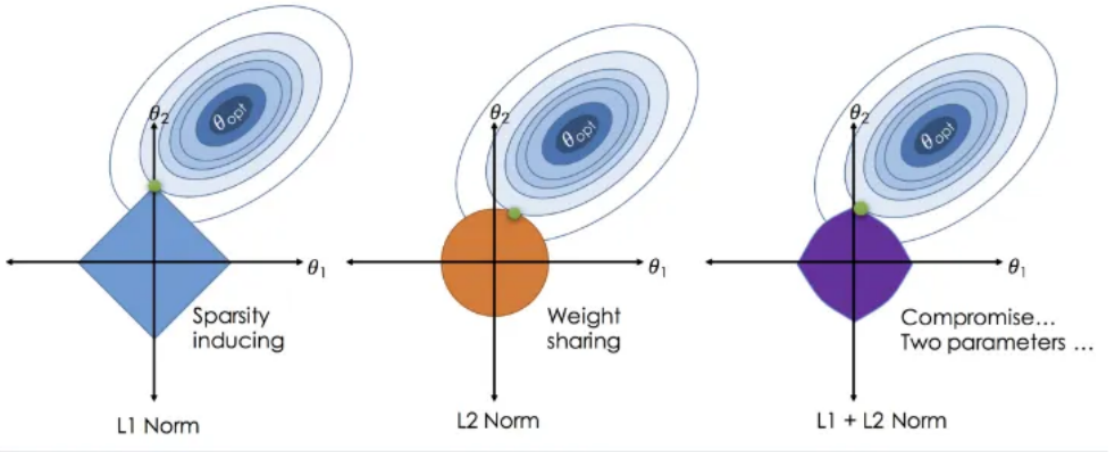
### Catatan

Jika $\lambda = 0$, maka Ridge Regression sama seperti regresi linier biasa (tanpa regularisasi). Nilai $\lambda$ dapat diatur menggunakan teknik validasi seperti *cross-validation* untuk mendapatkan hasil terbaik.

#### Elastic Net Regression (Gabungan L1 dan L2)

**Elastic Net** adalah metode regresi yang menggabungkan penalti dari Ridge (L2) dan LASSO (L1). Tujuannya adalah untuk memperoleh manfaat dari keduanya:

- Seperti LASSO: dapat melakukan seleksi fitur (koefisien bisa nol)




**Gambar 4.4.** Perbedaan LASSO, Ridge dan Elastic Net

- Seperti Ridge: stabil dalam situasi di mana fitur saling berkorelasi

Gambar 4.4 menggambarkan perbedaan antara LASSO, Ridge dan Elastic Net.

### Fungsi Loss

Fungsi loss yang diminimalkan:

$$\text{Loss} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 + \lambda\left[\alpha\sum_{j=1}^{p}|w_j| + (1-\alpha)\sum_{j=1}^{p}w_j^2\right]$$

di mana:

- $\lambda$ mengatur seberapa kuat penalti diterapkan
- $\alpha$ mengatur rasio antara L1 dan L2:
  - $\alpha = 1$: sama seperti LASSO
  - $\alpha = 0$: sama seperti Ridge
  - $0 < \alpha < 1$: kombinasi L1 dan L2

### Kapan Menggunakan Elastic Net?

- Saat jumlah fitur sangat banyak dan berkorelasi
- Saat LASSO tidak stabil dan Ridge terlalu konservatif

### Contoh Implementasi dengan `scikit-learn`

```
from sklearn.linear_model import ElasticNet
```

```
from sklearn.metrics import mean_squared_error
import numpy as np

# Data sama seperti sebelumnya
np.random.seed(0)
n = 100
X = np.random.randn(n, 3)
true_coef = np.array([3, -2, 1.5])
y = X @ true_coef + np.random.normal(0, 0.5, size=n)

# Elastic Net regression
model = ElasticNet(alpha=0.1, l1_ratio=0.5)  # l1_ratio = alpha dari formula
model.fit(X, y)

print("Koefisien:", model.coef_)
print("Intersep:", model.intercept_)
print("MSE:", mean_squared_error(y, model.predict(X)))
```

#### Perbandingan dan Kapan Menggunakan

Ketiga metode regularisasi — Ridge, LASSO, dan Elastic Net — memiliki kekuatan dan kelemahan masing-masing. Pemilihan metode yang tepat bergantung pada karakteristik data dan tujuan analisis.

| Metode | Karakteristik dan Kapan Digunakan |
|---|---|
| **Ridge (L2)** | Menyusutkan semua koefisien tetapi tidak membuatnya nol. Cocok jika semua fitur dianggap penting dan tidak ingin ada yang dihilangkan. Stabil terhadap multikolinearitas. |
| **LASSO (L1)** | Melakukan seleksi fitur dengan menyusutkan beberapa koefisien menjadi nol. Cocok saat hanya sedikit fitur yang dianggap penting (sparse model). Kurang stabil jika fitur sangat berkorelasi. |
| **Elastic Net (L1 + L2)** | Kombinasi terbaik dari Ridge dan LASSO. Cocok untuk data dengan banyak fitur yang saling berkorelasi. Dapat mengontrol keseimbangan antara penalti L1 dan L2. |

**Tabel 4.1.** Perbandingan Ridge, LASSO, dan Elastic Net

**Ringkasan:**

- Gunakan **Ridge** jika Anda lebih ingin mengurangi varians dan mempertahankan semua fitur.
- Gunakan **LASSO** jika Anda ingin menghasilkan model yang lebih sederhana dan otomatis melakukan seleksi fitur.

- Gunakan **Elastic Net** jika fitur sangat banyak dan saling berkorelasi — atau jika ingin menggabungkan kekuatan Ridge dan LASSO.

Dalam praktiknya, pemilihan parameter $\lambda$ dan $\alpha$ dapat dilakukan melalui teknik validasi seperti *cross-validation* untuk mendapatkan hasil terbaik.

## Referensi dan Bahan Bacaan Lanjutan


- Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, O'Reilly Media, 2022. [Bab 2 tentang regresi linier].
- scikit-learn User Guide – Linear and Ridge Regression: https://scikit-learn.org/stable/modules/linear_model.html
- Coursera – Machine Learning by Andrew Ng (Stanford): https://www.coursera.org/learn/machine-learning (Bagian awal membahas regresi linier dan regresi logistik dengan penjelasan yang sangat visual)
- Google ML Crash Course – Regression: https://developers.google.com/machine-learning/crash-course
- Hastie, Tibshirani, Friedman. *The Elements of Statistical Learning*, 2nd edition, Springer, 2009. https://web.stanford.edu/~hastie/ElemStatLearn/

# BAB 5
# Clustering

## 5.1 Konsep Clustering

*Clustering* adalah proses mengelompokkan data ke dalam grup (klaster) sedemikian rupa sehingga:

- Data dalam satu klaster memiliki kemiripan yang tinggi.
- Data dari klaster yang berbeda memiliki perbedaan yang signifikan.

Secara umum, clustering dapat dianggap sebagai proses menemukan struktur tersembunyi dalam data tanpa bantuan label. Pendekatan ini berguna ketika kita ingin memahami distribusi data, melakukan segmentasi, atau mempersiapkan data untuk algoritma pembelajaran lebih lanjut. Gambar 5.1 memberikan ilustrasi proses clustering.

#### Representasi Klaster

Setiap klaster biasanya direpresentasikan oleh:

- **Centroid**: Titik tengah dari klaster (misalnya dalam K-Means).
- **Himpunan anggota**: Sekumpulan data yang termasuk dalam klaster tersebut.

#### Ukuran Kemiripan

Untuk menentukan kemiripan antar data, beberapa metrik yang umum digunakan:

- **Jarak Euclidean:** Umum digunakan untuk data numerik.

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

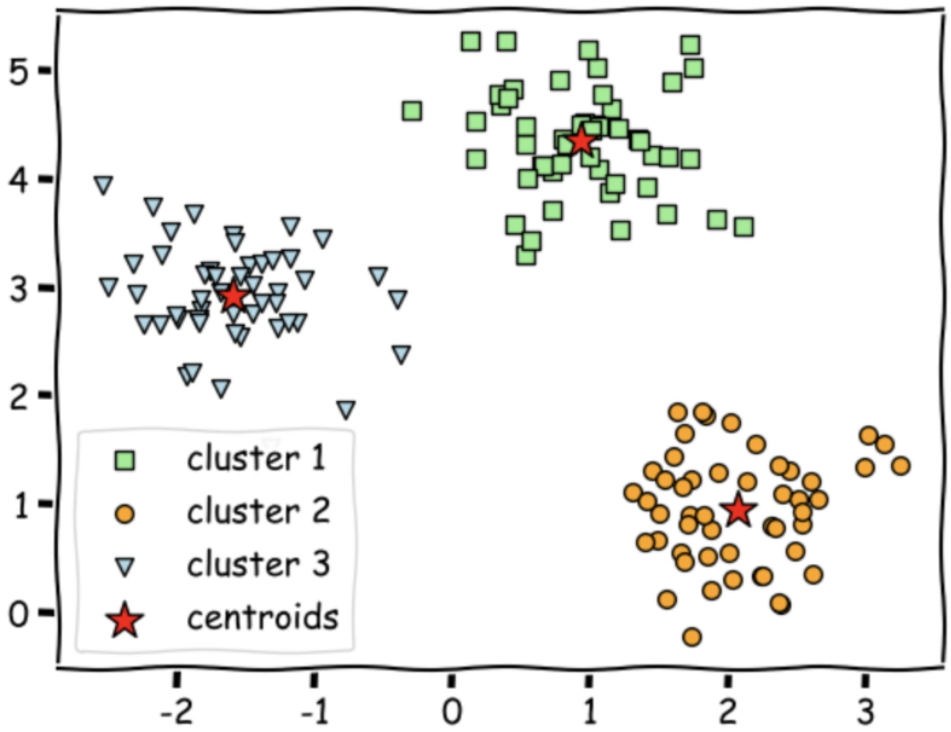
- **Jarak Manhattan (city block):**

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

- **Jarak cosine:** Cocok untuk data vektor arah, seperti dokumen.

$$\text{Cosine Similarity} = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|}$$

dimana $\|\mathbf{x}\|$ menyatakan panjang vektor $x$




**Gambar 5.1.** Ilustrasi Cluster (sumber: Naukri.com)

#### Jenis Clustering Berdasarkan Struktur

Terdapat beberapa pendekatan clustering berdasarkan bagaimana struktur klaster dibentuk:

1. **Partitional clustering:** Membagi data ke dalam $k$ klaster, seperti K-Means.
2. **Hierarchical clustering:** Membangun struktur seperti pohon (dendrogram).
3. **Density-based clustering:** Klaster ditentukan berdasarkan kepadatan data, misalnya DBSCAN.
4. **Model-based clustering:** Diasumsikan bahwa data berasal dari kombinasi distribusi probabilistik, seperti Gaussian Mixture Model.

Pada modul ini kita akan fokus pada dua metode utama: K-Means dan Hierarchical Clustering.

## 5.2 Metrik Evaluasi Clustering

Berbeda dengan klasifikasi, di mana kita memiliki label yang diketahui untuk mengevaluasi akurasi model, evaluasi hasil clustering lebih menantang karena umumnya dilakukan tanpa label. Oleh karena itu, metrik evaluasi clustering terbagi menjadi dua kategori utama:

- **Internal Evaluation:** Mengukur kualitas klaster berdasarkan data dan hasil klaster itu sendiri (tanpa label).
- **External Evaluation:** Menggunakan label kebenaran (jika tersedia) untuk mengevaluasi hasil clustering.

Pada konteks pembelajaran tak terawasi, fokus utama adalah evaluasi *internal*. Berikut ini beberapa metrik populer:

#### Inertia (Within-Cluster Sum of Squares)

Digunakan dalam K-Means, inertia mengukur total kuadrat jarak antara setiap titik dengan pusat klasternya:

$$\text{Inertia} = \sum_{i=1}^{k} \sum_{x \in C_i} \|\mathbf{x} - \mu_i\|^2$$

Semakin kecil inertia, semakin baik pembentukan klaster (asalkan tidak overfitting).

#### Silhouette Score

Silhouette Score mengukur seberapa mirip sebuah titik data dengan klasternya sendiri dibandingkan dengan klaster lain. Untuk tiap titik $i$:

$$s\left(i\right) = \frac{b\left(i\right) - a\left(i\right)}{\max\{a\left(i\right), b\left(i\right)\}}$$

di mana:

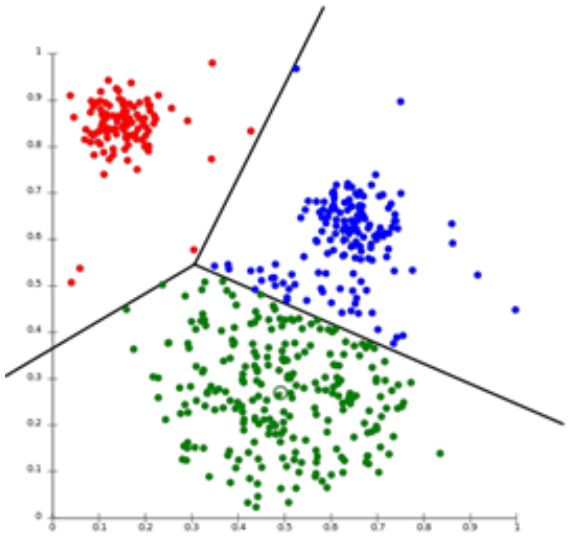
- $a\left(i\right)$ = rata-rata jarak antara $i$ dan semua titik dalam klasternya
- $b\left(i\right)$ = rata-rata jarak minimum antara $i$ dan semua titik di klaster lain

Nilai silhouette score berada di antara $-1$ dan 1:

- $s\left(i\right) \approx 1$: titik sangat cocok dengan klasternya
- $s\left(i\right) \approx 0$: titik berada di perbatasan dua klaster
- $s\left(i\right) < 0$: titik salah ditempatkan

#### Contoh Evaluasi Clustering dengan Scikit-learn

```
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.datasets import make_blobs

# Generate data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.6, random_state=0)

# Clustering
model = KMeans(n_clusters=4)
```



**Gambar 5.2.** Ilustrasi K-Means

```
model.fit(X)
labels = model.labels_

# Evaluation
print("Inertia:", model.inertia_)
print("Silhouette Score:", silhouette_score(X, labels))
```

## 5.3 K-Means Clustering

K-Means adalah salah satu algoritma clustering yang paling populer dan banyak digunakan. Tujuan utamanya adalah membagi data ke dalam $k$ klaster yang berbeda, di mana setiap data dimasukkan ke dalam klaster dengan centroid terdekat. Pada prinsipnya, K-Means membagi ruang dimensi tempat titik/vektor data berada menjadi $k$ buah ruangan/partisi berbeda (Gambar 5.2).

#### Langkah-langkah Algoritma K-Means

Algoritma bekerja dalam iterasi sebagai berikut:

1. Pilih $k$ titik awal (centroid) secara acak.
2. Untuk setiap titik data, hitung jaraknya ke semua centroid dan tentukan klaster terdekat.
3. Hitung centroid baru untuk setiap klaster berdasarkan rata-rata titik dalam klaster.
4. Ulangi langkah 2 dan 3 hingga konvergen (tidak ada perubahan signifikan pada centroid).

#### Fungsi Objektif K-Means

Tujuan dari K-Means adalah meminimalkan total jarak kuadrat dalam klaster:

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{\mathbf{x} \in C_i} \|\mathbf{x} - \mu_i\|^2$$

di mana:

- $C_i$ adalah himpunan anggota klaster ke-$i$
- $\mu_i$ adalah centroid dari $C_i$

#### Kelebihan dan Kekurangan K-Means

**Kelebihan:**

- Cepat dan efisien pada dataset besar
- Mudah dipahami dan diimplementasikan

**Kekurangan:**

- Harus menentukan jumlah klaster $k$ di awal
- Sensitif terhadap inisialisasi dan outlier
- Hanya cocok untuk bentuk klaster cenderung bulat atau seimbang

#### Contoh Implementasi Python (Scikit-learn)

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate data
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=0.7, random_state=42)

# Apply KMeans
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)

# Plot hasil
plt.scatter(X[:, 0], X[:, 1], c=kmeans.labels_, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            s=200, c='red', marker='X', label='Centroid')
plt.title('Hasil Clustering dengan K-Means')
plt.legend()
plt.show()
```

#### Menentukan Jumlah Klaster: Elbow Method

Salah satu tantangan dalam K-Means adalah menentukan nilai $k$ (jumlah klaster) yang optimal. **Elbow Method** adalah pendekatan visual yang umum digunakan untuk menentukan jumlah klaster berdasarkan nilai *inertia*.

**Inertia** mengukur total kuadrat jarak antara setiap titik dan centroid klasternya. Semakin besar $k$, maka inertia akan semakin kecil (karena klaster lebih sempit). Namun, setelah titik tertentu, penurunan inertia menjadi tidak signifikan. Titik "siku" atau "elbow" pada grafik adalah kandidat terbaik untuk jumlah klaster optimal.

### Langkah-langkah Elbow Method

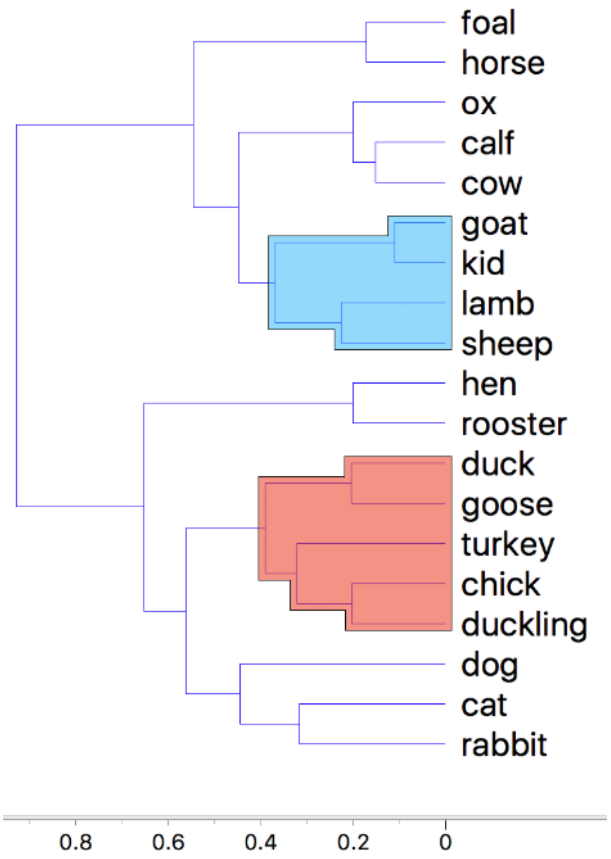
1. Jalankan K-Means dengan berbagai nilai $k$ (misalnya 1 sampai 10).
2. Simpan nilai inertia untuk setiap $k$.
3. Plot grafik $k$ vs inertia.
4. Cari titik di mana kurva mulai melandai — itulah "elbow".

### Contoh Implementasi Elbow Method

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs

# Generate data
X, _ = make_blobs(n_samples=300, centers=4, cluster_std=0.6, random_state=0)

inertias = []
K = range(1, 11)

for k in K:
    model = KMeans(n_clusters=k, random_state=0)
    model.fit(X)
    inertias.append(model.inertia_)

# Plot elbow graph
plt.figure(figsize=(8, 4))
plt.plot(K, inertias, 'bo-')
plt.xlabel('Jumlah Klaster (k)')
plt.ylabel('Inertia')
plt.title('Elbow Method untuk Menentukan k')
plt.grid(True)
plt.show()
```

### Catatan

Elbow Method bersifat visual dan subjektif. Dalam beberapa kasus, tidak terdapat "elbow" yang jelas. Alternatif lain yang bisa digunakan antara lain:




**Gambar 5.3.** Contoh Hierarchical Clustering

- **Silhouette Score**
- **Gap Statistic**

## 5.4 Hierarchical Clustering

Hierarchical Clustering adalah metode pengelompokan data yang membentuk struktur hierarki. Alih-alih menentukan jumlah klaster sejak awal seperti pada K-Means, hierarchical clustering menghasilkan pohon (dendrogram) yang menunjukkan bagaimana klaster-klaster terbentuk dari data, dan kita bisa memilih jumlah klaster berdasarkan struktur tersebut. Gambar 5.3 menunjukkan contoh hasil sebuah *hierarchical clustering*.

#### Perbedaan dengan K-Means

- **Penentuan jumlah klaster:** K-Means memerlukan jumlah klaster $k$ ditentukan sebelumnya, sedangkan hierarchical clustering tidak.
- **Struktur klaster:** K-Means menghasilkan partisi "fixed" (tetap), sementara hierarchical clustering menghasilkan struktur pohon (dendrogram), dimana kita bisa memilih untuk "memotong" pohon tersebut pada jarak/level tertentu untuk men-

dapatkan jumlah klaster yang berbeda-beda.

- ◇ **Kestabilan hasil:** K-Means dapat menghasilkan hasil berbeda tergantung inisialisasi centroid. Hierarchical clustering menghasilkan hasil yang deterministik (konsisten).
- ◇ **Kompleksitas:** Hierarchical clustering lebih mahal secara komputasi dibanding K-Means, khususnya untuk data besar.

#### Dua Pendekatan Umum

1. **Agglomerative (bottom-up):** Setiap titik data awalnya merupakan klaster tersendiri. Klaster paling mirip digabungkan hingga seluruh data menjadi satu klaster besar.
2. **Divisive (top-down):** Seluruh data dimulai dalam satu klaster, lalu dipisahkan secara bertahap.

#### Linkage: Ukuran Jarak antar Klaster

Untuk menentukan klaster mana yang akan digabungkan (atau dipecah), kita perlu mendefinisikan jarak antar dua klaster. Berikut metode linkage umum beserta rumusnya:

- ◇ **Single Linkage:** Jarak minimum antara dua titik dari dua klaster.

$$D(A, B) = \min_{\mathbf{x} \in A, \mathbf{y} \in B} \|\mathbf{x} - \mathbf{y}\|$$

- ◇ **Complete Linkage:** Jarak maksimum antara dua titik dari dua klaster.

$$D(A, B) = \max_{\mathbf{x} \in A, \mathbf{y} \in B} \|\mathbf{x} - \mathbf{y}\|$$

- ◇ **Average Linkage:** Rata-rata jarak antar semua pasangan titik dari dua klaster.

$$D(A, B) = \frac{1}{|A||B|} \sum_{\mathbf{x} \in A} \sum_{\mathbf{y} \in B} \|\mathbf{x} - \mathbf{y}\|$$

- ◇ **Ward's Method:** Menyusun klaster agar peningkatan total variansi dalam-klaster seminimal mungkin. Rumus jarak Ward biasanya dihitung sebagai:

$$D(A, B) = \frac{|A||B|}{|A| + |B|} \|\bar{\mathbf{x}}_A - \bar{\mathbf{x}}_B\|^2$$

di mana $\bar{\mathbf{x}}_A$ dan $\bar{\mathbf{x}}_B$ adalah centroid dari klaster $A$ dan $B$.

#### Visualisasi Dendrogram

Hierarchical clustering dapat divisualisasikan menggunakan *dendrogram*, yaitu diagram pohon yang menunjukkan urutan penggabungan klaster. Tinggi cabang menunjukkan jarak antar klaster saat penggabungan.

```
from sklearn.datasets import make_blobs
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt

X, _ = make_blobs(n_samples=100, centers=3, random_state=42)
linked = linkage(X, method='ward')  # bisa diganti dengan 'single', 'complete', 'aver

plt.figure(figsize=(10, 5))
dendrogram(linked)
plt.title('Dendrogram Clustering')
plt.xlabel('Data Point')
plt.ylabel('Jarak')
plt.show()
```

#### Pemotongan Dendrogram

Untuk mendapatkan hasil klaster tertentu, kita dapat "memotong" dendrogram pada jarak tertentu:

```
from scipy.cluster.hierarchy import fcluster

# Ambil 3 klaster
labels = fcluster(linked, t=3, criterion='maxclust')

plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='rainbow')
plt.title('Hasil Clustering dari Dendrogram')
plt.show()
```

## 5.5 Contoh dan Implementasi

Untuk memberikan gambaran nyata bagaimana metode clustering bekerja, mari kita bandingkan hasil clustering menggunakan dua algoritma berbeda — K-Means dan Hierarchical Clustering — pada dataset yang sama.

#### Dataset Sintetik

Kita akan menggunakan `make_blobs` dari `sklearn.datasets` untuk membuat data dengan 3 klaster terpisah secara visual.

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt

# Membuat data
X, _ = make_blobs(n_samples=300, centers=3, cluster_std=0.7, random_state=42)

plt.scatter(X[:, 0], X[:, 1], s=30)
plt.title("Data Sintetik")
plt.show()
```

#### Clustering dengan K-Means

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=42)
kmeans.fit(X)
labels_kmeans = kmeans.labels_

plt.scatter(X[:, 0], X[:, 1], c=labels_kmeans, cmap='viridis')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1],
            s=200, c='red', marker='X', label='Centroid')
plt.title("Hasil Clustering dengan K-Means")
plt.legend()
plt.show()
```

#### Clustering dengan Hierarchical (Agglomerative)

```
from scipy.cluster.hierarchy import linkage, fcluster

linked = linkage(X, method='ward')
labels_hier = fcluster(linked, t=3, criterion='maxclust')

plt.scatter(X[:, 0], X[:, 1], c=labels_hier, cmap='rainbow')
plt.title("Hasil Clustering dengan Hierarchical (Ward)")
plt.show()
```

#### Catatan Perbandingan

- **K-Means:** cepat dan efisien, cocok untuk data dengan bentuk klaster bulat dan ukuran relatif seimbang.
- **Hierarchical:** lebih fleksibel, dapat menangkap struktur bertingkat, tetapi lebih mahal secara komputasi.
- **Visualisasi dendrogram** bisa membantu memilih jumlah klaster optimal tanpa eksplisit menentukan $k$ sebelumnya.

## Referensi dan Bahan Bacaan Lanjutan

Berikut adalah beberapa referensi dan sumber belajar lanjutan yang dapat digunakan untuk memahami lebih dalam topik clustering:

1. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. *An Introduction to Statistical Learning* (2nd Edition). Springer, 2021.

    - Bab 10: Unsupervised Learning – membahas clustering dan metode PCA.
    - Tersedia gratis di: https://www.statlearning.com/

2. Google Developers Machine Learning Crash Course:

    - Materi interaktif dengan visualisasi untuk pemula.
    - https://developers.google.com/machine-learning/crash-course

3. Scikit-learn documentation (official):

    - Penjelasan dan dokumentasi lengkap untuk semua algoritma clustering.
    - https://scikit-learn.org/stable/modules/clustering.html

4. YouTube – StatQuest by Josh Starmer:

    - Video penjelasan sangat jernih untuk K-Means dan hierarchical clustering.
    - K-Means: https://youtu.be/4b5d3muPQmA
    - Hierarchical Clustering: https://youtu.be/7xHsRkOdVwo

# BAB 6
# Metode Pelatihan

## 6.1 Optimasi dengan Gradient Descent

Gradient Descent (GD) adalah algoritma optimasi yang paling umum digunakan dalam pelatihan model machine learning dan deep learning. Tujuan dari GD adalah untuk meminimalkan fungsi kerugian (loss function) dengan bergerak ke arah negatif gradien dari fungsi tersebut.

### Prinsip Dasar Gradient Descent

Jika $f(\theta)$ adalah fungsi kerugian (loss) yang ingin kita minimalkan terhadap parameter $\theta$, maka update parameter dilakukan dengan:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla f\left(\theta^{(t)}\right)$$

di mana:

- $\theta^{(t)}$ adalah nilai parameter pada iterasi ke-$t$
- $\eta$ adalah learning rate
- $\nabla f\left(\theta^{(t)}\right)$ adalah gradien fungsi kerugian terhadap $\theta$

### Visualisasi Fungsi Loss dan Lintasan Gradient Descent

Untuk membantu memahami bagaimana Gradient Descent bekerja, kita dapat memvisualisasikan bentuk permukaan fungsi loss dan lintasan pergerakan parameter menuju minimum global.

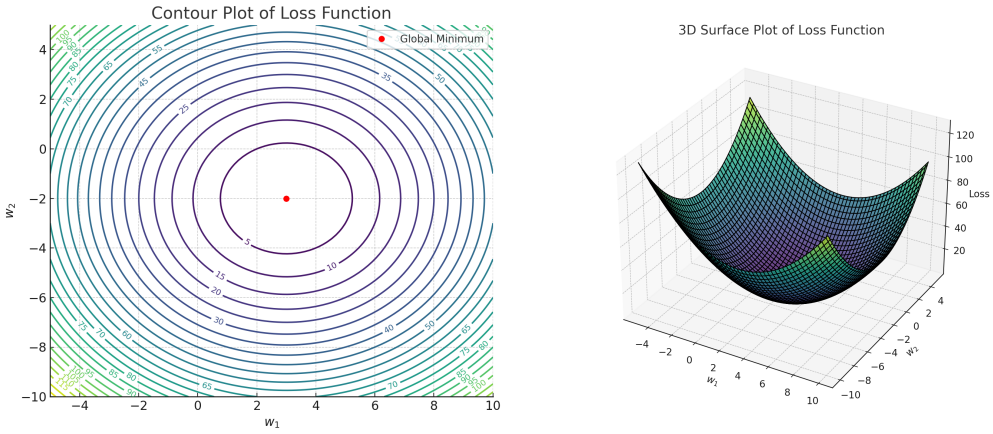
- Fungsi loss sederhana dapat ditulis sebagai:

$$L(w_1, w_2) = (w_1 - 3)^2 + (w_2 + 2)^2$$

Minimum global dari fungsi ini berada pada titik $(w_1, w_2) = (3, -2)$.

Simulasi Gradient Descent dari titik awal $(w_1, w_2) = (-4, 4)$ menunjukkan lintasan parameter menuju minimum ditunjukkan pada Gambar 6.2.
**Catatan:**

- Setiap langkah GD bergerak ke arah negatif gradien.




(a) Kontur fungsi loss

(b) Permukaan 3D loss

**Gambar 6.1.** Visualisasi fungsi loss dan proses optimasi dengan Gradient Descent.

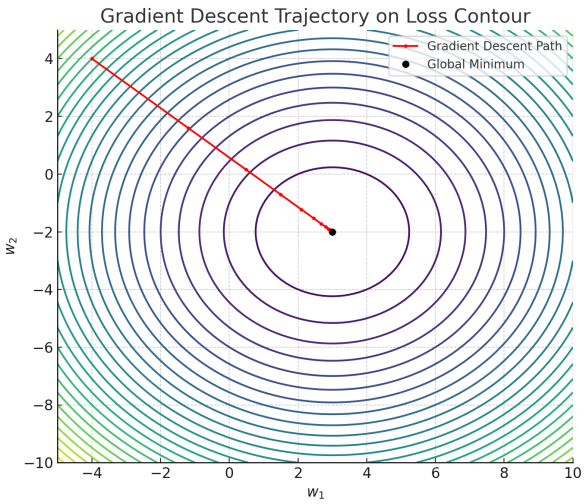
- Langkah yang terlalu besar bisa melampaui minimum, sedangkan terlalu kecil membuat proses lambat.
- Optimizer seperti `ADAM` menggabungkan momentum dan adaptasi ukuran langkah agar lebih stabil dan cepat.

### Batch Gradient Descent

Pada batch gradient descent, seluruh data digunakan untuk menghitung gradien sebelum parameter diperbarui. Ini membuat proses stabil namun bisa sangat lambat untuk dataset besar.

**Contoh Python:**

```
import numpy as np

# fungsi: f(x) = (x - 3)^2
def f(x):
    return (x - 3)**2

# turunan: f'(x) = 2(x - 3)
def grad_f(x):
    return 2 * (x - 3)

# inisialisasi
x = 0.0
eta = 0.1
history = []

for i in range(20):
    x = x - eta * grad_f(x)
```




**Gambar 6.2.** Lintasan Gradient Descent pada permukaan kontur loss.

```
    history.append(x)

print("Nilai minimum mendekati:", x)
```

### Stochastic Gradient Descent (SGD)

SGD memperbarui parameter berdasarkan satu data saja secara acak. Ini membuat proses lebih cepat namun fluktuatif.

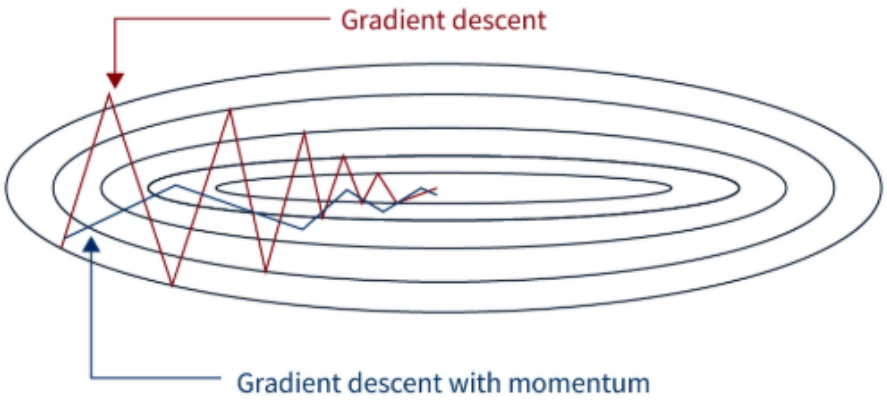
**Contoh ilustrasi Python:**

```
# simulasi data linear: y = 2x + noise
X = np.random.rand(100)
y = 2 * X + np.random.randn(100) * 0.1

w = 0.0  # inisialisasi bobot
eta = 0.05

for epoch in range(10):
    for i in range(100):
        xi, yi = X[i], y[i]
        grad = 2 * (w * xi - yi) * xi
        w -= eta * grad

print("Bobot w mendekati:", w)
```

### Mini-batch Gradient Descent

Mini-batch GD adalah kompromi antara batch dan SGD. Data dibagi dalam batch kecil (misal 32 atau 64), dan parameter diperbarui per batch.

```
batch_size = 16
```




**Gambar 6.3.** Ilustrasi momentum (sumber: scaler.com)

```
for epoch in range(10):
    perm = np.random.permutation(len(X))
    X_shuffled = X[perm]
    y_shuffled = y[perm]
    for i in range(0, len(X), batch_size):
        xb = X_shuffled[i:i+batch_size]
        yb = y_shuffled[i:i+batch_size]
        grad = 2 * np.mean((w * xb - yb) * xb)
        w -= eta * grad
```

#### Momentum

Salah satu kelemahan dari metode gradient descent standar adalah kecepatannya yang lambat saat menghadapi lembah yang sempit dan curam dalam satu arah, namun datar pada arah lainnya (contohnya: fungsi kerugian berbentuk seperti mangkuk memanjang). Dalam situasi seperti ini, GD bisa berosilasi maju-mundur di sepanjang arah curam dan bergerak sangat lambat ke arah minimum global.

**Momentum** adalah teknik yang diperkenalkan untuk mengurangi osilasi dan mempercepat konvergensi. Idenya mirip seperti fisika klasik: saat bola menggelinding ke bawah lereng, kecepatannya akan bertambah karena momentum.

Dengan momentum, kita tidak hanya mempertimbangkan gradien saat ini, tetapi juga "kecepatan" dari langkah-langkah sebelumnya. Ini membuat pergerakan ke arah gradien menjadi lebih mulus dan cepat, terutama dalam lembah sempit atau permukaan loss yang bergelombang. Gambar 6.3 menunjukkan ilustrasi dampak penerapan momentum terhadap proses optimasi.

**Rumus:**

$$v_t = \gamma v_{t-1} + \eta \nabla f\left(\theta^{(t)}\right)$$
$$\theta^{(t+1)} = \theta^{(t)} - v_t$$

di mana:

- $v_t$ adalah "kecepatan" akumulatif dari arah gradien sebelumnya
- $\gamma \in (0, 1)$ adalah koefisien momentum, biasanya dipilih 0.9
- $\eta$ adalah learning rate

Dengan adanya momentum, update parameter akan tetap "bergerak" dalam arah yang konsisten meskipun gradiennya kecil — dan dapat membantu keluar dari lokal minima kecil.

**Contoh Implementasi Python:**

```
w = 0.0       # parameter
v = 0.0       # momentum
eta = 0.05    # learning rate
gamma = 0.9   # momentum coefficient

for epoch in range(10):
    for i in range(100):
        xi, yi = X[i], y[i]
        grad = 2 * (w * xi - yi) * xi
        v = gamma * v + eta * grad
        w -= v

print("Bobot akhir:", w)
```

## 6.2 Algoritma Optimasi Modern

Meskipun Gradient Descent dan variasinya seperti momentum bekerja dengan baik, ada beberapa kekurangan pada pendekatan klasik, terutama dalam hal:

- Pengaturan learning rate yang sensitif.
- Kecepatan konvergensi berbeda untuk setiap dimensi/parameter.
- Performa lambat saat menghadapi fitur dengan skala berbeda.

Untuk mengatasi hal tersebut, berbagai algoritma optimasi modern telah dikembangkan, di antaranya: Adagrad, RMSprop, dan Adam.

#### Adagrad (Adaptive Gradient)

Adagrad secara otomatis menyesuaikan learning rate untuk setiap parameter berdasarkan seberapa sering parameter tersebut diperbarui.

**Prinsip:** Parameter yang sering mendapatkan gradien besar akan memiliki learning rate yang lebih kecil, sedangkan parameter yang jarang diupdate akan memiliki learning rate yang lebih besar.

**Rumus update:**

$$g_t = \nabla f(\theta_t)$$

$$G_t = G_{t-1} + g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

**Kelebihan:** Efektif untuk fitur sparse (jarang muncul). **Kekurangan:** Akumulasi gradien bisa menyebabkan learning rate terlalu kecil (mati).

#### RMSprop (Root Mean Square Propagation)

Untuk mengatasi penurunan learning rate yang terlalu agresif seperti pada Adagrad, RMSprop menggunakan **rata-rata bergerak** dari kuadrat gradien.

$$g_t = \nabla f(\theta_t)$$
$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

Nilai $\gamma$ biasanya 0.9. **Kelebihan:** Tidak menurunkan learning rate terlalu drastis.

#### Adam (Adaptive Moment Estimation)

Adam menggabungkan keuntungan dari momentum dan RMSprop, yaitu:

- Estimasi rata-rata pertama (momentum dari gradien)
- Estimasi rata-rata kedua (kuadrat gradien)

**Rumus:**

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

Nilai umum:

- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\epsilon = 10^{-8}$

**Kelebihan:**

- Sangat populer dan stabil untuk banyak arsitektur deep learning.
- Menangani sparse gradient dan fitur skala berbeda dengan baik.

#### Contoh Implementasi Adam di Python

Untuk ilustrasi sederhana: **Catatan: Biasanya Adam digunakan melalui library seperti `PyTorch` atau `TensorFlow`.**

```
m, v = 0, 0
beta1, beta2 = 0.9, 0.999
epsilon = 1e-8
w = 0.0

for t in range(1, 101):
    g = 2 * (w - 3)  # grad dari f(w) = (w - 3)^2
    m = beta1 * m + (1 - beta1) * g
    v = beta2 * v + (1 - beta2) * g**2

    m_hat = m / (1 - beta1**t)
    v_hat = v / (1 - beta2**t)

    w -= 0.1 * m_hat / (np.sqrt(v_hat) + epsilon)

print("Estimasi w mendekati:", w)
```

## 6.3 Pembagian Dataset: Latih, Validasi, dan Uji

Dalam proses pelatihan model machine learning, sangat penting untuk membagi data menjadi beberapa bagian yang berbeda agar model dapat belajar secara efektif dan dievaluasi secara adil. Umumnya, dataset dibagi menjadi tiga bagian utama:

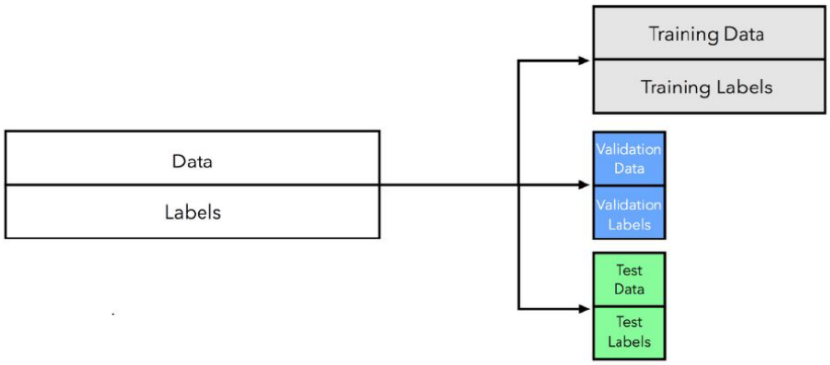
1. **Training set (data latih):** Digunakan untuk melatih model. Model mempelajari pola dari data ini.
2. **Validation set (data validasi):** Digunakan untuk mengevaluasi performa model selama proses pelatihan, terutama untuk tuning hyperparameter dan mencegah overfitting.
3. **Test set (data uji):** Digunakan untuk mengevaluasi performa akhir dari model pada data yang benar-benar baru dan belum pernah dilihat sebelumnya.

Gambar 6.4 menunjukkan ilustrasi pembagian data latih, validasi dan uji.

#### Proporsi Umum

Proporsi pembagian bisa bervariasi, namun secara umum:

- 60% data latih, 20% validasi, 20% uji
- 70% latih, 30% uji (tanpa validasi jika validasi silang digunakan)




**Gambar 6.4.** Pembagian data latih, validasi dan uji

#### Mengapa Perlu Dipisah?

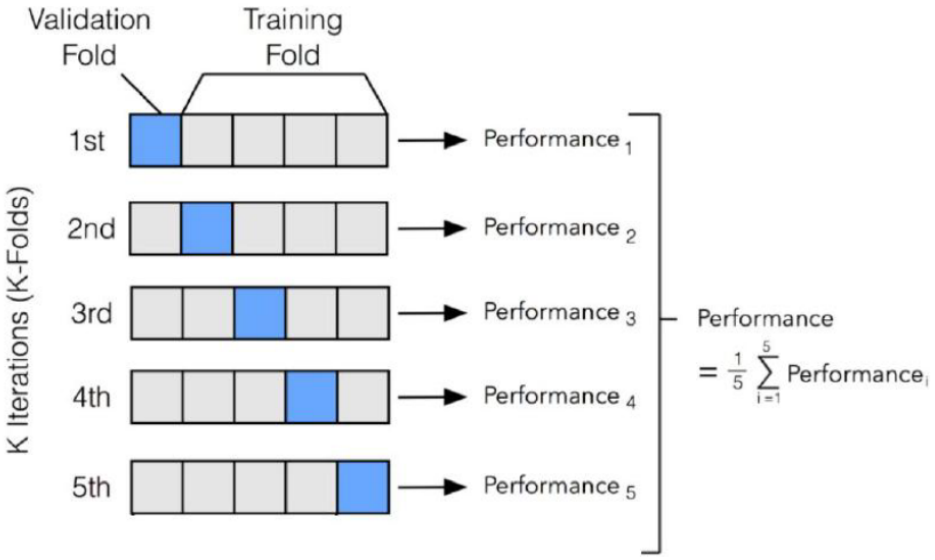
Tanpa memisahkan data uji, kita tidak bisa menilai apakah model benar-benar generalisasi ke data baru. Jika kita mengevaluasi pada data yang sama dengan data pelatihan, hasilnya bias karena model bisa saja hanya "menghafal" data latih (overfitting).

#### Contoh Implementasi di Python (dengan scikit-learn)

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

# Muat dataset
X, y = load_iris(return_X_y=True)

# Bagi menjadi data latih dan data uji (80:20)
X_train, X_temp, y_train, y_temp = train_test_split(X, y,
    test_size=0.4, random_state=42)

# Bagi sisanya menjadi validasi dan uji (masing-masing 50%)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
    test_size=0.5, random_state=42)

print("Ukuran data latih:", len(X_train))
print("Ukuran data validasi:", len(X_val))
print("Ukuran data uji:", len(X_test))
```

## 6.4 Validasi Silang (Cross Validation)

Validasi silang adalah teknik evaluasi model yang lebih andal dan efisien dibandingkan pembagian data secara tetap (hold-out). Tujuan utamanya adalah untuk memastikan bahwa performa model tidak bergantung pada subset data tertentu saja, sehingga hasil evaluasi menjadi lebih stabil dan umum. Gambar 6.5 menunjukkan ilustrasi validasi silang dengan K-Fold Cross-validation.




**Gambar 6.5.** Validasi silang dengan K-Fold

#### Hold-Out Validation

Ini adalah metode paling sederhana, di mana dataset dibagi menjadi dua atau tiga bagian (train-validation-test) seperti pada subbab sebelumnya. Kelemahan utamanya adalah performa model bisa sangat tergantung pada cara data dibagi — bisa jadi hasilnya bagus hanya karena data kebetulan "mudah".

#### K-Fold Cross Validation

K-Fold CV membagi data menjadi $K$ bagian (fold). Proses dilakukan sebanyak $K$ kali:

- Setiap kali, satu fold digunakan sebagai data validasi, dan sisanya digunakan sebagai data latih.
- Hasil evaluasi dari tiap fold dirata-ratakan untuk mendapatkan metrik akhir.

Biasanya $K = 5$ atau $K = 10$.
**Ilustrasi K-Fold dengan $K = 5$:**

$$\begin{aligned} &\text{Fold 1:} \quad \text{Val: } \mathrm{D}_1, \quad \text{Train: } \mathrm{D}_2 \cup D_3 \cup D_4 \cup D_5 \\ &\text{Fold 2:} \quad \text{Val: } \mathrm{D}_2, \quad \text{Train: } \mathrm{D}_1 \cup D_3 \cup D_4 \cup D_5 \\ &\qquad\qquad \ldots \end{aligned}$$

#### Stratified K-Fold

Untuk data klasifikasi, penting agar proporsi kelas seimbang di setiap fold. Stratified K-Fold menjaga distribusi kelas agar tetap representatif di setiap fold.

#### Leave-One-Out (LOO)

LOO adalah kasus ekstrem dari K-Fold dengan $K = n$ (jumlah data). Setiap data diuji satu per satu, sisanya dilatih. Sangat akurat tapi sangat mahal secara komputasi.

#### Contoh Implementasi K-Fold di Python

```
from sklearn.datasets import load_iris
from sklearn.model_selection import KFold, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import numpy as np

X, y = load_iris(return_X_y=True)

kf = KFold(n_splits=5, shuffle=True, random_state=42)
# Atau gunakan Stratified K-Fold untuk
# menjaga proporsi kelas di setiap lipatan
# kf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

scores = []
for train_index, val_index in kf.split(X):
    X_train, X_val = X[train_index], X[val_index]
    y_train, y_val = y[train_index], y[val_index]

    model = LogisticRegression(max_iter=1000)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    scores.append(accuracy_score(y_val, y_pred))

print("Akurasi rata-rata:", np.mean(scores))
```

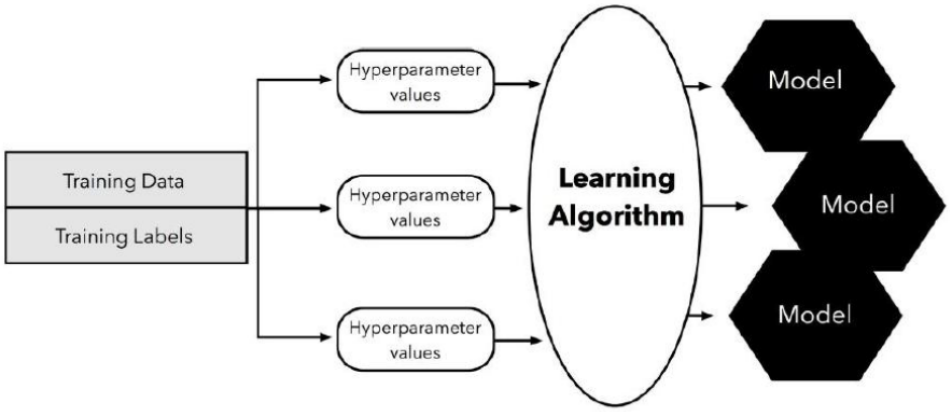
**Catatan: Kapan dan Mengapa Menggunakan Cross-Validation**

Cross-validation sangat berguna ketika ukuran dataset terbatas. Dengan membagi data menjadi beberapa lipatan, kita dapat:

- Menggunakan seluruh data untuk pelatihan dan evaluasi secara bergantian
- Mendapatkan estimasi performa model yang lebih stabil dan tidak bergantung pada satu pembagian tertentu
- Menghindari bias evaluasi yang dapat terjadi jika kita hanya mengandalkan satu `train/test split`

Namun, perlu diperhatikan bahwa:

- Cross-validation memerlukan waktu komputasi yang lebih besar, karena model dilatih berulang kali sebanyak jumlah lipatan ($k$).
- Oleh karena itu, CV sebaiknya digunakan saat dataset cukup kecil atau saat kita




**Gambar 6.6.** Ilustrasi hyperparameter tuning

ingin mengevaluasi model secara menyeluruh (misalnya dalam riset atau kompetisi).

## 6.5 Tuning Hyperparameter

Hyperparameter adalah parameter yang tidak dipelajari langsung oleh model dari data, melainkan harus ditentukan sebelum pelatihan dimulai. Contohnya meliputi:

- Learning rate pada algoritma optimasi
- Jumlah tetangga pada KNN
- Nilai $C$ dan $\gamma$ pada SVM
- Jumlah pohon pada Random Forest

Memilih kombinasi hyperparameter yang tepat sangat penting untuk mendapatkan model yang berkinerja baik dan mampu generalisasi. Untuk melakukan pemilihan ini, kita harus mencoba beberapa **kombinasi** nilai yang mungkin untuk semua hyperparameter, dan memilih yang menghasilkan model terbaik (Gambar 6.6).

### Grid Search

Metode paling umum dan sederhana: mencoba semua kombinasi dari beberapa nilai hyperparameter yang telah ditentukan sebelumnya.

**Contoh:** Untuk KNN, kita ingin mencoba nilai $k = 1$ hingga 10 dan menggunakan cross-validation untuk mengevaluasi performanya.

```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

param_grid = {'n_neighbors': np.arange(1, 11)}
grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5)
grid.fit(X, y)
```

```
print("Nilai k terbaik:", grid.best_params_)
print("Akurasi terbaik:", grid.best_score_)
```

#### Random Search

Daripada mencoba semua kombinasi, random search mencoba kombinasi secara acak dari ruang parameter yang telah ditentukan. Ini lebih efisien saat ruang parameter besar.

#### Bayesian Optimization (Lanjutan)

Merupakan pendekatan yang lebih cerdas dengan memodelkan fungsi objektif (misalnya akurasi) dan memilih kombinasi hyperparameter berdasarkan prediksi performa terbaik. Contoh tools: `Optuna`, `Hyperopt`, `skopt`.

## 6.6 Underfitting dan Overfitting

Salah satu tantangan dalam membangun model machine learning adalah mendapatkan model yang tidak terlalu sederhana maupun terlalu kompleks. Dua konsep penting dalam konteks ini adalah **underfitting** dan **overfitting**.

#### Underfitting

Underfitting terjadi ketika model terlalu sederhana untuk menangkap pola dalam data. Akibatnya, baik pada data latih maupun uji, akurasinya rendah.

Contoh penyebab underfitting:

- Model terlalu sederhana (misalnya regresi linier untuk pola nonlinier)
- Terlalu sedikit fitur
- Waktu pelatihan terlalu singkat

#### Overfitting

Overfitting terjadi ketika model terlalu kompleks dan belajar terlalu banyak dari data latih, termasuk noise. Model memiliki akurasi tinggi pada data latih, namun buruk pada data uji.

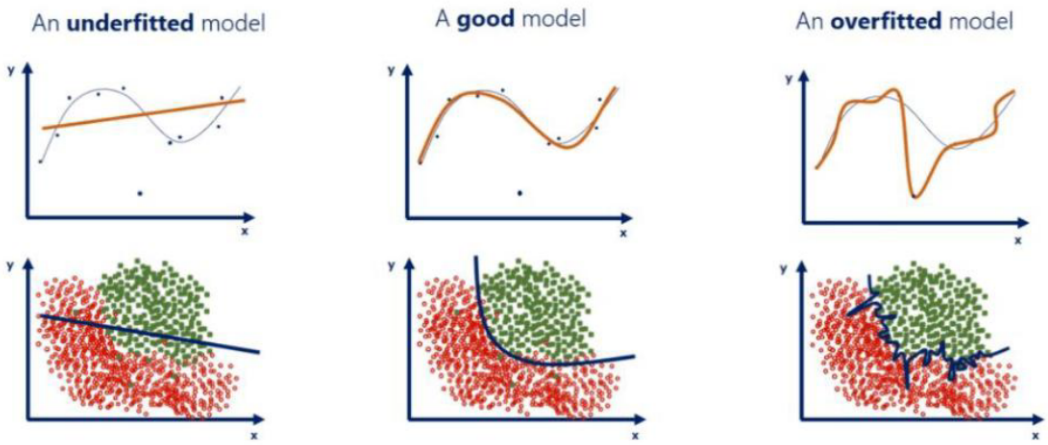
Contoh penyebab overfitting:

- Model terlalu kompleks (misalnya decision tree sangat dalam)
- Jumlah fitur terlalu banyak
- Jumlah data terlalu sedikit

Sebuah model yang baik adalah model yang mampu menghindari underfitting serta overfitting dan memiliki kemampuan *generalisasi* yang paling baik (Gambar 6.7).

#### Trade-off Bias dan Varians

- **Bias tinggi** → underfitting - **Varians tinggi** → overfitting




**Gambar 6.7.** Ilustrasi underfitting dan overfitting

Tujuan pelatihan adalah menemukan titik optimal antara bias dan varians.

#### Solusi terhadap Overfitting

- Menggunakan lebih banyak data
- Regularisasi (misalnya L1/L2) seperti pada LASSO/Ridge
- Early stopping
- Reduksi dimensi (PCA)
- Cross-validation

#### Learning Curve

Learning curve adalah grafik yang menunjukkan performa model (biasanya akurasi atau error) terhadap ukuran data latih. Ini dapat digunakan untuk mendiagnosis underfitting atau overfitting.

- Jika akurasi latih dan uji sama-sama rendah → underfitting
- Jika akurasi latih tinggi tetapi uji rendah → overfitting

#### Contoh Visualisasi Learning Curve (Python)

```
from sklearn.model_selection import learning_curve
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt
import numpy as np

train_sizes, train_scores, test_scores = learning_curve(
    LogisticRegression(max_iter=1000),
    X, y,
    train_sizes=np.linspace(0.1, 1.0, 5),
    cv=5
```

```
)

train_scores_mean = train_scores.mean(axis=1)
test_scores_mean = test_scores.mean(axis=1)

plt.plot(train_sizes, train_scores_mean, label="Train")
plt.plot(train_sizes, test_scores_mean, label="Validation")
plt.xlabel("Jumlah data latih")
plt.ylabel("Akurasi")
plt.legend()
plt.title("Learning Curve")
plt.show()
```

## Referensi dan Bahan Bacaan Lanjutan

Untuk mempelajari lebih lanjut tentang optimasi, overfitting, hyperparameter tuning, dan teknik pelatihan model lainnya, berikut beberapa sumber yang direkomendasikan:

- **Deep Learning Book – Ian Goodfellow, Yoshua Bengio, Aaron Courville** Bab 8 membahas algoritma optimasi berbasis gradient descent dan variannya. https://www.deeplearningbook.org/
- **CS231n: Optimization for Training Deep Models (Stanford University)** Penjelasan singkat dan visual tentang gradient descent, momentum, RMSprop, Adam. https://cs231n.github.io/optimization-1/
- **Scikit-learn User Guide: Model Evaluation and Tuning** Panduan tentang cross-validation, grid search, dan hyperparameter tuning. https://scikit-learn.org/stable/model_selection.html
- **Google Developers – Machine Learning Crash Course: Generalization** Materi interaktif tentang overfitting, underfitting, dan generalisasi. https://developers.google.com/machine-learning/crash-course/generalization/video-lecture
- **Machine Learning Mastery – Tutorials and Guides** Artikel dan contoh Python praktis untuk evaluasi dan tuning model. https://machinelearningmastery.com/

# BAB 7
# Studi Kasus

Pada bab ini, kita akan mempelajari penerapan nyata dari konsep-konsep machine learning yang telah dibahas sebelumnya. Terdapat dua studi kasus:

- Studi kasus klasifikasi menggunakan dataset diabetes (binary classification)
- Studi kasus regresi menggunakan dataset California Housing (regresi harga rumah)

Masing-masing studi kasus akan mencakup tahap:

1. Pembacaan dan eksplorasi data (EDA)
2. Praproses (imputasi, normalisasi, encoding)
3. Pemodelan dan pelatihan
4. Validasi silang dan evaluasi
5. Interpretasi hasil dan perbaikan model

## 7.1 7.1 Studi Kasus Klasifikasi: Prediksi Risiko Diabetes

Dalam studi kasus ini, kita akan membangun model klasifikasi untuk memprediksi apakah seseorang menderita diabetes atau tidak, berdasarkan fitur-fitur medis seperti tekanan darah, kadar glukosa, dan indeks massa tubuh.

Dataset yang digunakan adalah **Pima Indians Diabetes Dataset**, tersedia secara publik di banyak web, misalnya:
https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database

#### 1. Membaca Dataset dan Eksplorasi Awal

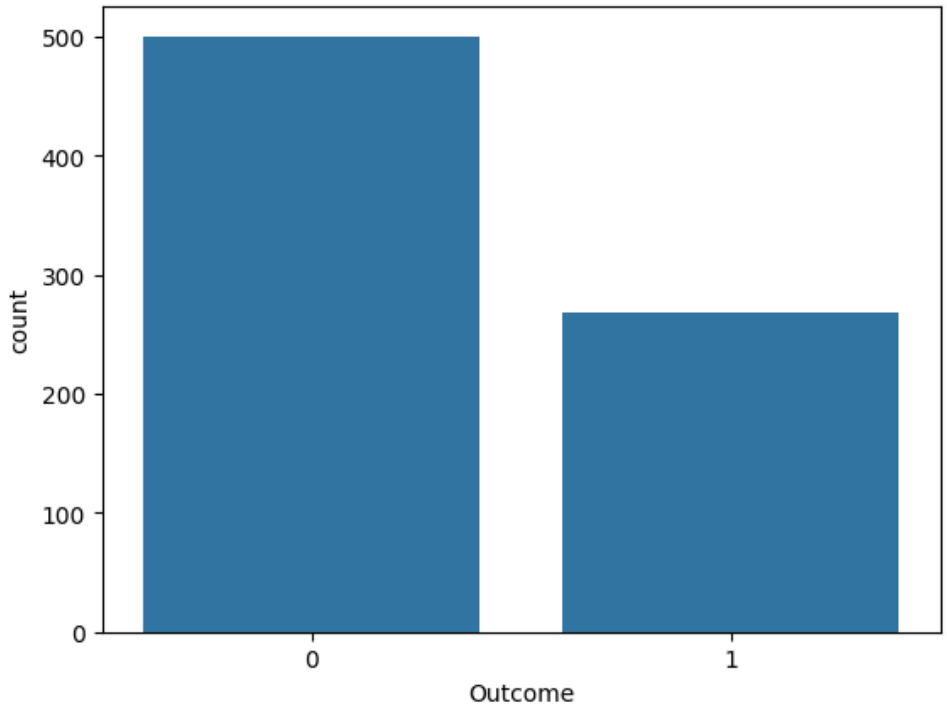
Langkah pertama adalah memuat data dan melihat struktur serta ringkasan statistik:

```
import pandas as pd
import seaborn as sns

df = pd.read_csv("diabetes.csv")
df.head()
df.describe()
df.info()
```

Visualisasi distribusi kelas target:

```
sns.countplot(data=df, x='Outcome')
```

Berikut adalah tampilan hasil plot dari perintah di atas:



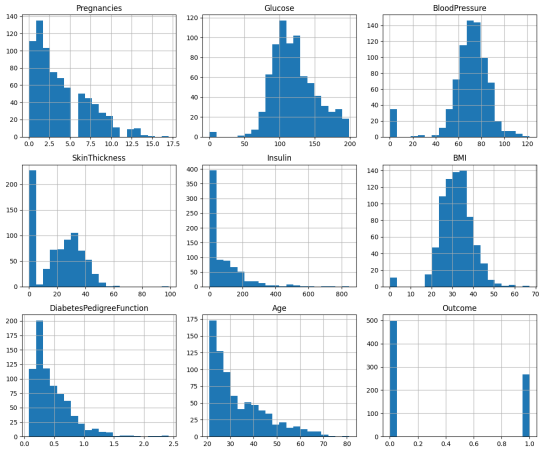

**Gambar 7.1.** Plot histogram kelas "*Outcome*"

**Catatan:** Variabel `Outcome` bernilai 0 (tidak diabetes) atau 1 (diabetes).

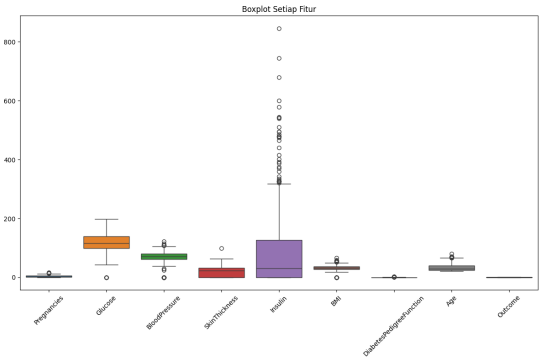
Kita juga bisa melakukan beberapa eksplorasi lain, misalnya melihat distribusi nilai setiap variabel serta melihat korelasi antarvariabel. Berikut adalah kode Python yang dapat digunakan.

```
# Histogram setiap variabel
df.hist(figsize=(12, 10), bins=20)
plt.tight_layout()

# Boxplot setiap fitur
plt.figure(figsize=(12, 8))
sns.boxplot(data=df)
plt.xticks(rotation=45)
plt.title("Boxplot Setiap Fitur")
plt.tight_layout()

# Plot korelasi antarvariabel
plt.figure(figsize=(10, 8))
corr = df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title("Matriks Korelasi Fitur")
plt.tight_layout()
```

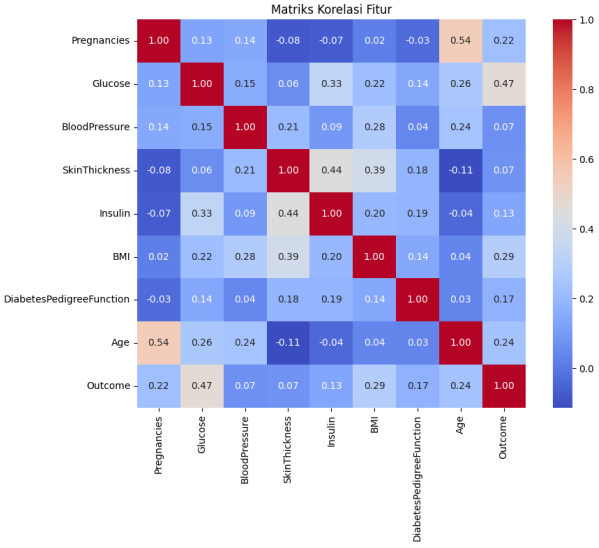
Gambar 7.2 berikut menunjukkan hasil visualisasi eksplorasi data pada dataset Pima Diabetes.




(a) Histogram fitur




(b) Boxplot fitur




(c) Korelasi antar fitur

**Gambar 7.2.** Eksplorasi awal data Pima: histogram, boxplot, dan korelasi antar fitur

### 2. Praproses Data: Penanganan Nilai Tidak Valid dan Outlier

Dataset Pima mengandung nilai nol untuk beberapa fitur seperti `Glucose`, `Insulin`, dan `BMI`. Nilai ini tidak realistis secara medis, sehingga perlu diimputasi/diisikan dengan nilai yang lebih realistis. Berikut langkah praproses yang dapat dilakukan:

- Nilai 0 pada fitur tertentu diganti dengan `NaN`

- Imputasi dilakukan menggunakan nilai **median** dari masing-masing kolom
- Outlier pada kolom `Insulin` dideteksi dengan metode IQR (interquartile range) dan diganti dengan median
- Semua fitur kemudian dinormalisasi menggunakan `StandardScaler`

**Kode Python:**

```
# 1. Identifikasi kolom dengan nilai 0 tidak valid
cols_zero_invalid = ["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]

# Hitung berapa banyak nilai 0
print("Jumlah nilai 0 sebelum imputasi:")
print((df[cols_zero_invalid] == 0).sum())

# 2. Ganti 0 dengan NaN
df[cols_zero_invalid] = df[cols_zero_invalid].replace(0, np.nan)

# 3. Imputasi dengan nilai median
df[cols_zero_invalid] = df[cols_zero_invalid].fillna(df[cols_zero_invalid].median())
```

Selain itu, kita juga melihat bahwa ada banyak nilai *outlier* pada variabel Insulin. Salah satu hal yang dapat kita lakukan adalah misalnya menggantikan nilai *outlier* tersebut dengan nilai lain (misalnya median seperti tadi).

**Kode Python:**

```
# 4. Deteksi dan penanganan outlier pada kolom Insulin (IQR method)
Q1 = df["Insulin"].quantile(0.25)
Q3 = df["Insulin"].quantile(0.75)
IQR = Q3 - Q1
lower = Q1 - 1.5 * IQR
upper = Q3 + 1.5 * IQR

# Tandai outlier insulin
outliers = (df["Insulin"] < lower) | (df["Insulin"] > upper)
print(f"Jumlah outlier pada Insulin: {outliers.sum()}")

# 5. (Opsional) Ganti outlier dengan median (bukan drop)
median_insulin = df["Insulin"].median()
df.loc[outliers, "Insulin"] = median_insulin
```

#### 3. Pelatihan Model

Selanjutnya, kita dapat mulai memodelkan klasifikasi pada dataset ini menggunakan dua metode: Logistic Regression serta SVM. Berikut adalah contoh kode Python untuk melakukan hal ini.

**Kode Python:**

```
X = df.drop(columns="Outcome")
y = df["Outcome"]

# Normalisasi fitur
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Pelatihan & pengujian dengan Logistic Regression + SVM
# Split train/test
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, rando

# Model Logistic Regression
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)
y_pred_logreg = logreg.predict(X_test)

# Model SVM
svm = SVC(kernel="rbf", C=1.0, gamma="scale")
svm.fit(X_train, y_train)
y_pred_svm = svm.predict(X_test)

# Evaluasi
print("Logistic Regression Accuracy:", accuracy_score(y_test, y_pred_logreg))
print(classification_report(y_test, y_pred_logreg))

print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

Hasil dari kode di atas mungkin akan menghasilkan keluaran seperti di bawah ini

```
Logistic Regression Accuracy: 0.6948051948051948
              precision    recall  f1-score   support

           0       0.74      0.81      0.78       100
           1       0.58      0.48      0.53        54

    accuracy                           0.69       154
   macro avg       0.66      0.65      0.65       154
weighted avg       0.69      0.69      0.69       154

SVM Accuracy: 0.7337662337662337
              precision    recall  f1-score   support

           0       0.77      0.85      0.81       100
           1       0.65      0.52      0.58        54
```

```
    accuracy                           0.73       154
   macro avg       0.71      0.68      0.69       154
weighted avg       0.73      0.73      0.73       154
```

Kita dapat juga menampilkan *confusion matrix* dari hasil yang didapatkan sebagai berikut:

```
# Hitung confusion matrix
# Confusion matrices
cm_logreg = confusion_matrix(y_test, y_pred_logreg)
cm_svm = confusion_matrix(y_test, y_pred_svm)

# Buat dataframe dari confusion matrix Logistic Regression
df_logreg_cm = pd.DataFrame(cm_logreg,
                            index=["Actual 0", "Actual 1"],
                            columns=["Predicted 0", "Predicted 1"])
print("Confusion Matrix - Logistic Regression")
print(df_logreg_cm)

# Buat dataframe dari confusion matrix SVM
df_svm_cm = pd.DataFrame(cm_svm,
                         index=["Actual 0", "Actual 1"],
                         columns=["Predicted 0", "Predicted 1"])
print("\nConfusion Matrix - SVM")
print(df_svm_cm)
```

#### 4. Evaluasi dan Interpretasi Hasil

Setelah melakukan pelatihan model, kita mengevaluasi performanya menggunakan metrik berikut:

- **Akurasi**: proporsi prediksi yang benar terhadap seluruh data uji.
- **Precision**: proporsi prediksi positif yang benar-benar benar (menghindari false positive).
- **Recall**: proporsi kasus positif yang berhasil ditangkap oleh model (menghindari false negative).
- **F1-score**: rata-rata harmonik dari precision dan recall, berguna saat distribusi kelas tidak seimbang.
- **Support**: jumlah sampel aktual dari masing-masing kelas.

**Interpretasi:**

- Kelas `0` (negatif diabetes) lebih mudah diprediksi oleh kedua model, dengan recall dan precision yang cukup tinggi.
- Kelas `1` (positif diabetes) lebih sulit diprediksi, terutama oleh Logistic Regression (recall hanya 0.48).

**Tabel 7.1.** Perbandingan Performansi Model

| Metrik | Logistic Regression | SVM |
|---|---|---|
| Akurasi | 0.69 | 0.73 |
| F1-score Kelas 0 | 0.78 | 0.81 |
| F1-score Kelas 1 | 0.53 | 0.58 |
| Macro F1-score | 0.65 | 0.69 |
| Weighted F1-score | 0.69 | 0.73 |

- Model SVM sedikit lebih baik dalam mengenali kelas `1`, serta memiliki akurasi dan F1-score rata-rata yang lebih tinggi.
- Nilai `macro average` menggambarkan performa rata-rata antar kelas, tanpa memperhatikan distribusi.
- Nilai `weighted average` mempertimbangkan distribusi jumlah sampel di setiap kelas (dalam hal ini, kelas 0 lebih dominan).

**Kesimpulan:** model SVM menunjukkan performa yang lebih baik secara keseluruhan dibanding Logistic Regression untuk dataset ini.

#### 5. Refleksi dan Pengembangan Lanjutan

Setelah kita lakukan percobaan di atas, mungkin ada beberapa refleksi dan pertanyaan lanjutan sebagai berikut:

- Apakah fitur-fitur tertentu sangat mempengaruhi hasil?
- Apakah terjadi overfitting?
- Bagaimana hasilnya jika menggunakan model berbeda?

## 7.2 7.2 Studi Kasus Regresi: Prediksi Harga Rumah California

Studi kasus ini bertujuan membangun model regresi untuk memprediksi harga rumah di California berdasarkan fitur-fitur seperti jumlah kamar, kepadatan penduduk, dan luas tanah.

Dataset yang digunakan adalah **California Housing Dataset**, tersedia di `scikit-learn`.

#### 1. Membaca Dataset dan Eksplorasi Awal

**Kode Python:**

```
from sklearn.datasets import fetch_california_housing
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
# Load dataset
data = fetch_california_housing(as_frame=True)
df = data.frame
df.head()
df.describe()
```

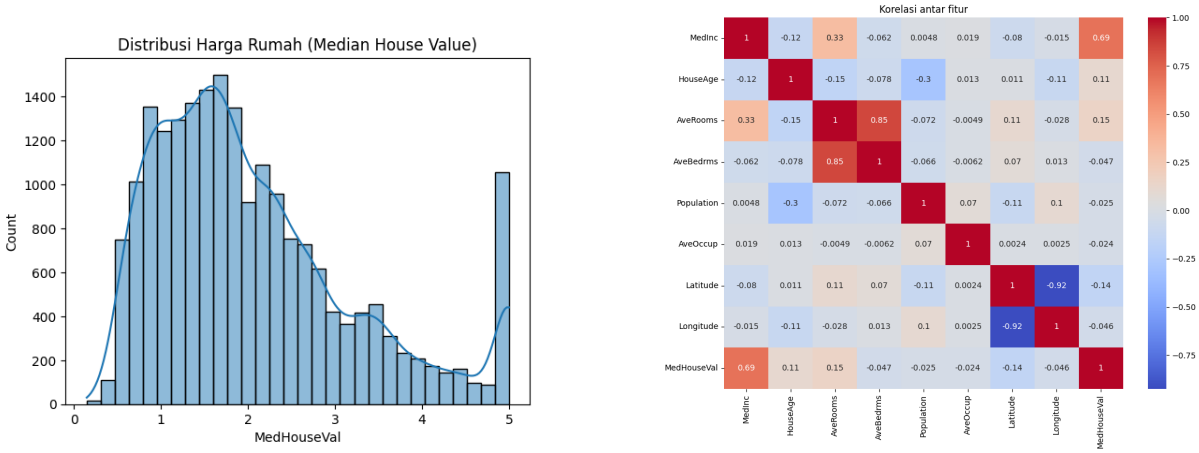
Visualisasi distribusi target:

```
sns.histplot(df['MedHouseVal'], bins=30, kde=True)
plt.title("Distribusi Harga Rumah (Median House Value)")
```

Visualisasi korelasi antar fitur:

```
plt.figure(figsize=(10, 8))
sns.heatmap(df.corr(), annot=True, cmap="coolwarm")
plt.title("Korelasi antar fitur")
plt.tight_layout()
```

Kedua kode di atas menghasilkan visualisasi pada Gambar 7.3.




(a) Histogram nilai target

(b) Korelasi antarfitur

**Gambar 7.3.** Eksplorasi awal data California Housing: distribusi nilai target dan korelasi antar fitur

### 2. Praproses Data

Karena semua fitur numerik dan tidak ada missing values, langkah praproses utama adalah:

- Normalisasi (standardisasi) semua fitur
- Split data menjadi training dan test set

**Kode Python:**

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = df.drop(columns="MedHouseVal")
y = df["MedHouseVal"]

# Normalisasi
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
```

#### 3. Pelatihan Model dan Evaluasi

Kita akan menggunakan dua model regresi: Linear Regression dan Decision Tree Regression. Metrik evaluasi yang digunakan meliputi:

- MAE (Mean Absolute Error)
- MSE (Mean Squared Error)
- RMSE (Root MSE)
- $R^2$ Score

**Kode Python:**

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# Linear Regression
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred_lr = lr.predict(X_test)

# Decision Tree
tree = DecisionTreeRegressor(random_state=42)
tree.fit(X_train, y_train)
y_pred_tree = tree.predict(X_test)

# Evaluasi fungsi
def evaluate(y_true, y_pred, model_name):
    print(f"{model_name} Evaluation:")
```

```
    print("MAE:", mean_absolute_error(y_true, y_pred))
    print("MSE:", mean_squared_error(y_true, y_pred))
    print("RMSE:", np.sqrt(mean_squared_error(y_true, y_pred)))
    print("R²:", r2_score(y_true, y_pred))

evaluate(y_test, y_pred_lr, "Linear Regression")
evaluate(y_test, y_pred_tree, "Decision Tree")
```

#### 4. Visualisasi Hasil Prediksi

**Kode Python:**

```
# Plot prediksi vs aktual
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_lr, alpha=0.5, label="Linear Regression")
plt.scatter(y_test, y_pred_tree, alpha=0.5, label="Decision Tree", color='red')
plt.plot([0, 5], [0, 5], '--k')
plt.xlabel("Harga Aktual")
plt.ylabel("Prediksi")
plt.legend()
plt.title("Prediksi vs Nilai Aktual")
plt.tight_layout()
```

Gambar hasil prediksi akan menunjukkan seberapa dekat hasil prediksi model terhadap nilai sebenarnya. Garis diagonal $y = x$ menunjukkan prediksi sempurna.

**Catatan:** Model Linear Regression umumnya menghasilkan prediksi lebih stabil namun mungkin kurang fleksibel, sedangkan Decision Tree dapat lebih menangkap nonlinearitas tetapi rawan overfitting.

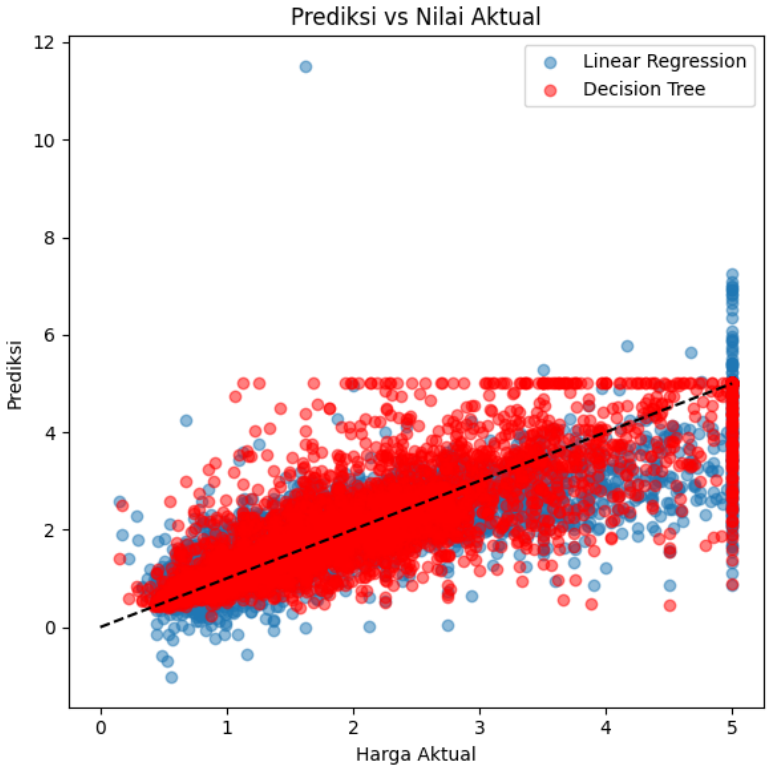
### 7.2.1 7.3 Refleksi dan Langkah Selanjutnya

Melalui dua studi kasus di bab ini, kita telah mempelajari implementasi nyata dari algoritma Machine Learning untuk dua jenis tugas utama:

- **Klasifikasi**: menggunakan dataset Pima Diabetes untuk memprediksi status diabetes seseorang.
- **Regresi**: menggunakan dataset California Housing untuk memprediksi harga rumah berdasarkan fitur numerik.

Setiap studi kasus mencakup tahapan lengkap:

1. Eksplorasi data (EDA) dan visualisasi awal
2. Praproses data, seperti imputasi, normalisasi, dan penanganan outlier
3. Pelatihan model (Logistic Regression, SVM, Linear Regression, Decision Tree)
4. Evaluasi performa model menggunakan metrik yang sesuai




**Gambar 7.4.** Plot prediksi model regresi terhadap nilai aktual

### Apa yang Telah Dipelajari

- Menginterpretasikan distribusi data dan korelasi antar fitur
- Menerapkan praproses data yang sesuai dengan konteks masalah
- Melatih dan mengevaluasi model klasifikasi dan regresi
- Memahami arti dari metrik seperti akurasi, F1-score, MAE, RMSE, dan $R^2$

### Langkah Selanjutnya

Siswa diharapkan dapat:

- Menerapkan pipeline analisis ini pada dataset lain
- Mencoba model tambahan seperti KNN, Random Forest, atau Gradient Boosting
- Mengeksplorasi teknik tuning hyperparameter dan validasi silang yang lebih kompleks

### Sumber Belajar dan Latihan Tambahan

Untuk memperdalam pemahaman dan melatih keterampilan secara langsung, berikut adalah beberapa sumber online yang sangat berguna:

- **Kaggle** (https://www.kaggle.com): platform kompetisi dan pembelajaran ML

dengan banyak dataset, notebook, dan tutorial.

- **Scikit-learn Documentation** (https://scikit-learn.org/stable/documentation.html): referensi lengkap tentang semua fungsi dan model di scikit-learn.
- **Google Colab** (https://colab.research.google.com): lingkungan interaktif untuk menjalankan Python notebook secara gratis dengan GPU/TPU.
- **DataCamp** (https://www.datacamp.com) dan **Coursera** (https://www.coursera.org): platform pembelajaran online dengan kursus interaktif di bidang Data Science dan ML.

**Rekomendasi Latihan:**

- Ambil dataset lain dari Kaggle (misalnya prediksi kanker payudara, kredit, atau harga mobil), lalu lakukan proses serupa: EDA → praproses → modeling → evaluasi.
- Uji perbedaan hasil antara model sederhana dan kompleks (misalnya: Logistic Regression vs Random Forest).
- Bandingkan hasil sebelum dan sesudah hyperparameter tuning.

Dengan menyelesaikan bab ini, siswa telah memperoleh pondasi kuat untuk membangun proyek Machine Learning sederhana secara mandiri, serta melangkah ke studi lanjut seperti deep learning, NLP, atau computer vision yang akan dibahas di modul-modul berikutnya.

# Bibliografi


[DBCR19] David M Diez, Christopher D Barr, and Mine Cetinkaya-Rundel, *Openintro statistics*, 3rd ed., OpenIntro, 2019.

[Dev23] Google Developers, *Machine learning crash course*, https://developers.google.com/machine-learning/crash-course, 2023.

[DFO20] Marc Peter Deisenroth, A Aldo Faisal, and Cheng Soon Ong, *Mathematics for machine learning*, Cambridge University Press, 2020.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT Press, 2016, Book in preparation for MIT Press.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The elements of statistical learning*, Springer Series in Statistics, Springer New York Inc., New York, NY, USA, 2001.

[ID17] Barbara Illowsky and Susan Dean, *Introductory statistics*, OpenStax, 2017.

[JWHT13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, *An introduction to statistical learning*, Springer, 2013.

[Uni23] Duke University, *Data science math skills*, https://www.coursera.org/learn/datasciencemathskills, 2023.

[Was04] Larry Wasserman, *All of statistics: A concise course in statistical inference*, Springer, 2004.